Architekturen betrieblicher Anwendungssysteme

Vom Geschäftsprozess zur Softwarearchitektur

Lehrstuhl für Wirtschaftsinformatik
Prozesse und Systeme
Universität Potsdam

Chair of Business Informatics
Processes and Systems
University of Potsdam

Univ.-Prof. Dr.–Ing. habil. Norbert Gronau
Lehrstuhlinhaber | Chairholder

Mail	August-Bebel-Str. 89 | 14482 Potsdam | Germany
Visitors	Digitalvilla am Hedy-Lamarr-Platz, 14482 Potsdam
Tel	+49 331 977 3322
E-Mail	ngronau@lswi.de
Web	lswi.de

Lernfragen

- Was ist eine Softwarearchitektur und welche Bestandteile hat diese?
- Welche Ziele werden mit dem Software Architekturmanagement verfolgt?
- Was ist Metamodellierung und in welchem Zusammenhang steht diese mit der MDA?
- Was ist MDA und welche Ziele werden damit verfolgt?
- Wie können Softwarearchitekturen bewertet werden?
- Was ist ein Softwaremuster?
- Welche Ziele werden mit dem Softwaremuster verfolgt?
- Wie werden Softwaremuster dokumentiert?
- Warum sind Schichtenarchitekturen so erfolgreich?
- Welche Bestandteile hat eine Service-orientierte Architektur?

Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- Bewertungskriterien

Einführung in Architekturmuster

- Schichtenarchitekturen
- Integrationsarchitekturen
- Weitere Architekturmuster

Definitionen von Software-Architekturen

Shaw und Garlan 1996

- SA enthält **die Beschreibung von Elementen** aus denen das System gebaut wird. Sie definiert die **Interaktion zwischen den Elementen**, beschreibt, wie die Elemente eingesetzt und zusammengesetzt werden können (**Pattern**) und gibt Bedingungen für die Anwendung der Elemente an.

Hruschka und Starke 2006

- Die Architektur eines Softwaresystems definiert dessen **Komponenten und deren Zusammenwirken** über Schnittstellen. Sie beschreibt die Struktur von Komponenten. Architektur betrachtet sowohl **statische als auch dynamische Aspekte** und zeigt damit sowohl den Bauplan als auch den Ablaufplan für Software auf.

Hasselbring 2006

- Eine Softwarearchitektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren **Beziehungen zueinander und zur Umgebung** sowie den Prinzipien, die den **Entwurf** und die **Evolution** der Systeme bestimmen.

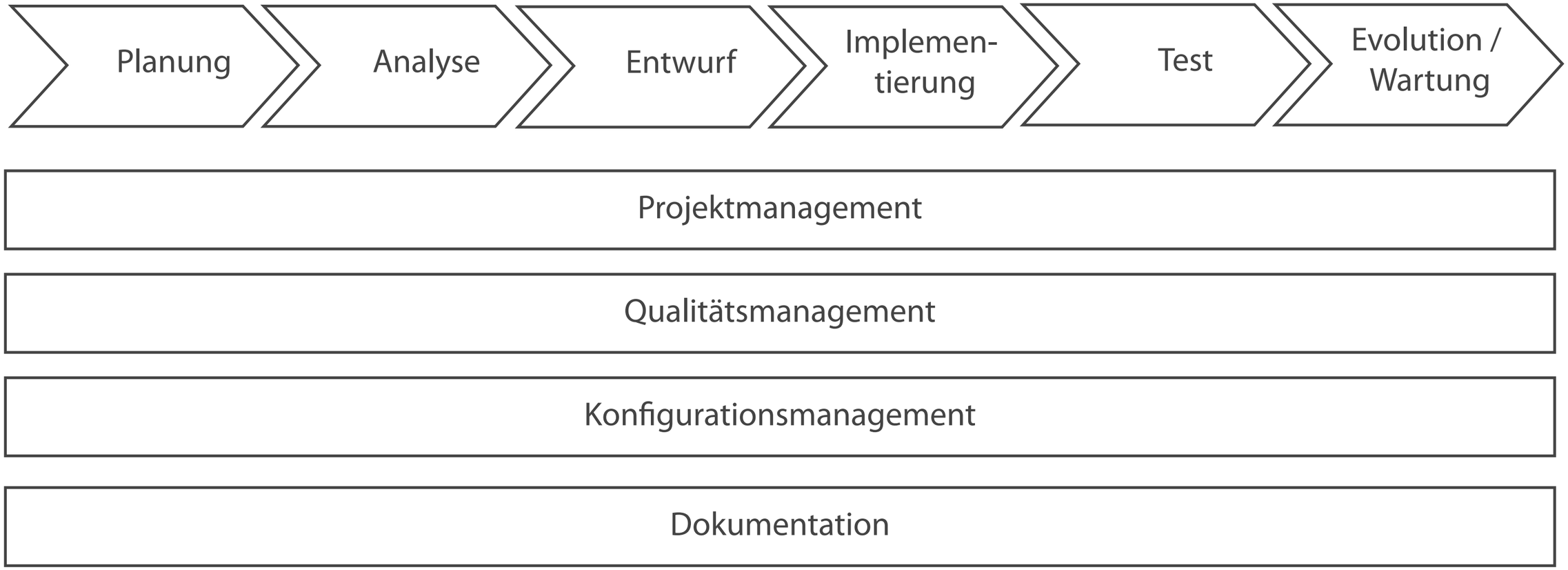
Zusammenfassung

- Darstellung von Komponenten
- Beziehung zwischen den Komponenten
- Beziehung zur Umgebung
- Struktur eines Anwendungssystems
- Programmierung im Großen
- Verdeutlicht Zusammenhänge und Strukturen

