



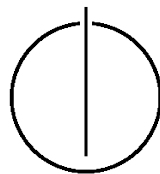
FAKULTÄT FÜR INFORMATIK

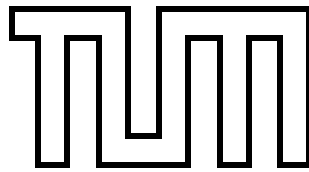
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

**Identifikation, Quantifizierung und
Visualisierung von Komplexitätsmerkmalen
einer Unternehmensarchitektur**

Christian Heindl





FAKULTÄT FÜR INFORMATIK

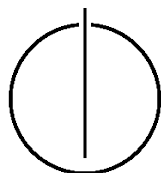
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Diplomarbeit in Informatik

Identifikation, Quantifizierung und Visualisierung von
Komplexitätsmerkmalen einer
Unternehmensarchitektur

Identification, Quantification and Visualization of
Complexity properties of an Enterprise Architecture

Autor: Christian Heindl
Aufgabensteller: Prof. Dr.rer.nat. Florian Matthes
Betreuer: Dipl.Inf. Christopher Schulz
Datum: 15. Januar 2010



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

München, den 15. Januar 2010

.....
(*Unterschrift des Kandidaten*)

“Komplexität ist einfach zu erzeugen, aber schwer zu beherrschen.”
Motto des “TelekomForum“-Jahreskongresses 2006

Danksagung

Bei meinem Betreuer **Christopher Schulz** bedanke ich mich herzlich für die aufopfernde Unterstützung, die vielen Informationsmaterialien und Denkanstöße sowie die häufigen konstruktiven Diskussionen.

Besonderen Dank haben auch meine Eltern verdient, die mich in all den Jahren des Studiums immer tatkräftig unterstützt haben. Meiner Freundin **Katrin Janott** gilt ein ganz herzlicher Dank für ihre tägliche und liebevolle Unterstützung als auch für ihr Vertrauen in den letzten Monaten der Diplomarbeit.

Besonderer Dank gilt auch meinen Geschäftspartnern **Florian Sager** und **Bernhard Heindl**, die im letzten Jahr viele meiner Kundenprojekte übernehmen und Überstunden leisten mussten.

Zusammenfassung

Bei der Betrachtung von Anwendungslandschaften im Unternehmensarchitekturmanagement wird oft von Komplexität gesprochen. Dennoch finden sich in der Literatur wenige konkrete Definitionen von Komplexität für den Kontext von Anwendungslandschaften.

Diese Arbeit erörtert die Ursachen von Komplexität. Ausgehend von Publikationen im zum Unternehmensarchitekturmanagement verwandten Fachbereichen werden Eigenschaften von Anwendungslandschaften herausgefiltert, die als Komplexitätstreiber identifiziert werden können.

Diese Eigenschaften werden dann in einem Informationsmodell gegliedert und zusammengefasst, welches die Quantifizierung der Komplexitätstreiber vereinfacht. Auf Basis dieses Informationsmodell werden Metriken definiert, die als Indikator für Komplexität in Anwendungslandschaften gelten.

Weiterhin werden bestehende Visualisierungsformen für Anwendungslandschaften auf Ihre Tauglichkeit für die Visualisierung der identifizierten Komplexitätstreiber diskutiert. Auch werden neue Wege zur Visualisierung von quantitativen Eigenschaften einer Anwendungslandschaft aufgezeigt und exemplarisch angewandt.

Inhaltsverzeichnis

Zusammenfassung	vii
Übersicht	xi
I Einführung und Grundlagen	1
1 Einführung	3
1.1 Motivation	3
1.2 Aufgabenstellung	3
2 Grundlagen	5
2.1 Unternehmensarchitekturmanagement	5
2.1.1 Unternehmensarchitektur	5
2.1.2 IT-Unternehmensarchitektur und Anwendungslandschaft	7
2.1.3 Softwarekartographie	9
2.1.4 Clusterung nach Domänen	11
2.2 Komplexität	12
2.2.1 Arten von Komplexität	13
2.2.2 Optimale Komplexität und Komplexitätsreduktion	20
2.3 Metriken	21
2.3.1 Grundlagen	21
2.3.2 Anforderungen	24
II Komplexitätsbestimmung	27
3 Komplexitätsmetriken in der Softwaretechnik	29
3.1 Zyklomatische Komplexität	29
3.2 Kohäsion	30
3.3 Kopplung	32
3.4 Halstead Metrik	34
3.5 Strukturelle Komplexität	35
3.6 Softwarearchitekturkomplexität	36
4 Komplexitätsmetriken für Anwendungslandschaften	37
4.1 Identifikation von Komplexitätsattributen	37
4.1.1 Anwendungsbezogene Attribute	37
4.1.2 Schnittstellenbezogene Attribute	43

4.1.3	Abhängigkeitsbezogene Attribute	44
4.1.4	Organisationsbezogene Attribute	45
4.2	Konzeptionelles Informationsmodell	47
4.2.1	Modellbegriff	48
4.2.2	Informationsmodell	49
4.2.3	Abhängigkeit vom Abstraktionsgrad	52
4.3	Quantifikation von Komplexitätsmerkmalen	53
4.3.1	Hinweise zur Notation	54
4.3.2	Metriken	55
4.4	Zusammenfassung der Metriken	77
5	Visualisierung von Komplexitätsmerkmalen	79
5.1	Grundlagen	79
5.2	Visualisierungsansätze für Anwendungslandschaften	81
5.3	Visualisierung der Komplexitätsmerkmale	82
5.3.1	Metrikschicht für Softwarekarten	82
5.3.2	Komplexitätsmetrik Viewpoint	83
5.3.3	Darstellung als TreeMap	84
5.3.4	Visualisierung von mehreren Metriken	85
6	Diskussion und Ausblick	89
	Anhang	93
	A Glossar	93
	Literaturverzeichnis	95

Übersicht

Teil I: Einführung und Grundlagen

KAPITEL 1: EINFÜHRUNG

Führt in die grundlegende Materie dieser Arbeit ein und legt die Ziele dieser Arbeit dar.

KAPITEL 2: GRUNDLAGEN

Vermittelt die Grundlagen zum Unternehmensarchitekturmanagement, das Verständnis von Komplexität und Basiswissen für die Konstruktion und Anwendung von Metriken.

Teil II: Komplexitätsbestimmung

KAPITEL 3: KOMPLEXITÄTSMETRIKEN IN DER SOFTWARETECHNIK

Komplexitätsmetriken aus dem verwandten Bereich der Softwaretechnik werden exemplarisch vorgestellt, um sie später auf Anwendungslandschaften transformieren zu können.

KAPITEL 4: KOMPLEXITÄTSMETRIKEN FÜR ANWENDUNGSLANDSCHAFTEN

In diesem Kapitel werden Komplexitätsmerkmale einer Anwendungslandschaft identifiziert und anschließend als Komplexitätsindikatoren quantifiziert.

KAPITEL 5: VISUALISIERUNG VON KOMPLEXITÄTSMERKMALEN

Auf Basis der entwickelten Kennzahlen werden Möglichkeiten der Visualisierung aufgezeigt und sowohl in bestehende Rahmenwerke integriert als auch neuartige Varianten entwickelt.

KAPITEL 6: DISKUSSION UND AUSBLICK

Die zuvor erarbeiteten Ergebnisse werden in einer Zurückschau diskutiert und Anregungen für anknüpfende und erweiternde Arbeiten gegeben.

Teil I

Einführung und Grundlagen

1 Einführung

1.1 Motivation

In modernen Unternehmen wachsen die Anforderungen an die IT: von einer unterstützenden Organisationseinheit ausgehend hat sich die IT inzwischen zu einem integralen Bestandteil von Geschäftsmodellen gewandelt. Die gestiegenen Anforderungen wie etwa eine zügige Anpassung der IT an geänderte Geschäftsprozesse, eine allzeit zu gewährleistende Verfügbarkeit, ein möglichst kostengünstiger Betrieb bei gleichzeitig höherer Funktionsintegrität und Automatisierung haben Einfluss auf das Management des IT-Betriebs.

Im Besonderen treffen diese Anforderungen alle geschäftsunterstützenden Softwareanwendungen, die im Gesamten die Anwendungslandschaft bilden. Durch zahlreiche (meist parallele) Projekte wird die Anwendungslandschaft verändert; zumeist ist es jedoch unklar, welche Änderungen der Projekte Komplexität in die Anwendungslandschaft induzieren oder welche Änderungen die Komplexität verringern. Die Komplexität beeinflusst allerdings nachhaltig die Möglichkeiten zur Wartung und Weiterentwicklung der Anwendungen selbst, aber auch die der Anwendungslandschaften.

Um den großen Herausforderungen Preis-, Wettbewerbs- und Flexibilitätsdruck [ALL09] in Unternehmen besser begegnen zu können, lohnt sich ein aktives Komplexitätsmanagement. Auf Ebene der Anwendungslandschaften existieren dazu allerdings noch keine quantitative Ansätze, die zum Verständnis der Komplexität beitragen. Dazu müssen zunächst die Grundlagen für eine detaillierte Betrachtung geschaffen werden, die Komplexität auf Ebene der Anwendungslandschaft verständlich und greifbar macht. Erst dann kann ein aktives Komplexitätsmanagement greifen, das durch Metriken zielgerichtet gesteuert werden kann.

1.2 Aufgabenstellung

Ziel dieser Arbeit ist es daher Merkmale von Anwendungslandschaften zu identifizieren und deren Relevanz als Treiber der Komplexität zu untersuchen. Dabei sind Konzepte verwandter Fachgebiete (z.B. Softwaretechnik, Graphentheorie) zu studieren und ihre Portierbarkeit in den Kontext von Anwendungslandschaften zu prüfen.

In einem zweiten Schritt müssen die Attribute quantitativ erfass- und vergleichbar gemacht werden. Dazu sollen geeignete Metriken und Indikatoren entwickelt und diskutiert werden. Grundlage hierfür soll ein konzeptuelles Modell sein, das für alle Metriken angewendet und passend auf den Bedarf einer einzelnen Metrik verschlankt werden kann.

Abschließend sollen auch Vorschläge zur Visualisierung der Komplexitätsindikatoren erarbeitet werden.

2 Grundlagen

2.1 Unternehmensarchitekturmanagement

Jedes (Groß-)Unternehmen ist vergleichbar mit einer Stadt: Es gibt Bewohner, die Ihre Interessen vertreten, einen Stadtrat, der Entscheidungen über die Fortentwicklung der Stadt trifft, es gibt Infrastrukturen, die das Leben in der Stadt beeinflussen, es gibt Beziehungen zwischen den Bewohnern und Verbindungen zu anderen Städten.

Gerade in großen Städten kommt es leicht zu Wildwuchs: es bilden sich Bezirke, mit sehr guter Infrastruktur heraus, andere Bezirke werden vernachlässigt und beginnen marode zu werden; es kommt zu Konflikten zwischen den Bürgern, zwischen den Stadtbezirken und zwischen den Bezirksausschüssen. Der Stadtrat kann leicht den Überblick über die Zusammenhänge zwischen den Bezirken, deren Infrastruktur und den zugewiesenen Aufgaben und Ziele des Bezirks verlieren.

Ähnliche Strukturen sind auch in Unternehmen zu finden: es gibt einen Vorstand, verschiedene Organisationseinheiten, Mitarbeiter, Gewerkschaften, Personalräte und viele weitere Gremien und Stakeholder. Daher kann im Unternehmen auch der Überblick schnell verloren gehen; die Zusammenhänge und Abhängigkeiten der Prozesse untereinander, die grundlegende Infrastruktur vor allem im IT-Bereich und die Ausrichtung der Organisationseinheiten an den Zielen des Unternehmens gerät aus dem Blick. Um dies im Unternehmen besser zu gestalten, flexibler auf die immer häufiger vorkommenden Änderungen der Geschäftsanforderungen reagieren zu können und um ein gezielteres Handeln innerhalb der gesamten Unternehmung zu fördern werden Initiativen zum Unternehmensarchitekturmanagement (*engl.:* Enterprise Architecture Management) eingeführt.

Idealerweise wird durch das Unternehmensarchitekturmanagement die Ausrichtung der IT-Infrastruktur an den strategischen Geschäftszielen adressiert: Wie können die Geschäftsziele möglichst optimal von der IT-Infrastruktur unterstützt werden? Wie kann die IT-Infrastruktur weiterentwickelt werden, um dem Unternehmen nicht nur Kosten zu verursachen, sondern zu einer effizienteren Abwicklung der Geschäftsprozesse zu verhelfen oder gar zusätzliche Umsätze zu generieren?

Daneben werden aber auch technologische Ziele bei der Einführung des Unternehmensarchitekturmanagements verfolgt. So erwarten Unternehmen davon auch einen hohen Beitrag zur Standardisierung von IT-Prozessen, die Reduktion von IT-Kosten, die Integration von Anwendung und Daten als auch eine bessere technologische Zukunftssicherheit und Stabilität [Lo08].

2.1.1 Unternehmensarchitektur

Ein zentraler Bestandteil um die Aufgaben des Unternehmensarchitekturmanagement zu erfüllen ist das Verständnis der Unternehmensarchitektur. Der IEEE-Standard 1471-2000 definiert den Begriff der (Software-)Architektur wie folgt:

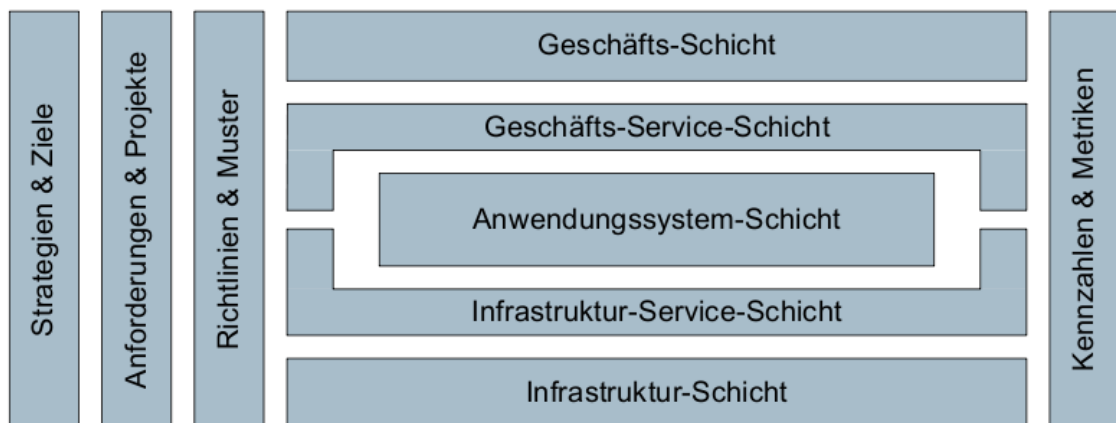
Definition: Architektur The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [MEH07]

Übersetzung Der fundamentale Aufbau eines Systems, verkörpert in seinen Komponenten, ihren Beziehungen zueinander und den Prinzipien, die sein Design und seine Entwicklung bestimmen. [Ke06]

Es ist zunächst nicht relevant ob eine Architektur von extern geplant in ein System induziert wurde oder ob ein System ohne Planung entstanden ist. Es ist lediglich die Struktur des Systems als Architektur relevant. Die Architektur wohnt dem System inne und nicht ggf. vorhandenen externen Architekturbeschreibungen.

Im Kontext von Unternehmen müssen in einer Unternehmensarchitektur also alle zugehörigen Elemente, seien es Prozesse, Organisationsstruktur, Anwendungssysteme und Infrastrukturkomponenten in einer Architektur(beschreibung) geordnet werden. Wittenburg definiert eine Unternehmensarchitektur wie folgt:

Definition: Unternehmensarchitektur Die Unternehmensarchitektur ist die kohärente und ganzheitliche Architektur eines Unternehmens, die nicht nur die Informationstechnologie sondern ebenso betriebswirtschaftliche Elemente umfasst. Dabei umfasst die Architektur nicht nur die einzelnen Elemente des Unternehmens selbst, wie beispielsweise die Organisationsstruktur, die Geschäftsprozesse, die Anwendungen und die Infrastrukturelemente, sondern auch ihre Verbindungen und Querschnittselemente, wie Strategien und Ziele, Anforderungen und Projekte, Richtlinien und Muster sowie Kennzahlen und Metriken. [Wi07]



([Wi07])

Abbildung 2.1: Schichten und Querschnittsfunktionen einer Unternehmensarchitektur

Abbildung 2.1 diversifiziert eine Unternehmensarchitektur in verschiedenen Schichten und Querschnittsfunktionen, die im Unternehmensarchitekturmanagement zu berücksichtigen sind. Es werden dabei fünf Hauptschichten identifiziert, die nach Wittenburg [Wi07] folgende Semantik besitzen:

- **Geschäfts-Schicht** beinhaltet Geschäftsprozesse und Aufbauorganisation
- **Geschäfts-Service-Schicht** verbindet die Geschäfts-Schicht mit der Anwendungssystemschicht und definiert Geschäftsobjekte, -services und deren Service-Level.
- **Anwendungssystem-Schicht** erbringt die von der Geschäfts-Service-Schicht geforderten Geschäftsservices und implementiert die Geschäftsobjekte mit Hilfe von IT-Anwendungen.
- **Infrastruktur-Service-Schicht** verbindet die Anwendungssystem-Schicht mit der Infrastruktur-Schicht in dem erforderliche Infrastruktur-Services und Service-Level für die Anwendungen definiert werden.
- **Infrastruktur-Schicht** erbringt die von der Infrastruktur-Service-Schicht geforderten Infrastruktur-Services (z.B. Middleware, Hardware) und ermöglicht so erst die Funktion der darüberliegenden Schichten.

Daneben werden die Querschnittsfunktionen Strategien & Ziele , Anforderungen & Projekte , Richtlinien & Muster als auch Kennzahlen & Metriken aufgezeigt, die die Elemente aller Schichten beeinflussen.

Für die vorliegende Arbeit ist jedoch nur ein Teilaspekt interessant: hauptsächlich ist die Anwendungssystem-Schicht Ziel aus Sicht der Kennzahlen & Metriken Funktion Gegenstand dieser Arbeit. Durch Metriken erhält das Unternehmensarchitekturmanagement erst die Informationen und Aussagen, um das Unternehmen richtig steuern zu können. Daher wird zunächst in der folgenden Sektion die IT-Unternehmensarchitektur und die Anwendungssystem-Schicht, die eng verknüpft mit dem Begriff der Anwendungslandschaft ist, erklärt und beleuchtet.

2.1.2 IT-Unternehmensarchitektur und Anwendungslandschaft

Der Bereich der IT-Unternehmensarchitektur ist eine Untermenge der Unternehmensarchitektur, welche sich auf die Bereiche konzentriert, in denen IT-Systeme (Anwendungen, Infrastruktur) relevant sind. In modernen Unternehmen sind allerdings fast alle Prozesse sehr stark von der Unterstützung durch IT-Systeme durchdrungen, so dass eine Unterscheidung zwischen IT-Unternehmensarchitektur und Unternehmensarchitektur nicht sofort ersichtlich ist. Der Hauptunterschied liegt in der Kompetenz Geschäftsstrategien und -ziele zu definieren als auch Entscheidungen bzgl. der Aufbauorganisation zu treffen; diese Kompetenzen liegen klar im Verantwortungsbereich der Unternehmensarchitekten und bedingen keinen Einfluss der IT-Unternehmensarchitekten. Keller definiert IT-Unternehmensarchitektur wie folgt:

Definition: IT-Unternehmensarchitektur "IT-Unternehmensarchitektur ist derjenige Teil der Unternehmensarchitektur, den die IT-Funktion in einem Unternehmen ausmachen darf, ohne wegen Kompetenzüberschreitung von anderen Unternehmenseinheiten außerhalb der IT erfolgreich politisch attackiert zu werden. Im besten Fall sind IT-Unternehmensarchitektur und Unternehmensarchitektur identisch und einheitlich organisiert." [Ke06]

Die Anwendungslandschaft als Teil der IT-Unternehmensarchitektur ist jedoch unabhängig vom Aufbau des Unternehmens:

Definition: Anwendungslandschaft Als Anwendungslandschaft (Informationssystemlandschaft, IS-Landschaft) wird die gewachsene Gesamtheit aller Anwendungen eines Unternehmens verstanden, die zur Unterstützung der Durchführung von Geschäftsprozessen betrieben werden. [De06]

Eine alternative Definition, die in dieser Arbeit genutzt wird und zusätzlichen Fokus auf die Kommunikationsbeziehungen zwischen den Anwendungen legt, stammt von Wittenburg:

Definition: Anwendungslandschaft Die Anwendungslandschaft ist die Gesamtheit der betrieblichen Anwendungen inklusive der Kommunikationsbeziehungen zwischen den Anwendungen in einem Unternehmen. [Wi07]

Die Anwendungslandschaft ist Basis aller IT-Unterstützung des Unternehmens. Sie umfasst alle Anwendungen eines Unternehmens als auch deren Kommunikationsbeziehungen untereinander. Ordnet man die Anwendungslandschaft in das Schichtenmodell aus Abbildung 2.1 ein, so ist die Anwendungssystem-Schicht sehr eng verknüpft mit der Anwendungslandschaft.

Auf der Seite zur Geschäfts-Services-Schicht werden Beziehungen zwischen Anwendungen und den Geschäftsobjekten hergestellt. Beispielsweise wird referenziert, welche Anwendungen welchen Geschäftsprozess unterstützt oder welche Anwendung welcher Organisationseinheit dient, von welcher Organisationseinheit sie weiterentwickelt oder betrieben wird.

Auf der Seite zur Infrastruktur-Schicht werden hingegen Beziehungen zwischen den Anwendungen und den virtuellen oder physischen Infrastrukturelementen hergestellt, z.B. welche Anwendung auf welcher Hardware ausgeführt wird. Gerade letzteres ist in den aufkommenden Cloud-Umgebungen aber oft nicht abzubilden.

Daneben unterliegt die Anwendungslandschaft einer zeitlichen Entwicklung: IT-Projekte erweitern und verändern die Anwendungslandschaft. So können neue Anwendungsversionen entwickelt werden, Anwendungen zusammengeführt oder abgeschaltet werden. Auch die Vielzahl unterschiedlicher Projektabhängigkeiten induzieren dem Management der IT-Unternehmensarchitektur Komplexität.

Für diese Arbeit wichtigster Bestandteil der Anwendungslandschaft sind aber die Anwendungen selbst und deren Beziehungen untereinander. Wie in späteren Kapiteln noch festgestellt wird, sind gerade diese Beziehungen für die Komplexität in Anwendungslandschaften verantwortlich.

Um die Menge der Anwendungen der Anwendungslandschaft zu erfassen, muss noch der Begriff der Anwendung definiert werden. In dieser Arbeit wird Anwendung wie folgt verstanden:

Definition: Anwendung Eine Anwendung ist ein in Software implementiertes System, das mindestens einen Geschäftsprozess unterstützt.

Analog zu [Wi07] zählen daher Middleware- und Datenbankkomponenten nicht als Anwendung, sondern als Infrastruktur, die Services für die Anwendungen erbringen.

2.1.3 Softwarekartographie

Um Anwendungslandschaften visualisieren zu können hat sich ein eigener Forschungsbereich entwickelt: die Softwarekartographie. In Anlehnung an die im Städtebau verwendeten Bebauungspläne wurden deren Konzepte auf Anwendungslandschaften übertragen. Spezielle Softwarekarten sollen Überblick über die Anwendungslandschaft bieten. Für verschiedene Aspekte oder Interessen - sogenannte Concerns - gibt es verschiedene Ausprägungen von Softwarekarten.

Definition: Softwarekarte Eine graphische Repräsentation der Anwendungslandschaft oder Ausschnitte selbiger. Eine Softwarekarte setzt sich zusammen aus einer oder mehrerer Schichten, die verschiedene Aspekte visualisieren. [MW04a]

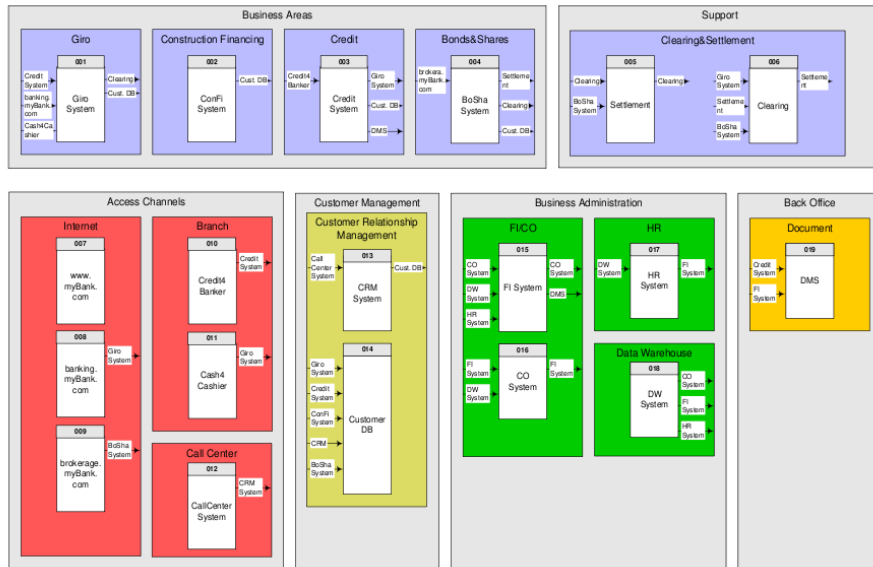
Wittenburg [Wi07] hat dazu einen systematischen Ansatz zur Softwarekartographie entwickelt. Er hat einen Grundstein für die automatische Generierung von Softwarekarten gelegt. Mit der Definition eines abstrakten Informationsmodells und einer Methode zur Transformation von konkreten Modellinstanzen in konkrete (unterschiedliche) Softwarekarten müssen nun lediglich Informationen über die Anwendungslandschaft in einem Informationsmodell gesammelt werden. Anschließend können Softwarekarten für unterschiedliche Concerns automatisch generiert werden. Dazu stehen verschiedene Werkzeuge zur Verfügung, z.B. das SyCaTool [Er06] [SYC09].

Als Beispiel zeigt etwa Abbildung 2.2 eine Clusterkarte, die die Beziehungen zwischen Anwendungen und den Organisationseinheiten herstellt, in denen eine Anwendung genutzt wird. Zudem werden die Abhängigkeiten zwischen den Anwendungen dargestellt. Diese Darstellung erleichtert das Erfassen der Zusammenhänge zwischen Anwendungen und vor allem zwischen Anwendung und der nutzenden Organisationseinheit(en).

Ein weiteres Beispiel einer Softwarekarte ist die in Abbildung 2.3 dargestellte Intervallkarte. Im Gegensatz zur Clusterkarte liegt hier der Fokus vor allem auf der Darstellung der zeitlichen Abhängigkeiten: Wann wird eine Anwendungen in einer bestimmten Version eingesetzt? Gerade zur Planung von mehreren parallelen Projekten (Projektportfoliomanagement, PPM) ist diese Karte hilfreich, um Projekte zu identifizieren, die konkurrierende Änderungen an der Anwendungslandschaft vornehmen möchten oder um Abhängigkeiten zwischen Projekten leichter aufzudecken.

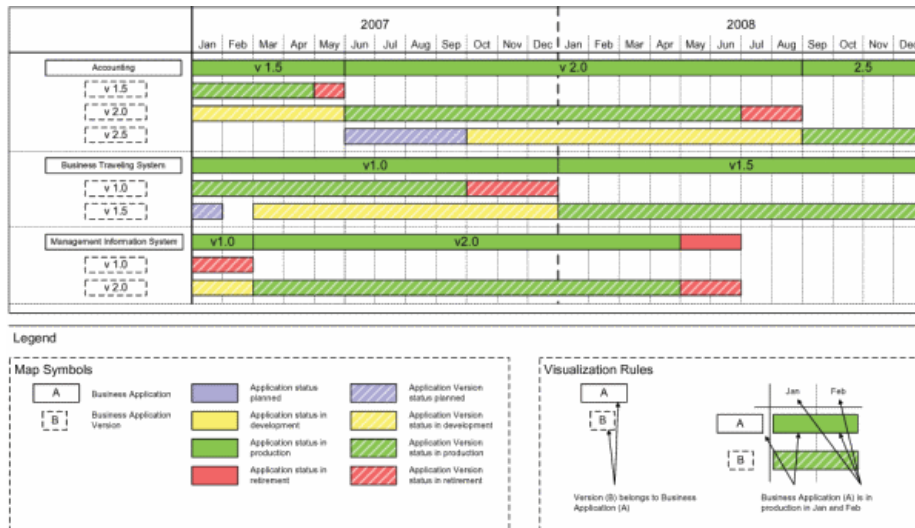
Für die hier vorliegende Arbeit von besonderer Relevanz ist die Darstellung als Graph wie in Abbildung 2.4 zu sehen. Unabhängig von Beziehungen zu Geschäftsprozessen oder Organisationseinheiten werden die Anwendungen und ihre Abhängigkeiten untereinander als Graph dargestellt. Diese Darstellung ist besonders geeignet um automatisierte Berechnungen auf dem Graphen ausführen zu lassen; dabei ist es aber erforderlich, dass der Graph nicht nur visuell vorliegt, sondern in einem entsprechenden Graphmodell computerlesbar vorliegt.

2 Grundlagen



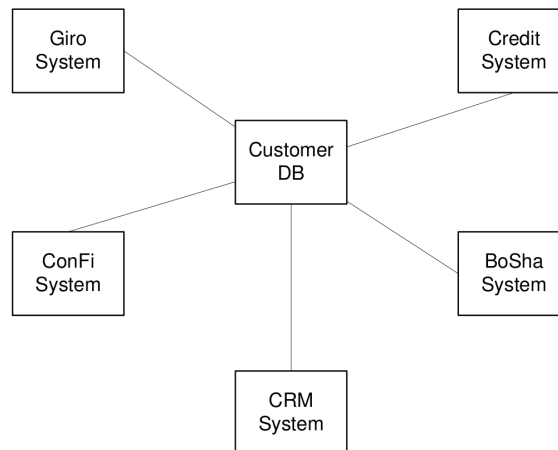
(SEBA-Master Vorlesung, SS 2009, TUM, sebis)

Abbildung 2.2: Clusterkarte



(SEBA-Master Vorlesung, SS 2009, TUM, sebis)

Abbildung 2.3: Intervallkarte



(SEBA-Master Vorlesung, SS 2009, TUM, sebis)

Abbildung 2.4: Anwendungslandschaft als Graph

2.1.4 Clusterung nach Domänen

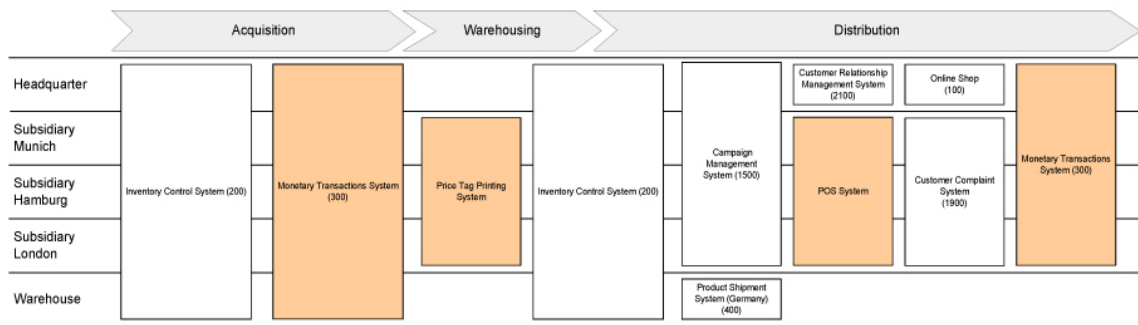
Oftmals ist nur die Betrachtung eines Teilbereichs einer Anwendungslandschaft notwendig, da z.B. ein Projekt nur Auswirkungen auf einen kleinen Teilbereich hat. Dazu können Anwendungslandschaften in unterschiedliche Domänen aufgespalten werden. Eine Domäne ist eine thematische Gruppierung einer Anzahl von Anwendungen, die auch hierarchisch gegliedert sein kann, d.h. eine Domäne selbst kann wiederum aus mehreren Domänen zusammengesetzt sein. Durch die Betrachtung lediglich begrenzter Ausschnitte der Anwendungslandschaft wirkt sich dies positiv auf die Kosten der Datenerhebung im Zuge der Visualisierung oder Anwendung von Metriken auf der Anwendungslandschaften aus.