Quelle: Hasselbring 2006; Hruschka und Starke 2006, Shaw und Garlan 1996

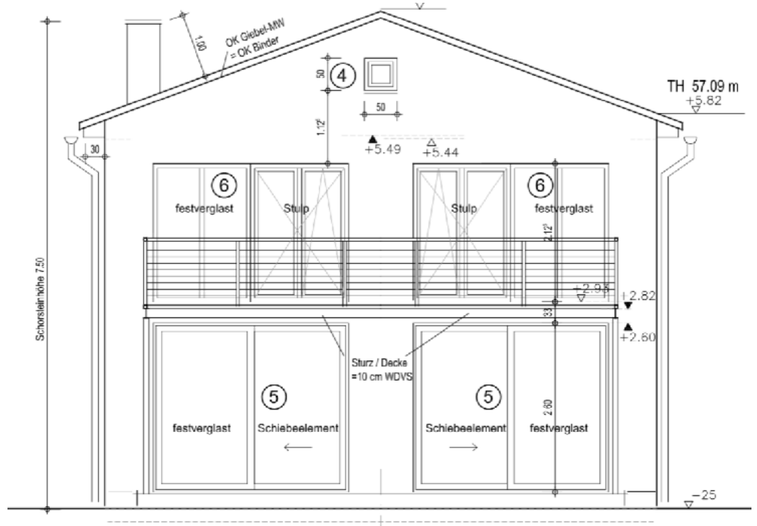
Einordnung der Software-Architektur in das Software Engineering

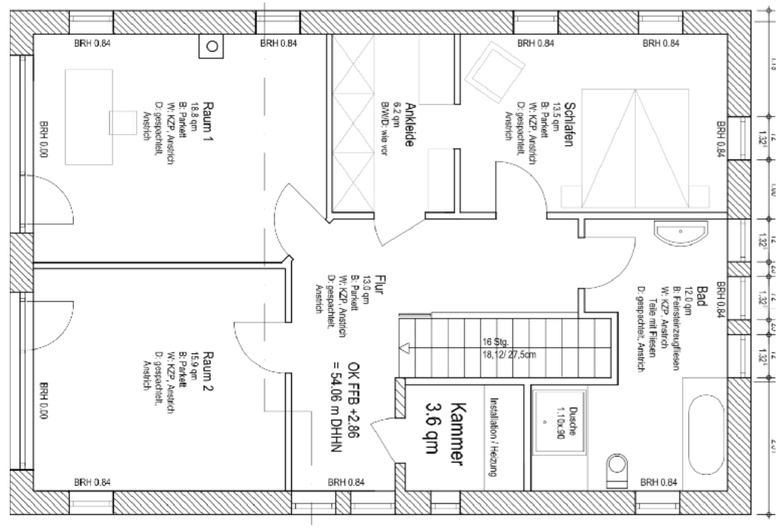
Software Engineering definiert fünf bis sechs Kernphasen und begleitende Prozesse



Software-Architekturen sind für den gesamten Software-Lebenszyklus relevant

Analogie zur Gebäudearchitektur





Lageplan

- Übersicht des Gesamtsystems
- Lokalisation einer einzelnen Komponente
- Verbindungen zwischen den Elementen

Ansichten

- Spezifische Sicht auf eine Komponente

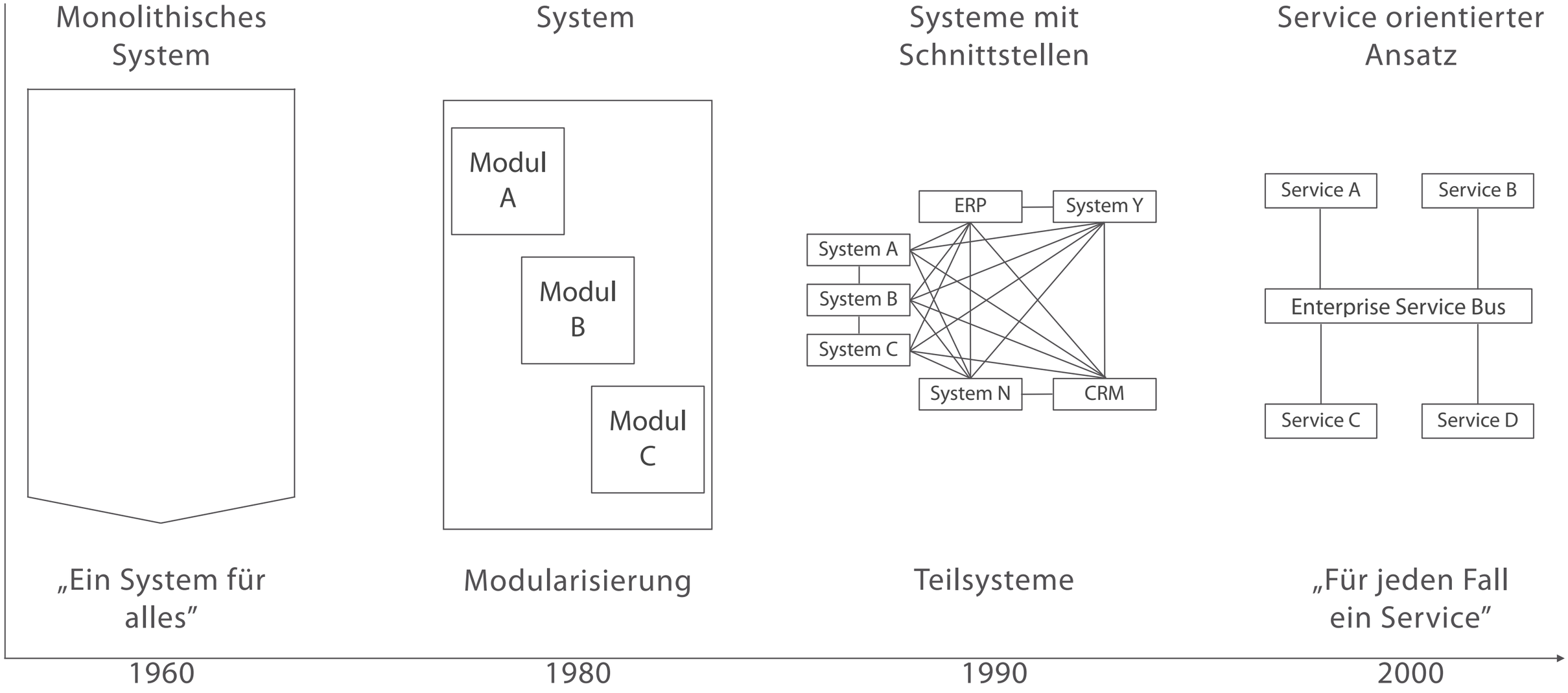
Ebenen / Grundriss

- Darstellung des Aufbaus einer Komponente

Verschiedene Sichtweisen ermöglichen einen Überblick über den Betrachtungsgegenstand, ähnlich ist es bei der Software-Architektur.

Quelle: Getty Museum 2022

Historische Entwicklung von Systemarchitekturen



Quelle: Vogel et al. 2009, S. 216-252

Vorteile und Probleme der Zerlegung in einzelne Bausteine

Vorteile

- Wiederverwendung einzelner Bausteine
- Einzelne Bausteine können einfacher gewartet und ausgetauscht werden
- Können arbeitsteilig entwickelt werden

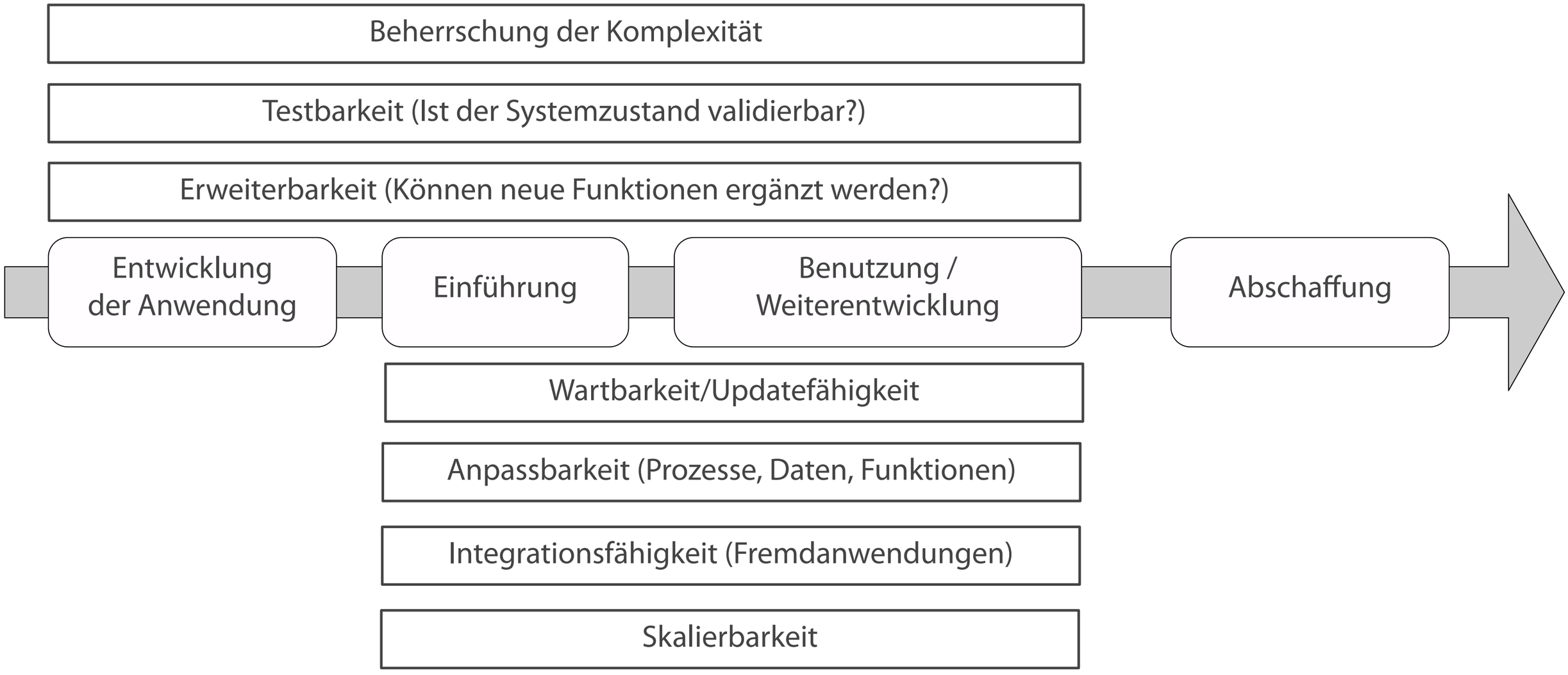
Probleme

- Kommunikationsaufwand zwischen den Bausteinen steigt mit der Anzahl an
- Erhöhter Integrationsaufwand zur Verbindung der einzelnen Bausteine
- Ggf. verringerter Austausch zwischen Abteilungen die für einzelne Module zuständig sind