Eine einfache Aufspaltung wurde bereits in der Clusterkarte in Abbildung 2.2 gezeigt: Die Aufspaltung der Anwendungslandschaft in Organisationseinheiten. Wenn ein Projekt lediglich Elemente einer einzigen Organisationseinheit betrifft, die keine Relationen zu anderen Anwendungen anderer Organisationseinheiten besitzen, so ist der Teilbereich der Anwendungslandschaft ausreichend für eine hinreichend genaue Betrachtung.

Eine andere Variante ist in Abbildung 2.5 gezeigt: Hier wird nach Geschäftsprozessen und Organisationseinheiten segmentiert. Sofern lediglich Anwendungen im Geschäftsprozessschritt "Acquisition" betroffen sind, reicht eine Betrachtung aller diesem Geschäftsprozessschritt aus.

Prinzipiell sind Segmentierungen nach anderen Gesichtspunkten ebenso möglich, genauso wie eine hierarchische Anordnung der Domänen: Organisationseinheiten können so noch weiter gegliedert werden. Ein Beispiel ist in Abbildung 2.6 wiedergegeben.

Neben der Aufspaltung nach Organisationseinheiten oder Geschäftsprozessen existieren eine Reihe weiterer Clustering Möglichkeiten: Domänen können sowohl nach fachlichen aber z.B. auch technischen Gesichtspunkten gebildet werden. Beyer et al. [BN05] beispielsweise versuchen Softwareartefakte anhand der Häufigkeit ihrer Änderungen zu clustern; analog könnten in Anwendungslandschaften auch Anwendungen anhand ihrer Änderungsfrequenz in Domänen organisiert werden. Aier und Winter versuchen sogar eine automatische Berechnung von Domänen auf Basis eines Modells der Abhängigkei-



(SEBA-Master Vorlesung, SS 2009, TUM, sebis)

Abbildung 2.5: Prozessunterstützungskarte



(Eigene Darstellung)

Abbildung 2.6: Hierarchische Domains einer Anwendungslandschaft

ten zwischen den Anwendungen. Zur Berechnung dieser verwenden sie ihr Modell der Alignment-Architektur [AW09].

Wie später noch vorgestellt wird, ist diese Aufspaltung nach Domänen sehr hilfreich bei der Betrachtung der Komplexität von Anwendungslandschaften.

2.2 Komplexität

Das Wort Komplexität stammt vom lateinischen Wort *complexari* ab, welches mit umfassen oder umarmen übersetzt werden kann [DUD01]. Etwas "komplexes" umfasst also mehrere Elemente.

Mehrere zusammenhängende Elemente können als ein System, das begrenzt ist und von seiner Umwelt abgegrenzt werden kann, angesehen werden. Übertragen auf Systeme, bezeichnet Komplexität diejenige Eigenschaft eines Systems, welche eine genaue Abschätzung des Gesamtverhaltens erschwert, selbst wenn vollständige Informationen über die einzelnen Subsysteme und deren Abhängigkeiten untereinander bekannt sind [Ha08]. In einem komplexen System treten also Seiteneffekte auf, die entweder nicht oder nur sehr schwer beschreibbar sind.

Auf allgemeiner Ebene ist eine Definition schwer zu fassen. Der französische Philosoph Edgar Morin hat eine negative Definition für Komplexität gegeben: "Complexity is what is not simple" ("Komplexität ist, was nicht einfach ist") [Mo74]. Was aber ist einfach? Wie ist Einfachheit definiert?

Wissenschaftstheoretiker weisen hier auf Ockhams Rasiermesser (*engl.*: Occam's razor) hin: dieses wissenschaftliche Prinzip fordert zur Sparsamkeit bei der Erklärung von Sachverhalten auf. Mit je weniger Axiomen ein Sachverhalt erklärt und vollständig beschrieben werden kann, desto besser sei die Beschreibung [Ba09]. Einfachheit ist damit also mit der

Kürze einer Beschreibung gegeben. Im Umkehrschluss ist also ein Sachverhalt dann komplex, wenn dieser nur durch viele Axiome beschrieben werden kann.

Zuvor wurde bereits genannt, dass ein System dann komplex ist, wenn das Systemverhalten von mehr abhängt als aus den Einzelverhalten der Subsysteme abgeleitet werden kann. Auf der anderen Seite stellt Morin fest, dass ein komplexes System nicht nur mehr als die Summe seiner Subsysteme sein muss, sondern dass ein Gesamtsystem auch einer Substraktivität unterliegen kann: unter Umständen können Eigenschaften eines Subsystems von einem anderen Subsystem blockiert und unterdrückt werden [Mo05]. Es wird dabei auch von Dominanz bzw. Interferenz von Subsystemen gesprochen. Daher ist es ebenso wichtig das Verhalten des Gesamtsystems zu begreifen als auch das der Subsysteme um ein insgesamtes Verständnis entwickeln zu können.

In einem weiteren Definitionsversuch von Komplexität stellt Morin vor allem einen Zusammenhang mit Unsicherheit, Zufall und Unordnung her [Mo05]. Diese Begriffe überlappen sich zugegebenermaßen, allerdings greift auch Diederichs diesen Zusammenhang [Di04] auf und beschreibt ebenfalls eine Abhängigkeit der Komplexität von Zufall und Ordnung. Der Zusammenhang dazu wird allerdings schnell intuitiv klar: einem Betrachter eines komplexen Systems erscheint das beobachtete Verhalten des System oft nicht nachvollziehbar; also zufällig oder ungeordnet.

Komplexität ist demnach vom Betrachter abhängig. Für einen Automechaniker ist ein Motor klar und er kennt die möglichen Verhaltensweisen. Für einen Laien hingegen stellt ein Motor ein komplexes System dar. Diese Subjektivität wird auch in vielen Veröffentlichungen belegt: es wird zwischen subjektiver und objektiver Komplexität unterschieden [GSK05]. Die subjektive Komplexität hängt dabei vom Betrachter ab, dessen Wissen und Kenntnisstand [Ba02]. Bei der subjektiven Komplexität wird die Komplexität eines Systems intuitiv definiert. Dabei ist meist der (intuitive) Grad der Komplexität abhängig vom notwendigen Aufwand ein bestimmtes System vollständig zu verstehen und es beherrschen zu können [Ba02], [Li08]. Im Gegenteil dazu versuchen objektive Definitionen von Komplexität diese unabhängig vom Betrachter anhand von Systemeigenschaften zu messen.

In (Fremd-)Wörterbüchern findet man beim Eintrag Komplexität auch die Bedeutung "Vielschichtigkeit" [DUD94]. Komplexität ist nicht einfach nur durch Vielschichtigkeit definiert, vielmehr ist der Begriff selbst vielschichtig und lässt sich nicht allgemein definieren. Es existieren daher für verschiedene Domänen unterschiedliche, spezielle Definitionen und Arten von Komplexität. Im Folgenden werden einige dieser Arten von Komplexität aufgezeigt und näher erläutert.

2.2.1 Arten von Komplexität

Komplexität von Algorithmen

In der theoretischen Informatik wird der Begriff Komplexität hauptsächlich bei der Bewertung und Diskussion von Algorithmen verwendet. Bei der Bewertung werden Algorithmen anhand ihres Ressourcenverbrauchs in Abhängigkeit von der Eingabelänge klassifiziert. Es können unterschiedliche Ressourcen betrachtet werden: am gebräuchlichsten ist

die Betrachtung der Laufzeit (Zeitkomplexität) und des Speicherverbrauchs (Platzkomplexität). Es können aber auch andere Ressourcen wie die Anzahl der Vergleiche, die Anzahl der Multiplikationen oder die Schachtelungstiefe zur Bewertung herangezogen werden.

Die Komplexität eines Algorithmus beschreibt die Schwierigkeit ein bestimmtes Problem zu lösen; gleichzeitig beschreibt die Komplexität eines Algorithmus auch dessen Skalierbarkeit. Sie beantwortet die Frage nach dem Verhalten eines Algorithmus bezüglich einer bestimmten Resource, wenn sich die Eingabelänge verändert: Führt beispielsweise eine Verdoppelung der Eingabelänge auch zu einer Verdoppelung der Laufzeit oder zu einer signifikant längeren Laufzeit? In dieser Betrachtung können allerdings wiederum verschiedene Aspekte analysiert werden. Oft unterscheidet sich nämlich der Ressourcenverbrauch im schlechtesten Fall (*engl.*: worst-case) deutlich vom Ressourcenverbrauch im durchschnittlichen Fall (*engl.*: average-case). So haben viele Algorithmen oft eine sehr schlechte worst-case Laufzeit werden aber dennoch produktiv eingesetzt, da ihre average-case Laufzeit besser als bei anderen Algorithmen (mit besserem worst-case Verhalten) ist.

Die Komplexität eines Algorithmus wird als Funktion in Abhängigkeit von der Eingabelänge angegeben. Es ist jedoch schwierig eine allgemeingültige Funktion zu finden, die den genauen Ressourcenverbrauch angibt; ebenso wäre es schwierig eine genaue, allgemeingültige Funktion mit der eines anderen Algorithmus direkt zu vergleichen. Daher wird vielmehr das asymptotische Verhalten der von einem Algorithmus verwendeten Resource aufgezeigt. Zur Notation werden die Landau-Symbole genutzt. Damit ist es auch möglich obere und untere Schranken anzugeben.

Analog beschäftigt sich die Komplexitätstheorie mit der Bewertung und Klassifizierung von Problemen: es werden Probleme in verschiedene Klassen eingeteilt. Interessant ist dabei aber zunächst die Einteilung in lediglich zwei verschiedene Komplexitätsklassen: NP und P . P ist die Klasse von Entscheidungsproblemen, die mit einer deterministischen Turingmaschine in polynomieller Zeit gelöst werden können; oft werden die Probleme dieser Klasse auch praktisch lösbar bezeichnet. NP hingegen bezeichnet eine Klasse von Entscheidungsproblemen, die von einer nicht-deterministischen Turingmaschine in polynomieller Zeit entschieden werden können, d.h. praktisch nicht effizient lösbar sind. Eine große, bisher ungelöste Frage der Informatik ist jedoch, ob N und NP nicht vielleicht dieselben Elemente beinhalten. Für eine detaillierte Betrachtung sei hier allerdings auf die Literatur verwiesen, z.B. auf die Arbeiten von Cook [Co71] oder Karp [Ka72].

Beispiel: Tiefensuche in einem Wald In der Graphentheorie müssen oft alle Knoten in einem Wald aus Bäumen gefunden werden, die bezüglich eines Startknotens zum gleichen Baum gehören. Dazu kann die Tiefensuche eingesetzt werden. Dabei wird zunächst ein Startknoten ausgewählt (der zum gewünschten Baum gehört) und dann jeweils alle Kinder markiert, die von diesem Startknoten aus erreichbar sind. Dies wird jeweils rekursiv für jeden markierten, noch nicht bearbeiteten Knoten durchgeführt. Listing 2.1 zeigt einen Tiefensuch-Algorithmus in Pseudo-Code.

Listing 2.1: Tiefensuche (DFS) in einem Baum

```
1 void DFS(graph G, node v) {  
2     v.visited = true;  
3  
4     for(c in v.children) {
```

```

5     if(!c.visited) {
6         DFS(G, c);
7     }
8 }
9 }
10
11
12 graph g = random_graph(); // Baum generieren
13 node s = random_node(g); // Startknoten
14
15 for(v in s.children) {
16     if(!v.visited) {
17         DFS(G, v);
18     }
19 }

```

Zur Analyse der Zeitkomplexität (worst-case) gilt folgende Abschätzung: der Algorithmus muss im schlechtesten Fall (z.B. Wald bestehend aus einem Baum, der keine Äste besitzt) jeden Knoten und jede Kante besuchen. Sofern V = Menge aller Knoten und E = Menge aller Kanten ist, ist die Zeitkomplexität $O(|V| + |E|)$; die Laufzeit wächst also mit längerer Eingabe linear.

Produktkomplexität

Bei der Entwicklung und Produktion von technikenthaltenden Produkten wird verstärkt der Begriff des Komplexitätsmanagements geprägt, welches durch die gestiegene Komplexität technischer Produkte notwendig wurde. In der Literatur werden verschiedene Ansätze diskutiert und Sichtweisen von Produktkomplexität und Komplexität im Produktlebenszyklus gegeben.

In [Sc01] wird Komplexität durch die Komplexitätstreiber Masse und Dynamik definiert: Ersteres wird zum einen durch die Vielzahl und die Vielfalt der verschiedenen Elemente und Beziehungen charakterisiert, letzteres, die Dynamik über Vieldeutigkeit und vor allem die Veränderlichkeit von Fakten während des Produktlebenszyklus. Dabei spielen hauptsächlich Markteinflüsse eine Rolle: so führt der Konkurrenzdruck zu einer höheren Dynamik um Produkte schneller auf den Markt bringen zu müssen und erhöht somit automatisch die Komplexität des Produktentwicklungsprozesses.

Kolmogorow-Komplexität (Informationskomplexität)

In der Informationstheorie wird eine Definition verwendet die u.a. auf Kolmogorow zurückgeht. Demnach ist Komplexität über die Länge der kürzesten, aber dennoch vollständigen Systembeschreibung definiert. Voraussetzung ist, dass eine Systembeschreibung als Zeichenkette oder Bitfolge dargestellt werden kann [MV97], [Di04].

Grundidee ist, dass der Informationsgehalt einer Informationseinheit (= Zeichenkette) durch deren Länge beschrieben werden kann. Dies wird dann als Komplexität der Information verstanden. Der Ansatz geht davon aus, dass ein einfaches System weniger Sys-

tembeschreibung benötigt, als ein komplexes. Man vergleiche dazu auch das Prinzip des Ockham'schen Rasiermesser (vgl. Abschnitt 2.2).

Zur korrekten Bestimmung dieser Komplexitätsart ist es allerdings notwendig, dass die Zeichenkette nicht mehr weiter komprimierbar ist. Das Problem liegt aber in der Tatsache, dass ein Beweis der Minimalität (bzw. der Unkomprimierbarkeit) unmöglich ist. Es kann lediglich das Gegenteil bewiesen werden, in dem ein Gegenbeweis angestrengt wird [Di04]. Auf der anderen Seite ist aber auch der Beweis der Vollständigkeit einer Systembeschreibung schwierig. Bei künstlich erzeugten Systemen mag dies noch gelingen, bei empirisch beobachteten Systemen kann aber eine Vollständigkeit nicht nachgewiesen werden.

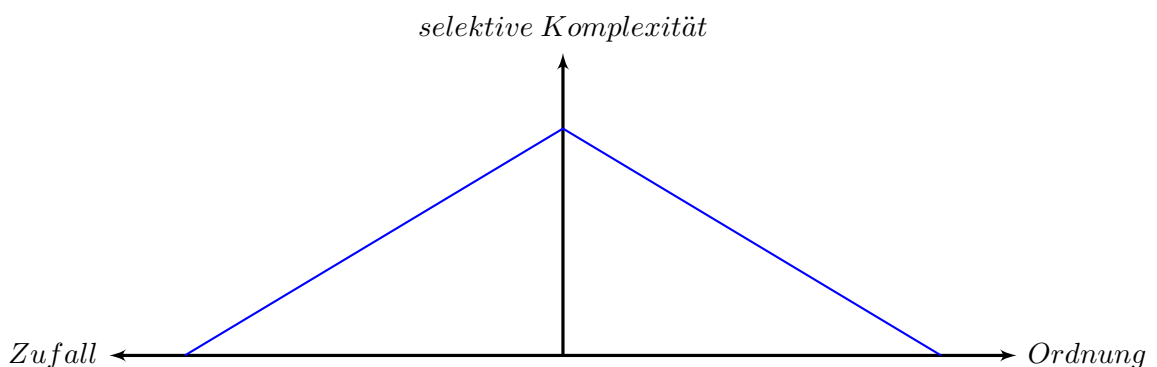
Selektive Komplexität (Systemtheoretische Komplexität)

Diederichs [Di04] schlägt das Konzept der selektiven Komplexität vor. Diese basiert auf einigen systemtheoretischen Überlegungen:

Ein System, das aus n Elementen besteht, kann aus Sicht der Kombinatorik maximal $n^2 - (n - 1)!$ Abhängigkeiten besitzen. Ein System ist normalerweise nicht sinnvoll, wenn alle diese Abhängigkeiten existieren. Für ein sinnvolles System werden die Abhängigkeiten daraus selektiert.

Betrachtet man dies im Kontext der systemtheoretischen Umwelt sind zwischen System und Umwelt theoretisch unendlich viele Abhängigkeiten denkbar, die entstehen wenn man alles mit allem verknüpfen würde. Im System sind jedoch lediglich bestimmte Abhängigkeiten aus allen möglichen ausgewählt und bilden so eine höhere Ordnung bzw. eine Struktur.

Diederichs geht davon aus, dass nicht die Zunahme der Größe die Komplexität wachsen lässt, sondern die Zunahme von Struktur bzw. der Einschränkungen, denen die Auswahl der möglichen Abhängigkeiten unterliegt. Sowohl rein durch Zufälligkeiten bestimmte Strukturen als auch vollkommen bestimmte Strukturen führen zu einer niedrigen Komplexität. Die Zufälligkeit der Abhängigkeiten muss jedoch erkannt werden. "Die Eingrenzung der Möglichkeiten bestimmt, ob hohe Komplexität entstehen kann" [Di04]. Abbildung 2.7 zeigt die Zusammenhänge zwischen Komplexität und vorhandener Struktur in einem System.



(Eigene Darstellung in Anlehnung an [Di04])

Abbildung 2.7: Selektive Komplexität

Softwarekomplexität

Im Bereich der Softwaretechnik wird Komplexität in der Literatur umfassender diskutiert. Im Laufe der Zeit sind verschiedenste Metriken v.a. im Unterbereich Codekomplexität entstanden. Zuse gibt in [Zu94] einen Überblick verfügbarer Metriken; einige davon werden noch in Abschnitt 3 vorgestellt.

Dennoch ist der Zugang zur Komplexität im Bereich Softwaretechnik nicht einfach. Softwarekomplexität wird meist zunächst in verschiedene Unterkategorien aufgeteilt: So unterscheidet Lilienthal zunächst zwischen probleminhärenter (auch: essentielle) und lösungsabhängiger (auch: akzidentelle) Komplexität [Li08].

Erstere zielt dabei auf die Komplexität ab, die dem durch die Software zu lösenden Problem innewohnt. Im Softwareentwicklungsprozess lässt sich die probleminhärente Komplexität normalerweise nicht beeinflussen; lediglich durch grundsätzliche Änderungen der Softwareanforderungen kann die probleminhärente Komplexität verringert oder erhöht werden.

Die Ursachen der probleminhärenten Komplexität liegen dabei jeweils in der Anwendungsdomäne: Während für die Planung der Vorlesungen einer Universität die Vielzahl an Studenten, Studiengängen, Professoren und Hörsäle bewältigt werden muss ist etwa bei einem Routenplaner die Komplexität algorithmisch und in den zu Grunde liegenden Daten beheimatet.

Die lösungsabhängige Komplexität kann im Gegensatz dazu jedoch beeinflusst werden, da diese z.B. durch ungünstige Entwurfsentscheidungen entsteht. Eine gute, einfache, dem Problem angepasste Softwarearchitektur trägt daher zu einer niedrigen lösungsabhängigen Komplexität bei.

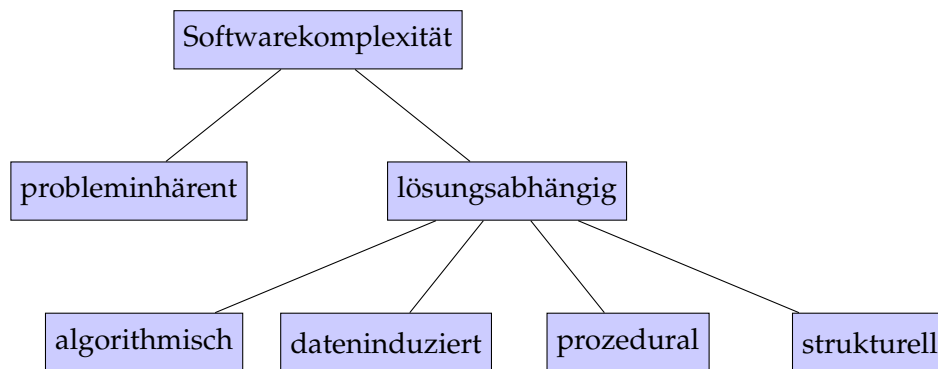
Unnötige lösungsabhängige Komplexität entsteht beispielsweise dann, wenn in einer Software zur Raumplanung einer Universität zusätzlich in den Entwurfsentscheidungen davon ausgegangen wird, dass es feste Klassenverbände gibt, um die Software z.B. auch an Schulen einsetzen zu können. Dies führt zusätzliche Komplexität in den Programmcode und die Dokumentation ein. Auch für den Anwender wird zusätzliche lösungsabhängige Komplexität erzeugt.

Weiterhin ist die lösungsabhängige Komplexität von der probleminhärenten Komplexität beeinflusst bzw. von dieser induziert. In Abbildung 2.8 ist die Aufgliederung noch detaillierter dargestellt: Die lösungsabhängige Komplexität wird nämlich wiederum in verschiedene Arten unterteilt, wie z.B. von Jones in [Jo91] ausgeführt wird. Die Darstellung ist derweil nicht vollständig; viele weitere Unterscheidungen können eingeführt werden wie etwa bei Jones nachgelesen werden kann.

Besonderes Augenmerk soll jedoch der strukturellen Komplexität zu Teil werden: Diese Art der Komplexität ist wesentlich für die Softwareentwicklung und wird vor allem auch im späteren Verlauf der Arbeit als Grundlage für die Komplexität einer Anwendungslandschaft herangezogen.

Strukturelle Komplexität

Strukturelle Komplexität in Softwaresystemen ist durch Muster und Abhängigkeiten eines System bedingt [Jo91]. Sowohl die strukturelle Bindung innerhalb eines Moduls (Kohäsi-



(Eigene Darstellung in Anlehnung an [Li08], [Jo91])

Abbildung 2.8: Einflüsse auf die Softwarekomplexität

on) als auch die strukturelle Bindung zwischen Modulen (Kopplung) determinieren die Komplexität [Di04].

Ähnlich wie bei der selektiven Komplexität bestimmt das Vorhandensein von Mustern die Komplexität: tritt in einem System eine Struktur stark und wiederkehrend auf, so ist dessen Komplexität gering. Tritt nur wenig Struktur auf, so ist die Komplexität des Systems hoch. So bedingt beispielsweise eine hohe Ordnung wie sie z.B. in natürlichen Kristallen vorzufinden ist eine niedrige Komplexität.

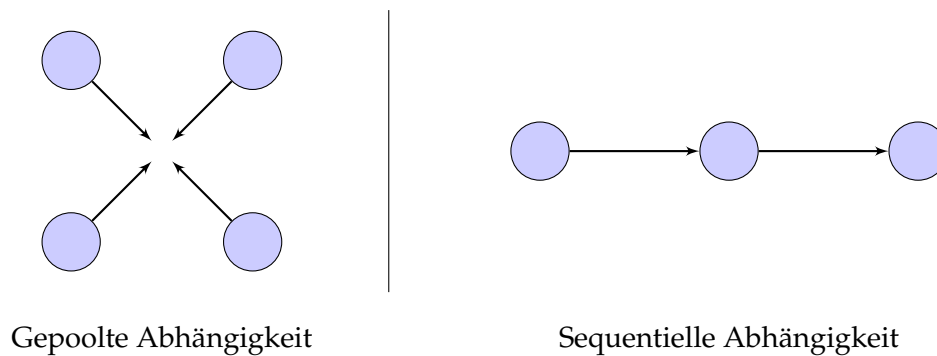
Viele Metriken zur Komplexitätsbestimmung in Softwareprodukten basieren auf den Ideen der strukturellen Komplexität. Dies wird durch die leichte Abbildung von Programmcode und Softwarearchitekturen auf einfache Graphstrukturen bedingt; zudem eignen sich solche Graphen gut zur Verarbeitung durch Graphalgorithmen. Die Abbildung auf einen Graphen ist jedoch für jede Metrik unterschiedlich; je nach Fokus wird Quellcode auf Graphen portiert, Klassen und deren Abhängigkeiten gemappt oder auf Ebene von Modulen/Paketen abstrahiert.

Nach Backlund [Ba02] sind neben der Anzahl der Teilsysteme v.a. die Anzahl und Art der Abhängigkeiten zwischen den Strukturelementen Komplexitätstreiber. Williams [Wi03] unterscheidet daher drei verschiedene Abhängigkeitstypen:

Gepoolte Abhängigkeit Die erste und einfachste Form nach Williams [Wi03] ist die gepoolte Abhängigkeit, wie in Abbildung 2.9 links dargestellt. Dabei existieren mehrere Subsysteme die unabhängig voneinander einen Beitrag zum Gesamtsystem leisten.

Sequentielle Abhängigkeit Im Gegensatz zur gepoolten Abhängigkeit existieren die Subsysteme jetzt nicht mehr unabhängig voneinander sondern interagieren miteinander und hängen voneinander ab. So bilden die Subsysteme bei sequentieller Abhängigkeit eine einfache Kette; jedes Subsystem ist jeweils abhängig von einem anderen wie in Abbildung 2.9 rechts dargestellt.

Eine Beispiel aus der Praxis ist hier etwa der Pipe-Mechanismus in POSIX basierenden Systemen: Ein Programm ist lediglich darauf angewiesen Ausgaben der Standardausgabe an ein anderes Programm weiterzugeben. Die Programme bilden so eine einfache Kette, welche lediglich abhängig von der Standardeingabe-Schnittstelle abhängig sind.



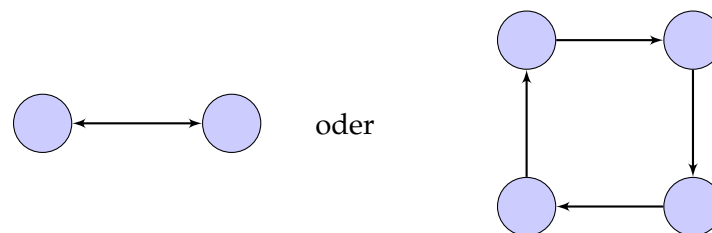
Gepoolte Abhängigkeit

Sequentielle Abhängigkeit

(Eigene Darstellung in Anlehnung an [Wi03])

Abbildung 2.9: Gepoolte und sequentielle Abhängigkeit

Reziproke Abhängigkeit Eine erweiterte Form sind reziproke Abhängigkeiten wie in Abbildung 2.10 illustriert. Es treten nicht nur Abhängigkeiten in eine Richtung auf sondern auch bidirektionale oder zyklische Abhängigkeiten. So sind Rückkopplungen möglich, die das Systemverhalten entscheidend beeinflussen können. Gerade diese machen reziproke Abhängigkeiten zu einem Komplexitätstreiber.



oder

(Eigene Darstellung in Anlehnung an [Wi03])

Abbildung 2.10: Reziproke Abhängigkeit

Verstehenskomplexität

Eine sehr stark subjektive Form der Komplexität ist die Verstehenskomplexität. Verstehenskomplexität entsteht dann, wenn eine Person eine Aufgabe mit oder an einem Softwaresystem ausführt [Li08]. Dabei bezieht die Person einen individuellen Grad an Komplexität auf die Aufgabe oder das Softwaresystem.

Die Verstehenskomplexität ist abhängig von der Fachkompetenz, der Erschließbarkeit und der Softwarekomplexität. Für alle drei Faktoren können Maßnahmen zur Komplexitätsreduktion ergriffen werden: mit Weiterbildung der Mitarbeiter, gute Dokumentation und optimale Werkzeugunterstützung sind nur einige der möglichen Maßnahmen genannt.

Für das Verständnis der Erschließbarkeit hilft eine Betrachtung der strukturbildende Prozesse im menschlichen Gehirn aus Sicht der kognitiven Psychologie. Die kognitive Psychologie beschäftigt sich mit den Methoden, mit der sich das menschliche Gehirn Wissen

aneignet und Problemlösungen erzeugt. Aus der kognitiven Psychologie sind drei strukturbildende Prozesse bekannt [Li08]:

- **Chunking** bezeichnet das Aggregieren und Gruppieren von kleineren Informationseinheiten zu größeren Informationseinheiten im Gehirn, welche später als Ganzes wieder abgerufen werden können.
- **Bildung von Hierarchien** bezeichnet die Fähigkeit des menschlichen Gehirns beim Verstehen und Abspeichern von komplexen Informationen diese hierarchisch (z.B. in Form von Bäumen) zu organisieren. Ein Beispiel dafür ist der erfolgreiche Einsatz von Mind-Map-Methoden zur Wissensspeicherung.
- Der **Aufbau von Schemata** hilft beim Versuch komplexe Systeme zu Verstehen: hält sich ein System an gewohnte Schemata oder gewohnte Denkmuster so kann ein System schneller verstanden werden als ein System das nach (dem Betrachter) unbekanntes Schemata arbeitet. Die Existenz von wiederkehrenden Schemata in einem System vereinfacht daher das Verstehen desselbigen und lässt es weniger komplex erscheinen.

Die Erschließbarkeit kann durch Unterstützung dieser kognitiven Prozesse verbessert werden: so können wiederkehrende Muster und Hierarchiestrukturen das Verstehen einer Anwendung, von Softwarecode und einer Architektur erleichtern. Dies führt zu einer geringeren Verstehenskomplexität.

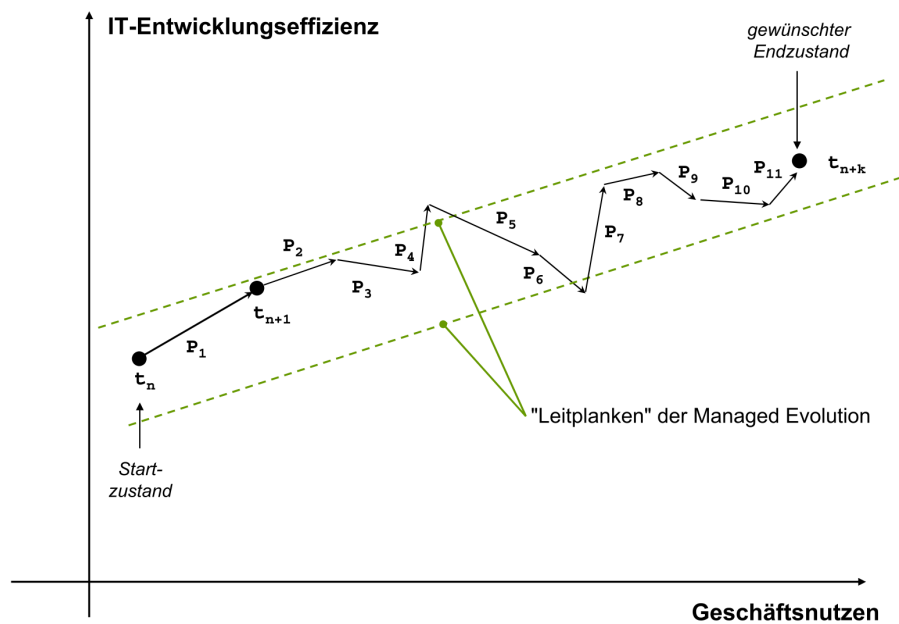
Idealerweise wird beim Verstehen eines Sachverhalts sowohl die linearen, verbalen Fähigkeiten der linken Gehirnhälfte mit den holistischen, nicht-linearen Fähigkeiten der rechten Gehirnhälfte kombiniert [Hu09].