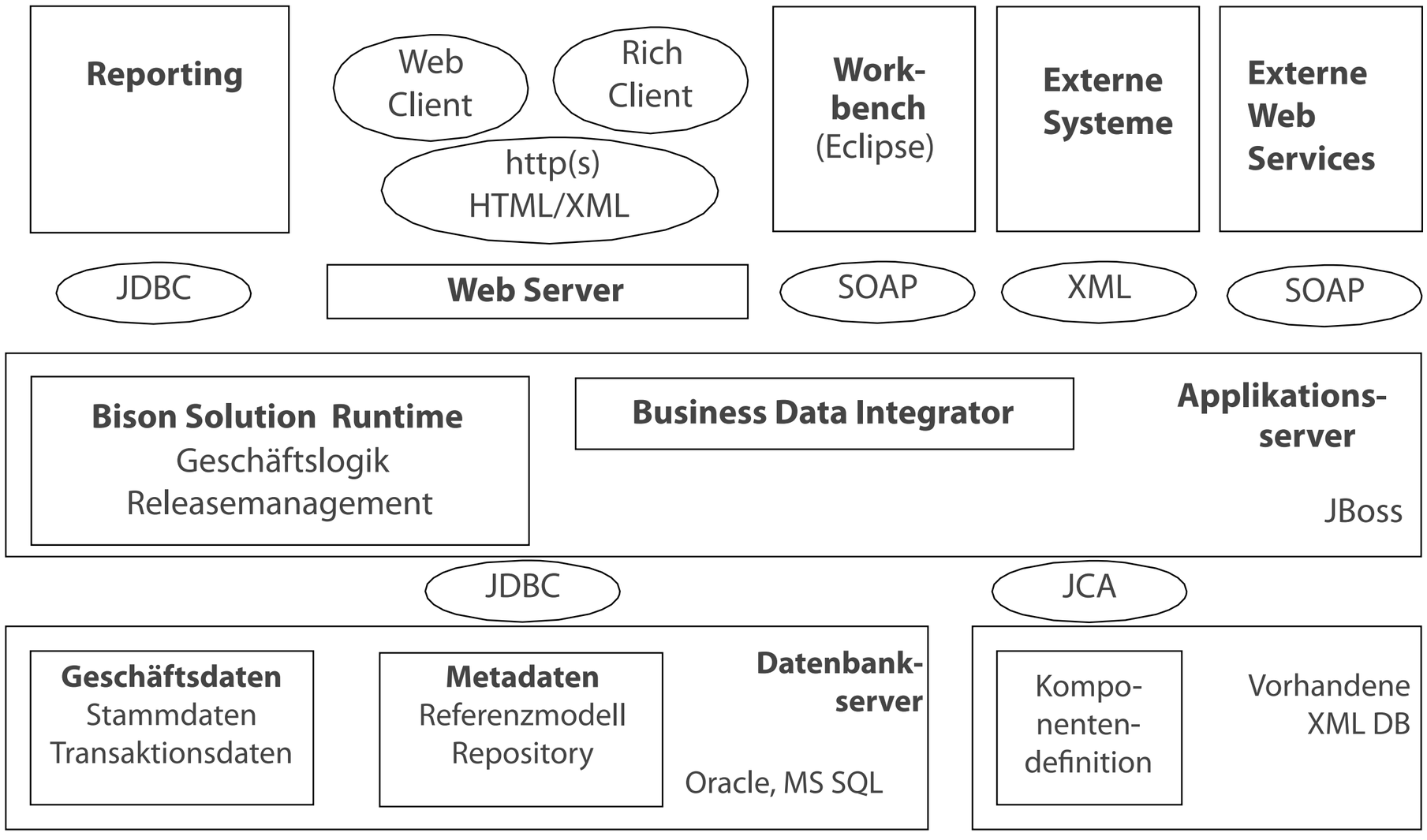
Eine Architektur wird benötigt, die festlegt, wie die einzelnen Bausteine miteinander kommunizieren

Quelle: Vogel et al. 2009, S. 216-252

Ziele einer Softwarearchitektur im Lebenszyklus von Standardsoftware



Architektur eines modernen ERP-Systems auf Basis von JAVA



Quelle: Bison AG 2020

Einführung von Software-Architekturen

Aufbau von Softwarearchitekturen

Einführung in die Model Driven Architecture

Bewertungskriterien

Einführung in Architekturmuster

Schichtenarchitekturen

Integrationsarchitekturen

Weitere Architekturmuster

Qualitätskriterien für Softwarearchitekturen

Ordnungsmäßigkeit

- Betrachten und Einhalten von anwendungsspezifischen Normen, Vereinbarungen, gesetzlichen Bestimmungen

Richtigkeit

- Aufgabenorientierte Zusammensetzung von Funktionen aus Teilfunktionen

Interoperabilität

- Schnittstellenbetrachtung von Komponenten und der Umgebung der Komponenten
- Standardisierte Beschreibung der Schnittstellen

Sicherheit

- Verhindern von unberechtigten Zugriffen auf Programme und Daten

Quelle: Starke 2017

Weitere Qualitätskriterien für Softwarearchitekturen

Zuverlässigkeit

- Erkennen von Abhängigkeiten durch globale Betrachtung des Gesamtsystems
- Effiziente Fehlerbehandlung und Nachvollziehbarkeit

Benutzbarkeit

- Einbeziehung aller Interessensgruppen auch für spezifische Bedürfnisse
- Standardisierung der Bedienelemente für Wiedererkennung und schnelle Erlernbarkeit

Effizienz (Performance)

- Überprüfung des Verhältnisses zwischen Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel
- Skalierbarkeit

Übertragbarkeit

- Verwendung von Mustern

Quelle: Starke 2017

Tragweite der Analogie zur Gebäudearchitektur

Gemeinsamkeiten zwischen Gebäude- und Software-Architektur

- Abstraktion und Fokussierung auf eine spezifische Darstellung innerhalb einer Sicht
- Viele Sichten zur umfassenden Architektur, nicht eine Sicht für alle Aspekte
- Darstellung des Planungsprozesses und Grundlage für die Umsetzung
- Verdeutlichung von Aufbau und Zusammenhang der Softwareelemente

Unterschiede zwischen Gebäude- und Software-Architektur

- Architekturkosten bei Bauprojekten nur ein kleiner Teil der Gesamtkosten
- Software-Architektur bei IT Projekten der größte und wichtigste Teil
- Gebäudearchitektur als greifbares Produkt, höherer Abstraktionsgrad bei Softwarearchitekturen

Allgemeiner Aufbau einer Software-Architektur

Modellsammlung

- Referenzmodelle und Ableitung der Architektur
- Grafische Modelle (FMC, UML, ...)
- Unterschiedliche Sichten
- Dokumentation der Modelle und Elemente

Technologie

- Technologieauswahl
- Technologieeigenschaften
- Dokumentation der Technologieentscheidungen

Umgebung

- Andere Hardware, Software, Plattformabhängigkeit
- Weitere Einflussfaktoren die für eine Designentscheidung benötigt werden
- Dokumentation der Umweltfaktoren / Elemente

Quelle: Starke 2017

Vier Arten von Sichten

Kontextsicht

- Zeigen den Zusammenhang des Systems mit seiner Umgebung aus der Vogelperspektive
- Schnittstellen nach außen
- Interaktion mit wichtigen Stakeholdern
- Notation z.B. durch Use Cases