2.2.2 Optimale Komplexität und Komplexitätsreduktion

Oft ist es nicht dienlich eine minimale Komplexität anzustreben. Wie Schuh in [Sc01] beschreibt, erfordern Entitäten im wirtschaftlichen Handeln eine Komplexität ungleich der minimalen Komplexität ist.

Als Beispiel soll die Entwicklung eines Videorekorders dienen. Werden etwa die Funktionen des Videorekorders auf das Abspielen und Aufnehmen von Videos auf Videokassetten reduziert, wird ein geringer Komplexitätsgrad erreicht. Allerdings werden dadurch auch Kundenbedürfnisse an das Produkt nicht berücksichtigt, da z.B. eine Timer-Funktion nicht berücksichtigt wurde und das Aufnehmen von Fernsehsendungen nur im Beisein des Benutzers funktioniert. Ein derart abgespeckter Videorekorder wird vom Markt kaum akzeptiert werden und zu einem finanziellen Mißerfolg führen.

Was ist jedoch der Grad der optimalen Komplexität? Allgemein lässt sich dies nicht bestimmen und muss für jedes System individuell festgelegt werden. Einen Ansatz zur Erreichung eines optimalen Komplexitätsgrades für Anwendungslandschaften wurde von Murer et al. [MWF08] vorgestellt: die Managed Evolution. Ziel ist es die Komplexität von Änderungen an der Anwendungslandschaft kontrolliert innerhalb eines geeigneten Rahmens zu halten, dem Komplexitätskorridor. Dafür wird für jedes Projekt dessen Beitrag zu Geschäftsnutzen und IT-Entwicklungseffizienz durch geeignete Metriken bewertet. Jedes



(MWF08)

Abbildung 2.11: Evolutionskorridor der Managed Evolution

Entwicklungsprojekt soll in beiden Dimensionen einen positiven Wertbeitrag erbringen. Ein Beispiel für die Evolution einer Anwendung ist in Abbildung 2.11 dargestellt.

Nach Schmidt [Sc08] kommt es bei gewachsenen Systemen immer zu einem Komplexitätswachstum, sofern keine expliziten Maßnahmen zu deren Reduktion angesetzt werden. Die Managed Evolution adressiert dies und versucht eine durch Komplexität verursachte Starrheit (auch: Resistance-To-Change, Verkrustung, Architekturerosion) zu vermeiden. Es ist daher notwendig bei der Fortentwicklung von Systemen ausgewogen sowohl in Feature-Erweiterungen als auch in architekturerhaltende Projekte zu investieren.

2.3 Metriken

Für ein genaues Verständnis eines Sachverhalts müssen Sachverhalte analysiert und dessen Eigenschaften quantifiziert werden. Erst durch die Quantifizierung und Messung der Eigenschaften durch definierte, einheitliche Metriken können genaue Aussagen über einen Sachverhalt getroffen werden; ein effektives Management erfordert eine Grundlage bestehend aus Kennzahlen und Methoden. Tom DeMarcos berühmter Ausspruch "What you cannot measure, you cannot control" [De82] weist kompakt und einprägsam auf diesen Umstand hin. Um Komplexität daher einmal beherrschen zu können, müssen zunächst Metriken definiert werden, die Komplexitätsattribute messen können. Im Folgenden werden zunächst die Grundlagen zum Aufstellen geeigneter Metriken gelegt, die für die Quantifizierung von Komplexität in Anwendungslandschaften in Abschnitt 4.3 herangezogen werden.

2.3.1 Grundlagen

Definition: Metrik (auch: Kennzahl) Metriken erfassen Sachverhalte quantitativ und in konzentrierter Form. Merkmale einer Metrik sind Informationscharakter, Quantifizierbarkeit und Informationsform. Der Informationscharakter zeigt, dass eine Beurteilung von Sachverhalten und Zusammenhängen möglich ist. Quantifizierbarkeit bedeutet, dass Sachverhalte auf metrischen Skalen gemessen werden können und dadurch "genaue" Aussagen möglich sind. Die Informationsform führt dazu, dass komplexe Sachverhalte komprimiert und auf einfache Art dargestellt werden [Ku07].

Der Informationscharakter einer Metrik drückt aber nicht nur aus, dass eine Beurteilung eines Sachverhalts möglich wird, sondern auch dass zunächst Informationen über den Sachverhalt vorhanden sein müssen. Eine Metrik besitzt daher auch einen bestimmten Informationsbedarf, der spezifiziert welche Eigenschaften des Sachverhalts ermittelt werden müssen, um die Metrik anwenden zu können.

Weiterhin impliziert eine Metrik auch die Existenz von metrischen Skalen, die nicht nur für die durch den Informationsbedarf definierten Eigenschaften gelten, sondern auch für die Repräsentation der Metrik gelten. Es können allerdings jeweils unterschiedliche metrische Skalen verwendet werden. Die Berechnungsvorschrift einer Metrik muss allerdings die metrischen Skalen berücksichtigen und dem Ergebnis eine entsprechend korrekte Einheit zuweisen.

Kütz schlägt vor Metriken in einem Steckbrief zu dokumentieren. Dort werden sowohl allgemeine Rahmendaten wie Bezeichnung, Beschreibung und Stakeholder als auch Beschreibungen zu Datenermittlung (Datenquellen, Messverfahren, Messpunkte), Berechnungsvorschrift und Präsentation (Darstellung, Aggregationsstufen) beschrieben und formal festgehalten. Dies erleichtert vor allem die Kommunikation und Weitergabe von Kennzahlen. Details hierzu können in [Ku07] nachgelesen werden.

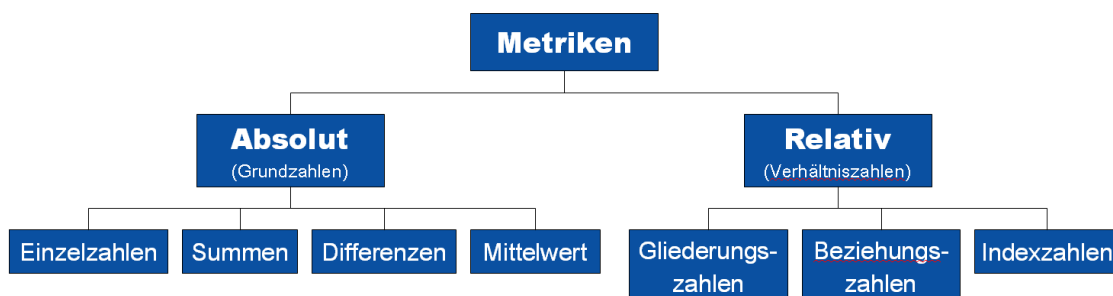
Aus der Messtheorie sind verschiedene sogenannte Skalenniveaus bekannt: Eigenschaften eines Messobjekts können in verschiedene Skalenniveaus eingeordnet werden. Allgemein werden vier Skalenniveaus unterschieden [Ka03]:

- **Nominalskala** Nominalskalen definieren eine einfache Kategorisierung der Eigenschaften eines Messobjekts. Elemente können nicht verglichen werden, sondern lediglich einer Kategorie zugeordnet werden. Beispiel wäre die Kategorisierung nach Geschlecht; Messobjekte können dort nur den Werten "Mann" oder "Frau" zugeordnet werden.
- **Ordinalskala** Ordinalskalen erlauben zusätzlich zur Kategorisierung der Eigenschaften eines Messobjekts den Vergleich zwischen den Kategorien, also ob ein Objekt einer Kategorie größer, kleiner oder gleich einem anderen Objekt ist. Beispielsweise ist die Klassifizierung von Automobilen im Sinne der Umweltplakette eine Ordinalskala: Die Kategorien rot, gelb oder grün sind klar definiert und erlauben einen Vergleich. Rot ist schlechter als eine gelbe Plakette, welche wiederum schlechter ist als eine grüne.

- **Intervallskala** Zusätzlich zu den Eigenschaften von Ordinalskalen bieten Intervallskalen noch eine Aussage zur Differenz zwischen den Skaleneinheiten. Bei Datumsangaben etwa kann die Differenz zweier Angaben angegeben werden, z.B. "2 Tage", "3 Jahre 25 Wochen und 4 Tage".
- **Verhältnisskala** Das höchste Maß an Mess- und Vergleichbarkeit erlauben Verhältnisskalen. Zusätzlich zur Differenzbildung existiert in Verhältnisskalen ein (absoluter) Nullpunkt. Beispiele hierfür sind z.B. Eigenschaften wie Gewicht, Zeitdauer in Sekunden, Temperatur (in Kelvin) oder Entfernung zwischen zwei Orten.

Im Folgenden interessieren zur Bestimmung der Komplexität von Anwendungslandschaften hauptsächlich Eigenschaften, deren Quantifizierung mindestens den Anforderungen einer Ordinalskala genügen.

Da sich Metriken aus verschiedenen Eigenschaften zusammensetzen, gibt es auch für Metriken eine ähnliche Einordnung. Zunächst wird zwischen absoluten und relativen Metriken unterschieden. Erstere werden auch Grundzahlen genannt, letztere auch Verhältnisskennzahlen. Beide Kategorisierungen können noch weiter aufgespalten werden, wie in Abbildung 2.12 dargestellt.



(Eigene Darstellung in Anlehnung an [Pr08])

Abbildung 2.12: Systematisierung von Metriken

Absolute Metriken oder Grundzahlen sind Metriken, die unabhängig und ohne Vermengung mit anderen Metriken sind und geben einen Sachverhalt direkt wieder. Es können folgende Untertypen unterschieden werden [Pr08]:

- **Einzelzahlen** geben Eigenschaften eines Sachverhalts ohne Veränderung wieder, z.B. Anzahl Fahrgäste einer U-Bahn.
- **Summen** summieren gleichartige Eigenschaftswerte mehrerer zum Sachverhalt gehöriger Entitäten auf, z.B. Summe der offenen Posten
- **Differenzen:** Bilden die Differenz aus mehreren Eigenschaftswerten, z.B. Guthaben als Differenz zwischen Gutschriften und Abbuchungen.

- **Mittelwerte:** Bilden den Mittelwert von gleichartigen Eigenschaftswerten mehrerer zum Sachverhalt gehöriger Entitäten, z.B. Durchschnittliche Länge eines Telefongesprächs

Relative Metriken (auch: Verhältniszahlen) hingegen sind Metriken, die unterschiedliche Sachverhalte miteinander in Bezug setzen. Verhältniszahlen verwenden dazu oben definierte Grundzahlen. Auch hier lassen sich verschiedene Untertypen unterscheiden [Pr08]:

- **Gliederungszahlen** geben eine Beziehung zwischen Teilmenge und Gesamtmenge wieder. So werden vor allem strukturelle Eigenschaften der Gesamtmenge deutlich. Beispiel: Eigenkapitalquote (im Vergleich zum Gesamtkapital)
- **Beziehungszahlen** geben eine Beziehung zwischen ungleichartigen Eigenschaften des Sachverhalts wieder. Beispiel: Nettogewinn pro Aktie, Kilometer pro Stunde
- **Indexzahlen** vereinfachen das Ablesen der zeitlichen Entwicklung einer (absoluten) Metrik indem zu einem Stichtag der Wert der (absoluten) Metrik als Referenz gesetzt wird und im Folgenden die Werte relativ zur Referenz angegeben werden. Beispiel: Wertentwicklung eines Aktiendepots.

Manchmal reicht eine Metrik allerdings nicht aus um einen Sachverhalt umfassend darzustellen und als Grundlage einer Entscheidung zu verwenden. Metriken können daher zu einem Metriksystem zusammengefasst werden. Ziel ist dabei Mehrdeutigkeiten und Abhängigkeiten zwischen Einzelmetriken zu erfassen und einschätzbar zu machen [Ku07].

2.3.2 Anforderungen

Für die Definition von sinnvollen Kennzahlen sind mehrere Anforderungen zu erfüllen. Die folgende Liste enthält dabei die wichtigsten Punkte, die in der Literatur diskutiert werden:

- **Zielorientierung** Nach Kütz [Ku07] sind Kennzahlen jeweils von einem bestimmten Concern eines Stakeholders abgeleitet und dienen dazu, diesen Concern kontrollier- und steuerbar zu machen. Preißler betont zudem, dass nicht nur ein Concern adressiert wird, sondern dass durch die Metrik ein konkreter Bezug zwischen Geschäftszielen und kritischen Erfolgsfaktoren hergestellt werden soll [Pr08].
- **Korrektheit und Vollständigkeit der Modellbildung** Nur mit einem vollständigen und korrekten Modell des Sachverhalts ermöglicht eine Metrik oder ein Metriksystem eine fehlerfreie Interpretation und in der Folge richtige Entscheidungen [Ku07].
- **Minimalität** Ein Metriksystem muss nach [Ku07] minimal sein, um die Kosten für die Datenerhebung zu minimieren aber auch um das Metriksystem für einen Stakeholder überschaubar zu halten. Dies impliziert für eine einzelne Metrik ebenfalls, dass nur Eigenschaften des Sachverhalts betrachten sollte, die für den Zweck der Metrik relevant sind; zudem sollte die Berechnungsvorschrift einfach sein, so dass ein Stakeholder die Zusammensetzung der Metrik verstehen kann.

- **Wirtschaftlichkeit** Preißler [Pr08] fordert für Metriken im Umfeld des wirtschaftlichen Handelns zudem ein positives Kosten-Nutzen Verhältnis: die Kosten für die Datenerhebung einer Metrik darf deren Nutzen nicht übersteigen. Dies impliziert bei Metriksystemen automatisch die zuvor geforderte Minimalität.
- **Aktualität** Metriken müssen einen Aktualitätsanspruch erheben, um einen sinnvollen Beitrag für Entscheidungsprozesse im laufenden Geschäft bieten zu können. Es muss dabei gerade bei betriebswirtschaftlichen Metriken vor allem darauf geachtet werden, dass z.B. Änderungen in der Bilanzierung oder Gewinnermittlung keinen Einfluss auf die Metrik haben können (bzw. alte vorhandene Werte einer Metrik im neuen Umfeld Neuberechnet werden). Hilfreich sind aber nicht nur rückwärts gewandte Metriken, sondern auch Metriken, die eine Prognose bzw. SOLL-Vergleiche erlauben. [Pr08]
- **Veränderungssensibilität** Sollte sich das zu steuernde System verändern, muss eine Metrik ebenfalls darauf möglichst frühzeitig und vor allem erkennbar reagieren. Andernfalls ist eine Steuerung nicht möglich [Ku07].
- **Vergleichbarkeit** Vor allem für betriebswirtschaftliche Kennzahlen ist eine Vergleichbarkeit wünschenswert. Ein Benchmark zwischen zwei unterschiedlichen (d.h. disjunkten), aber gleichartigen Sachverhalten soll möglich sein, wodurch Aussagen der Form "A ist besser als B" zulässig werden [Ku07].
- **Korrekte Ermittlung** "Kennzahlen sind nur so gut, wie es die Qualität des Ausgangsmaterials zulässt" [Pr08]. Entscheidend für die Aussagekraft einer Metrik ist daher die Qualität der Datenbasis und der Prozess der Berechnung. Je nach Geschäftsorganisation können leicht falsche oder gar manipulierte Daten in eine Metrik einfließen. Eine kritische Auseinandersetzung der Stakeholder mit der Metrik und der Datenerhebung ist daher notwendig.

Diese Anforderungen stellen eine Grundauswahl dar. Dennoch ist es nicht leicht, alle Anforderungen gleichermaßen zu erfüllen. Vor allem die Korrektheit der Modellbildung ist schwer nachzuweisen. Auf der anderen Seite ist aber auch die Forderung nach Aktualität nicht immer erfüllbar, da Metriken nicht beliebig weit in die Vergangenheit an neue Gegebenheiten angepasst werden können.

Teil II

Komplexitätsbestimmung

3 Komplexitätsmetriken in der Softwaretechnik

In der Softwaretechnik werden oft ähnliche Strukturen und Konzepte verwendet wie im Management von Anwendungslandschaften. Das Management von IT-Unternehmensarchitekturen betrachtet Softwaresysteme und deren Abhängigkeiten untereinander von einem höheren Aggregationslevel als die Softwaretechnik. Während die Softwaretechnik sich auf die Ebene von Modulen, Paketen und Klassen eines Softwaresystems konzentriert, welche zusammen eine Anwendung bilden, aggregiert die Sicht der Anwendungslandschaft alle diese Einheiten und deren Abhängigkeiten zu anderen Softwaresystemen auf ein höheres Level. Die darin enthaltenen, grundlegenden Modelle und Konzepte bleiben aber sehr ähnlich und vergleichbar. Im Folgenden werden daher Softwaremetriken vorgestellt, deren Konzepte auch auf Anwendungslandschaften aufgrund des unterschiedlichen Abstraktionsgrades transformiert werden können.

In der Softwaretechnik wurden bereits in den 70ern Metriken entwickelt, die die Komplexität von Software messen sollten. Angesetzt wurde das damals moderne Konzept von Softwaremodulen, wobei bereits unterschieden wurde zwischen Intra- und Intermodularer Komplexität. Erstere beschäftigt sich mit den Zusammenhängen und Eigenschaften innerhalb eines Moduls, während intermodulare Komplexität den Fokus die gesamte Anwendung legt. Von Zuse werden einige dieser Metriken kompakt zusammengefasst [Zu94]. Im Folgenden sollen die wichtigsten davon näher vorgestellt werden.

3.1 Zyklomatische Komplexität

Eine der ersten und zugleich eine der bekanntesten Softwaremetriken ist die zyklomatische Komplexität, die von McCabe [Mc76] bereits 1976 vorgeschlagen wurde. Getrieben von den vergleichsweise hohen Kosten für das Testen und das Warten von Software versuchte McCabe eine Metrik zu finden, die existierende, aber ungeeignete Metriken ersetzen sollte.

Beispielsweise wurde vor McCabe die Anzahl der Zeilen pro Modul/Programm auf einen festen Wert von z.B. 50 Zeilen oder 2 Seiten limitiert. Diese Metriken versuchen zwar die Verstehenskomplexität zu reduzieren, allerdings auf eine ungeeignete Art und Weise: Nach McCabe ist nämlich ursächlich nicht der Codeumfang, sondern die Anzahl der möglichen Kontrollflüsse durch das Programm für die Komplexität verantwortlich. Die Anzahl dieser kann nämlich auch bei einem kleinen Programm erhebliche Ausmaße annehmen, so dass Tests und Wartung zeit- und somit kostenintensiv sind.

Grundlage der Berechnung der zyklomatischen Komplexität ist daher der Kontrollflussgraph eines Moduls. Aus diesem kann die zyklomatische Komplexität $V(G)$ wie folgt bestimmt werden:

$$V(G) = e - n + 2p$$

mit

- e := Anzahl der Kanten
- n := Anzahl der Knoten
- p := Anzahl der Zusammenhangskomponenten

Je geringer $V(G)$ desto geringer ist auch die Komplexität. Die zyklomatische Komplexität ist ≥ 1 . $V(G) = 1$ bedeutet, dass ein Modul keine Verzweigungen im Kontrollfluss besitzt, also linear abgearbeitet wird.

Die Bestimmung der zyklomatischen Komplexität kann bei vorhandenem Kontrollflussgraph mit Hilfe des Algorithmus von Tarjan [Ta72] in linearer Zeit vorgenommen werden.

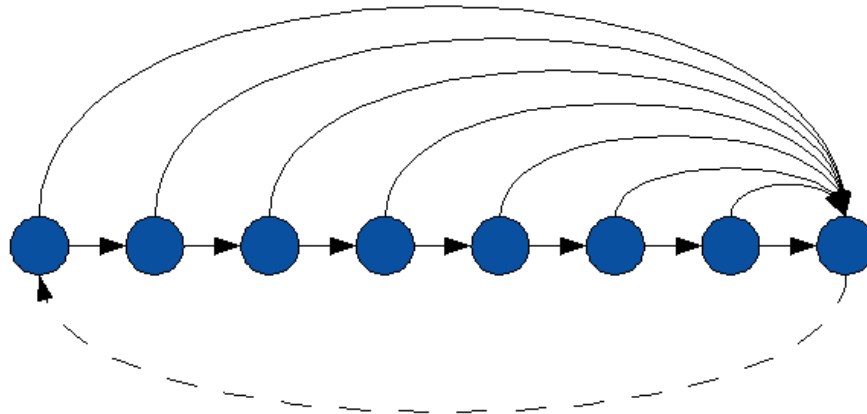
Die zyklomatische Komplexität bestimmt die Komplexität nicht immer in einer nachvollziehbaren Art und Weise. In manchen Fällen stimmt $V(G)$ nicht mit der subjektiven Einschätzung der Komplexität eines Codefragments überein. Beispielsweise liefern bereits einfache `switch`-Ausdrücke oder `if-then-else`-Blöcke vergleichsweise hohe Werte für $V(G)$. Das folgende Beispiel (Listing 3.1, der dazugehörige Kontrollflussgraph ist in Abbildung 3.1 dargestellt) ist für den Betrachter nicht komplex, hat aber bereits eine zyklomatische Komplexität $V(G) = 8$. Ab einer zyklomatischen Komplexität von 10 empfiehlt McCabe die Aufspaltung in mehrere Module. Auch [Ta06] weist auf den Umstand hin, dass eine hohe zyklomatische Komplexität alleine kein zusätzliches Risiko darstellt oder eine Aufspaltung des Moduls rechtfertigt.

Listing 3.1: Beispielprogramm mit zyklomatischen Komplexität $V(G)$ von 8

```
1
2 char* get_name(int id) {
3     switch(id) {
4         case 1: return "Hofmann";
5         case 2: return "Schmidt";
6         case 3: return "Schneider";
7         case 4: return "Fischer";
8         case 5: return "Weber";
9         case 6: return "Meyer";
10        case 7: return "Wagner";
11    }
12    return "unbekannt";
13 }
```

3.2 Kohäsion

Ebenfalls 1976 wurde die Kohäsion eines Softwaremoduls als Komplexitätstreiber von Myers identifiziert [Zu94]. Kohäsion ist der Grad des funktionellen Zusammenhangs in-



(Eigene Darstellung)

Abbildung 3.1: Kontrollflussgraph für Listing 3.1

nerhalb eines Moduls; hohe Kohäsion liegt z.B. dann vor, wenn funktionell wie logisch zusammenhängende Funktionalitäten innerhalb eines Moduls gebündelt werden und nicht über das gesamte Softwaresystem verteilt werden. Eine hohe Kohäsion ist deshalb erstrebenswert; je niedriger die Kohäsion, desto höher sind die logischen Abhängigkeiten zwischen den Modulen. Dieses Phänomen wird Kopplung genannt und wird im nachfolgenden Abschnitt behandelt.

Zu Folge ist die Kohäsion nicht hinreichend gut zu messen. In der Literatur wird daher Kohäsion oft nur mit Hilfe von mehreren Indikatoren gemessen. Ott und Thuss messen dafür die Indikatoren *Coverage*, *Overlap*, *Tightness*, *Parallelism*, *MinCoverage* und *MaxCoverage* [OT93]. Diese geben zusammengefasst in einem Metriksystem einen guten Überblick über den Grad der Kohäsion.

Grundlage zur Bestimmung dieser Metriken sind *slices* eines Moduls. Ein Slice eines Moduls ist abhängig von einer Anweisung s und einer zu betrachtenden Variable v . Der Slice ist dann die Menge aller Anweisungen und Prädikate, die den Wert von v bis zur Anweisung s beeinflussen können. Um die Metriken zu berechnen, verwenden Ott und Thuss alle Slices, die einen Output-Parameter oder einen Rückgabewert beeinflussen und betrachten lediglich die Slices am Ende (*end-slice*) eines Moduls (bzw. nach der letzten Anweisung eines Moduls/Prozedur).

Im Folgenden werden die Einzelmetriken von Ott und Thuss [OT93] kurz beschrieben. Als Notation wird V_O als die Menge der Output-Parameter und Rückgabevariablen des Moduls M verwendet. SL_i ist der end-slice eines Moduls M in Bezug auf die Variable $v_i \in V_O$. Die Länge eines Moduls ist mit $length(M)$ notiert, welches die Anzahl der ausführbaren Anweisungen in M darstellt. Weiterhin ist SL_{int} als Schnittmenge aller Slices definiert:

$$SL_{int} = \bigcap_{i=1}^{V_O} SL_i$$

Coverage Errechnet die mittlere Länge der Slices und setzt diese in Bezug zur Länge des Moduls:

$$Coverage(M) = \frac{1}{|VO|} \sum_{i=1}^{|VO|} \frac{SL_i}{length(M)}$$

Overlap Die Überlappung gibt das mittlere Verhältnis zwischen der Anzahl der Anweisungen in jedem Slice zur Anzahl der Größe aller Slices wieder:

$$Overlap(M) = \frac{1}{|VO|} \sum_{i=1}^{|VO|} \frac{|SL_{int}|}{|SL_i|}$$

Tightness Die Dichte eines Modules kann durch das Verhältnis der Anzahl der Anweisungen, die in jedem Slice enthalten sind, zur Gesamtanzahl der Anweisungen im Modul ausgedrückt werden:

$$Tightness(M) = \frac{|SL_{int}|}{length(M)}$$

Parallelism Der Parallelismus errechnet die Anzahl der Slices, die weniger Anweisungen als ein Threshold τ paarweise gemeinsam haben. Falls $\tau = 0$ wird die Anzahl der Slices berechnet die vollständig unabhängig voneinander sind:

$$Parallelism(M) = |\{SL_i \text{ mit } |SL_i \cap SL_j| \leq \tau \forall j \neq i\}|$$

MinCoverage Im Gegensatz zur Coverage-Metrik wird hier nur der kleinste Slice ins Verhältnis zur Länge des gesamten Moduls gesetzt:

$$MinCoverage(M) = \frac{1}{length(M)} \min |SL_i|$$

MaxCoverage Genau gegensätzlich zu MinCoverage wird hier der größte Slice mit der Länge des gesamten Moduls verglichen:

$$MaxCoverage(M) = \frac{1}{length(M)} \max |SL_i|$$

Alle Metriken zusammen können einen guten Eindruck über den Grad der Kohäsion geben; allerdings bleibt es schwierig die Kohäsion direkt in einer Metrik auszudrücken.

3.3 Kopplung

Eng verwandt mit der zuvor diskutierten Kohäsion ist die Kopplung. Bei der Kohäsion standen die funktionellen und logischen Zusammenhänge innerhalb eines Moduls im Mittelpunkt; bei der Kopplung hingegen werden die Abhängigkeiten zwischen den Modulen

in den Vordergrund gerückt. Die Kopplung zweier Module kann als Grad der Verbindung bzw. Abhängigkeit voneinander gesehen werden. Auch dies wurde bereits früh in den 70ern als Komplexitätstreiber identifiziert [Zu94]. Enge (bzw. hohe) Kopplung erschwert zum einen die Wartung von Anwendungen, da Änderungen schwieriger durchzuführen sind und sich so leichter Fehler in die Anwendung einschleichen. Zum anderen leidet die Wiederverwendung von einzelnen Modulen: durch enge Kopplung können Module nicht mehr unabhängig von einem anderen Modul in einem anderen Kontext wiederverwendet werden.

Zur Messung der Komplexität einer Anwendung auf Basis des der Anwendung zugrundeliegenden Grades der Kopplung hat Bowles 1983 mehrere Metriken entwickelt [Zu94]. Er hat dazu zum einen Metriken entwickelt, die auf (Sub-)Modul Ebene die Kopplung bestimmen, zum anderen können diese Metriken aber auch zu einer Gesamtmetriken für die Kopplung aggregiert werden.

Kopplung auf Modulebene Bowles bestimmt die Kopplung zwischen Module auf Basis der Anzahl der gemeinsam benutzten Variablen und der jeweils zwischen den Modulen abhängigen Parametern [Zu94]:

- a = Anzahl der formalen Parameter
- b = Anzahl der globalen Variablen, die mit Super-Modulen gemeinsam benutzt werden
- c = Anzahl der Parameter, die zwischen dem zu betrachtenden und seinen Submodulen weitergeleitet werden
- d = Anzahl der globalen Variablen, die mit Submodulen gemeinsam verwendet werden

Daraus konstruiert Bowles folgende einfache Metrik, die sowohl in der Detailplanung als auch auf fertigen Code angewendet werden kann:

$$CM = 1 - \frac{1}{1 + a + 2b + c + 2d}$$

Kopplung auf Anwendungsebene Die Komplexität auf Anwendungsebene wird durch Wiederverwendung der CM -Metrik errechnet. Zunächst wird die Systemkomplexitätsmatrix SC errechnet, die aus Ausgangspunkt für vier verschiedene Metriken verwendet wird. Die Konstruktion der Matrix SC ist wie folgt definiert [Zu94]:

- SC hat die Größe $|M| * |M|$, wobei M die Menge aller Module ist
- $SC_{i,j}$ beschreibt die Beziehung des Modules i zu Module j ; der Wert wird dabei wie folgt berechnet: $SC_{i,j} = |P_{i,j}| + 2 * |SH_{i,j}|$ mit $P_{i,j}$ = Menge der Parameter die von Modul i an Modul j übergeben werden und $SH_{i,j}$ = Menge der globalen, gemeinsam benutzten Variablen der Module i und j .

Um nun die Komplexität zu bestimmen schlägt Bowles vier Metriken vor:

- Gesamt Systemkomplexität: $SSC = \sum_{m,n \in M} SC(m, n)$
- Mittlere Systemkomplexität: $AVC = \sum_{m,n \in M} \frac{SC(m,n)}{|M|}$
- Maximale Modulkomplexität: $MAC = \max(SC)$
- Minimale Modulkomplexität: $MIC = \min(SC)$

Ähnlich wie bei den Kohäsions-Metriken müssen diese vier Metriken als Metriksystem aufgefasst werden und dienen als Indikator für die Kopplung innerhalb eines Systems und der damit implizierten Komplexität.

3.4 Halstead Metrik

Ein Jahr nachdem McCabe die Metrik zur zyklomatischen Komplexität entwickelt hat, beschreibt Halstead eine ähnliche, statische Metrik zur Messung von Codekomplexität [Ha77]. Seine Ausführungen basieren auf der Annahme, dass auch immaterielle und abstrakte Objekte wie Computerprogramme zu einem gewissen Ausmaß den Naturgesetzen unterliegen. Sein Ansatz ist universal anwendbar, da er auf vier Grundzahlen basiert, die in Programmen jeder Programmiersprache bestimmt werden können:

- $\eta_1 =$ Anzahl der eindeutigen Operatoren im Quellcode
- $\eta_2 =$ Anzahl der eindeutigen Operanden im Quellcode
- $N_1 =$ Häufigkeit aller Operatoren im Quellcode
- $N_2 =$ Häufigkeit aller Operanden im Quellcode

Davon ausgehend werden verschiedene Metriken aufgestellt, die als Indikatoren für die Komplexität dienen. Die wichtigste dabei ist die *Program-Level* Metrik, die versucht die Verstehenskomplexität auf Ebene einer Anwendung direkt zu messen. Dazu definiert Halstead das Program-Level L wie folgt:

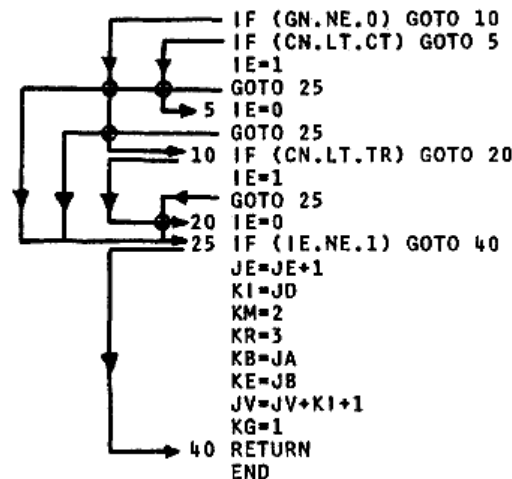
$$L = \frac{2}{\eta_1} \frac{\eta_2}{N_2}$$

Halstead weißt aber auch daraufhin, dass die Metrik nur dann aussagekräftig ist, wenn alle Personen, die die Metrik beeinflussen oder auswerten (also z.B. Programmierer, Reviewer, Projektmanager), auf dem gleichen Wissenslevel über die Programmiersprache sind. Andernfalls müsste der Wissensunterschied noch in der Metrik berücksichtigt werden.