Bausteinsicht

- Statische Struktur der Architekturbausteine des Systems, Subsysteme, Komponenten und deren Schnittstellen zueinander
- Notation z.B. durch UML-Klassendiagramme

Laufzeitsicht

- Beschreibt das Zusammenwirken der Bausteine zur Laufzeit
- Dynamische Strukturen
- Notation z.B. durch UML-Sequenz, Aktivitäts- oder Kollaborations/Kommunikationsdiagramme

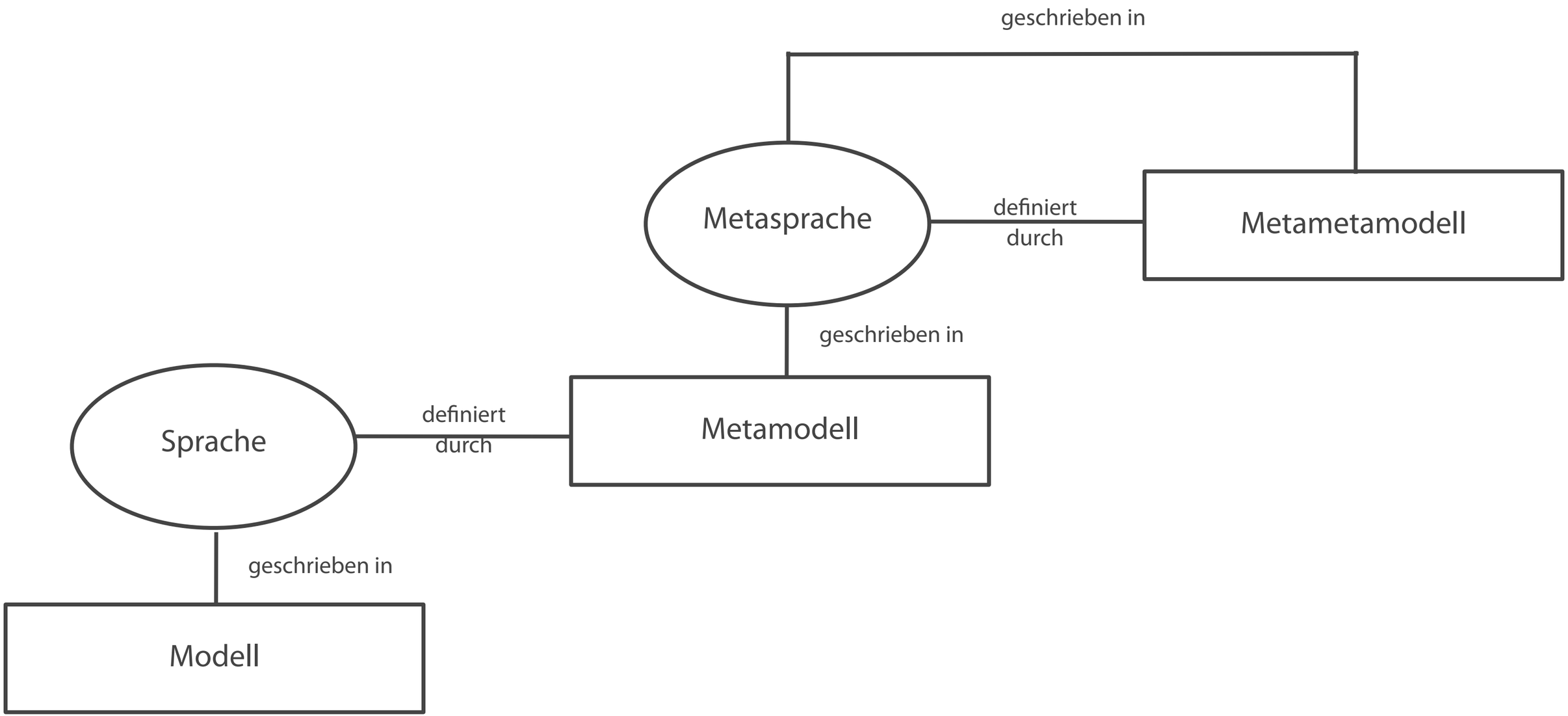
Verteilungssicht

- Infrastruktursicht
- Beschreibung der Hardwarekomponenten (Rechner, Prozessoren, Netztopologien
- System aus Betreibersicht
- Notation z.B. durch UML-Einsatzdiagramme

Die verschiedenen Sichten bilden ein umfassendes Bild der Software-Architektur.

Quelle: Starke 2017

Verschiedene Modellebenen im Überblick



Quelle: Object Management Group 2022a

Modellbegriff

Wesentliche Modellmerkmale

- Verkürzungsmerkmal - Beschränkung auf bestimmte Aspekte, die für den jeweiligen Zweck des Modells von Bedeutung sind
- Abbildungsmerkmal - Bestimmte Verhaltensweisen und Strukturen eines Systems werden durch Abbildung auf das Modell übertragen
- Pragmatisches Merkmal - Zweckbestimmung des Modells haben Einfluss auf die Verkürzungen und die Beziehungen zwischen Modell und realer Sache

Modelle

- Class, Responsibility and Collaboration-Karten (CRC-Karten)
- UML-Modelle
- eEPK als ARIS-Modelle
- BPML für Geschäftsabläufe
- Entity-Relationship-Modelle für Datenstrukturmodelle
- Deployment-Descriptor einer EJB-Komponente
- XML-Datei

Modelle sind zur Beschreibung von Software und zur Planung der Entwicklung notwendig

Quelle: Reussner 2006, S. 105f

Metamodell

Definition und Beschreibung

- Ein Metamodell ist ein Modell, das eine Menge anderer Modelle definiert, die als Instanzen des Metamodells bezeichnet werden.
- Ein Metamodell ist selbst wiederum in einer Metamodellsprache verfasst
- Eine Instanz ist konform zum Metamodell

Beispiele

- UML-Metamodell beschreibt die einzelnen Diagrammtypen und deren Verwendung
- XML-Schema beschreibt XML-Datei
- Meta Object Facility (MOF), standardisiert durch die OMG

Metamodellebene

- Definiert, die Elemente eines Modells
- Dient der Beschreibung von Modellierungssprachen