Problematisch ist jedoch die Interpretation dieser Metrik als Komplexitätsmaß: aus Sicht der Messtheorie sind die Metriken von Halstead zwar korrekt, allerdings geben sie nicht die Komplexität wieder, sondern geben tatsächlich nur etwa das Merkmal "Länge" eines Programms wieder [Fe94].

3.5 Strukturelle Komplexität

Ein Ansatz zur Messung der strukturellen Komplexität wurde 1979 von Woodward et al. vorgeschlagen. Anders als die beiden vorigen Metriken zu Kohäsion und Kopplung basiert ihr Ansatz auf Kontrollflussgraphen. Das Interesse liegt hierbei hauptsächlich auf "Knoten" bzw. "Kreuzungen" des Kontrollflusses, die nach Woodward et al. ein direktes Maß für die Komplexität darstellen [WHH79].



([WHH79])

Abbildung 3.2: Beispiel Fortran Programm mit 4 Knoten

Ein Knoten liegt genau dann vor, wenn für ein Paar von Kontrollflusssprüngen, gegeben durch die Tupel (a, b) und (p, q) mit $a, b, p, q \in \mathbb{N} \hat{=} \text{Zeilennummern}$ eine der folgenden Bedingungen gilt:

- $\min(a, b) < \min(p, q) < \max(a, b) \wedge \max(p, q) > \max(a, b)$
- $\min(a, b) < \max(p, q) < \max(a, b) \wedge \min(p, q) < \min(a, b)$

In Abbildung 3.2 sind die Knoten eines Beispielprogramms dargestellt.

Für Sprachen mit mehreren Anweisungen pro Zeile muss die Zählweise von a, b, c, d entsprechend einer Totalordnung angepasst werden.

Ein Nachteil diese Metrik ist, dass eine Anwendung auf moderne Programmiersprachen nicht trivial ist. Die Metrik basiert auf dem prozeduralen Fortran, das konditionale Verzweigungen, Schleifen oder Funktionsaufrufe mit GOTO oder CALL Konstrukten abbildet. In modernen Sprachen müssen zuerst if-then-else, do while und Funktionsaufrufe in ein solches Schema umgeschrieben werden. Woodward et al. beschreiben zwar auch eine Methode die direkt auf dem Kontrollflussgraphen arbeitet, allerdings ist diese Vergleichsweise aufwendig zu berechnen.

Neben McCabe und Woodward et al. versucht auch Jose Luis Roca die strukturelle Komplexität zu messen. Roca vermisst in den beiden existierenden Ansätzen allerdings jeweils die Berücksichtigung der Eingabeparameter und deren Auswirkungen auf die möglichen Programmflüsse. Roca entwickelt daher ein Maß für die strukturelle Komplexität welches die Wahrscheinlichkeit von Programmflüssen einbezieht [Ro96].

3.6 Softwarearchitekturkomplexität

Neuere Ansätze, wie der von Lilienthal [Li08], beschäftigen sich mit der Bestimmung von Komplexität auf Ebene von Softwarearchitekturen. Lilienthal versucht sich der Thematik Komplexität über drei Faktoren zu nähern:

- **Mustertreue** Falls in einer Architektur Muster immer wieder auftreten, ist nach Lilienthal die Verstehenskomplexität geringer, da das menschliche Gehirn die vorhandenen Schemata leicht erkennen kann.
- **Modularität** Ein modulares System trägt auch dann zu einer geringen Komplexität bei, wenn die Modulstruktur logisch und funktional-zusammenhängend gewählt wurde. Dies wird z.B. durch den Prozess des Chunkings der kognitiven Psychologie gestützt.
- **Geordnetheit** "Eine Architektur ist dann geordnet, wenn die aggregierten Benutzt- und Vererbungsbeziehungen zwischen den Architekturelementen einen gerichteten azyklischen Graphen bilden." [Li08]

Für die Geordnetheit stehen durch Lilienthal auch Metriken zur Verfügung. Datenbasis dieser Metriken ist aber nicht mehr wie bei den zuvor vorgestellten Metriken ein Kontrollflussgraph. Anstatt dessen werden die Graphstrukturen einer Architektur als Grundlage wiederverwendet und dienen als Eingabe in den Berechnungsalgorithmus.

Die Geordnetheit setzt sich aus verschiedenen Faktoren zusammen, die Lilienthal jeweils mit einer eigenen Metrik misst: das Zyklenausmaß, die Zyklenreichweite, der Zyklenumfang und die Verflochtenheit.

- Das **Zyklenausmaß** gibt die Anzahl der an allen Zyklen beteiligten Elemente auf verschiedenen Ebenen wieder. Betrachtet werden die Klassenebene, Paketebene und Subsystemebene, deren Zyklen jeweils einzeln gezählt werden.
- Die **Zyklenreichweite** untersucht den Einfluss der jeweils untergeordneten Ebene auf die übergeordnete Ebene: welche Zyklen auf Paketebene werden z.B. durch Zyklen auf Klassenebene induziert? Analog werden auch Zyklen auf Subsystemebene auf deren Ursprung in der Paketebene untersucht.
- Der **Zyklenumfang** bestimmt die Anzahl der an den einzelnen Zyklen beteiligten Elemente auf Klassen- und Subsystemebene. Diese Metrik unterscheidet sich insofern vom Zyklenausmaß, als dass bei ähnlich großen Anwendungen mit ähnlichem Zyklenausmaß eine Anwendung für den Programmierer weniger komplex erscheint, wenn viele kleine Zyklen vorhanden sind. Die Änderungen an kleinen Zyklen haben überschaubare Abhängigkeiten, während die Änderungen an einem großen Abhängigkeitszyklus weniger leicht durchschaubar und beherrschbar ist.
- Nach Lilienthal ist **Verflochtenheit** durch das Ausmaß der bidirektionalen Abhängigkeiten auf Klassen-, Paket- und Subsystemebene bestimmt. Bidirektionale Abhängigkeiten tragen insofern zur Komplexität einer Anwendung bei, als dass diese oft nicht einfach oder nur durch Redesign und Neuimplementierung aufzulösen sind.

4 Komplexitätsmetriken für Anwendungslandschaften

Komplexität definiert sich unter anderem aus dem ‐Ineinander vieler Merkmale‐ [DUD07]. Eine direkte Messung von Komplexität in einer Kennzahl ist nicht m3glich; es k3nnen lediglich Metriken aufgestellt werden, welche versuchen verschiedene Aspekte und Merkmale von Komplexitat zu charakterisieren [Fe94].

Im Folgenden werden daher verschiedene Attribute einer Anwendungslandschaft identifiziert, die die Komplexitat dieser beeinflussen.

Auf Basis der Attribute werden dann Metriken entwickelt, die unterschiedliche Sichten auf und Aspekte von Komplexitat in Anwendungslandschaften charakterisieren. Die Metriken basieren auf einem generischen Informationsmodell, das in den nachsten Sektionen ebenfalls beschrieben wird.

4.1 Identifikation von Komplexitatsattributen

Die komplexitatstreibenden Attribute werden im Folgenden in vier Kategorien eingeteilt: anwendungsbezogene Attribute geben Merkmale wieder, die direkt in der Anwendung oder in der Menge aller Anwendungen beobachtet werden k3nnen. Schnittstellenbezogene Attribute verdeutlichen Komplexitatstreiber, die durch die von Anwendungen zur Verf3gung gestellten Schnittstellen existieren. Daran ankn3pfend werden Attribute identifiziert, die abhangigkeitsbezogen sind, also aus den konkreten Schnittstellennutzungen entstehen. Weiterhin werden noch einige Attribute gelistet, die ausgehend von der Geschaftorganisation Einfluss auf die Komplexitat einer Anwendungslandschaft haben.

4.1.1 Anwendungsbezogene Attribute

Anzahl der Anwendungen

Dass Komplexitat von der Gr33e eines Systems abhangt, wird in der Literatur mehrmals beschrieben [Di04] [Wi03] [Kr09]. Die Gr33e beeinflusst die Verstehenskomplexitat insofern, als dass die Aufnahmefahigkeit des Menschen limitiert ist. Aber nicht die Gr33e allein ist der Komplexitatstreiber, sondern die Diversitat der Systemelemente, wie folgendes Beispiel veranschaulicht:

Beispiel Diederichs [Di04] verwendet das Beispiel von verschiedenen Waldern: Ein Laubwald in der gema3igten Zone erscheint aufgrund der wenigen verschiedenen Baumarten wesentlich weniger komplex als ein tropischer Urwald, in dem vielfaltige Baumarten auftreten.

Auf Anwendungslandschaften lässt sich dies in verschiedene Faktoren transformieren: einer dieser Faktoren ist die Anzahl der unterschiedlichen Anwendungen, die in einer Anwendungslandschaft vorkommen ein anderer die Instanzanzahl. Zunächst wird nur die Anzahl der unterschiedlichen Anwendungen betrachtet.

Im Verlauf der Arbeit wird die Anzahl der Anwendungen auch verkürzt *Anwendungszahl* genannt. Zunächst muss jede eigenständige, im Unternehmen genutzte Software als Anwendung gezählt werden. Eine Anwendung steht dabei stellvertretend für alle Installationen einer Software. Ab wann wird allerdings eine Anwendung als eigenständig in die Zählung mit aufgenommen? Um diese Frage beantworten zu können müssen noch die weiteren Faktoren der Anwendungsversion und -konfiguration betrachtet werden.

Anzahl der Anwendungsversionen

In Korrelation mit der Anzahl der Anwendungen muss auch unterschieden werden, wie viele verschiedene Versionen einer Anwendung (parallel) eingesetzt werden. Bei der Ermittlung der Anwendungszahl tritt dies in Vordergrund: Ab wann muss eine Anwendung als eigenständig gezählt werden? Ab wann hat sich die Anwendung soweit fortentwickelt (z.B. durch neue Versionen), dass sie nicht mehr mit gleichen Anwendungen in anderen Versionen zusammen betrachtet werden kann und daher eine gesonderte Betrachtung erforderlich ist?

Durch den Einsatz von Versionierungsschemas kann diese Frage leichter beantwortet werden. Versionen werden beliebig durch die Entwickler vergeben und geben keine Auskunft zur Kompatibilität zwischen den Versionen. Im Zweifelsfall muss jede eingesetzte Anwendungsversion als eigenständige Anwendung betrachtet werden.

Das Versionierungsschema des Apache Portable Runtime Projects [APR09] ist hier allerdings vorteilhaft: Jede Major-Version sollte als eigenständige Anwendung erfasst werden, da eine Kompatibilität zwischen verschiedenen Major-Versionen einer Software nicht garantiert ist. Somit ist eine einheitliche Betrachtung einer Software mit verschiedenen eingesetzten Major-Versionen nicht mehr möglich. Dieses Versionierungsschema ist auch als *Degree of compatibility* bekannt.

Werden Versionsnummern oder -namen nicht anhand der Kompatibilität vergeben (z.B. motiviert durch Marketingstrategien) muss die Prüfung auf Kompatibilität durch den Datenerheber vorgenommen werden. Dies bedeutet oft einen signifikanten Mehraufwand und ist nur schwer möglich. Die Existenz eines geeigneten Changelogs [GNU09] oder vergleichbarer Dokumentation ist hilfreich.

Anzahl der Anwendungskonfigurationen

Neben den Anwendungsversionen kann aber auch die Konfiguration einer Anwendung unterschiedliches Verhalten hervorrufen, das eine gesonderte Betrachtung in der Anwendungslandschaft nötig macht. So gibt es Beispiele bei denen Anwendungen je nach Konfiguration sehr unterschiedlich reagieren: beispielsweise kann ein Apache Webserver als normaler HTTP-Server konfiguriert werden, gleichzeitig aber auch als HTTP-Proxy.

Bei der Ermittlung der Anwendungszahl muss daher abgewägt werden, ab wann eine Konfiguration als eigenständige Anwendung zählt. Nachdem es allerdings für verschiede-

ne Konfiguration leider keine Versionierungsschemata gibt, entfällt diese Entscheidungshilfe.

Eine Softwareinstallation sollte genau dann als eigenständige Anwendung gezählt werden, wenn sich das Verhalten der Softwareinstallation konfigurationsbedingt signifikant vom Verhalten anderer (gleicher) Softwareinstallationen mit anderer Konfiguration unterscheidet.

Anzahl der Anwendungsinstanzen

Eine Anwendung kann in einem Unternehmen mehrfach installiert sein; eine installierte Anwendung wird im Folgenden auch Anwendungsinstanz (kurz: Instanz) genannt. Die Anzahl der Anwendungsinstanzen für eine bestimmte Anwendung wird auch im Folgenden kurz *Instanzzahl* genannt.

An einem kleinen Beispiel aus der Musik werden die unterschiedlichen Dimensionen von Anwendungszahl und Instanzzahl deutlich. Enthält ein Musikstück viele unterschiedliche Noten die gespielt werden müssen, wird vom interpretierenden Musiker mehr abverlangt als bei wenigen unterschiedlichen Noten. Auf der anderen Seite wird dem Musiker ebenfalls mehr Können abverlangt, wenn er mehrmals eine gleiche Note spielen muss. Beides zusammen - also viele unterschiedliche Noten von denen jeweils viele gespielt werden müssen - vervielfacht die Komplexität.

Überträgt man dieses Beispiel nun auf den Bereich der Anwendungslandschaften so würde die Anzahl der unterschiedlichen Noten der Anwendungszahl entsprechen und die Anzahl der gleichen zu spielenden Noten der Instanzzahl. Dabei kann dies aber etwa pro einzelner Takt betrachtet werden, dies würde etwa einer Analyse nach Standorten entsprechend, annähernd analog dazu würde etwa die Aufgliederung in verschiedene parallele Stimmen einer Aufgliederung in verschiedene Geschäftsprozesse entsprechen.

Eine Anwendungsinstanz ist ein Objekt von der durch die Attribute Anwendungsname, -version und -konfiguration definierten Anwendungsklasse; aufgrund der Installation der Anwendung auf einem physischen System besitzt eine Anwendungsinstanz ferner einen Ort.

Alter der Anwendungen

Eine Eigenschaft der Anwendungen ist auch deren Alter: Während dem Lebenszyklus einer Anwendung verändert sich deren Umwelt kontinuierlich. Sowohl die Infrastruktur (z.B. Hardware) als auch das Unternehmen verändert sich [Kr09]. Das Alter einer Anwendung kann in zwei Formen zu einem Komplexitätstreiber werden:

Sofern die Anwendung während dem Lebenszyklus nicht einer Weiterentwicklung unterzogen wird, verändert sich aber ggf. dessen Umwelt. Eine Diskrepanz zwischen der Weiterentwicklung zwischen der Anwendung und der Umwelt führt auf längere Sicht zu sogenannten Legacy-Anwendungen - Anwendungen die zwar für den Betrieb des Unternehmens zwingend notwendig sind, die aber nicht mehr weiterentwickelt oder an die neue Umwelt angepasst werden können [Bi99]. Häufig sind heute im produktiven Umfeld noch Anwendungen anzutreffen, die in Cobol oder Fortran geschrieben wurden; eine Weiterentwicklung ist heutzutage wegen fehlendem Wissen über die Programmiersprache

und die Anwendung nicht mehr in einem positiven Kosten-Nutzen Verhältnis zu bewerkstelligen [En09].

Auf der anderen Seite kann aber auch die stetige Fortentwicklung einer Anwendung die Komplexität beeinflussen. Die kontinuierliche Weiterentwicklung von Anwendungen führt zwangsweise zu einem Komplexitätswachstum [Sc08]. Nachvollziehbar wird dies, wenn lediglich Änderungen durchgeführt werden, die neue Beziehungen und Abhängigkeiten innerhalb wie außerhalb eines Systems einführen und keine architekturhaltenden Änderungen eingebracht werden.

Für beide Probleme gibt es Lösungen, wobei gerade für das erstere die Ansätze noch nicht weit genug gehen: Engels plädiert daher für eine realistischere Einschätzung der Lebenszyklen einer Anwendung und mahnt dazu, Methoden für die langfristige Entwicklung von Anwendungen zu entwickeln. Diese müssten zum einen modellgetrieben sowohl in der Anforderungs- als auch in der Entwicklungs- und Wartungsphase als auch optimal durch Werkzeuge unterstützt sein [En09].

Für die Bestimmung des Alters einer Anwendung können mehrere Daten im Lebenszyklus einer Anwendung herangezogen werden. Eine Anwendung durchläuft innerhalb einer Unternehmung mehrere Phasen: zu Beginn gibt es die Idee und eine Entscheidung zur Notwendigkeit einer bestimmten Anwendung, welche dann in einer weiteren Phase entweder selbst entwickelt wird oder von Fremdanbietern eingekauft und evaluiert wird. Anschließend wird die Anwendung eingeführt und in Betrieb genommen, später im Verlauf des Betriebs muss diese gewartet und weiterentwickelt werden. Am Ende des Lebenszyklus wird die Anwendung abgeschaltet oder abgeschafft [Kr09]. Der Beginn der Entwicklung als auch der Beginn des Produktivbetriebs können als Geburtsstunde zur Bestimmung des Alters herangezogen werden: während zweiteres Datum lediglich die Betriebsdauer festhält würde ersteres Datum zusätzlich noch die technologische Fortentwicklung während der Entwicklungsphase festhalten, welche ggf. nicht mehr Einfluss auf das endgültige Produkt genommen hat. In dieser Arbeit wird das Alter jedoch als Beginn des Produktivbetriebs einer Anwendung gesehen; Weiterentwicklungen der Software, welche keine Inkompatibilitäten mit Vorversionen einführen, bedingen keine neue Zählweise.

Inhärente Komplexität der Anwendungen

Inhärente Komplexität bezeichnet die Komplexität, die der Anwendung innewohnt. Diese ist zum einen der essentiellen Komplexität des durch die Anwendung zu lösenden Problems geschuldet, zum anderen der Komplexität die durch Entwicklung der Anwendung zufällig - akzidentiell - entsteht (z.B. durch ungünstige Entwurfsentscheidungen, zu viel Funktionalität). Dies beeinflusst allerdings auch die Komplexität auf Ebene der Anwendungslandschaft, da sich Entscheidungen zur Änderung, Ersatz oder Abschaltung von Anwendungen auch über die Interna der konkreten Anwendung stützen müssen. Je höher die inhärente Komplexität desto schwieriger, komplexer sind solche Entscheidungen zu treffen. Aber auch im Falle des Ersatzes einer Anwendung, ist eine inhärent-komplexe (v.a. essentiell-komplexe) Anwendung schwieriger zu ersetzen, da die Vielschichtigkeit einer Anwendung nicht erkannt wird.

Dabei ist zunächst der Standpunkt interessant: Wäre es ggf. nicht möglich kleinere Anwendungen, die viele Abhängigkeiten zueinander besitzen, zu einer größeren Anwendung zu aggregieren? Die dann entstehende Anwendungslandschaft besteht zunächst aus weniger Elementen, welches den Aufwand zum Verstehen geringer macht. Ganzheitlich gesehen, hat sich die Komplexität allerdings durch die Aggregation nicht verändert, da immer noch die gleichen Services von der Anwendungslandschaften erbracht werden müssen. Es ist daher festzustellen, dass die Granularität der Anwendungen und daraus folgend auch deren inhärente Komplexität die Komplexität der Anwendungslandschaft beeinflusst.

Soll eine Anwendungslandschaft sinnvoll mit einer anderen Landschaft verglichen werden, so muss der Abstraktionsgrad, die Granularität der Anwendungen gleich sein. Dies ist allerdings schwer zu bewerkstelligen, da sich die Sichtweisen verschiedener Unternehmensarchitekten unterscheiden.

Nutzungsgrad von Standardanwendungen

Der Einsatz von Standardanwendungen - unabhängig ob Branchenstandards oder globale Standards - beeinflusst die Komplexität. Zunächst kann durch den Einsatz von Standardanwendungen die Verstehenskomplexität verringert werden, da kein gesondertes Wissen über die Anwendung selbst vorhanden sein muss. Der Zweck einer Anwendung und deren Funktion innerhalb der Anwendungslandschaft erschließt sich von selbst.

Dies gilt allerdings nur, falls Standardanwendungen ohne spezielle Anpassungen eingesetzt werden. Sofern Standardanwendungen jedoch zur Erfüllung der betrieblichen Bedürfnisse modifiziert oder erweitert wurden wird die Komplexität der Anwendungslandschaft als auch die Kosten für die Wartung gesteigert [SK09].

Standardisierungsgrad von Anwendungstechnologien

Zur Herabsetzung der Verstehenskomplexität gibt es Ansätze die Anwendungslandschaft zu standardisieren; einer der Eigenschaften, die einer Standardisierung unterliegen können sind die Technologien, die in den Anwendungen eingesetzt werden [Br09], [RGA07], [AD05].

Technologien in diesem Sinne können z.B. Middleware-Technologien, Programmiersprachen und -frameworks, Verschlüsselungsalgorithmen (z.B. RSA, 3DES, AES), Datenbanktechnologien (Objektrelationale Mapper wie JPA, Hibernate, TopLink, Konnektoren wie ODBC, JDBC, ADO.NET oder Oracle Net), Signaturmethoden (PGP, S/MIME, DSA), Hashtechnologien (MD5, SHA-2) oder ähnliches sein. Wichtig hierbei ist, dass je mehr Anwendungen die gleichen Technologien verwenden - also nach den gleichen Mustern und Schemata arbeiten - desto geringer ist die Verstehenskomplexität.

Generell muss ein hoher Standardisierungsgrad aber nicht optimal sein: so warnt Gartner Research vor einer "Überstandardisierung" [Bu09b]. Standardisierung sei nur sinnvoll, sofern auch die Geschäftsanforderungen damit erfüllt werden können.

Nutzungsgrad von generischen Architekturstilen

Architekturstile für Anwendungen in betrieblichen Informationssystemen beeinflussen die Verstehenskomplexität einer Anwendung, aber auch die Verstehenskomplexität die auftritt, wenn eine gesamte Anwendungslandschaft und deren Abhängigkeiten zur Infrastrukturschicht analysiert werden.

Als Architekturstil bezeichnet werden Lösungsstrukturen, denen ein Anwendungssystem folgen soll. Es werden dabei Einschränkungen des durch Programmiersprachen ausdrückbaren eingeführt; es werden nur bestimmte definierte Beziehungen zwischen Architekturelementen erlaubt [Li08].

Die 3-Tier Schichtenarchitektur ist ein prominentes Beispiel eines Architekturstils: in einer Präsentationsschicht wird alle Funktionalität zur Ein- und Ausgabe von Daten gebündelt; die Daten stammen aus einer Logikschicht, die die Geschäftsprozesse abbildet und ihrerseits persistente Daten aus einer Datenhaltungsschicht bezieht. Diese klare Trennung der Zuständigkeiten erleichtert die Wartung und die Verteilung der Entwicklung auf spezialisierte Teams.

Redundanz

Redundanz bezeichnet allgemein einen Überfluss bzw. das Vorhandensein von weglassbaren Elementen [DUD05]. Dies kann sowohl für vorhandene Funktionen oder Daten der Fall sein; beide Arten bedingen eine gesonderte Betrachtung der redundanten Teile einer Anwendungslandschaft und führen so zu erhöhter Komplexität.

Redundanz von Funktionen Redundanz von Funktionen liegt dann vor, wenn mehrere Anwendungen die gleichen (Teil-)Funktionen bereitstellen. Vor allem beim Management von M&A (Merger & Acquisition) Projekten können Redundanzen auftreten, da z.B. im Supply Chain Management (SCM), dem Customer Relationship Management (CRM) oder dem Human Resources Management (HRM) oftmals in den zusammenzuführenden Unternehmen bereits unterschiedliche Lösungen vorhanden sein können.

Beispielsweise könnten aber auch in einem Internetgeschäftsmodell (zugekaufte) Shopanwendungen über ähnliche Funktionen verfügen wie eine interne Anwendung zur Bestellabwicklung. Für einen Unternehmensarchitekten ist dabei der Fakt relevant, dass verschiedene Anwendungen den gleichen Dienst erbringen können, ein Problem, das den Aufwand zum Verstehen der Anwendungslandschaft, negativ beeinflusst.

Redundanz von Daten Redundanz von Daten bezeichnet, dass die Verwaltung der Informationen, die einer bestimmten Entität angehören, über mehrere Anwendungen verteilt ist. Es gibt also mehrere Kopien der gleichen Daten in verschiedenen Anwendungen. Diese Daten müssen dann über alle Anwendungen synchronisiert werden; eine schwierige Aufgabe, wenn die Daten von mehreren Anwendungen auch verändert werden dürfen.

Vor allem in M&A Projekten treten Redundanzen auf. Ein Unternehmensarchitekt muss abschätzen dort können, welche der redundanten Anwendungen oder Daten verändert, migriert oder konsolidiert werden sollen. Buxmann et al. haben diese Schwierigkeit aufgenommen und ein Entscheidungsmodell vorgeschlagen, welches sich gerade für M&A

Projekte eignet und den Unternehmensarchitekten unterstützt [MB07]. Dennoch bleibt die Entscheidung komplex, da zunächst einige Parameter wie die Standardisierungskosten oder der monetäre Nutzen einer Anwendung bestimmt werden muss. Vor allem letzterer ist in vielen Organisationen schwer zu bestimmen; Buxmann et al. weisen sogar selbst darauf hin, dass der Einsatz ihres Modells mehr Kosten - hauptsächlich zur Erhebung der Informationen zu den verschiedenen Parametern - verursachen könnte als es Nutzen bringt.

Kapselung von Daten

Die Kapselung von Daten (*engl.*: information hiding) ist in der Softwaretechnik eines der wichtigsten Instrumente zur Vermeidung von Komplexität [La06]. Von Kapselung von Daten spricht man, wenn auf Informationen nur über definierte, fachliche Schnittstellen zugegriffen werden kann und kein Direktzugriff auf interne Datenstrukturen (z.B. einer Klasse) möglich ist. Auch etwa der Direktzugriff auf den persistenten Datenspeicher (z.B. ein relationales Datenbankmanagementsystem) ist nur für die Komponenten erlaubt, die Besitzer einer Entität sind. Eine Komponente darf aber jeweils wieder nur auf Datenbankstrukturen zugreifen, die unmittelbar zur verwalteten Entität gehören.

Auch auf Anwendungslandschaftsebene kann eine Kapselung von Daten erfolgen; der Zugriff auf Informationen findet in einer serviceorientierten Architektur (SOA) nur über wohldefinierte Services und Schnittstellen statt; der Zugriff auf Datenbanken einer anderen Anwendung ist nicht erlaubt.

Die strukturelle Komplexität wird insofern davon beeinflusst, als dass die konsequente Kapselung von Daten zu implementierungsunabhängigen Schnittstellen führt, welche auch langfristig stabil sind [Ha07].

4.1.2 Schnittstellenbezogene Attribute

Alter der Schnittstellentechnologie

Ähnlich zum Alter einer Anwendung kann auch separat das Alter bestimmter Schnittstellentechnologien betrachtet werden. In Umgebungen, in denen busartige Architekturen zur Kommunikation zwischen Anwendungen genutzt werden (z.B. ein Enterprise Service Bus) beeinflusst das Alter der dort eingesetzten Schnittstellentechnologie die Komplexität.

In Anlehnung an Engels [En09] altert die Schnittstellentechnologie unabhängig von deren Anwendung. Daher sind auch dort geeignete Maßnahmen zu treffen, die den Alterungsprozess vermeiden oder verlangsamen, um die Bildung von Legacy-Schnittstellen zu verhindern.

Komplexität entsteht vor allem dann, wenn eine Schnittstelle ersetzt werden muss oder gar ein neuer Service Bus eingeführt werden muss. Die Änderung einer Schnittstelle führt zusätzlich zu Änderungen in allen davon abhängigen Anwendungen; bei der Einführung eines neuen Service Bus müssen ebenfalls ggf. alte Schnittstellen zum Service Bus aktualisiert werden.

Nutzungsgrad von Schnittstellenmuster

Die Existenz und Nutzung von standardisierten Schnittstellenmuster trägt zur Reduktion der strukturellen Komplexität bei. Durch standardisierte Interaktionsmuster und -technologien wird vorgeschrieben, wie Anwendungen untereinander kommunizieren dürfen [Ha07]. Hilfreich ist es Schnittstellen in Normalform [HJ06] zu fordern; daneben müssen aber auch die erlaubten Notifikations- und Datenflusstypen standardisiert werden.

Lilienthal hebt zudem hervor, dass durch die Einhaltung von Interaktionsmuster nicht nur die strukturelle Komplexität gesenkt werden kann, sondern vor allem auch die strukturbildenden Prozesse des Chunkings und dem Aufbau von Schemata zu einer geringeren Verstehenskomplexität führen [Li08].

4.1.3 Abhängigkeitsbezogene Attribute

In der Softwaretechnik wurden Abhängigkeiten zwischen Modulen bereits mehrfach diskutiert und deren Bedeutung im Zusammenhang der Komplexität dargelegt [Mc76], [Ro96], [Ba02].

Die Abhängigkeiten zwischen Anwendungen stellen eine höhere Abstraktion der Abhängigkeiten von Modulen. Daher ist es naheliegend ebenfalls die Abhängigkeiten der Anwendungen in der Anwendungslandschaft im Rahmen der Komplexität näher zu untersuchen.

Anzahl der Abhängigkeiten

Zunächst kann die Anzahl der Abhängigkeiten als komplexitätsbeeinflussend identifiziert werden. Eine Abhängigkeit ist durch das Tupel aus einer Clientanwendung und einer durch eine Serveranwendung bereitgestellte Schnittstelle definiert. Zur Bestimmung der Anzahl der Abhängigkeiten (auch: *Abhängigkeitszahl*) werden einfach die Anzahl solcher Tupel innerhalb der Anwendungslandschaft gezählt.

Abhängigkeitsmuster

Es ist allerdings nicht nur die Anzahl der Abhängigkeiten signifikant für die Komplexität einer Anwendungslandschaft, sondern vor allem auch die Art und Topologie derselbigen. Eine Sequenz von in Reihe geschalteten Elementen ist einfacher handhabbar und (intuitiv) weniger komplex als ein System, bei dem alle beteiligten Element mit allen jeweils anderen Elementen interagieren. Im ersten Fall bedingt eine Änderung der Schnittstelle eines Elements höchstens die Änderung eines anderen Elements. Im letzteren Fall jedoch hat die Änderung einer Elementschnittstelle potentiell Auswirkung auf alle anderen Elemente. Weitere Typen wurden bereits in Abschnitt 2.2.1 erläutert.

Daneben gibt es aber zusätzliche Merkmale im Abhängigkeitsgraphen, die es zu untersuchen gibt. Lilienthal fordert für eine geringe Komplexität von Architekturen eine Geordnetheit des Abhängigkeitsgraphen, die an den Attributen Zyklenausmaß, Zyklenreichweite, Zyklenumfang und Verflochtenheit (vgl. Abschnitt 3.6) festgemacht werden kann [Li08]. Generell kann die Verwendung von Mustern innerhalb einer Anwendungslandschaft auch deren Verstehenskomplexität verringern.

Notifikationstyp der Abhängigkeiten

Relevant für das Verstehen von Abhängigkeiten ist auch, auf welcher Seite der Abhängigkeit Server- und Clientrolle definiert sind. Anhand der Datenflussrichtung lässt sich dies nicht direkt ablesen. Daten können auf Anforderung (Polling) vom Server an den Client übertragen werden, oder regelmäßig vom Server an den Client geschoben werden (Push) oder in einer Event-Driven-Architecture jeweils nur dann übertragen werden, wenn dies tatsächlich notwendig ist.

Wichtigkeit erlangt dies für die Komplexität in dem Sinne, als dass zum vollständigen Verstehen der Abhängigkeiten (bzw. im Großen dann der Anwendungslandschaft) diese Typunterscheidung wichtig ist, um den Datenfluss korrekt zu verstehen als auch das Gesamtverhalten abzuleiten. Treten innerhalb einer Anwendungslandschaft viele verschiedene Notifikationstypen auf, so ist die Verstehenskomplexität höher als bei einheitlichen Notifikationstypen.

Nutzungsgrad von Anwendungslandschaftsmuster

Die Einführung von Mustern in Architekturen verringert die Verstehenskomplexität; daher bedingt die Standardisierung von Mustern innerhalb von Anwendungslandschaften einen Einfluss auf die Komplexität derselbigen.

Hagen beschreibt in [Ha07] einen Ansatz der Credit Suisse zur Reduktion von Komplexität innerhalb der Anwendungslandschaft der Bank. Dort wird versucht, Anwendungen nach fachlichen Domänen zu gruppieren. Innerhalb der Domänen ist eine hohe Kopplung zwischen den Anwendungen erlaubt; zwischen Domänen kann allerdings nur über einen generischen Integrationsbus kommuniziert werden. Dies führt zwischen den Domänen zu einer losen Kopplung. Diese Architekturvorgaben stellen ein Anwendungslandschaftsmuster dar.

Einen ähnlichen Ansatz verfolgt Quasar Enterprise der Capgemini sd&m [En08]: Hier werden Methoden, Regeln und Muster zur Verfügung gestellt, mit Hilfe derer sich eine optimale Architektur für die Anwendungslandschaft aus der Geschäftsarchitektur abbilden lässt. IT-Architektur wird hier weniger als Managementaufgabe verstanden sondern es werden konkrete Methoden zur Gestaltung von Anwendungslandschaften vorgeschlagen (wie z.B. die Etablierung einer Normalform für Services [HJ06]).

4.1.4 Organisationsbezogene Attribute

Anzahl der Stakeholder

“Viele Köche verderben den Brei”, mit diesem Spruch könnte man den Einfluss vieler Stakeholder auf die Komplexität der Anwendungslandschaft beschreiben. Aber auch in der Literatur wird die Vielzahl von Systemelementen als Komplexitätstreiber (für Projekte) identifiziert [Ba96]: je mehr unterschiedliche Interessensvertreter durch eine Anwendungslandschaft befriedigt werden müssen, desto mehr Konflikte muss es zwischen den Interessen geben. Doch je mehr Konflikte aufgelöst werden müssen, desto höher ist auch die Komplexität eines Vorhabens.

Anzahl und Dauer der Projekte

Verändern viele Projekte gleichzeitig die Anwendungslandschaft, ergeben sich viele Abhängigkeiten zwischen den Projekten selbst und zwischen Projekten und dem status quo der Anwendungslandschaft. Die zeitliche Abfolge spielt eine wesentliche Rolle: allein die Fragestellung, welches Projekt wann, parallel zu welchen anderen Projekten, mit welchen Ressourcen und mit welchen Abhängigkeiten zu einem bestimmten Status der Anwendungslandschaft durchgeführt werden kann, ist Teil der Arbeit eines Projektportfolio-Managers.

Neben der Anzahl der Projekte und deren Abhängigkeiten untereinander, die koordiniert werden müssen, erläutert Hirzel auch die Projektdauer und die hohe Wirtschaftsdynamik, die ein ständiges reorganisieren der Projektplanungen erfordern, als Komplexitätstreiber [Hi06].

Diversität der Geschäftsprozesse

Die Anzahl der verschiedenen Anwendungen hängt u.a. davon ab, wieviele verschiedene Geschäftsprozesse unterstützt werden müssen. Müssen etwa bei Organisationen wie Yamaha sowohl Prozesse zur Planung, Produktion und Verkauf von Motorrädern als auch für den Bau und Verkauf von Musikinstrumenten unterstützt werden, ist eine höhere Anzahl an Anwendungen notwendig; die beiden Geschäftsbereichen weisen bei den benötigten Anwendungen nämlich nur geringe Überlappungen auf.

Heterogenität der Geschäftsorganisation

Zusätzliche Komplexität bringt auch die Heterogenität der Geschäftsorganisation: abhängig von der geographischen Verteilung, aber auch von der Aufbauorganisation in unterschiedlichen Organisationseinheiten wird die Komplexität beeinflusst. Die geographische Verteilung erschwert hauptsächlich die Kommunikation zwischen verteilten Standorten: neben der aufwendigeren Kommunikation über Telefon, E-Mail oder Brief im Vergleich zur Face-To-Face Kommunikation erschweren unterschiedliche Arbeitszeiten an Standorten in verschiedenen Zeitzonen den Informationsaustausch der Mitarbeiter. Auch kulturelle Unterschiede können Risiken in sich bergen.

Daneben können Unterschiede in der Aufbauorganisation Komplexität in die Anwendungslandschaft induzieren: Soll etwa eine Anwendung zur Freigabe von Geldtransaktionen in zwei verschiedenen Organisationseinheiten eingesetzt werden, wobei einmal eine strikte Hierarchie zur Zahlungsfreigabe durchlaufen werden muss, auf der anderen Seite jedoch lediglich eine sehr flache Hierarchie existiert, so muss die Anwendung beide Organisationsstruktur unterstützen. Diese höheren Anforderungen schlagen sich so in der inhärenten Komplexität der Anwendung nieder und so (je nach Betrachtung) auch in der Komplexität der Anwendungslandschaft bzw. auch in anderen Eigenschaften wie der Wartbarkeit und Evolvability.

Inhärente Komplexität des Geschäfts

Maßgeblich beeinflusst aber die essentielle Komplexität des Geschäfts selbst auch die Komplexität der Anwendungslandschaft. Durch komplexe Geschäftsprozesse wird folglich auch Komplexität in die Anwendungslandschaft induziert. Ein Automobilhersteller besitzt allein aufgrund des komplexen Produkts auch komplexe Prozesse bei der Bestellabwicklung, welche durch Anwendungen unterstützt werden müssen. Andererseits sind bei einem Online-Buchhandel vergleichsweise einfache Prozesse für weniger komplexe Anwendungen verantwortlich.

Clustering der Anwendungslandschaft

Wie in Abschnitt 2.1.4 bereits beschrieben, können Anwendungslandschaften auf Basis verschiedener fachlicher oder technischer Kategorisierungen in Cluster eingeteilt werden. Für die Verstehenskomplexität ist die Einteilung in Cluster positiv: ein Betrachter kann sich auf den für ihn relevanten Teil konzentrieren und kann diesen Teil einfacher verstehen als ohne entsprechende Clusterbildung. Dies wird vor allem durch den strukturbildenden Prozess des Chunkings aus der kognitiven Psychologie gestützt [Li08].

Anzahl geographischer Betriebsorte

Auf der Seite zur Infrastrukturschicht ist es wichtig, die Anzahl der verschiedenen Betriebsorte einer Anwendung zu kennen. Die Verteilung einer Anwendung auf verschiedene Standorte erschwert nicht nur das Management der Anwendung, sondern beeinflusst deren (akzidentielle) Komplexität bereits während der Entwicklung (z.B. Verteilung der Softwareartefakte).

Anzahl geographischer Nutzungsorte

Ebenfalls komplexitätsbeeinträchtigend ist die Anzahl der geographischen Nutzungsorte [Wi07]: neben dem komplizierteren Deployments über verschiedenen Organisationseinheiten hinweg, müssen oft zusätzliche Faktoren bei der Entwicklung und Wartung beachtet werden, wie z.B. zusätzliche Sprachlokalisierungen pro Standort, zusätzliche Währungen, verschiedene steuerliche Systeme oder andere Zahlenformatierungen. Auch unterschiedliche Nutzungszeiträume müssen bei der Planung von Wartungsarbeiten berücksichtigt werden.

Qualität der Dokumentation

Um für Außenstehende die Komplexität zu erniedrigen eignet sich eine Architekturdokumentation. Ganz entscheidend ist die Qualität der Dokumentation. Nur eine klare, prägnante und eindeutige Dokumentation hilft einem Außenstehenden sich in die Anwendungslandschaft einzuarbeiten.

Zur Aufrechterhaltung der Dokumentation und der Qualität der Dokumentation sind allerdings geeignete Prozesse im Unternehmensarchitekturmanagement aufzusetzen. Auf

der einen Seite erhöhen diese Prozesse den Aufwand bei Änderungen in der Anwendungslandschaft auf der anderen Seite trägt die kontinuierliche Fortschreibung der Architekturdokumentation zu einem besseren Verständnis der Anwendungslandschaft bei.

Ein Problem ist jedoch den richtigen Detailgrad für die Dokumentation zu finden. Aier et al. weisen in [Ai08] darauf hin, dass eine zu detaillierte Architekturdokumentation auch zur Unwartbarkeit einer Unternehmensarchitektur führen kann.

4.2 Konzeptionelles Informationsmodell

In Anlehnung an den EAM Pattern Catalog [Bu08] und die Anforderungen an Metriken für Anwendungslandschaften aus [LS07] sollen die gerade identifizierten Eigenschaften und Attribute in einem Informationsmodell - einem sogenannten I-Pattern - zusammengefasst werden. Im vorigen Abschnitt wurden verschiedene Merkmale identifiziert, die als Komplexitätstreiber einzustufen sind. Daraus werden nun im Folgenden die wesentlichen Entitäten, die zur Komplexitätsbestimmung beitragen, extrahiert und zu einem Informationsmodell kombiniert. Die später beschriebenen Metriken werden auf dieses Informationsmodell zurückgreifen.

4.2.1 Modellbegriff

Als Basis für das Informationsmodell wird der Modellbegriff nach Stachowiak [St73] verwendet, der den allgemeinen Modellbegriff an drei Hauptmerkmalen definiert:

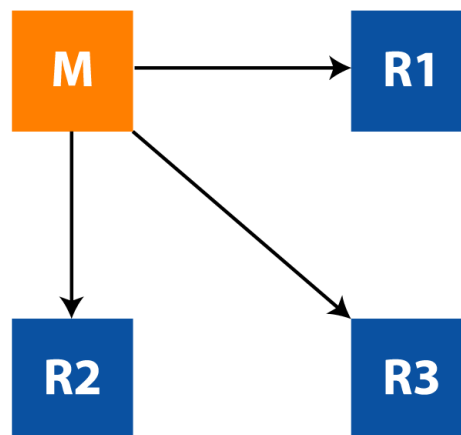
- **Abbildungsmerkmal** Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.
- **Verkürzungsmerkmal** Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaftern und/oder Modellbenutzern relevant erscheinen.
- **Pragmatisches Merkmal** Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion
 - a) für bestimmte – erkennende und/oder handelnde, modellbenutzende – Subjekte
 - b) innerhalb bestimmter Zeitintervalle und
 - c) unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.

Das Abbildungsmerkmal stellt sicher, dass alle Modelle auf ein natürliches oder künstliches Original zurückzuführen sind; allerdings kann eine Rückführung über mehrere hierarchische Abstraktionsebenen führen: Metamodelle allgemein beschreiben den Aufbau von anderen, konkreteren Modellen.

Die verkürzte Darstellung eines Originals ist notwendig, um dem Modell eine gewisse Sichtweise bzw. einen Zweck zu verschaffen: Ein Modell soll die Sicht auf ein Original zu einem bestimmten Zweck vereinfachen.

Modelle können sowohl textueller oder graphischer Natur sein. In der Informatik und verwandten Fachbereichen werden zur Modellierung von Prozessen und Architekturen oft graphische Repräsentationen verwendet, während in der Philosophie oder den Sprachwissenschaften textuell repräsentierte Modelle überwiegen.

Es ist wichtig zu erwähnen, dass eine reine graphische Darstellung selbst noch kein Modell darstellt. Erst wenn einer graphischen Darstellung eine definierte Notation, eine Syntax und eine passende Semantik zu Grunde liegen, kann eine graphische Darstellung ein Modell sein.



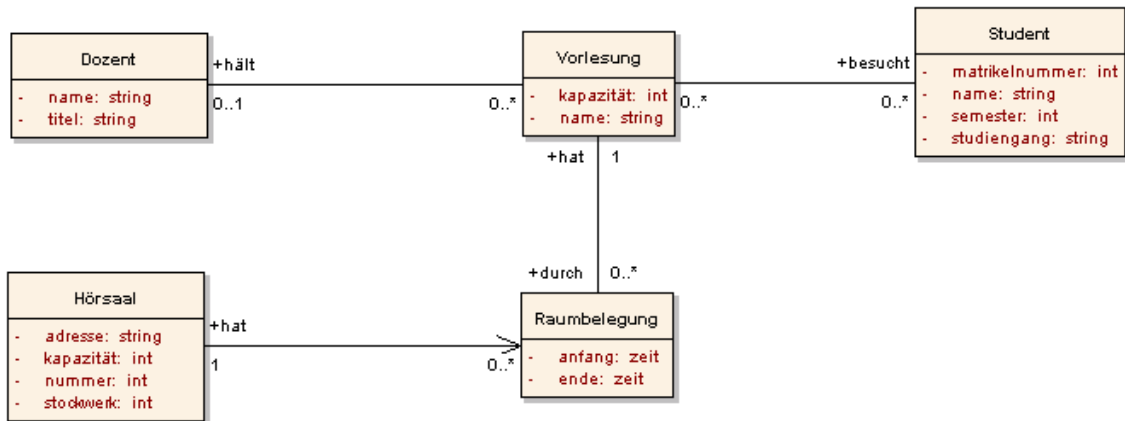
(Eigene Darstellung)

Abbildung 4.1: Graphische Darstellung einer Datenbankumgebung

Anhand von Abbildung 4.1 kann der Unterschied sehr deutlich. Die graphische Darstellung erschließt sich einem Betrachter ohne zusätzlichem Wissen oder Kommentare des Modellautors nicht. Erst wenn die einzelnen Elemente der graphischen Darstellung mit einer Semantik belegt werden, wird das einfache Schema einer Datenbankserver-Infrastruktur deutlich: ein orangenes Quadrat steht für einen Datenbankmaster, ein blaues für ein Datenbankreplika. Die Pfeile stehen für die Datenflussrichtung und die Bezeichner innerhalb der Quadrate sind die Namen der Rechner, auf denen die Datenbanksysteme laufen. Damit eine graphische Darstellung zum Modell wird, ist es also unbedingt notwendig die verwendete Notation, Syntax und Semantik zu kennen oder z.B. in einer Legende zu hinterlegen.

In der Informatik gibt es viele verschiedene Modellnotationen. Eine der gängigen Notationen ist die Unified Modeling Language (UML) [UML09]. Sie wird vor allem im Bereich der Softwaretechnik zur Modellierung von Anwendungsstruktur, -verhalten und Anwendungsarchitektur verwendet, findet allerdings auch Einsatz zur Prozessmodellierung.

UML ist dabei selbst ein sogenanntes Metamodell: Metamodelle sind "Übermodelle", also Modelle von Modellen. UML z.B. definiert ein Modell, wie andere Modelle konstruiert werden können. Es definiert also eine konkrete Syntax und eine Semantik, wie ein Modell von Entitäten aufgebaut werden könnte. In Abbildung 4.2 ist z.B. ein einfaches, sehr allgemein gehaltenes Klassendiagramm zu sehen, welches in einer Software zur Verwaltung des Vorlesungsbetriebs einer Universität zu finden sein könnte.



(Eigene Darstellung)

Abbildung 4.2: Einfaches Domänenmodell des Studienbetriebs einer Universität in UML-Notation

4.2.2 Informationsmodell

Um eine Datenbasis für den Einsatz von Komplexitätsmetriken zu erhalten, bedarf es einem strukturierten Prozess zur Datenerhebung. Neben dem Vorgehen zur Datenerhebung - welcher hier nicht weiter erläutert wird - wird ein strukturiertes Informationsmodell benötigt, nach welchem Daten erfasst werden müssen.

Definition: Informationsmodell Ein Informationsmodell (*engl.*: Information Model) ist ein Modell, welches die Semantik von Informationsobjekten, die möglichen Beziehungen zwischen den Informationsobjekten und die Attribute von Informationsobjekten definiert. Ein Informationsobjekt ist ein Abbild eines existierenden Objekts in einem betrieblichen Objektsystem [Wi07].

Im Folgenden wird ein Informationsmodell entwickelt, welches Eigenschaften einer Anwendungslandschaft in einem Informationsmodell abbildet, die im vorigen Abschnitt als Komplexitätstreiber identifiziert worden sind und nachfolgend quantifiziert werden. Dabei finden sich im Modell nur diejenigen Eigenschaften und Parameter, die für die nachfolgenden Metriken von Relevanz sind.

Betrachtet und ordnet man die oben genannten komplexitätstreibenden Merkmale und versucht diese in existierende Begriffssysteme einzuordnen ([Bu08], [VST05], [Br09]) kristallisieren sich folgende für das Informationsmodell relevante Entitäten heraus:

Anwendung (auch: BusinessApplication [Bu08], IT-Application Block [VST05]) Eine Anwendung ist ein Softwaresystem innerhalb einer Anwendungslandschaft; sie stellt Schnittstellen für andere Anwendungen bereit (Schnittstellenexport), nutzt Schnittstellen anderer Anwendungen (Schnittstellenimport). Die Anwendung ist dabei ein abstrakter Stellvertreter für Anwendungsinstanzen einer bestimmten Produktfamilie, Anwendungsversion und Anwendungskonfiguration, von denen mehrere real und parallel in Betrieb sein können (was sich im Attribut Instanzzahl niederschlägt). Eine Anwendung ist ferner Mitglied

in einer Domäne und nutzt ggf. mehrere Technologien.

Eine Anwendung besitzt folgende Attribute:

- **Alter** Einsatzdauer der Anwendung in Jahren.
- **AnzahlBetriebsorte** Anzahl der Orte, an denen die Anwendung installiert oder gehostet ist.
- **AnzahlNutzungsorte** Anzahl der Orte an denen eine Anwendung verwendet wird [Wi07].
- **ImplementiertArchitekturstil** Boolesches Flag, das wiedergibt ob eine Anwendung sich nach einem im Unternehmen definierten standardisierten Architekturstil richtet (*TRUE*) oder nicht (*FALSE*).
- **InhärenteKomplexität** Kennzahl für die inhärente Anwendungskomplexität.
- **Instanzzahl** Anzahl der installierten Instanzen der Anwendung.
- **Konfigurationstyp** Eindeutiges, stellvertretendes Kürzel für die Konfiguration der Anwendung; gleiche Versionen der Anwendung, die aber konfigurationsbedingt ein signifikant anderes Verhalten besitzen, müssen ein anderes Kürzel besitzen.
- **Produktfamilie** Eindeutiges Kürzel, das die Zugehörigkeit zu einer Softwareproduktfamilie kennzeichnet (z.B. SugarCRM, Microsoft Dynamics).
- **Standardanwendung** Boolesches Flag, ob die Anwendung ein Standardprodukt eines Fremdherstellers ist (*TRUE*). Ist *FALSE* falls die Anwendung eine Individualentwicklung oder signifikant an die Geschäftsprozesse angepasst worden ist.
- **Version** Eindeutiges Kürzel für die verwendete Anwendungsversion; falls es mehrere Versionen mit kompatiblen Schnittstellen gibt, so muss dieses Attribut ein Stellvertreter für alle diese Versionen sein (z.B. 5.2).

Eine Anwendung ist genau dann in Abhängigkeit von Version und Konfigurationstyp als eigenständige Anwendung zu erfassen, wenn

- die exportierten Schnittstellen der Anwendung nicht (rückwärts-)kompatibel zu früheren oder parallel installierten Instanzen der Anwendung sind

ODER

- das Verhalten der Anwendung signifikant von anderen parallel installierten Instanzen der Anwendung abweicht.

Schnittstelle (auch: Interface [Bu08], IS-Service [VST05]) Eine Schnittstelle ist eine technische Implementierung eines (ggf. nicht aktiv genutzten) Kommunikationskanals, der von einer Anwendung exportiert wird. Andere Anwendungen können diesen Kommunikationskanal verwenden (importieren).

Eine Schnittstelle besitzt folgende Attribute:

- **ImplementiertSchnittstellenmuster** Boolesches Flag, das *TRUE* ist für den Fall, dass die Schnittstelle einem im Unternehmen definiertem Schnittstellenmuster folgt, anderweitig *FALSE*.

Domäne Eine Domäne untergliedert eine Anwendungslandschaft in verschiedene, hierarchische Segmente. Die Gliederung ist dabei nicht vorgeben: Domänen können z.B. nach Organisationseinheiten oder Prozessen gebildet werden. Die Unterteilung nach Domänen kann aber auch durch andere Unterteilungen fachlicher, technischer oder organisatorischer Natur entspringen. Alle Domänen einer Anwendungslandschaft sind in einem Baum organisiert; d.h. es gibt eine Wurzel, von der aus man alle Unterdomänen erreichen kann.

Technologie Eine Technologie ist entweder eine Software, die zur Infrastrukturebene gehört (z.B. Datenbank, Betriebssystem, Middleware, Applikationsserver) oder ein technisches Verfahren (z.B. Verschlüsselungs-, Hashalgorithmen) (vgl. [Br09]).

Eine Technologie besitzt folgende Attribute:

- **Standardisiert** Boolesches Flag, ob eine Technologie innerhalb des Unternehmens als Standard deklariert wurde (*TRUE*) oder nicht (*FALSE*).

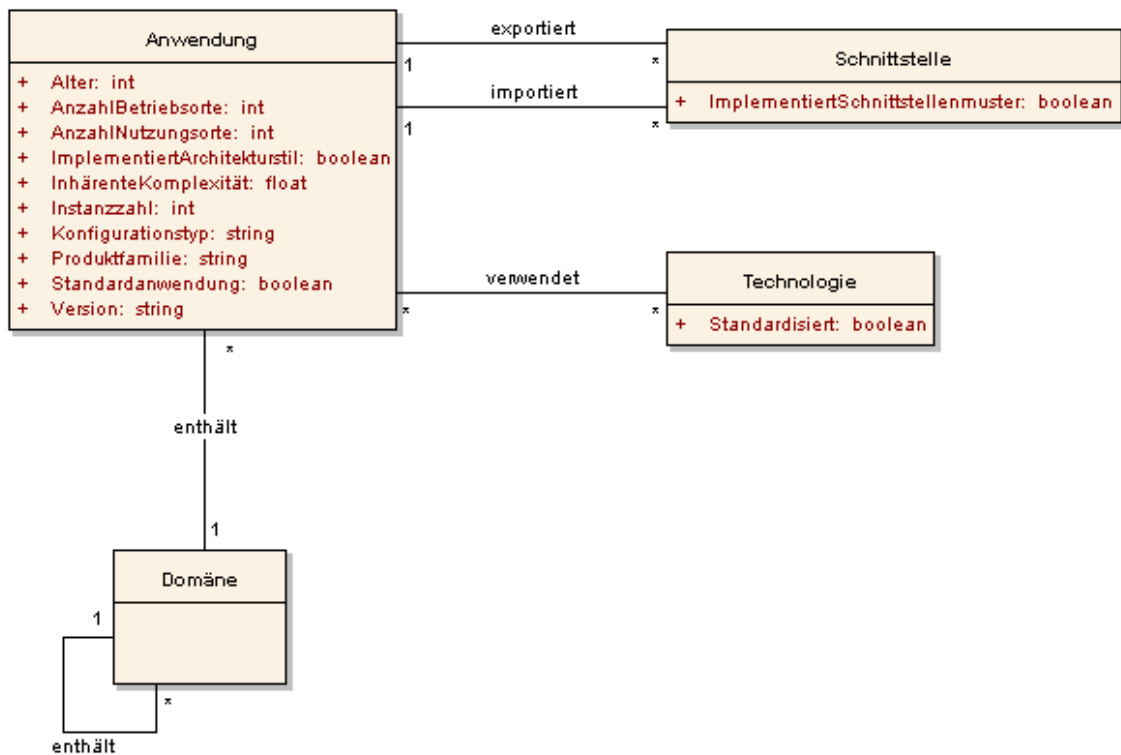
Abbildung 4.3 zeigt die Zusammenhänge der Entitäten graphisch und verdeutlicht die Kardinalitäten zwischen den Informationsentitäten.

4.2.3 Abhängigkeit vom Abstraktionsgrad

Wie bereits in Abschnitt 4.1.1 beschrieben ist eine Betrachtung der Komplexität von Anwendungslandschaften oft nur sinnvoll, sofern entweder ein passender Abstraktionsgrad gefunden wurde und/oder aber die inhärent Komplexität der Einzelanwendungen mit berücksichtigt wird.

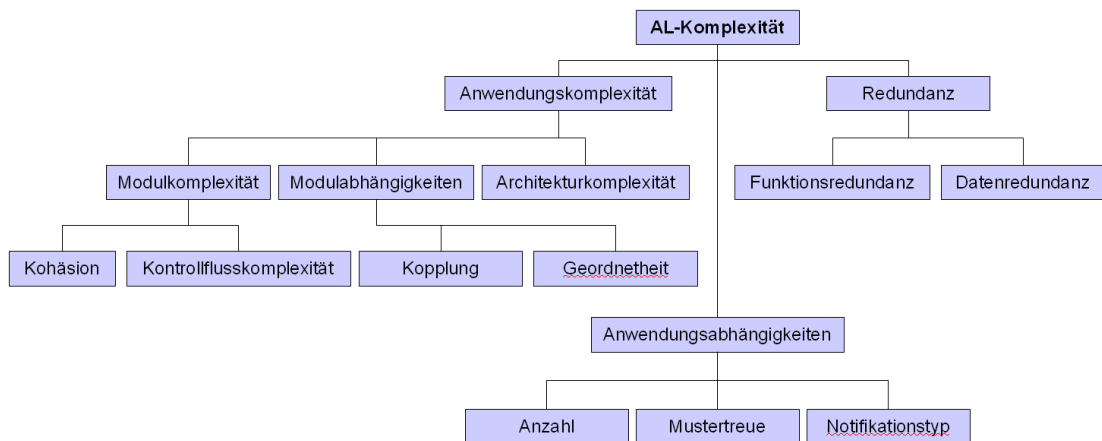
Aus diesem Grund bietet es sich an Komplexität nicht nur auf Ebene der Anwendungslandschaft zu betrachten, sondern eine ganzheitliche Betrachtung Bottom-Up durchzuführen: Um eine sinnvolle Aussage über die Komplexität einer Anwendungslandschaft zu treffen, muss die inhärente Komplexität der Anwendungen miteinbezogen werden, welche sich selbst wiederum durch die Komplexitätskennzahlen der darin enthaltenen Module oder Komponenten zusammensetzt. Auch Iacob und Jonkers benutzen ein ähnliches Modell um Parameter Bottom-Up über die gesamte IT-Architektur zu propagieren [IJ06].

In Abbildung 4.4 wurde versucht die Einzelgrößen der Komplexität miteinander in Beziehung zu setzen und deren Abstraktionsstufen und Abhängigkeiten voneinander zu verdeutlichen. Die Darstellung ist bei weitem nicht vollständig und beschränkt sich auf die in dieser Arbeit beschriebenen Merkmale.



(Eigene Darstellung)

Abbildung 4.3: Informationsmodell



(Eigene Darstellung)

Abbildung 4.4: Abstraktionsstufen der Komplexität

Nachdem die Bestimmung der Komplexitätsparameter bis ins Detail in der Praxis kaum durchführbar ist bzw. in keinem positiven Kosten-Nutzen-Verhältnis steht, verzichten die meisten nachfolgenden Metriken auf diesen ganzheitlichen Ansatz.

4.3 Quantifikation von Komplexitätsmerkmalen

Auf Basis des eben entwickelten Informationsmodell werden im Folgenden nun verschiedene Metriken vorgeschlagen, die eine Bewertung der Komplexität einer Anwendungslandschaft ermöglichen sollen. Dabei werden die bereits identifizierten Komplexitätsmerkmale quantifiziert. Bei einigen erwähnten Merkmalen (wie z.B. bei der Kapselung von Daten) ist eine Quantifizierung aber nicht möglich.

Die Metriken werden steckbriefartig in Anlehnung an [Ku07] strukturiert. Die verwendete Struktur ist allerdings vereinfacht und verzichtet auf die Auflistung Stakeholders, der Nennung von Soll- und Toleranzwerten, die je nach Organisation und Geschäft sehr unterschiedlich sein können. Zusätzlich wird für jede Metrik diskutiert, inwiefern ein Herunterbrechen der Komplexität auf einzelne Domänen möglich ist.

4.3.1 Hinweise zur Notation

Für die Notation werden folgende Kürzel und Notationen verwendet:

$AL = \{A_1, A_2, A_3, \dots, A_n\}$ Menge aller Anwendungen in der Anwendungslandschaft
$\mathbb{D} = \{D_1, D_2, D_3, \dots, D_m\}$ mit <ul style="list-style-type: none"> • $D_1, D_2, D_3, \dots, D_n \subseteq AL$ • $D_1, D_2, D_3, \dots, D_n$ paarweise disjunkt • $\bigcup_{d \in \mathbb{D}} d = AL$ Menge aller Domänen der Anwendungslandschaft. Jedes der D_i enthält alle Anwendungen, die der gleichen Domäne angehören. Für jede Domäne gibt es eine Menge D_i .
$\mathbb{S} = \{S_1, S_2, S_3, \dots, S_n\}$ mit <ul style="list-style-type: none"> • $S_1, S_2, S_3, \dots, S_n \subseteq AL$ • $S_1, S_2, S_3, \dots, S_n$ paarweise disjunkt • $\bigcup_{s \in \mathbb{S}} s = AL$ Menge aller Produktfamilien. Jedes der S_i enthält alle Anwendungen, die zur gleichen Produktfamilie gehören. Für jede Produktfamilie gibt es eine Menge S_i .

I	Menge aller Schnittstellen aller Anwendungen in AL
$I(a)$	Menge der importierten Schnittstellen der Anwendung $a \in AL$
$E(a)$	Menge der exportierten Schnittstellen der Anwendung $a \in AL$
$T(a)$	Menge der benutzten Technologien der Anwendung $a \in AL$
$D(a)$	Liefert die Domäne, der Anwendung $a \in AL$

Sofern (Klassen-)Attribute aus dem Informationsmodell referenziert werden, wird folgende Syntax verwendet: $[Entität].[Attribut]$. Wobei $[Entität]$ für eine Objektinstanz einer der Entitäten des Informationsmodell steht und $[Attribut]$ für den Namen des auszulesenden Attributs.

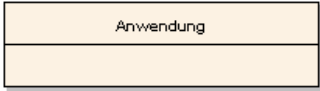
Steckbriefschema Die Metriken werden in folgendem steckbriefartigem Schema aufgelistet:


Name:	Beschreibender Name der Metrik
Kennung:	Kürzel der Metrik
Fokus:	Gibt die Ebenen an, auf der eine Auswertung der Metrik möglich ist. Mögliche Werte: Anwendungslandschaft, Domäne, Anwendung
Beschreibung:	Die Beschreibung der Metrik gibt Auskunft über die Zielsetzung der Metrik.
Definition:	Mathematische Definition der Metrik
Zielwert:	Interpretationshilfe für die Metrik; gibt an, ob ein hoher oder niedriger Wert anzustreben ist und ggf. in welchem Wertebereich sich die Metrik bewegen kann.
Aggregation:	Definiert die Berechnungsvorschriften auf Basis von Anwendungen oder Domänen und gibt hinweise zur Aggregation der einzelnen Metrikergebnisse der Domänen zu einem Gesamtwert für der Anwendungslandschaft
Informationsmodell:	An die Metrik angepasstes Informationsmodell in UML-Notation
Bemerkungen:	Diskussion und Bewertung der Metrik
Beispiel:	Beispielauswertung der Metrik mit Grafik und Beispielberechnung

4.3.2 Metriken

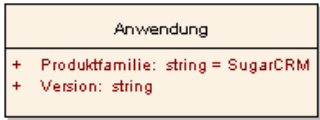

Analog zur Gliederung der Komplexitätsattribute, erfolgt im Folgenden eine Unterteilung in anwendungsbezogene, schnittstellenbezogene, abhängigkeitsbezogene und organisationsbezogene Metriken.