Class

Modellebene

- Definiert die Elemente eines Systems
- Dient der Beschreibung eines Softwaresystems

Termin

Instanzebene

- Beschreibt ein instanziiertes, also zur Laufzeit auftretendes Objekt

Arzttermin

Quelle: Reussner 2006, S.106f, Object Management Group 2022a

Instanziierungen der Modellebenen

M3 - Metamodellsprache

- Beispiel: MOF

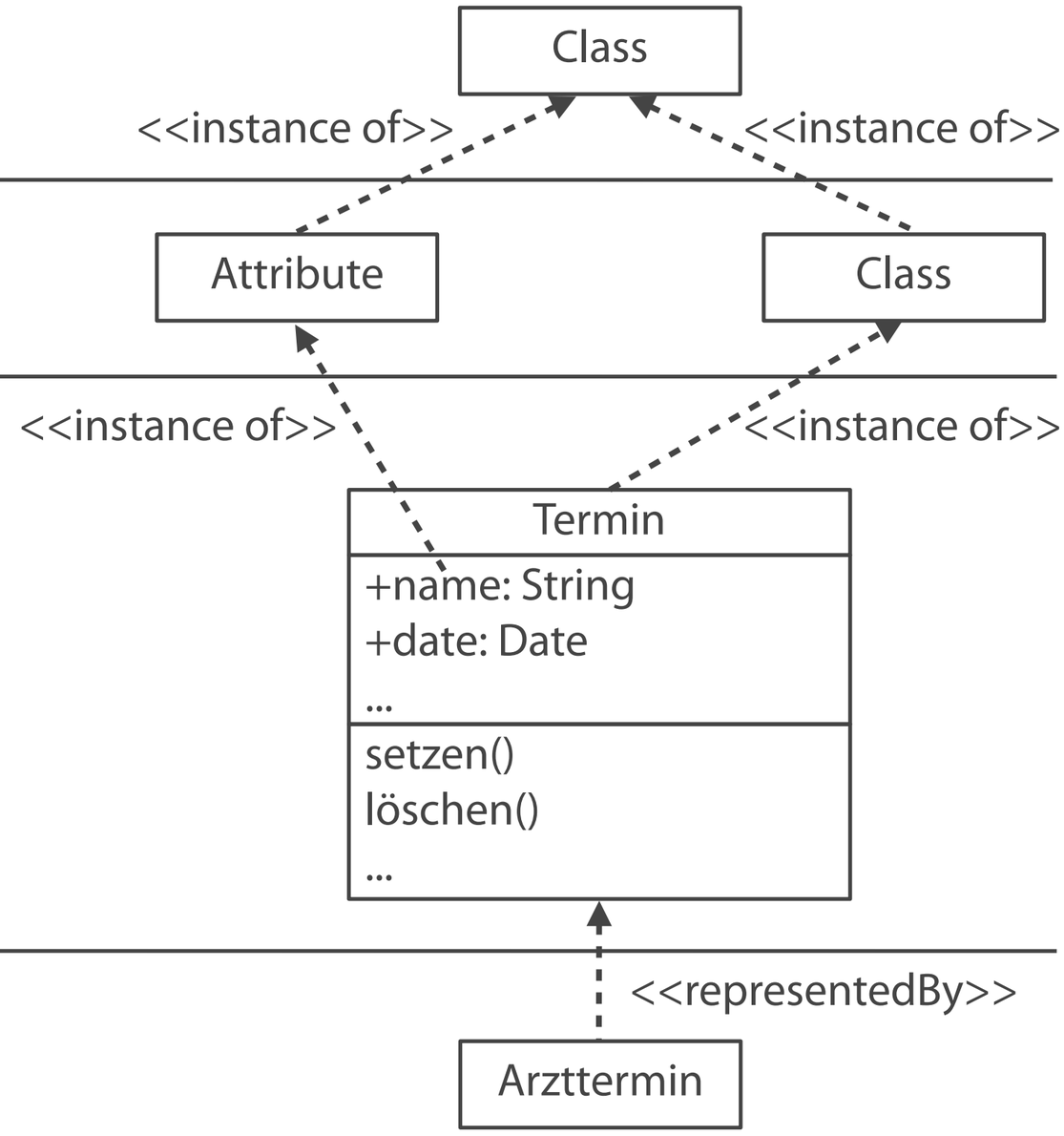
M2 - Modellsprache

- Beispiel: UML

M1 - Modellsprache

- Beispiel: Klassendiagramm

M0 - Instanzbeschreibung



Quelle: Object Management Group 2022a

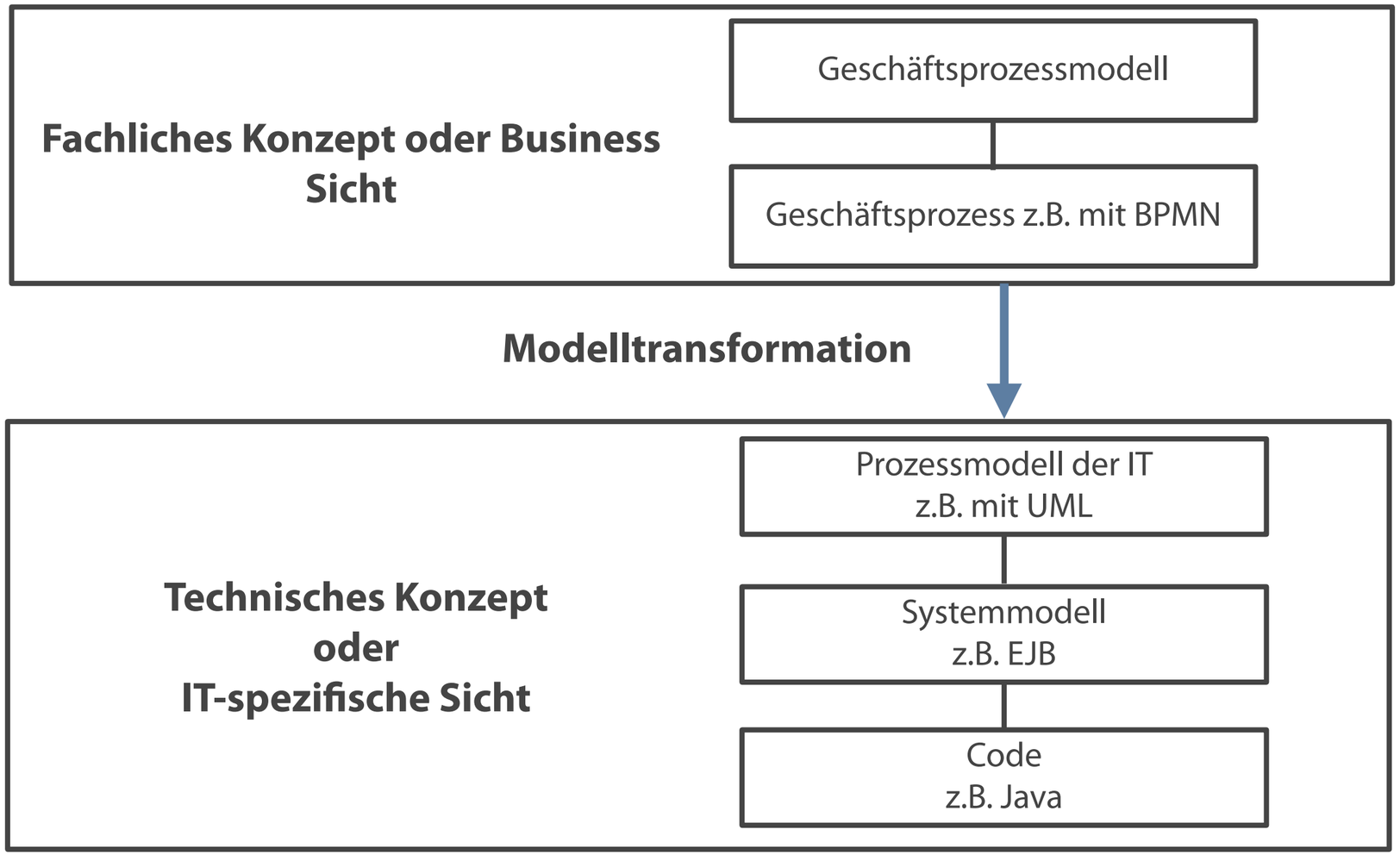
Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- **Einführung in die Model Driven Architecture**
- Bewertungskriterien

Einführung in Architekturmuster

- Schichtenarchitekturen
- Integrationsarchitekturen
- Weitere Architekturmuster

Überführung von Modellen



- Abbildung der Geschäftsprozesse eines Unternehmens im Anwendungssystem
- Wissen über die Geschäftsprozesse von besonderer Wichtigkeit
- Ein Ansatz zur anforderungsgerechten Architekturentwicklung ist die Model Driven Architecture

Modelltransformation ist eine typische Compilerfunktion

Quelle: Kempa 2005, S. 299

Entwicklung von Informationssystemen

Situation

- SOLL-Geschäftsprozessmodelle liegen vor, z.B. in Form von EKPs
- Informationssysteme dienen der Automatisierung dieser Geschäftsabläufe im Unternehmen

Häufiges Vorgehen

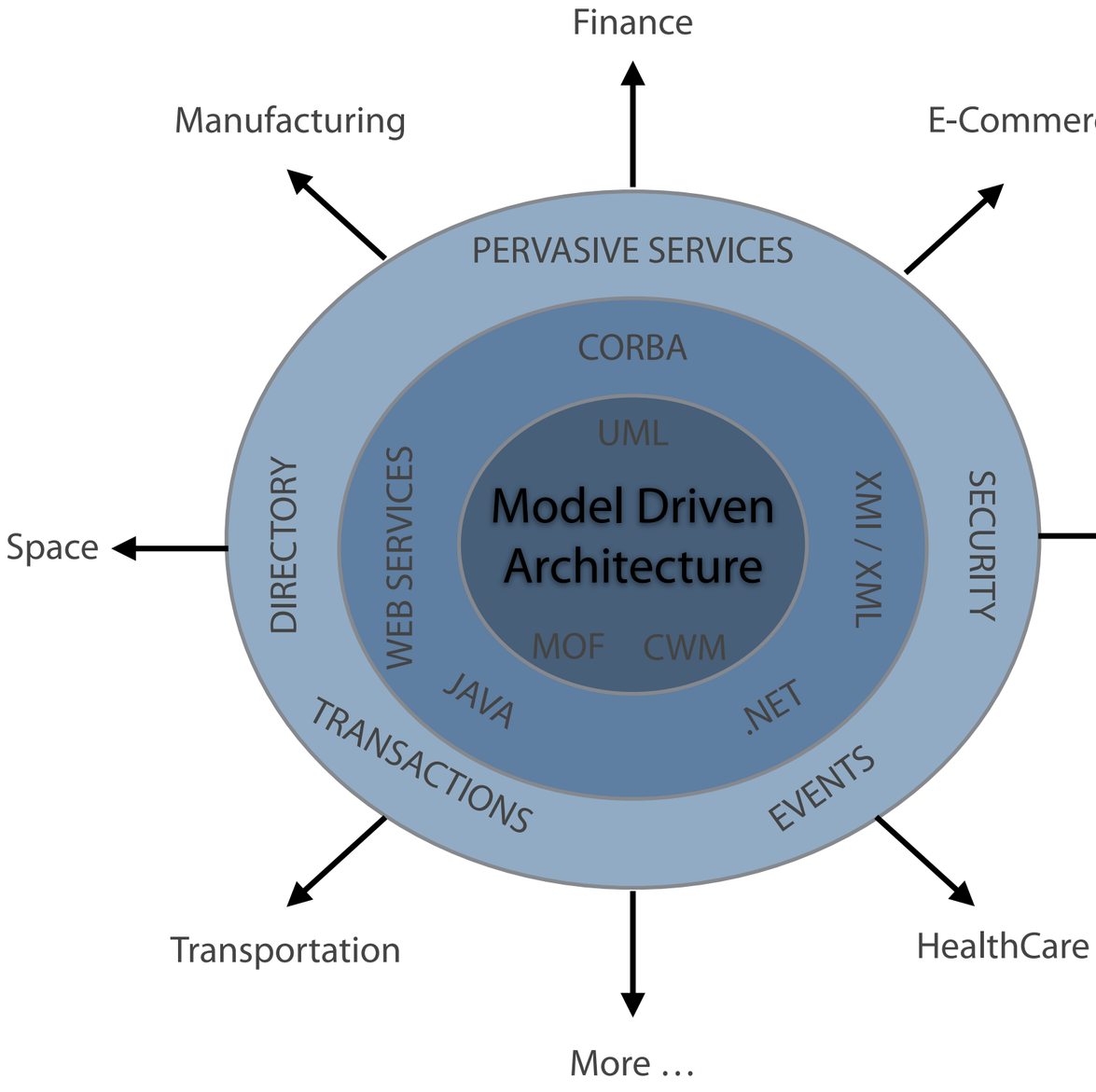
- Manuelle Transformation der Geschäftsmodelle in IT-Modelle
- IT-Modelle werden weiter manuell in plattformspezifische Modelle und schließlich in ausführbaren Quellcode transformiert