Anwendungsbezogene Metriken

Name:	Größe der Anwendungslandschaft
Kennung:	<i>AZ</i> , auch: Anwendungszahl
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	In Anlehnung an die Zählmetrik aus [VST05] bestimmt <i>AZ</i> die Größe der Anwendungslandschaft anhand der Anzahl der benutzten Anwendungen. Dies ist wie bereits beschrieben ein Indikator für Komplexität.
Definition:	$AZ = AL $
Zielwert:	Um Komplexität auf Ebene der Anwendungslandschaft zu vermeiden ist ein möglichst geringer Wert für <i>AZ</i> anzustreben.
Aggregation:	Die Metrik <i>AZ</i> kann auch für einzelne Domänen der Anwendungslandschaft betrachtet werden. <i>AZ</i> für die Domäne <i>D</i> ist wie folgt bestimmt: $AZ(D) = D $ <p>Die Komplexitätswerte der Domänen können aufsummiert werden; es gilt folgender Zusammenhang:</p> $AZ = \sum_{D \in \mathbb{D}} AZ(D)$
Informationsmodell:	
Bemerkungen:	<i>AZ</i> ist sehr einfach zu bestimmen. Eine geringe Anzahl an Anwendung lässt die Anwendungslandschaft weniger komplex erscheinen. Es muss jedoch der gewählte Abstraktionsgrad der Anwendungslandschaft mit berücksichtigt werden, da grobgranulare Anwendungen lediglich zu einer Kapselung der Komplexität innerhalb der Anwendungslandschaft führen.

Beispiel:	 <p style="text-align: center;">$AZ = 4$</p>
-----------	---

Name:	Durchschnittliche Versionsanzahl
Kennung:	<i>AVGVZ</i>
Fokus:	Anwendungslandschaft, Domäne, Anwendung
Beschreibung:	Bestimmt die durchschnittliche Anzahl verschiedener, installierter Anwendungsversionen auf Ebene der Anwendungslandschaft.
Definition:	$AVGVZ = \sum_{S \in \mathcal{S}} \frac{VZ(S)}{ \mathcal{S} }$ <p>mit</p> $VZ(S) = \{a \in S \mid \forall b \in S : b.Version \neq a.Version\} , S \in \mathcal{S}$
Zielwert:	Um Komplexität auf Ebene der Anwendungslandschaft zu vermeiden ist ein möglichst geringer Wert für <i>AVGVZ</i> anzustreben.



<p>Aggregation:</p>	<p>Die Metrik $AVGVZ$ kann auch für einzelne Domänen der Anwendungslandschaft betrachtet werden. $AVGVZ$ für die Domäne D ist wie folgt bestimmt:</p> $AVGVZ(D) = \frac{\sum_{S \in \mathbb{S}_D} VZ(S)}{ \mathbb{S}_D }$ <p>mit $\mathbb{S}_D = \bigcup_{S \in \mathbb{S}} S \cap D$</p> <p>Die Komplexitätswerte der Domänen können aufsummiert werden; es gilt folgender Zusammenhang:</p> $AVGVZ \leq \sum_{D \in \mathbb{D}} AVGVZ(D)$ <p>Auf Ebene einer einzelnen Anwendung bzw. einer Produktfamilie S ist die (durchschnittliche) Versionszahl durch $VZ(S)$ gegeben.</p>
<p>Informationsmodell:</p>	
<p>Bemerkungen:</p>	<p>Diese Metrik ist ein Indikator sowohl für die Komplexität, als auch die Diversität einer Anwendungslandschaft. Die Erfassung der einzelnen, installierten Anwendungsversionen ist jedoch aufwendig und drückt das Kosten-Nutzen Verhältnis.</p>
<p>Beispiel:</p>	 <p>$AVGVZ = 1,5$</p>

<p>Name:</p>	<p>Durchschnittliche Konfigurationsanzahl</p>
<p>Kennung:</p>	<p>$AVGKZ$</p>
<p>Fokus:</p>	<p>Anwendungslandschaft, Domäne, Anwendung</p>
<p>Beschreibung:</p>	<p>Bestimmt die durchschnittliche Anzahl verschiedener, sich signifikant voneinander unterscheidenden Anwendungskonfigurationen auf Ebene der Anwendungslandschaft.</p>

<p>Definition:</p>	$AVGKZ = \sum_{S \in \mathbb{S}} \frac{KZ(S)}{ \mathbb{S} }$ <p>mit</p> $KZ(S) = \{a \in S \mid \forall b \in S : b.Konfigurationstyp \neq a.Konfigurationstyp\} , S \in \mathbb{S}$
<p>Zielwert:</p>	<p>Um Komplexität auf Ebene der Anwendungslandschaft zu vermeiden ist ein möglichst geringer Wert für <i>AVGKZ</i> anzustreben.</p>
<p>Aggregation:</p>	<p>Die Metrik <i>AVGKZ</i> kann auch für einzelne Domänen der Anwendungslandschaft betrachtet werden. <i>AVGKZ</i> für die Domäne <i>D</i> ist wie folgt bestimmt:</p> $AVGKZ(D) = \frac{\sum_{S \in \mathbb{S}_D} KZ(S)}{ \mathbb{S}_D }$ <p>mit $\mathbb{S}_D = \bigcup_{S \in \mathbb{S}} S \cap D$</p> <p>Die Komplexitätswerte der Domänen können aufsummiert werden; es gilt folgender Zusammenhang:</p> $AVGKZ \leq \sum_{D \in \mathbb{D}} AVGKZ(D)$ <p>Auf Ebene einer einzelnen Anwendung bzw. Produktfamilie <i>S</i> ist die durchschnittliche Konfigurationszahl durch <i>KZ(S)</i> gegeben.</p>
<p>Informationsmodell:</p>	<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p style="text-align: center;">Anwendung</p> <ul style="list-style-type: none"> + Konfigurationstyp: string + Produktfamilie: string = SugarCRM </div>



Bemerkungen:	<p>Diese Metrik besitzt einen hohen Informationsbedarf, der das Kosten-Nutzen Verhältnis negativ beeinflusst. Die Bestimmung ist allerdings auch interessant, als dass diese Metrik gleichzeitig Indikator für die Diversität einer Anwendungslandschaft ist. Alternativ zu der oben beschriebenen Bestimmung von $KZ(S)$ kann für eine höhere Detailtreue $KZ(S)$ wie folgt bestimmt werden:</p> $KZ(S) = \{a \in S \mid \forall b \in S : b.Konfigurationstyp \neq a.Konfigurationstyp \wedge a.Version = b.Version\} , S \in \mathbb{S}$ <p>Hier werden die verschiedene Konfigurationstypen zusätzlich pro eingesetzter Anwendungsversion unterschieden.</p>
Beispiel:	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="border: 1px solid black; padding: 5px; width: 30%; text-align: center;"> <p><u>:Anwendung</u></p> <p>Konfigurationstyp = Default Produktfamilie = JBoss AS</p> </div> <div style="border: 1px solid black; padding: 5px; width: 30%; text-align: center;"> <p><u>:Anwendung</u></p> <p>Produktfamilie = Apache Konfigurationstyp = Webserver</p> </div> <div style="border: 1px solid black; padding: 5px; width: 30%; text-align: center;"> <p><u>:Anwendung</u></p> <p>Produktfamilie = Apache Konfigurationstyp = Proxy</p> </div> </div> <p>$AVGKZ = 1,5$</p>

Name:	Durchschnittliche Anwendungsinstanzen
Kennung:	<i>AVGIZ</i>
Fokus:	Anwendungslandschaft, Domäne, Anwendung
Beschreibung:	Bestimmt die durchschnittliche Anzahl installierter Anwendungen innerhalb der Anwendungslandschaft
Definition:	$AVGIZ = \sum_{a \in AL} \frac{a.Instanzzahl}{ AL }$
Zielwert:	Um Komplexität auf Ebene der Anwendungslandschaft zu vermeiden ist ein möglichst geringer Wert für <i>AVGIZ</i> anzustreben.
Aggregation:	<p>Die Metrik <i>AVGIZ</i> kann auch für einzelne Domänen der Anwendungslandschaft betrachtet werden. <i>AVGIZ</i> für die Domäne <i>D</i> ist wie folgt bestimmt:</p> $AVGIZ(D) = \sum_{a \in D} \frac{a.Instanzzahl}{ D }$ <p>Auf Ebene einer einzelnen Anwendung <i>a</i> ist die durchschnittliche Instanzzahl durch <i>a.Instanzzahl</i> gegeben.</p>



Informationsmodell:	
Bemerkungen:	Dient vor allem auch als Indikator für den durchschnittlichen Aufwand für den Rollout neuer Anwendungsversionen innerhalb der Anwendungslandschaft.
Beispiel:	 <p>$AVGIZ = 300$</p>

Name:	Durchschnittliches Anwendungsalter
Kennung:	AA
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Das Alter gibt einen Hinweis auf ggf. notwendige Erneuerungsmaßnahmen bzw. gilt als Indikator für die Häufigkeit Anwendungserneuerung und -überarbeitungen in der Vergangenheit.
Definition:	$AA = \sum_{a \in AL} \frac{a.Alter}{ AL }$
Zielwert:	Je geringer der Wert von AA desto jünger ist die Anwendungslandschaft.
Aggregation:	<p>Die Metrik kann auch auf Ebene von Domänen angewendet werden. Für eine Domäne D gilt folgende Definition:</p> $AA(D) = \sum_{a \in D} \frac{a.Alter}{ D }$ <p>Auf Basis der einzelnen Domänen kann eine obere Schranke für den Wert der gesamten Anwendungslandschaft bestimmt werden:</p> $AA \leq \sum_{D \in \mathbb{D}} AA(D)$



4 Komplexitätsmetriken für Anwendungslandschaften

Informationsmodell:	 <pre> classDiagram class Anwendung { + Alter: int } </pre>
Bemerkungen:	<p>Das Alter kann in unterschiedlicher Weise interpretiert werden: eine junge Anwendungslandschaft kann für ein junges Unternehmen stehen oder für eine runderneuerte Anwendungslandschaft, in der die Einzelanwendungen noch viele Fehler enthalten können.</p> <p>Eine alte Anwendungslandschaft kann auf der einen Seite für bewährte und ausgereifte Einzelanwendungen stehen oder auf eine schlecht wartbare, verkrustete Anwendungslandschaft hindeuten. Je nach Organisation, kann eine alte Anwendungslandschaft aber auch auf selten geänderte Geschäftsprozesse hinweisen.</p>
Beispiel:	 <p> $K_{GZ} = \frac{8+2+5}{3} = 5$ </p>

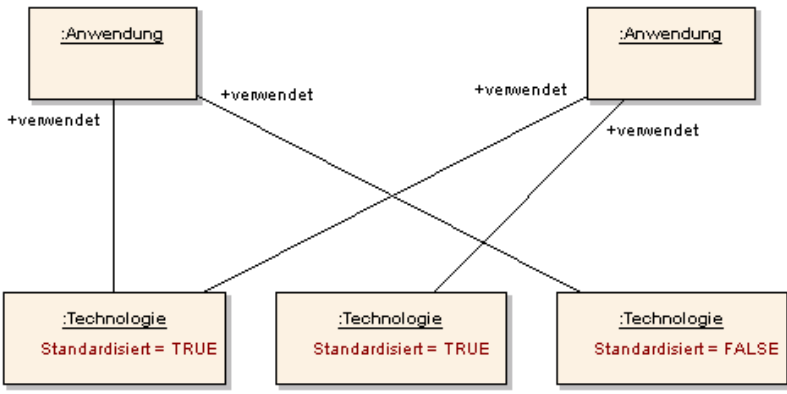
Name:	Inhärente Komplexitäten
Fokus:	Anwendungslandschaft, Domäne, Anwendung
Kennung:	<i>INH</i>
Beschreibung:	<p>Wie bereits beschrieben, hängen die meisten vorgestellten Metriken ab vom Abstraktionsgrad der Anwendungslandschaft. In dieser Metrik soll dies nun aufgelöst werden in auch die inhärente Komplexität der einzelnen Anwendungen mitbetrachtet wird, wie im Abschnitt 4.2.3 bereits angesprochen.</p> <p>Vorraussetzung für die Anwendung dieser Metrik ist das Vorliegen der inhärenten Komplexitätsparameter der einzelnen Anwendungen, die z.B. nach bekannten Codemetriken bestimmt werden können [Zu94].</p>
Definition:	$INH = \sum_{a \in AL} a.inhärenteKomplexität$
Zielwert:	Ein niedriger Wert dieser Metrik ist erstrebenswert.

<p>Aggregation:</p>	<p>Diese Metrik eignet sich per Design für eine Bottom-Up Betrachtung: Zunächst werden die Komplexitäten der Einzelanwendungen bestimmt, dann die für Domänen und anschließend für die gesamte Anwendungslandschaft. Für die Domäne D kann INH wie folgt bestimmt werden:</p> $INH(D) = \sum_{a \in D} a.inhärenteKomplexität$ <p>Diese können dann wiederum für die Anwendungslandschaft aggregiert werden:</p> $INH = \sum_{D \in \mathbb{D}} INH(D)$ <p>Auf Ebene einer einzelnen Anwendung a ist der inhärente Komplexität durch $a.inhärenteKomplexität$ gegeben.</p>
<p>Informationsmodell:</p>	 <pre> classDiagram class Anwendung { + InhärenteKomplexität: float } </pre>
<p>Bemerkungen:</p>	<p>Die Bestimmung der inhärenten Komplexität von Anwendungen kann an die Gegebenheiten im Unternehmen angepasst werden. Zu beachten bei der Auswahl der geeigneten Metrik sei jedoch, dass diese mit höherer Komplexität einen höheren Wert liefern und dass alle Anwendungen in der Anwendungslandschaft dem gleichen Komplexitätsmaß unterzogen werden. Beispiele für geeignete Metriken sind z.B. die zyklomatische Komplexität [Mc76], Kohäsions- [OT93] und Kopplungsmetriken [Zu94] als auch Metriken zur Bestimmung der strukturellen Komplexität wie in [Ro96] oder [WHH79] beschrieben. Für eine ganzheitliche Sicht auf die Anwendungslandschaft sollte jedoch wie in Abschnitt 4.2.3 beschrieben neben der inhärenten Komplexität der Einzelanwendung zusätzlich noch Metriken einbezogen werden, die die Abhängigkeiten zwischen den Anwendungen berücksichtigen.</p>
<p>Beispiel:</p>	 <p>$INH = 90$</p>



4 Komplexitätsmetriken für Anwendungslandschaften

Name:	Nutzungsgrad von Standardanwendungen
Kennung:	<i>NS</i>
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Der Nutzungsgrad von Standardanwendungen gibt den Anteil an (unveränderten) Standardanwendungen innerhalb der Anwendungslandschaft wieder.
Definition:	$NS = \frac{ SS }{ AL }$ <p>mit</p> $SS = \{a \in AL \mid a.\text{Standardanwendung} = TRUE\}$
Zielwert:	Ein möglichst hoher Wert (nahe 1) entspricht einer hohen Nutzung von Standardanwendungen. Ein hoher Wert ist erstrebenswert.
Aggregation:	<p>Auf Ebene von Domänen kann die Metrik wie folgt definiert werden:</p> $NS(D) = \frac{ SS_D }{ D }, D \in \mathbb{D}$ <p>mit</p> $SS_D = \{a \in D \mid a.\text{Standardanwendung} = TRUE\}$
Informationsmodell:	 <pre> classDiagram class Anwendung { + Standardanwendung: boolean } </pre>
Bemerkungen:	Die Nutzung von vielen Standardanwendungen erleichtert das Verstehen der Anwendungslandschaft für Außenstehende, erschwert jedoch das Abbilden von speziellen Geschäftsprozessen eines Unternehmens.
Beispiel:	 <p><i>NS</i> = 0.25</p>

Name:	Standardisierungsgrad der Anwendungstechnologien
Kennung:	<i>SAT</i>
Fokus:	Anwendungslandschaft, Domäne, Anwendung
Beschreibung:	Der Standardisierungsgrad einer Anwendungslandschaft kann in Anlehnung an die Spread-Metrik von [Br09] bestimmt werden.
Definition:	$SAT := \sum_{a \in AL} \frac{STD(a)}{ AL }$ <p>mit</p> $STD(a) := \frac{ \{t \in T(a) \mid t.Standardisiert = TRUE\} }{ T(a) }$ <p><i>STD(a)</i> entspricht dem Standardisierungsgrad der Anwendung <i>a</i>.</p>
Zielwert:	Der Wertebereich der Metrik ist das Intervall [0, 1]. Je höher der Standardisierungsgrad, desto niedriger ist die Komplexität, da die Anwendungslandschaft mehr Muster ($\hat{=}$ Standards) befolgt. Dies vermindert den notwendigen Aufwand für das Verstehen der Landschaft, da das menschliche Gehirn den Aufbau von Schemata leicht beherrscht [Li08].
Aggregation:	Die Metrik kann auf Domänen einer Anwendungslandschaft angewendet werden. Die Komplexität einer Domäne <i>D</i> errechnet sich wie folgt:
Informationsmodell:	<pre> classDiagram class Anwendung class Technologie { + Standardisiert: boolean } Anwendung "x" -- "x" Technologie : +verwendet </pre>

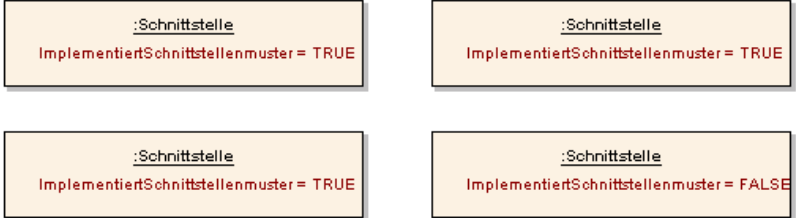
<p>Bemerkungen:</p>	<p>Der Einsatz dieser Metrik erfordert eine saubere Erfassung aller verwendeten Technologien und die Definition von Standardtechnologien. Brückmann et al. [Br09] beschreiben dazu einen geeigneten Ansatz. Da dieser jedoch hauptsächlich Technologien in Kategorien einordnet, muss der Ansatz für diese Metrik ein wenig modifiziert werden:</p> <p>Brückmann et al. teilen die Technologien (z.B. RSA, AES) in (hierarchische) Rubriken (z.B. Verschlüsselungsalgorithmen) ein. Jeder Technologie wird einer der folgenden Stati zugeordnet: <i>proposed</i>, <i>productive</i>, <i>standard</i> und <i>retired</i>. Für die einzelnen Rubriken kann dann ein Standardisierungsgrad berechnet werden, der sich auch hierarchisch Bottom-Up akkumulieren lässt.</p> <p>In unserem Ansatz vereinfachen wird das Modell, so dass keine Unterteilung der verwendeten Technologien in Rubriken erforderlich ist und auch lediglich die Technologien in <i>Standard</i> und <i>Nicht-Standard</i> eingeteilt werden müssen. Dies vermindert die Kosten zur Ermittlung dieser Metrik.</p>
<p>Beispiel:</p>	 <p>$SAT = 0,75$</p>

Name:	Standardisierungsgrad der Anwendungsarchitekturstile
Kennung:	SA
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Gibt den Verbreitungsgrad von standardisierten Architekturstilen innerhalb der Anwendungslandschaft wieder.

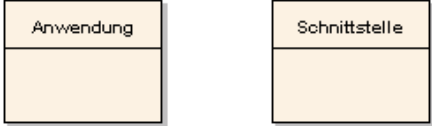

<p>Definition:</p>	$SA = \frac{ SA }{ AL }$ <p>mit</p> $SA = \{a \in AL \mid a.ImplementiertArchitekturstil = TRUE\}$
<p>Zielwert:</p>	<p>Für ein einfacheres Verständnis der Anwendungslandschaft ist eine hohe Verbreitung von standardisierten Anwendungsarchitekturstilen förderlich. Ein hoher Wert (nahe 1) ist daher erstrebenswert.</p>
<p>Aggregation:</p>	<p>Auf Domänenebene kann diese Metrik ebenfalls berechnet werden; es gilt für Domäne D:</p> $SA(D) = \frac{ SA_D }{ D }, D \in \mathbb{D}$ <p>mit</p> $SA_D = \{a \in D \mid a.ImplementiertArchitekturstil = TRUE\}$
<p>Informationsmodell:</p>	 <pre> classDiagram class Anwendung { + ImplementiertArchitekturstil: boolean } </pre>
<p>Bemerkungen:</p>	<p>Die Datenerhebung für diese Metrik ist nicht trivial; zunächst muss in einer Organisation definiert sein, welche Architekturstile als standardisiert gelten und als Referenz für neue Projekte gelten sollen. Weiterhin muss gesichert sein, dass der Architekturstil im Nachhinein festgestellt werden kann; gerade bei eingekauften Standardanwendungen ist dies meist nicht zuverlässig möglich. Aber selbst mit einer vorhandenen Architekturdokumentation, aus der der Architekturstil bestimmt werden kann, kann die tatsächliche Anwendung Abweichungen davon aufweisen, sofern die Architekturdokumentation nicht mehr aktuell ist (z.B. durch spätere Weiterentwicklungen, unsaubere Programmierung).</p>
<p>Beispiel:</p>	 <p>$SA = 0.75$</p>

Schnittstellenbezogene Metriken

Name:	Grad der Schnittstellenstandardisierung
Kennung:	<i>SI</i>
Fokus:	Anwendungslandschaft, Domäne, Anwendung
Beschreibung:	Maß zum Einsatzgrad von standardisierten Schnittstellenmustern.
Definition:	$SI = \frac{ STDI }{ I }$ <p>mit</p> $STDI = \{i \in I \mid i.ImplementiertSchnittstellenmuster = TRUE\}$
Zielwert:	Für eine geringe Verstehenskomplexität ist ein hoher Wert (nahe 1) anzustreben.
Aggregation:	<p>Auf Anwendungsebene kann der Standardisierungsgrad der exportierten Schnittstellen berechnet werden. Dafür gilt:</p> $SI(A) = \frac{ STDI_A }{E(A)}$ <p>mit</p> $STDI_A = \{i \in E(A) \mid i.ImplementiertSchnittstellenmuster = TRUE\}$ <p>Für eine Domäne kann ebenfalls ein Wert berechnet werden:</p> $SI(D) = \frac{ STDI_D }{ I_D }$ <p>mit</p> $I_D = \bigcup_{A \in D} E(A)$ $STDI_D = \{i \in I_D \mid i.ImplementiertSchnittstellenmuster = TRUE\}$
Informationsmodell:	<pre> classDiagram class Anwendung class Schnittstelle { + ImplementiertSchnittstellenmuster: boolean } Anwendung "1" -- "*" Schnittstelle : +exportiert </pre>

Bemerkungen:	Für die Datenerhebung und die anschließende Anwendung ist es entscheidend, dass vordefinierte Schnittstellenmuster (z.B. nach [HJ06]) innerhalb der Organisation definiert wurden. Desweiteren ist eine entsprechend ausführliche und aktuelle Architekturdokumentation notwendig; v.a. bei eingekauften Standardanwendungen ist dies aber nicht immer gewährleistet.
Beispiel:	 <p>$SI = 0,75$</p>


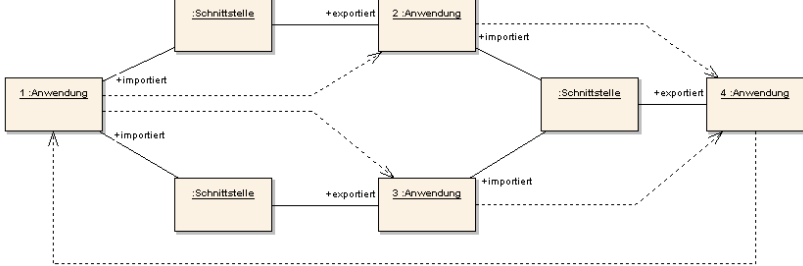
Name:	Schnittstellenausmaß
Kennung:	IA
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Diese Metrik betrachtet neben der Anwendungszahl auch die Anzahl der Schnittstellen der Anwendungen. Die Idee dieser Metrik geht dabei auf einen Ansatz von Niemann [Ni05] zurück, der die Komplexität in Abhängigkeit von $ AL $ und $ I $ definiert.
Definition:	$IA = f(AL , I) = AL I $
Zielwert:	Ein niedriger Wert für IA ist erstrebenswert, da die Anwendungslandschaft insgesamt geringer ist.

<p>Aggregation:</p>	<p>Ist eine detaillierte Betrachtung einer Domäne notwendig, kann diese Metrik auch auf einzelne Domänen wie folgt angewendet werden:</p> $IA(D) = D I_D $ <p>mit</p> $I_D = \bigcup_{A \in D} I(A)$ <p>Die Komplexitätswerte der Domänen können aufsummiert werden; es gilt folgender Zusammenhang:</p> $IA \leq \sum_{D \in \mathbb{D}} IA(D)$
<p>Informationsmodell:</p>	
<p>Bemerkungen:</p>	<p>Niemann [Ni05] definiert lediglich die Abhängigkeit der Komplexität von AL und I. Wie die Funktion zur konkreten Berechnung aussieht ist jedoch nicht gegeben. Es wurde daher in Anlehnung an Baccarinis [Ba96] Auffassung von Komplexität, die sich auf die Eckpunkte Differenzierung - also der Anzahl unterschiedlicher Elemente - und Konnektivität bzw. Abhängigkeiten der Elemente stützt, ein einfacher Ansatz gewählt, diese beiden Größen in einer Berechnung zu kombinieren. Dies hat zur Folge, dass der Informationsbedarf sehr gering ist. Durch die einfache Berechnung ist diese Metrik allerdings auch in ihrer Aussagekraft beschränkt.</p>
<p>Beispiel:</p>	 <p>$IA = 6$</p>

Abhängigkeitsbezogene Metriken

Name:	Anzahl der Abhängigkeiten
Kennung:	<i>DZ</i>
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Gibt die Anzahl der Abhängigkeiten zwischen den Anwendungen wieder.
Definition:	$DZ = \sum_{a \in AL} I(a) $
Zielwert:	Je geringer der Wert von <i>DZ</i> desto geringer die Verstehenskomplexität. Ein niedriger Wert ist daher anzustreben.
Aggregation:	<p>Die Anzahl der Abhängigkeiten kann auch auf Ebene von einzelnen Domänen betrachtet werden. Dazu wird folgende Hilfsfunktion definiert:</p> <p>$I(a, D) :=$ Menge aller importierten Schnittstellen der Anwendung <i>a</i>, die innerhalb der Domäne <i>D</i> exportiert werden.</p> <p>Für die Metrik auf Ebene der Domäne $D \in \mathbb{D}$ gilt dann:</p> $DZ(D) = \sum_{a \in D} I(a, D) $
Informationsmodell:	
Bemerkungen:	Die Erhebung der Daten ist einfach; allerdings ist die Aussagekraft beschränkt, da die Komplexität im Wesentlichen nicht nur von der Anzahl der Abhängigkeiten, sondern vor allem von durch Abhängigkeiten gebildete Muster und Strukturen abhängt.
Beispiel:	<p>$DZ = 2$</p>

Name:	Zyklomatische Komplexität von Anwendungslandschaften
Kennung:	<i>MC</i>
Beschreibung:	Vasconcelos [VST05] transformiert den Ansatz der zyklomatischen Komplexität auf Anwendungslandschaften. Die vorgeschlagene Metrik bedarf aber einer speziellen und komplizierten Datenbasis, weshalb hier eine simplifizierte Form vorgeschlagen wird. Diese ist der ursprünglichen Idee der zyklomatischen Komplexität ähnlicher.
Definition:	$MC = e - AL + 2p$ <p>mit</p> <ul style="list-style-type: none"> • $e = \sum_{A \in AL} I(A) \hat{=}$ Anzahl aller Abhängigkeiten (d.h. Schnittstellenimporte) • $p \hat{=}$ Anzahl der starken Zusammenhangskomponenten auf dem Graph der Anwendungslandschaft
Zielwert:	Wie schon bei den McCabe zugrundeliegenden Code-Metriken, soll auch bei Anwendungslandschaften die zyklomatische Komplexität möglichst niedrig sein, um ein einfacheres Management zu ermöglichen.
Aggregation:	<p>Die zyklomatische Komplexität kann auch auf einzelne Domänen einer Anwendungslandschaft berechnet werden. Dazu wird jeweils nur der Graph für die Anwendungen und Abhängigkeiten einer Domäne berechnet:</p> $MC(D) = e - A(D) + 2p$ <p>mit</p> <ul style="list-style-type: none"> • $e = \sum_{A \in D} I(A) \hat{=}$ Anzahl aller Abhängigkeiten (d.h. Schnittstellenimporte) • $p \hat{=}$ Anzahl der starken Zusammenhangskomponenten auf dem Graph der Domäne <i>D</i> der Anwendungslandschaft <p>Problematisch ist jedoch die Berechnung bzw. Abschätzung der zyklomatischen Komplexität mit Hilfe der Einzelkomplexitätswerte der Segmente. Dies gelingt nur sofern die Domänen disjunkt sind mit folgendem Zusammenhang, der auch von McCabe [Mc76] beschrieben wird:</p> $MC = \sum_{D \in \mathbb{D}} MC(D)$

Informationsmodell:	
Bemerkungen:	<p>Die Bestimmung der starken Zusammenhangskomponenten kann wie folgt erreicht werden: Zunächst muss der Graph der Anwendungslandschaft aus dem vorliegenden Informationsmodell errechnet werden. Dazu wird jede Anwendungsinstanz zu einem Knoten umgewandelt. Jede importierende Schnittstellenbeziehung wird dann als gerichtete Kante von der importierenden Anwendung zur exportierenden Anwendung zum Graph hinzugefügt. Mit Hilfe von Tarjans Algorithmus [Ta72] können dann die starken Zusammenhangskomponenten in linearer Zeit berechnet.</p>
Beispiel:	 <p>Die durchgezogenen Kanten entsprechen den Kanten im Informationsmodell; die gestrichelten Kanten entsprechen den vereinfachten Kanten zur direkten Anwendung für die McCabe-Metrik. Die Rückwärtskante von Anwendung 4 zu Anwendung 1 wurde automatisch eingefügt (nach topologischer Sortierung). $MC = 2$</p>

Name:	Zyklenausmaß
Kennung:	ZA
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Das Zyklenausmaß lehnt sich an eine Metrik aus [Li08] an, welche zur Bestimmung der Geordnetheit von Softwarearchitekturen dient. Diese Metrik wurde aufgrund der analogen Strukturen hier auf Anwendungslandschaften transformiert.
Definition:	$ZA :=$ Anzahl der Anwendungen in Abhängigkeitszyklen
Zielwert:	Ein möglichst niedriger Wert ist anzustreben.

4 Komplexitätsmetriken für Anwendungslandschaften

Aggregation:	Diese Metrik kann auch lokal für Domänen betrachtet werden. Dazu wird lediglich ein Teilgraph des Abhängigkeitsgraphs berechnet/betrachtet, der alle zur Domäne gehörenden Anwendungen enthält. In diesem Teilgraphen, werden dann ebenfalls Zyklen gesucht und die dazugehörigen Anwendungen gezählt.
Informationsmodell:	
Bemerkungen:	Die Bestimmung der Zyklen kann wie folgt erreicht werden: Zunächst muss der Graph der Anwendungslandschaft aus dem vorliegenden Informationsmodell errechnet werden. Dazu wird jede Anwendungsinstanz zu einem Knoten umgewandelt. Jede importierende Schnittstellenbeziehung wird dann als gerichtete Kante von der importierenden Anwendung zur exportierenden Anwendung zum Graph hinzugefügt (vgl. Beispiel). Gleicher Informationsbedarf wie <i>MC</i> .
Beispiel:	<p>$ZA = 0$ (kein Zyklus)</p>



Name:	Verflochtenheit
Kennung:	VF
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Das Maß der Verflochtenheit lehnt sich an eine Metrik aus [Li08] an, welche zur Bestimmung der Geordnetheit von Softwarearchitekturen dient. Diese Metrik wurde aufgrund der analogen Strukturen zwischen Software- und IT-Unternehmensarchitekturen hier auf Anwendungslandschaften transformiert.
Definition:	$VF :=$ Anzahl der direkt, bidirektionalen Abhängigkeiten innerhalb von Zyklen des Anwendungslandschaftgraphs.

Zielwert:	Ein möglichst niedriger Wert ist anzustreben.
Aggregation:	Diese Metrik kann auch lokal für Domänen betrachtet werden. Dazu wird lediglich ein Teilgraph des Abhängigkeitsgraphs berechnet/betrachtet, der alle zur Domäne gehörenden Anwendungen enthält. In diesem Teilgraphen, werden dann ebenfalls Zyklen gesucht und die darin enthaltenen direkt, bidirektionalen Abhängigkeiten gezählt.
Informationsmodell:	
Bemerkungen:	Die Bestimmung der Zyklen kann wie folgt erreicht werden: Zunächst muss der Graph der Anwendungslandschaft aus dem vorliegenden Informationsmodell errechnet werden. Dazu wird jede Anwendungsinstanz zu einem Knoten umgewandelt. Jede importierende Schnittstellenbeziehung wird dann als gerichtete Kante von der importierenden Anwendung zur exportierenden Anwendung zum Graph hinzugefügt (vgl. Beispiel). Gleicher Informationsbedarf wie <i>MC</i> und <i>ZA</i> .
Beispiel:	<p>$VF = 1$</p>


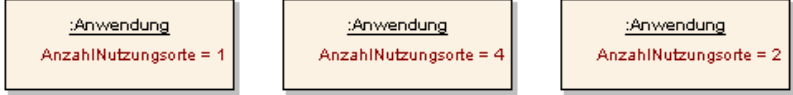
Organisationsbezogene Metriken

Name:	Durchschnittliche Anzahl an Betriebsorten
Kennung:	<i>AVGHL</i>
Fokus:	Anwendungslandschaft, Domäne

4 Komplexitätsmetriken für Anwendungslandschaften

Beschreibung:	Die Anzahl an Betriebsorten widerspiegelt den Aufwand für den Rollout von neuen Anwendungsversionen innerhalb der Anwendungslandschaft.
Definition:	$AVGHL = \sum_{A \in AL} \frac{A.AnzahlBetriebsorte}{ AL }$
Zielwert:	Je geringer der Wert von <i>AVGHL</i> desto weniger aufwändig ist das Deployment von neuen Anwendungsversionen, da weniger Betriebsorte mit der neuen Anwendungsversion versorgt werden müssen.
Aggregation:	Die Metrik kann auch auf Ebene von Domänen angewendet werden. Für eine Domäne <i>D</i> gilt folgende Definition: $AVGHL(D) = \sum_{A \in D} \frac{A.AnzahlBetriebsorte}{ D }$
Informationsmodell:	 <pre> classDiagram class Anwendung { + AnzahlBetriebsorte: int } </pre>
Bemerkungen:	Diese Metrik stellt auch einen Indikator für den Aufwand zum Rollout neuer Versionen oder Anwendungen dar.
Beispiel:	 $AVGHL = \frac{1+3+2}{3} = 2$

Name:	Durchschnittliche Anzahl an Nutzungsorten
Kennung:	<i>AVGUL</i>
Fokus:	Anwendungslandschaft, Domäne
Beschreibung:	Die Anzahl an Nutzungsorten widerspiegelt den Aufwand für den Rollout von neuen Anwendungsversionen innerhalb der Anwendungslandschaft, aufgrund der ggf. notwendigen Schulungsaufwände des Personals in den nutzenden Organisationseinheiten.

Definition:	$AVGUL = \sum_{A \in AL} \frac{A.AnzahlNutzungsorte}{ AL }$
Zielwert:	Je geringer der Wert von <i>AVGUL</i> desto weniger aufwändig ist das Deployment von neuen Anwendungsversionen, da potentiell weniger Schulungsaufwände anfallen.
Aggregation:	Die Metrik kann auch auf Ebene von Domänen angewendet werden. Für eine Domäne <i>D</i> gilt folgende Definition: $AVGUL(D) = \sum_{A \in D} \frac{A.AnzahlNutzungsorte}{ D }$
Informationsmodell:	 <pre> classDiagram class Anwendung { + AnzahlNutzungsorte: int } </pre>
Bemerkungen:	Diese Metrik stellt auch einen Indikator für den Aufwand zum Rollout neuer Versionen oder Anwendungen dar.
Beispiel:	 <p> $AVGUL = \frac{1+3+2}{3} = 2$ </p>

4.4 Zusammenfassung der Metriken

Die vorgestellten Metriken alleine sind kein allumfassendes Maß für die Komplexität einer Anwendungslandschaft. Sie evaluieren jeweils nur einen bestimmten Aspekt von Komplexität. Daher werden die Metriken nun noch im Folgenden in einer Übersicht zusammengetragen.

Die Erfassung der Domänen im Informationsmodell ist jeweils nur notwendig, falls die Betrachtung auf Domänenebene erwünscht ist. Daher sind die Einträge in der Spalte "Informationsmodell Domäne" in Klammern markiert.

4 Komplexitätsmetriken für Anwendungslandschaften

Metrik	Fokus	Betrachtungsebene		Informationsmodell				
		AL	Domäne	Anwendung	Anwendung	Schnittstelle	Technologie	Domäne
<i>AZ</i>	Größe des Systems	X	X		X			(X)
<i>AVGVZ</i>	Größe des Systems	X	X	X	X			(X)
<i>AVGKZ</i>	Größe des Systems	X	X	X	X			(X)
<i>AVGIZ</i>	Größe des Systems	X	X	X	X			(X)
<i>AA</i>	Legacy-Anteil	X	X		X			(X)
<i>INH</i>	Holistische Komplexität	X	X	X	X			(X)
<i>NS</i>	Standardkonformität	X	X		X			(X)
<i>SAT</i>	Standardkonformität	X	X	X	X		X	(X)
<i>SA</i>	Standardkonformität	X	X		X			(X)
<i>SI</i>	Standardkonformität	X	X		X	X		(X)
<i>IA</i>	Größe des Systems	X	X		X	X		(X)
<i>DZ</i>	Größe des Systems	X	X		X	X		(X)
<i>MC</i>	strukturelle Komplexität	X	X		X	X		(X)
<i>ZA</i>	strukturelle Komplexität	X	X		X	X		(X)
<i>VF</i>	strukturelle Komplexität	X	X		X	X		(X)
<i>AVGHL</i>	Artefaktverteilung	X	X		X			(X)
<i>AVGUL</i>	Artefaktverteilung	X	X		X			(X)

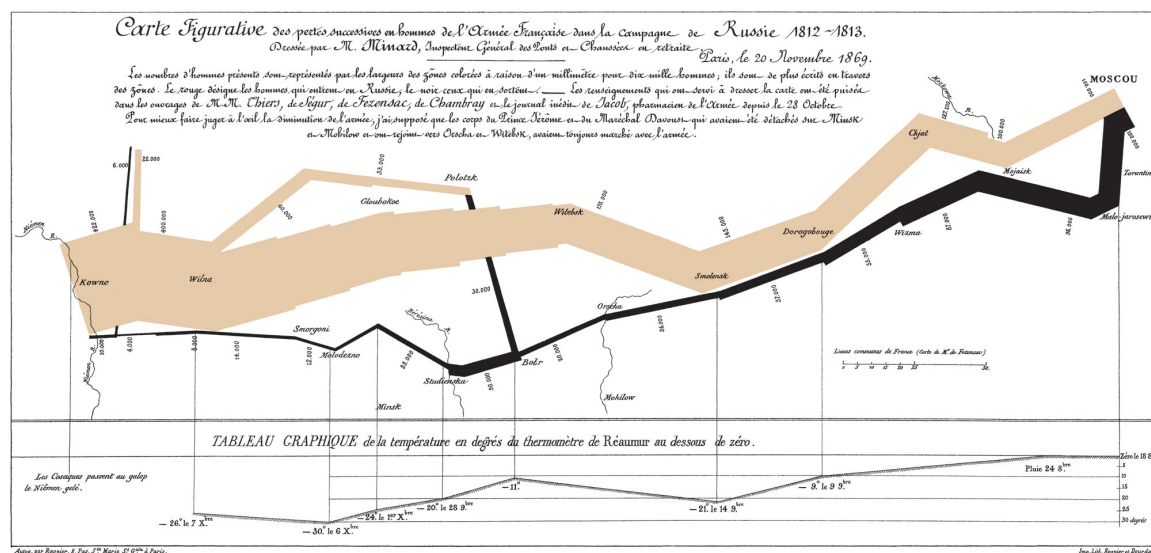
5 Visualisierung von Komplexitätsmerkmalen

Durch die Quantifizierung der Komplexitätsmerkmale einer Anwendungslandschaft ergeben sich eine Vielzahl von unterschiedlichen Kennzahlen für Anwendungen, Domänen und die Anwendungslandschaft selbst. In einer umfangreichen Anwendungslandschaft mit mehreren Domänen und einer Vielzahl von Anwendungen ist es allerdings schwierig aus der Masse an Zahlen, die Komplexitätszentren herauszufinden. Um diesen Prozess zu vereinfachen, kann eine Visualisierung der Metriken helfen.

Daher werden im folgenden Abschnitt verschiedene Visualisierungsvarianten entwickelt und diskutiert, die zum Beispiel in das SyCaTool [SYC09] integriert werden könnten.

5.1 Grundlagen

Die Visualisierung von Informationen ist im Vergleich zur Informatik schon länger im wissenschaftlichen Diskurs [Sp07]. Ein klassisches Beispiel einer frühen Visualisierung ist die Karte von Minard, einem französischen Ingenieur, der 1861 [Tu83] den Russlandfeldzug von Napoleon visualisierte (Abbildung 5.1). Durch simple Darstellungsformen verstehen die meisten Betrachter die Grafik ohne Nachfragen.



(<http://upload.wikimedia.org/wikipedia/commons/2/29/Minard.png>, 7. Dezember 2009)
 Abbildung 5.1: Minard Karte von Napoleons Russlandfeldzug

Die Linienstärke symbolisiert die Heeresstärke, die Farbe unterscheidet zwischen Vormarsch (braun) und Rückzug (schwarz), zusätzlichen sind noch markante Wegpunkte ver-

zeichnet.

Zentraler Punkt der Visualisierung von Informationen ist alleine die menschliche kognitive Aktivität beim Erwerb einer Erkenntnis aus einer Visualisierung. Ein computer-gestütztes Vorgehen ist nicht Bestandteil der Visualisierung, hat aber zum breiten Einsatz von Visualisierungen in den letzten 15 - 20 Jahren enorm beigetragen [Sp07]. In dieser Arbeit wird Visualisierung daher auch wie in folgender Definition verstanden:

Definition: Visualisierung Visualisierung beschreibt den Vorgang des Formens eines mentalen Modells oder Bildes eines Sachverhalts [Sp07].

Informationsvisualisierung darf trotz der enormen Unterstützung durch Computer allerdings nicht den Empfänger der kodierten Daten aus den Augen verlieren: den Menschen. Spence fasst drei Kernprobleme bei Visualisierungen zusammen, denen beim Erstellen von Visualisierungen Rechnung getragen werden muss:

- **Inattentional Blindness** bezeichnet das Phänomen, dass das menschliche Gehirn wesentliche Ereignisse innerhalb einer Szene, die er beobachtet oft nicht wahrnimmt. Beispielsweise nahmen viele Probanden, die auf einem Video die Anzahl der Pässe einer Basketballübung zählen sollten, eine im gleichen Video durch das Bild gehende Person nicht wahr [Sp07].
- **Change Blindness** bezeichnet die Unfähigkeit des menschlichen Gehirns, kleine und mittlere Veränderungen in Grafiken oder Bildern erst nach genauerer Betrachtung konkret benennen zu können. Abbildung 5.2 etwa zeigt zwei ähnliche Bilder mit einem Unterschied, der den meisten Betrachtern nicht sofort auffällt.
- **Cognitive Collage** betont, dass Visualisierungen von kognitiven Strukturen abhängig sind; diese kognitiven Strukturen können miteinander konkurrieren und zu falschen Aussagen führen. Beispielsweise führt die Frage "Was ist weiter westlich: Los Angeles oder Reno?" zu falschen Antworten: Reno ist weiter westlich als Los Angeles, da aber Los Angeles zu Kalifornien gehört und daher als gesamtes im Westen von Nevada liegt, lautet die Antwort fälschlicherweise oft Los Angeles [Sp07].

Weitere Kernprobleme bei der Visualisierung von Informationen sind Repräsentation und Präsentation der Daten. Die Repräsentation von Daten bezeichnet wie die Daten visuell enkodiert werden. Wichtige Rollen spielen dabei der Datentyp, die Datenkomplexität und die menschliche Wahrnehmung und Kognition. Spence liefert in [Sp07] eine breite Übersicht über verschiedene Möglichkeiten Daten zu repräsentieren.

Die Präsentation der Daten hingegen bezeichnet die Auseinandersetzung mit den Grenzen des Dargestellten wie z.B. der Darstellung auf begrenzt großem Raum oder innerhalb begrenzter Zeit [Sp07]. Dynamische Verläufe können nicht immer in einer Darstellung repräsentiert werden, sondern müssen über einen gewissen zeitlichen Raum dargestellt werden. Auf der anderen Seite besitzen mobile Geräte starke Einschränkungen in Bezug auf die Größe des Anzeigebereichs. Diese Einschränkungen müssen bei Visualisierungen betrachtet werden und ggf. durch Interaktivität aufgehoben werden.

Interaktivität in Visualisierungen ist eine neue Möglichkeit, die sich durch Computerunterstützung ergeben hat. Sie ermöglicht z.B. bei hohen Datendichten eine Entzerrung der



(Daniel Simons, <http://viscog.beckman.illinois.edu/flashmovie/1.php>, 7. Januar 2010)

Abbildung 5.2: Zwei unterschiedliche Bilder zur Verdeutlichung der Change Blindness

Dichte durch zoombare Benutzerschnittstellen [Tu83], [BS03]. Dadurch können vor allem hierarchische Daten besser für den Betrachter verständlich gemacht werden.

5.2 Visualisierungsansätze für Anwendungslandschaften

Die Visualisierung von Anwendungslandschaften ist bereits mehrfach diskutiert worden und ist im praktischen Einsatz [Bu09a]. Neben der Verwendung von UML können auch Softwarekarten, die zwar weniger detailliert sind, aber von höheren Managementebenen und nicht-technischen Stakeholdern verstanden werden, eingesetzt werden.

Unter anderen werden in [MW04a], [MW04b] und [Wi07] Grundlagen für die Modellierung und automatische Generierung von Softwarekarten auf Basis eines Informationsmodells gelegt. Wittenburg schlägt verschiedene Schichten einer Softwarekarte in Analogie zu thematischen Karten der Kartographie vor. Dabei werden wie in Abbildung 5.3 verschiedene Schichten über einen topographischen Grund angeordnet, die dann je nach Interessen des Stakeholders ein- bzw. ausgeblendet werden können [Wi07]. Als topographischer Kartengrund sind meist Gliederungen in Domänen gegeben, z.B. die Gliederung in Geschäftsprozesse oder Organisationseinheiten. Andere Domänengliederungen sind möglich. Aufsetzend werden dann in einer Schicht die Anwendungen aufgetragen, in einer weiteren Schicht die Abhängigkeiten zwischen den Anwendungen. Weiter schlägt Wittenburg bereits eine zusätzliche Schicht für die Darstellung von Metriken vor.

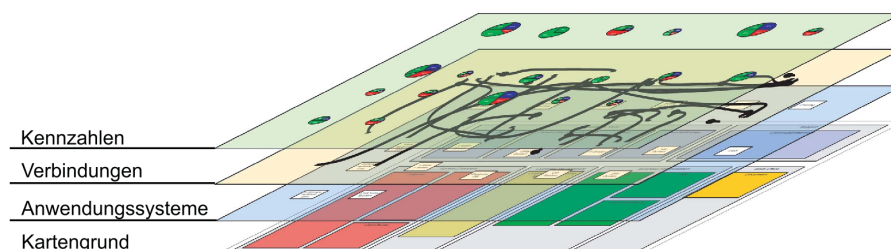


Abbildung 5.3: Schichten einer Softwarekarte

([Wi07])

Im Gegensatz dazu definiert Lankes in [La08] für seine Metriken eigene Viewpoints im Sinne des EAM Pattern Catalogs [Bu08]. Diese Viewpoints enthalten dabei alle zur Befriedigung eines bestimmten Concerns benötigten grafischen Gestaltungsmittel in einer einzigen Schicht. Ein Einblenden von zusätzlichen Informationen ist nicht möglich. Zusätzlich enthält ein Viewpoint eine detaillierte Definition der Gestaltungselemente und deren Gestaltungsregeln.

5.3 Visualisierung der Komplexitätsmerkmale

5.3.1 Metrikschicht für Softwarekarten

Softwarekarten können aus mehreren Schichten bestehen. In Anlehnung an [Wi07] wird daher nun ein Ansatz zur geeigneten Visualisierung von Metriken als Schicht einer Softwarekarte, die bei Bedarf ein- und ausgeblendet werden kann, entwickelt.

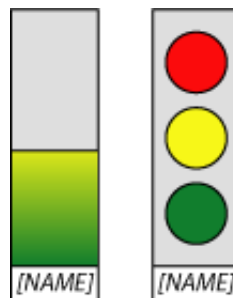
Die Metrikschicht ist abhängig vom topographischen Kartengrund, also abhängig von der Verortung der Gestaltungsmittelinstanzen. Die Gestaltungsmittelinstanzen können Anwendungen, Domänen (z.B. Organisationseinheiten oder Prozesse) oder Abhängigkeiten sein. Im speziellen Fall der in dieser Arbeit vorgestellten Metriken ist jedoch nur das Gestaltungsmittel der Domänen von Interesse. Die ausgewerteten Metriken werden an die jeweiligen Domänen angehängt bzw. annotiert, so dass der Betrachter sofort den Zusammenhang zwischen Domäne und dazugehörigen Metrik erfassen kann.



(Eigene Darstellung)

Abbildung 5.4: Einfache Annotationsvarianten für Softwarekarten

Zur Annotierung der Domänen gibt es mehrere Möglichkeiten. Abbildung 5.4 zeigt eine textbasierte Anzeige der Metrikwerte. Während die linke Variante schlicht den Wert- und den Namen der Metrik darstellt, kann bei der rechten Variante durch Farbkodierung (*engl.*: Color Coding) anhand der Farbe der Wert abgelesen werden. Farbkodierung ist allerdings nur für Metriken möglich, die auf einer Ordinalskala basieren und einen abgeschlossenen Wertebereich haben.



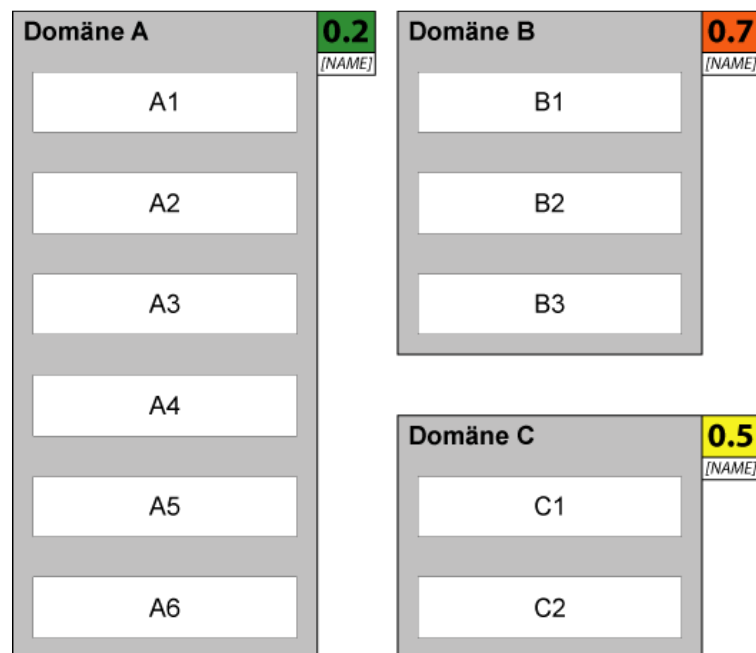
(Eigene Darstellung)

Abbildung 5.5: Grafische Annotationsvarianten für Softwarekarten

Eine weitere Möglichkeit sind graphische Indikatoren. In Abbildung 5.5 sind zwei Va-

rianten dargestellt, mit denen eine Softwarekarte annotiert werden kann. Durch die grafische Darstellung und die verwendete Farbkodierung, können schlechte Metrikergebnisse schnell in einer Softwarekarte lokalisiert werden. In der Abbildung links gibt die Länge und Farbe des Balkens den Wert der Metrik wieder; die rechte Variante vereinfacht die Darstellung auf eine Ampel, die drei Werte einnehmen kann: rot, gelb und grün. Um eine solche Einordnung der Metrik zu gewährleisten müssen allerdings die Metrikergebnisse entsprechend in drei Intervalle gesplittet werden.

Abbildung 5.6 zeigt eine mögliche Integration in eine Softwarekarte auf der Domänen und Anwendungen aufgezeigt sind.



(Eigene Darstellung)

Abbildung 5.6: Softwarekarte mit annotierten Metrikergebnissen

5.3.2 Komplexitätsmetrik Viewpoint

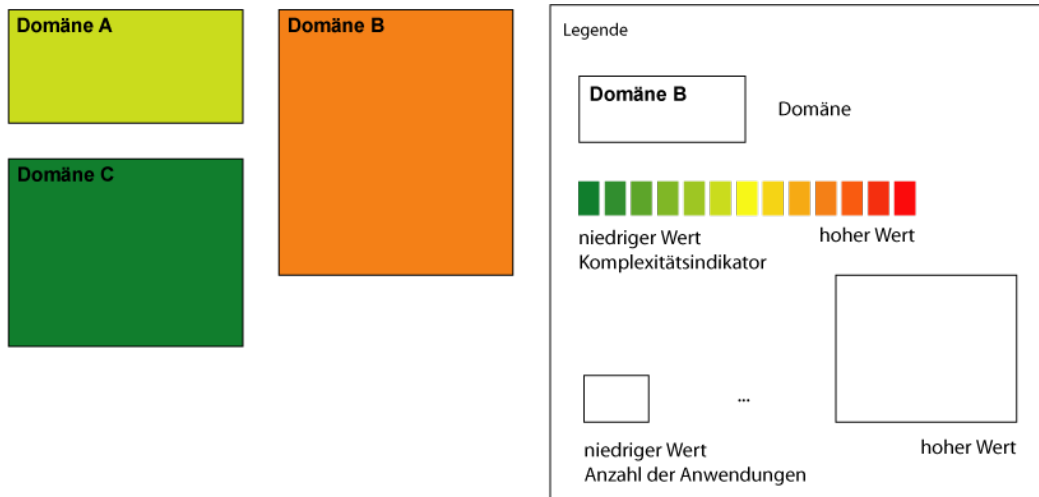
Auf Basis des Domain-Level Metrics Overview Viewpoint (V-91) von Lankes [La08] wird dieser nun angewendet für die vorgestellten Komplexitätsindikatoren.

Der Viewpoint besteht aus drei Gestaltungsmitteln: den Domänen, der Farbkodierung der Domänen und der Größe der Domänen. Die Farbkodierung bildet den Wert einer Komplexitätsmetrik ab; grün steht für einen niedrigen Komplexitätsindikator, rot für einen hohen Komplexitätsindikator. Die Größe der Domänen repräsentiert die Anzahl der Anwendungen in der Domäne.

Der Viewpoint zeigt jeweils nur den Wert der Metrik *AZ* und einem vom Unternehmensarchitekt beliebig wählbaren anderen Komplexitätsindikator dar. Denkbar wäre aber auch die Darstellung des Werts eines Indizes, der mehrere Komplexitätsmetriken vermengt.

Abbildung 5.7 zeigt eine beispielhafte Viewpoint-Instanzierung mit drei Domänen. Do-

mäne A ist kleine Domäne mit mittlerer Komplexität, Domäne C enthält mehr Anwendungen als A, besitzt aber dennoch weniger Komplexität. Domäne B ist im Beispiel die Domäne mit den meisten Anwendungen als auch mit der höchsten Komplexität.



(Eigene Darstellung in Anlehnung an [La08])

Abbildung 5.7: Komplexitätsmetrik Viewpoint

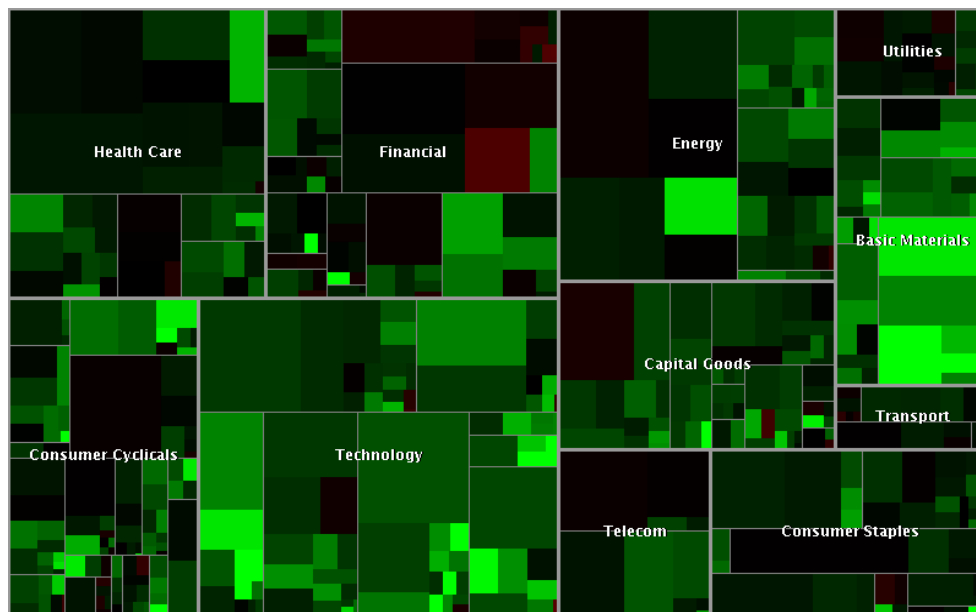
5.3.3 Darstellung als TreeMap

Für die Betrachtung der Komplexität von stark durch Domänen hierarchisch gegliederten Anwendungslandschaften oder für die Exploration von Metriken mit rekursiv-aggregierendem Charakter (z.B. durch Betrachtung der inhärenten Komplexität, vgl. Abschnitt 4.2.3) eignen sich die obigen Darstellungen nur bedingt. Daher wird im Folgenden ein Ansatz basierend auf sogenannten TreeMaps vorgestellt, die eine sehr gute Variante zur Darstellung von hierarchischen Sachverhalten sind [Sh91]. Auch Baker und Eick sehen mit TreeMaps vergleichbare Visualisierungen als geeignet für Komplexitätsmetriken an [BE99].

In Abbildung 5.8 ist beispielhaft eine TreeMap zur Analyse von Finanzmarktdaten abgebildet. Der Finanzmarkt wurde dazu hierarchisch in Branchen und dazugehörige Aktien von Unternehmen unterteilt. Die Größe der Rechtecke widerspiegelt den Marktanteil der Unternehmen, die Farbe die Performance der dazugehörigen Aktien über die letzten 52 Wochen. Durch anklicken der Rechtecke können Detailinformationen über die Aktie und das Unternehmen eingesehen werden, wie z.B. der konkrete Wert der Aktie und deren relative Entwicklung.

Für die Anwendung in Anwendungslandschaften zur Visualisierung von Interval- oder Verhältnismetriken können TreeMaps ebenfalls verwendet werden. Voraussetzung hierfür ist, dass Metriken Bottom-Up über die Domänenhierarchie berechnet werden können. Die meisten der zuvor vorgestellten Metriken unterstützen diese Berechnungsweise.

Zunächst muss aus der Anwendungslandschaft ein Baum generiert werden. Wurzel des Baumes ist die 0-Domäne, also eine Domäne von der aus alle Unterdomänen aus erreichbar sind. Die Kinder der Knoten sind dann jeweils die zugehörigen Unterdomänen. Die Knoten sind dabei zusätzlich mit dem Wert der Metrik für eine bestimmte Domäne als



(<http://www.smartmoney.com/map-of-the-market/>, 8. Dezember 2009)

Abbildung 5.8: Treemap zur Visualisierung von Finanzmarktdaten

Knotengewicht annotiert. Ein Beispielbaum ist in Abbildung 5.9 abgebildet. Mit Komma getrennt ist der jeweilige Metrikwert angegeben.

Zur Berechnung der Treemap-Visualisierung wird dieser Baum rekursiv im Sinne einer Tiefensuche durchgeschleift und der Zeichenbereich jeweils unterteilt: das Gesamtgewicht der Unterbäume einer Domäne wird beim Verteilen des benötigten Platzes berücksichtigt. Eine Domäne mit geringem Gewicht erhält weniger Zeichenplatz, als eine Domäne mit hohem Gewicht. Dadurch gewinnt der Betrachter einen schnellen Überblick über die Problemomänen, die hohe Metrikwerte aufweisen. Im Fall dieser Arbeit lassen sich so Komplexitätsschwerpunkte einfach erfassen.

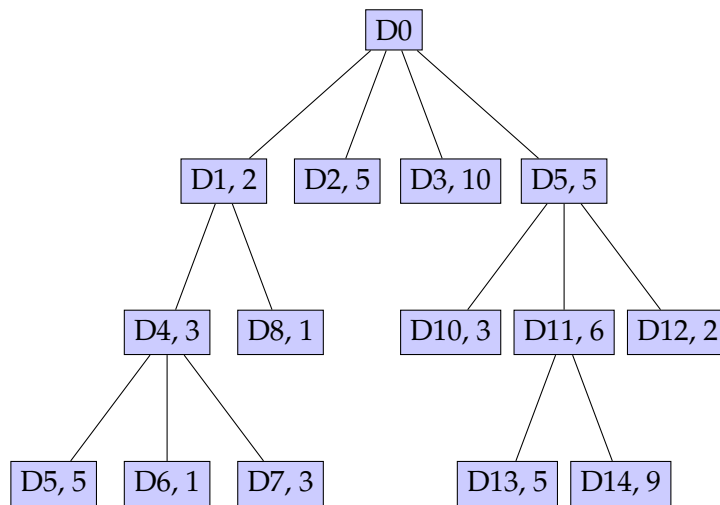
Eine beispielhafte Treemap für die in Abbildung 5.9 gezeigte Domänenhierarchie ist in Abbildung 5.10 dargestellt.

Die Wahl des genauen Algorithmus zur Darstellung der Treemap muss für den konkreten Anwendungsfall evaluiert werden. Bederson et al. kommen in [BSW03] zum Schluß, dass der Strip Treemap Algorithmus gute Ergebnisse liefert, v.a. in Bezug auf Lesbarkeit, der Auswahl der relativen Größen und der Stabilität.

5.3.4 Visualisierung von mehreren Metriken

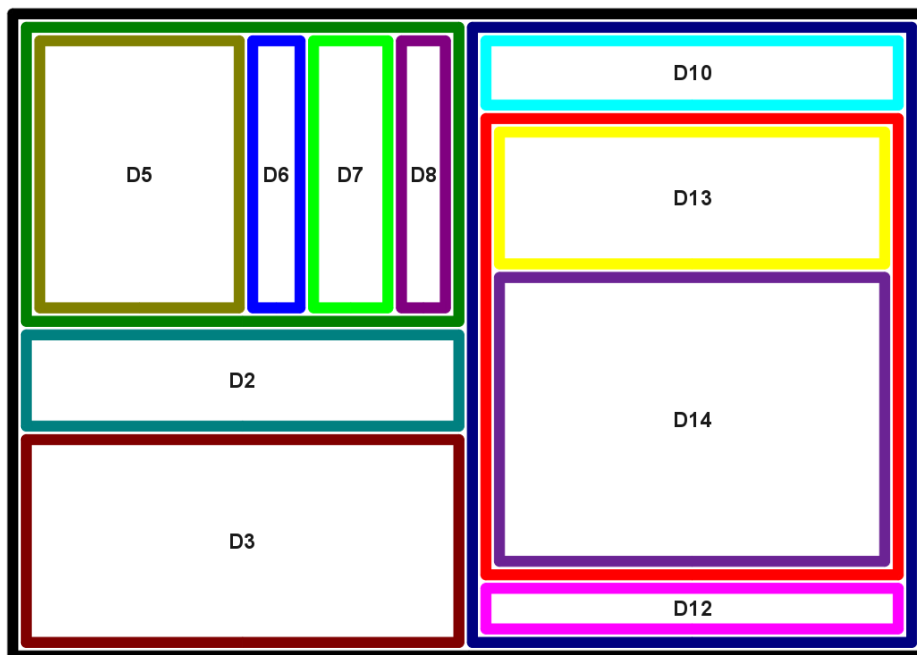
Nachdem gerade im Fall der vorgestellten Komplexitätsindikatoren mehrere Metriken gleichzeitig dargestellt werden sollen, müssen geeignete Varianten gefunden werden, Metriken parallel innerhalb einer Softwarekarte oder einer anderen Darstellungsform zu visualisieren.

Bei der Betrachtung der gesamten Anwendungslandschaft oder einer einzelnen Domäne, bieten Starplots eine geeignete Übersicht. Jede zu betrachtende Metrik ist dabei eine eigenständige Dimension, die jeweils sternförmig um einen gemeinsamen Ursprung herum angetragen werden [Sp07]. Die Metrikwerte werden dann jeweils auf den Dimensi-



(Eigene Darstellung)

Abbildung 5.9: Beispieldomänenhierarchie für TreeMap

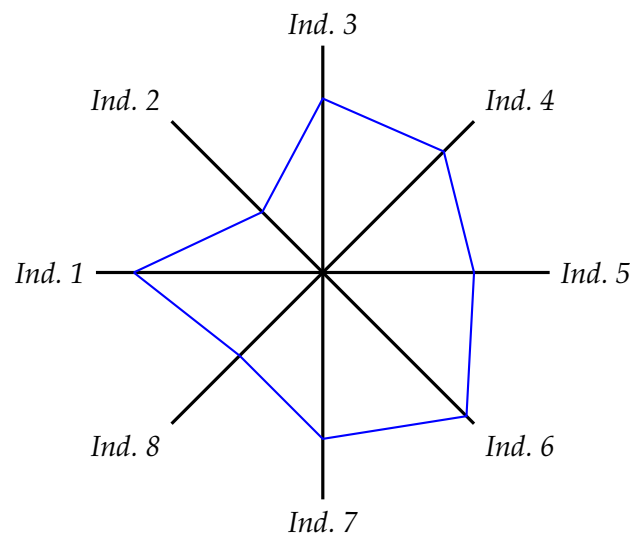


(Eigene Darstellung)

Abbildung 5.10: Beispielhafte TreeMap für die Domänenhierarchie aus Abbildung 5.9

onsachsen angezeichnet und mit den jeweils benachbarten Dimensionsachsen mit einer (farbigen) Linie verbunden. Voraussetzung ist allerdings, dass die eingezeichneten Metriken in einem normierten Wertebereich (z.B. $[0, 1]$) verankert sind. Ansonsten lassen sich die einzelnen Metrikerwerte schlecht vergleichen.

Abbildung 5.11 zeigt ein Beispiel eines Starplots mit acht verschiedenen Komplexitätsindikatoren (Ind. 1 - Ind. 8). Mit einem hohen Wert herausstechend ist der Komplexitätsindikator 6. Es sollten geeignete Maßnahmen getroffen werden, die Anwendungslandschaft so zu optimieren, dass (möglichst ohne Auswirkungen auf andere Indikatoren) dieser Indikatorwert verringert wird.



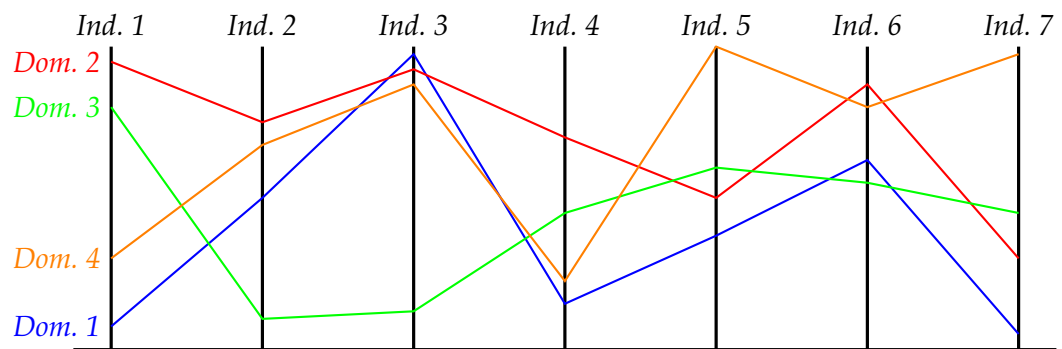
(Eigene Darstellung)

Abbildung 5.11: Starplot

Bei der Betrachtungen von mehreren Domänen gleichzeitig, eignet sich (neben einem Starplot) ein Multi-Koordinaten Plot (*engl.*: Parallel Coordinate Plot). Dabei erhalten die verschiedenen Metriken jeweils eine Koordinatenachse; die Koordinatenachsen werden parallel zueinander aufgetragen. Für jede Domäne wird dann ein Punkt auf den Koordinatenachsen aufgetragen und die zur gleichen Domänen gehörigen Punkte mit (farbigen) Linien verbunden. Dadurch können einzelne Domänen einfach miteinander verglichen werden [Sp07]. Voraussetzung hierfür ist, dass die verwendeten Metriken auf einen einheitlichen Wertebereich normiert wurden (z.B. $[0, 1]$).

Abbildung 5.12 zeigt ein Beispiel eines Multi-Koordinaten Plots, der vier Domänen über 7 verschiedene Indikatoren hinweg vergleicht.

Die Herausforderung bei Multi-Koordinaten Plots ist die Filterung der Domänen. Je nach Anzahl der Domänen können viele Linien den Überblick bzw. das Nachverfolgen einer einzelnen Linie erschweren. Es empfiehlt sich daher zusätzlich der Einsatz von interaktiven Filtern, mit Hilfe derer die Domänen gefiltert werden können, die eine hohe oder niedrige Komplexität aufweisen. Dabei werden ein oder mehrere Indikatoren so eingeschränkt, dass nur noch die Domänen angezeigt werden, die einen bestimmten Wertebereich erfüllen. Man vergleiche dazu auch die Ansätze von Inselberg [In99].



(Eigene Darstellung)

Abbildung 5.12: Parallel Coordinate Plot

Die hier vorgeschlagenen Visualisierungen sind unabhängig von den Komplexitätsmetriken aus den vorigen Sektionen. Sie können auf andere Metriken, die auf Intervall- oder Verhältnisskalen arbeiten, ebenfalls angewendet werden.

6 Diskussion und Ausblick

In dieser Arbeit wurden die Eigenschaften einer Anwendungslandschaft auf Basis einer Literaturrecherche identifiziert und erläutert, die ursächlich zu Komplexität von Anwendungslandschaften beitragen. Diese wurden in die Bereiche anwendungsbezogen, schnittstellenbezogen, abhängigkeitsbezogen und organisationsbezogen eingeordnet und gegliedert.

Diese Eigenschaften wurden dann in einem Informationsmodell zusammengefasst, das als Basis für die Quantifizierung von Komplexität dient. Dazu wurden Metriken aus der Softwaretechnik auf die Anwendung in Anwendungslandschaften transformiert und angepasst. Diese Metriken dienen als Indikator für die Komplexität einer Anwendungslandschaft.

In Kapitel 5 wurden Visualisierungsformen erörtert, die geeignet sind zur Darstellung der vorgeschlagenen Metriken. Dabei wurden sowohl bekannte Ansätze aus dem Bereich des Unternehmensarchitekturmanagements wie z.B. Softwarekarten oder Viewpoints im Sinne des EAM Pattern Catalog aufgegriffen als auch neue, innovative Darstellungen für den Einsatz in Anwendungslandschaften aufbereitet.

Mit dem Wissen über die komplexitätstreibenden Eigenschaften und deren Quantifizierbarkeit kann nun Komplexität in Anwendungslandschaften besser verstanden werden.

Ziel weiterführender Arbeiten muss nun die Evaluation der Metriken in einem produktiven Umfeld sein. Nach [Bu09a] nutzen 37% der Unternehmen Metriken im Bereich des Unternehmensarchitekturmanagements, 52% könnten Metriken einsetzen. Eine gute Basis zur Evaluation ist daher gegeben. Fraglich ist allerdings ob die Unternehmen den teilweise speziellen Informationsbedarf der vorgestellten Metriken liefern können. Anhand der Evaluation könnten die Metriken dann verfeinert werden oder auch die Frage beantwortet werden, welche Metriken sinnvoll zu einem Metriksystem zusammengefasst werden können. Denn nur eine geeignete Auswahl an validen Metriken ist für Unternehmen sinnvoll [DH08].

Nachdem Komplexität nun besser verstanden werden kann ist es von hohem Interesse Methoden zu entwickeln, die zu einer nachweisbaren Reduktion der Komplexität in einer Anwendungslandschaft führen und damit die Qualität einer Anwendungslandschaft verbessern können. Anhand der vorgestellten Metriken können diese Methoden dann evaluiert werden. Aber auch die Einbettung der Metriken in eine existierende Methode zur nachhaltigen Entwicklung von Unternehmensarchitekturen und Anwendungslandschaften würde eine sinnvolle Erweiterung dar (z.B. in [AD05], [MWF08]).

Ein weiteres Forschungsthema ergibt sich im Bereich der erarbeiteten Visualisierungen: eine Evaluierung derselbigen bietet sich an, um zu überprüfen, ob diese geeignet sind weniger zahlenfixierten Stakeholdern eine gute Darstellung der Komplexität einer Anwendungslandschaft zu bieten. Damit einhergehend kann auch ermittelt werden, inwiefern verschiedene Darstellungsformen von Anwendungslandschaften bereits Auswirkungen auf die subjektive Komplexität haben (vgl. auch [GSK05]).

Auch die Einordnung des Komplexitätsmanagement von Anwendungslandschaften zu einem bestimmten Zeitpunkt in den Planungsprozess des Unternehmensarchitekturmanagement ist ein offener Punkt. Eine Einbettung in verschiedene Ansätze wie z.B. dem von Aier et al. [Ai09] ist für die Zukunft denkbar.

Anhang

A Glossar

Anwendung Eine Anwendung ist ein in Software implementiertes System, das mindestens einen Geschäftsprozess unterstützt.

Anwendungslandschaft Die Anwendungslandschaft ist die Gesamtheit der betrieblichen Anwendungen inklusive der Kommunikationsbeziehungen zwischen den Anwendungen in einem Unternehmen. [Wi07]

Architektur Der fundamentale Aufbau eines Systems, verkörpert in seinen Komponenten, ihren Beziehungen zueinander und den Prinzipien, die sein Design und seine Entwicklung bestimmen. [Ke06]

Chunking bezeichnet das Aggregieren und Gruppieren von kleineren Informationseinheiten zu größeren Informationseinheiten im Gehirn, welche später als Ganzes wieder abgerufen werden können. [Li08]

Domäne Eine Domäne untergliedert eine Anwendungslandschaft in verschiedene, hierarchische Segmente. Die Gliederung ist dabei nicht vorgegeben: Domänen können z.B. nach Organisationseinheiten oder Prozessen gebildet werden. Die Unterteilung nach Domänen kann aber auch anderen Unterteilungen fachlicher, technischer oder organisatorischer Natur entspringen. Alle Domänen einer Anwendungslandschaft sind in einem Baum organisiert; d.h. es gibt eine Wurzel, von der aus man alle Unterdomänen erreichen kann.

Informationsmodell Ein Informationsmodell (*engl.*: Information Model) ist ein Modell, welches die Semantik von Informationsobjekten, die möglichen Beziehungen zwischen den Informationsobjekten und die Attribute von Informationsobjekten definiert. Ein Informationsobjekt ist ein Abbild eines existierenden Objekts in einem betrieblichen Objektsystem. [Wi07]

Informationssystem Bei Informationssystemen (IS) handelt es sich um soziotechnische ("Mensch-Maschine-") Systeme, die menschliche und maschinelle Komponenten (Teilsysteme) umfassen und zum Ziel der optimalen Bereitstellung von Information und Kommunikation nach wirtschaftlichen Kriterien eingesetzt werden. [Kr09]

IT-Infrastruktur physikalische oder virtuelle Infrastruktur, auf der die Anwendungssysteme der Anwendungslandschaft deployed werden können.

IT-Unternehmensarchitektur IT-Unternehmensarchitektur ist derjenige Teil der Unternehmensarchitektur, den die IT-Funktion in einem Unternehmen ausmachen darf, ohne wegen Kompetenzüberschreitung von anderen Unternehmenseinheiten außerhalb der IT erfolgreich politisch attackiert zu werden. Im besten Fall sind IT-Unternehmensarchitektur und Unternehmensarchitektur identisch und einheitlich organisiert. [Ke06]

- Kohäsion** Kohäsion ist der Grad des funktionellen Zusammenhangs innerhalb eines Moduls; hohe Kohäsion liegt z.B. dann vor, wenn funktionell wie logisch zusammenhängende Funktionalitäten innerhalb eines Moduls gebündelt werden und nicht über das gesamte Softwaresystem verteilt werden. [Zu94]
- Kopplung** Kopplung stellt die Abhängigkeiten zwischen den Modulen in den Vordergrund. Die Kopplung zweier Module kann als Grad der Verbindungen bzw. Abhängigkeit voneinander gesehen werden. [Zu94]
- Metrik** (auch: Kennzahl) Metriken erfassen Sachverhalte quantitativ und in konzentrierter Form. Merkmale einer Metrik sind Informationscharakter, Quantifizierbarkeit und Informationsform. Der Informationscharakter zeigt, dass eine Beurteilung von Sachverhalten und Zusammenhängen möglich ist. Quantifizierbarkeit bedeutet, dass Sachverhalte auf metrischen Skalen gemessen werden können und dadurch "genaue" Aussagen möglich sind. Die Informationsform führt dazu, dass komplexe Sachverhalte komprimiert und auf einfache Art dargestellt werden. [Ku07]
- Metrik, absolute** Absolute Metriken oder Grundzahlen sind Metriken, die unabhängig und ohne Vermengung mit anderen Metriken sind und geben einen Sachverhalt direkt wieder. [Pr08]
- Metrik, relative** (auch: Verhältniszahlen) Relative Metriken sind Metriken, die unterschiedliche Sachverhalte miteinander in Bezug setzen. Verhältniszahlen verwenden dazu absolute Metriken. [Pr08]
- Redundanz** Redundanz bezeichnet allgemein einen Überfluss bzw. das Vorhandensein von weglassbaren Elementen. [DUD05]
- Schnittstelle** (auch: Interface [Bu08], IS-Service [VST05]) Eine Schnittstelle ist eine technische Implementierung eines (ggf. nicht aktiv genutzten) Kommunikationskanals, der von einer Anwendung exportiert wird. Andere Anwendungen können diesen Kommunikationskanal verwenden (importieren).
- Softwarekarte** Eine graphische Repräsentation der Anwendungslandschaft oder Ausschnitte selbiger. Eine Softwarekarte setzt sich zusammen aus einer oder mehrerer Schichten, die verschiedene Aspekte visualisieren. [MW04a]
- Unternehmensarchitektur** (auch: : Enterprise Architecture) Die Unternehmensarchitektur ist die kohärente und ganzheitliche Architektur eines Unternehmens, die nicht nur die Informationstechnologie sondern ebenso betriebswirtschaftliche Elemente umfasst. Dabei umfasst die Architektur nicht nur die einzelnen Elemente des Unternehmens selbst, wie beispielsweise die Organisationsstruktur, die Geschäftsprozesse, die Anwendungen und die Infrastrukturelemente, sondern auch ihre Verbindungen und Querschnittselemente, wie Strategien und Ziele, Anforderungen und Projekte, Richtlinien und Muster sowie Kennzahlen und Metriken. [Wi07]
- Visualisierung** beschreibt den Vorgang des Formens eines mentalen Modells oder Bildes eines Sachverhalts. [Sp07]

Literaturverzeichnis

- [AD05] Stephan Aier und Turgut Dogan. Indikatoren zur Bewertung der Nachhaltigkeit von Unternehmensarchitekturen. In O. Ferstl, E. Sinz, S. Eckert und T. Isselhorst, Hrsg., *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*, Seiten 607–626. Physica, Heidelberg, 2005.
- [Ai08] Stephan Aier, Stephan Kurpjuweit, Christian Riege und Jan Saat. Stakeholderorientierte Dokumentation und Analyse der Unternehmensarchitektur. In Heinz-Gerd Hegering, Axel Lehmann, Hans J. Ohlbach und Christian Scheideler, Hrsg., *INFORMATIK 2008 - Beherrschbare Systeme - dank Informatik, Band 2*, Jgg. 133 of *LNI*, Seiten 559–565. GI, September 2008.
- [Ai09] Stephan Aier, Bettina Gleichauf, Jan Saat und Robert Winter. Complexity Levels of Representing Dynamics in EA Planning. In Antonia Albani, Joseph Barijs und Jan L. G. Dietz, Hrsg., *CIAO!/EOMAS*, Seiten 55–69, 2009.
- [ALL09] Michael Amberg, Michael Lang und Marc Laszlo. Die geschäftsfokussierte Informationstechnologie - Business-IT-Alignment als zentrales Steuerungsinstrument zur strategischen Ausrichtung der IT. Bericht, Detecon International GmbH, August 2009.
- [APR09] Apache APR Version Numbering, <http://apr.apache.org/versioning.html>, November 2009.
- [AW09] Stephan Aier und Robert Winter. Virtuelle Entkopplung von fachlichen und IT-Strukturen für das IT/Business Alignment – Grundlagen, Architekturgestaltung und Umsetzung am Beispiel der Domänenbildung. *Wirtschaftsinformatik*, 51(2):175–191, April 2009.
- [Ba96] David Baccarini. The concept of project complexity - a review. *International Journal of Project Management*, 14(4):201–204, August 1996.
- [Ba02] A. Backlund. The concept of complexity in organisations and information systems. *International Journal of Systems and Cybernetics*, 31(1):30–43, 2002.
- [Ba09] Alan Baker. Stanford Encyclopedia of Philosophy - Simplicity, <http://plato.stanford.edu/entries/simplicity/>, November 2009.
- [BE99] Marla J. Baker und Stephen G. Eick. Space-Filling Software Visualization. In Stuart K. Card, Jock D. Mackinlay und Ben Shneiderman, Hrsg., *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, 1. Auflage, February 1999.

- [Bi99] Jesús Bisbal, Deirdre Lawless, Bing Wu und Jane Grimson. Legacy Information Systems: Issues and Directions. *IEEE Software*, 16:103–111, 1999.
- [BN05] Dirk Beyer und Andreas Noack. Clustering Software Artifacts Based on Frequent Common Changes. In *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, Seiten 259–268, Washington, DC, USA, 2005. IEEE Computer Society.
- [Br09] Matthias Brückmann, Klaus-Manfred Schöne, Stefan Junginger und Diana Boudinova. Evaluating Enterprise Architecture Management Initiatives - How to Measure and Control the Degree of Standardization of an IT Landscape. 2009.
- [BS03] Benjamin B. Bederson und Ben Shneiderman. *The Craft of Information Visualization: Readings and Reflections*. Morgan Kaufmann, 1. Auflage, April 2003.
- [BSW03] Benjamin B. Bederson, Ben Shneiderman und Martin Wattenberg. Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. In *The Craft of Information Visualization: Readings and Reflections*, Kapitel 6, Seiten 257–278. Morgan Kaufmann, 1. Auflage, April 2003.
- [Bu08] Sabine Buckl, Alexander Ernst, Josef Lankes und Florian Matthes. Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Bericht, Chair for Informatics 19, Technische Universität München, February 2008.
- [Bu09a] S. Buckl, A. M. Ernst, J. Lankes, F. Matthes und C. M. Schweda. State of the Art in Enterprise Architecture Management. Bericht, Technische Universität München, Fakultät für Informatik, I19, March 2009.
- [Bu09b] Betsy Burton. Thirteen Worst Enterprise Architecture Practices. Report Nr. G00164424. Bericht, January 2009.
- [Co71] Stephen A. Cook. The complexity of theorem-proving procedures. *Proceedings Third Annual ACM Symposium on Theory of Computing*, Seiten 151–158, 1971.
- [De82] Tom DeMarco. *Controlling Software Projects: Management, Measurement and Estimation*. Yourdon, New York, NY, 1982.
- [De06] Gernot Dern. *Management von IT-Architekturen: Leitlinien für die Ausrichtung, Planung und Gestaltung von Informationssystemen*. Vieweg+Teubner, 2. Auflage, November 2006.
- [DH08] Bryce Dunn und Linh C. Ho. Selecting business-relevant metrics. In Annelies van der Veen, Hrsg., *IT Service Management Global Best Practices*, Kapitel 9.4. 2008.
- [Di04] Henner Diederichs. *Komplexitätsreduktion in der Softwareentwicklung - Ein systemtheoretischer Ansatz*. Books on Demand GmbH, 1. Auflage, 2004.
- [DUD94] *Duden, Das Große Fremdwörterbuch*. Dudenverlag, Mannheim, 1994.

-
- [DUD01] *Duden, Herkunftswörterbuch, Etymologie der deutschen Sprache*. Dudenverlag, Mannheim, 3. Auflage, 2001.
- [DUD05] *Duden - Das Fremdwörterbuch*, Jgg. 5. Dudenverlag, Mannheim, 8. Auflage, 2005.
- [DUD07] *Duden - Deutsches Universalwörterbuch*. Dudenverlag, Mannheim, 6. Auflage, 2007.
- [En08] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann und Jan-Peter Richter. *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten*. Dpunkt, 1. Auflage, 2008.
- [En09] Gregor Engels, Michael Goedicke, Ursula Goltz, Andreas Rausch und Ralf Reussner. Design for Future – Legacy-Probleme von morgen vermeidbar? *Informatik-Spektrum*, 32(5):393–397, October 2009.
- [Er06] Alexander M. Ernst, Josef Lankes, Christian M. Schweda und André Wittenburg. Using Model Transformation for Generating Visualizations from Repository Contents - An Application to Software Cartography. Bericht, TU München, Fakultät für Informatik, I19, November 2006.
- [Fe94] Norman Fenton. Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20(3):199–206, March 1994.
- [GNU09] Change Logs - GNU Coding Standards, http://www.gnu.org/prep/standards/html_node/Change-Logs.html, 2009.
- [GSK05] R. Groot, M. Smits und H. Kuipers. A Method to Redesign the IS Portfolios in Large Organisations. In *System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on*, Seiten 223–223a, 2005.
- [Ha77] Maurice H. Halstead. *Elements of Software Science*. Elsevier, New York, Amsterdam, 1977.
- [Ha07] Claus Hagen. Integrationsarchitektur der Credit Suisse. In Stephan Aier und Marten Schönherr, Hrsg., *Enterprise Application Integration - Flexibilisierung komplexer Unternehmensarchitekturen*, Seiten 63–83. GITO, April 2007.
- [Ha08] Holden Haertel. *Implizite Informationen: Sprachliche Ökonomie und interpretative Komplexität bei Verben*. studiae grammaticae. Akademie-Verlag, 2008.
- [Hi06] Matthias Hirzel. Herausforderungen des Projektportfolio-Managements. In Matthias Hirzel, Frank Kühn und Peter Wollmann, Hrsg., *Projektportfolio-Management - Strategisches und operatives Multi-Projektmanagement in der Praxis*, Seiten 11–20. Gabler Verlag, 1. Auflage, April 2006.
- [HJ06] Bernhard Humm und Oliver Juwig. Eine Normalform für Services. In Bettina Biel, Matthias Book und Volker Gruhn, Hrsg., *Software Engineering 2006. GI Edition Lecture Notes in Informatics (LNI)*, Seiten 99–110. Gesellschaft für Informatik, March 2006.

- [Hu09] Andy Hunt. *Pragmatisches Denken und Lernen. Refactor your Wetware!* Hanser Fachbuch, April 2009.
- [IJ06] Maria-Eugenia Iacob und Henk Jonkers. Quantitative Analysis of Enterprise Architectures. In *Interoperability of Enterprise Software and Applications*, Seiten 239–252. 2006.
- [In99] Alfred Inselberg. Multidimensional Detective. In Stuart K. Card, Jock D. Mackinlay und Ben Shneiderman, Hrsg., *Readings in Information Visualization - Using Vision to Think*, Seiten 107–114. Morgan Kaufmann, 1. Auflage, February 1999.
- [Jo91] Capers Jones. *Applied Software Measurement - Assuring Productivity and Quality*. McGraw-Hill, 1991.
- [Ka72] Richard M. Karp. Reducibility among combinatorial problems. In J. W. Thatcher und R. E. Miller, Hrsg., *Complexity of Computer Computations*. Plenum Press, 1972.
- [Ka03] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Professional, 2. Auflage, 2003.
- [Ke06] Wolfgang Keller. *IT-Unternehmensarchitektur. Von der Geschäftsstrategie zur optimalen IT-Unterstützung*. Dpunkt Verlag, October 2006.
- [Kr09] Helmut Krcmar. *Informationsmanagement*. Springer, 5. Auflage, November 2009.
- [Ku07] Martin Kuetz. *Kennzahlen in der IT: Werkzeuge für Controlling und Management*. Dpunkt Verlag, 2. Auflage, 2007.
- [La06] Wolfgang Lassmann. *Wirtschaftsinformatik: Nachschlagwerk für Studium und Praxis*. Springer, 2006.
- [La08] Josef Lankes. *Metrics for Application Landscapes*. Dissertation, Technische Universität München, 2008.
- [Li08] Carola Lilienthal. *Komplexität von Softwarearchitekturen, Stile und Strategien*. Dissertation, Universität Hamburg, July 2008.
- [Lo08] Jörg Lohmann, Tilo Böhm, Stefanie Leimeister, Klaus König und Helmut Krcmar. Ziele und Nutzenbeitrag eines unternehmensweiten Architekturmanagements: Ergebnisse einer empirischen Studie. In Heinz-Gerd Hegering, Axel Lehmann, Hans J. Ohlbach und Christian Scheideler, Hrsg., *INFORMATIK 2008 - Beherrschbare Systeme - dank Informatik, Band 2*, Seiten 552–558. GI, September 2008.
- [LS07] Josef Lankes und Christian M. Schweda. *Constructing Application Landscape Metrics: Why & How*. Bericht, Technische Universität München, Fakultät für Informatik, I19, 2007.

-
- [MB07] T. Miklitz und P. Buxmann. IT standardization and integration in mergers and acquisitions: a decision model for the selection of application systems. In Hubert Österle, Joachim Schelp und Robert Winter, Hrsg., *Proceedings of the 15th European Conference on Information Systems (ECIS 2007)*, Seiten 1041–1051, 2007.
- [Mc76] T. J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.
- [MEH07] Mark W. Maier, David Emery und Rich Hilliard. Systems and software engineering - Recommended practice for architectural description of software-intensive systems. Bericht, IEEE Standards Association, 2007.
- [Mo74] E. Morin. Complexity. *International Science Journal*, 26:555–582, 1974.
- [Mo05] E. Morin. *Restricted Complexity, General Complexity*. June 2005.
- [MV97] Li Ming und Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, New York, 1997.
- [MW04a] Florian Matthes und André Wittenburg. Softwarekarten zur Visualisierung von Anwendungslandschaften und ihren Aspekten - Eine Bestandsaufnahme. Bericht, TU München, Fakultät für Informatik, I19, March 2004.
- [MW04b] Florian Matthes und André Wittenburg. Softwarekartographie: Visualisierung von Anwendungslandschaften und ihrer Schnittstellen. In Peter Dadam und Manfred Reichert, Hrsg., *Informatik 2004 - Informatik verbindet, Band 2*, Seiten 71–75. GI, September 2004.
- [MWF08] Stephan Murer, Carl Worms und Frank J. Furrer. Managed Evolution. *Informatik-Spektrum*, Volume 31(6):537–547, December 2008.
- [Ni05] Klaus D. Niemann. *Von der Unternehmensarchitektur zur IT-Governance. Bausteine für ein wirksames IT-Management*. Vieweg, May 2005.
- [OT93] Linda M. Ott und Jeffrey J. Thuss. Slice Based Metrics for Estimating Cohesion. In *Proceedings of the IEEE-CS International Metrics Symposium*, Seiten 71–81, 1993.
- [Pr08] Peter R. Preißler. *Betriebswirtschaftliche Kennzahlen - Formeln, Aussagekraft, Sollwerte, Ermittlungsintervalle*. Oldenbourg, München, 2008.
- [RGA07] Gerold Riempp und Stephan Gieffers-Ankel. Application portfolio management: a decision-oriented view of enterprise architecture. *Information Systems and E-Business Management*, 5(4):359–378, September 2007.
- [Ro96] Jose L. Roca. An entropy-based method for computing software structural complexity. *Microelectronics and Reliability*, 36(5):609–620, May 1996.
- [Sc01] Günter Schuh. *Produktkomplexität managen - Strategien, Methoden, Tools*. Hanser, 2001.

- [Sc08] Christian Schmidt. *Management komplexer IT-Architekturen - Eine empirische Analyse am Beispiel der internationalen Finanzindustrie*. Dissertation, Technischen Universität Darmstadt, March 2008.
- [Sh91] Ben Shneiderman. Tree visualization with Tree-maps: A 2-d space-filling approach. Bericht, University of Maryland, Department of Computer Science, Human-Computer Interaction Laboratory, June 1991.
- [SK09] Eva P. Sekatzek und Helmut Krcmar. Messung der Standardnähe von betrieblicher Standardsoftware. *Wirtschaftsinformatik*, 51(3):273–283, June 2009.
- [Sp07] Robert Spence. *Information Visualization: Design for Interaction*. Prentice Hall, 2. Auflage, January 2007.
- [St73] Herbert Stachowiak. *Allgemeine Modelltheorie*. Springer-Verlag, Wien, 1973.
- [SYC09] SyCaTool - Software and System Cartography Tool, <http://www.matthes.in.tum.de/wikis/sebis/sycatool>, November 2009.
- [Ta72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.
- [Ta06] Tuukka Taipale. Comparison of Routing Software in Linux. Diplomarbeit, Helsinki University of Technology, September 2006.
- [Tu83] Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, April 1983.
- [UML09] Unified Modeling Language (UML), <http://www.uml.org>, October 2009.
- [VST05] André Vasconcelos, Pedro Sousa und José Tribolet. Information System Architecture Evaluation: From Software to Enterprise Level Approaches. In *12th European Conference On Information Technology Evaluation (ECITE 2005)*, September 2005.
- [WHH79] Martin R. Woodward, Michael A. Hennell und David Hedley. A Measure of Control Flow Complexity in Program Text. *IEEE Transactions on Software Engineering*, SE-5, January 1979.
- [Wi03] Terry Williams. *Management von komplexen Projekten: Projektrisiken durch quantitative Modellierungstechniken steuern*. Wiley-VCH, 1. Auflage, 2003.
- [Wi07] André Wittenburg. *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften*. Dissertation, TU München, 2007.
- [Zu94] Horst Zuse. Complexity Metrics/Analysis. In John J. Marciniak, Hrsg., *Encyclopedia of Software Engineering*, Jgg. 1, Seiten 131–165. John Wiley & Sons, New York, February 1994.