Häufiges Problem

- Fachliche Sicht lässt sich nicht immer in technische Sicht abbilden
- Kommunikationsprobleme zwischen den entsprechenden Abteilungen
- Wissen der Entwickler wandert ab

Lösung: Modell-Transformation weitestgehend automatisieren

Das Konzept der Model Driven Architecture



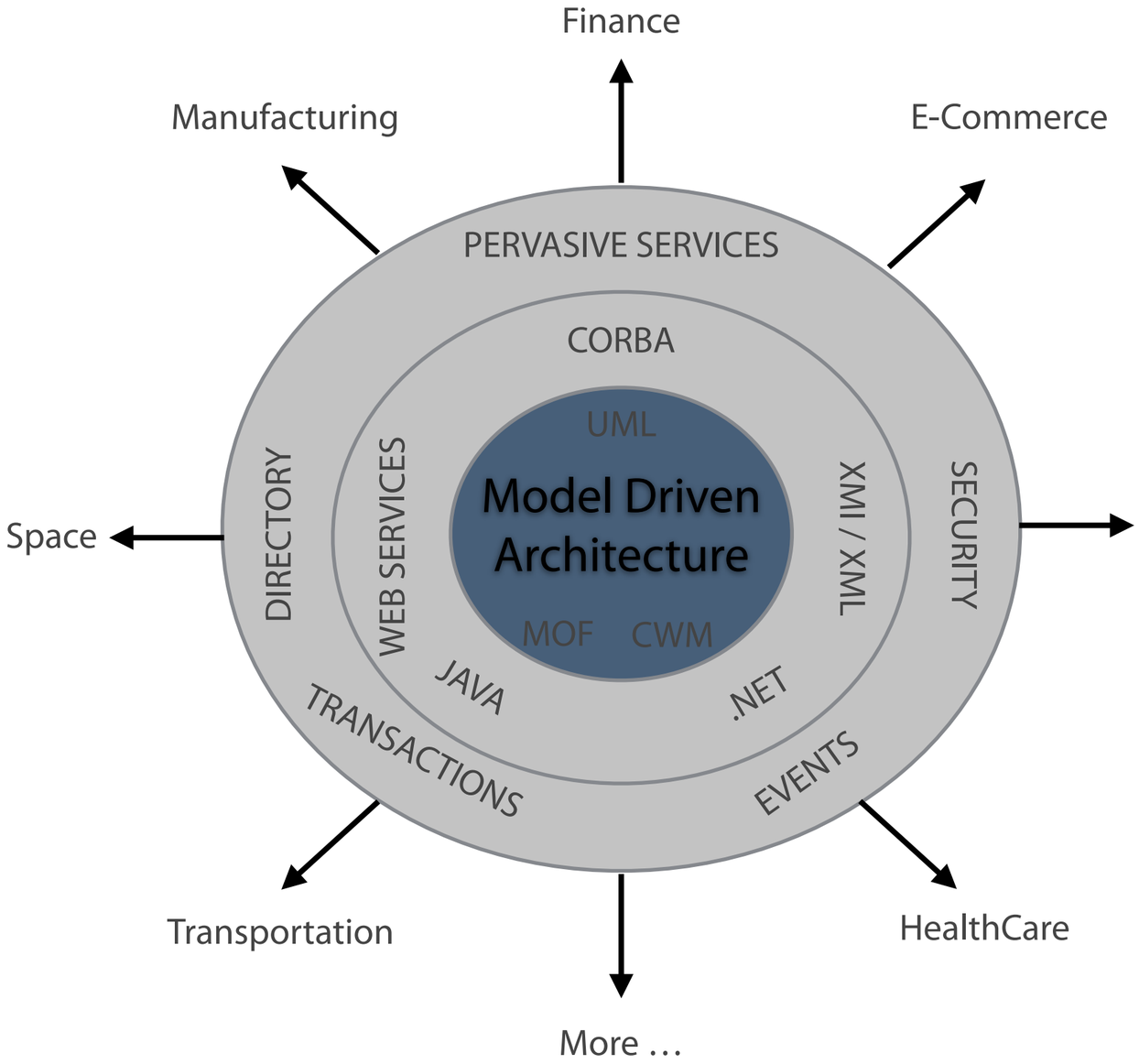
Bereiche der MDA

- Technologien für die Modellierung von Softwaresystemen
- Technologien für mögliche Zielplattformen einer Modelltransformation
- Pervasive Services = grundlegende Dienste, wie Sicherheit, Transaktion und Persistenz
- Überblick über verschiedene Märkte und Domänen, die von der modellgetriebene Entwicklung profitieren

Standardisierte Transformation vom plattformunabhängigen Modell zum Quellcode.

Quelle: Object Management Group 2022b

Das Konzept der Model Driven Architecture



Quelle: Object Management Group 2022b

UML (Unified Modeling Language)

- Modellierung von Struktur und Verhalten von Systemen

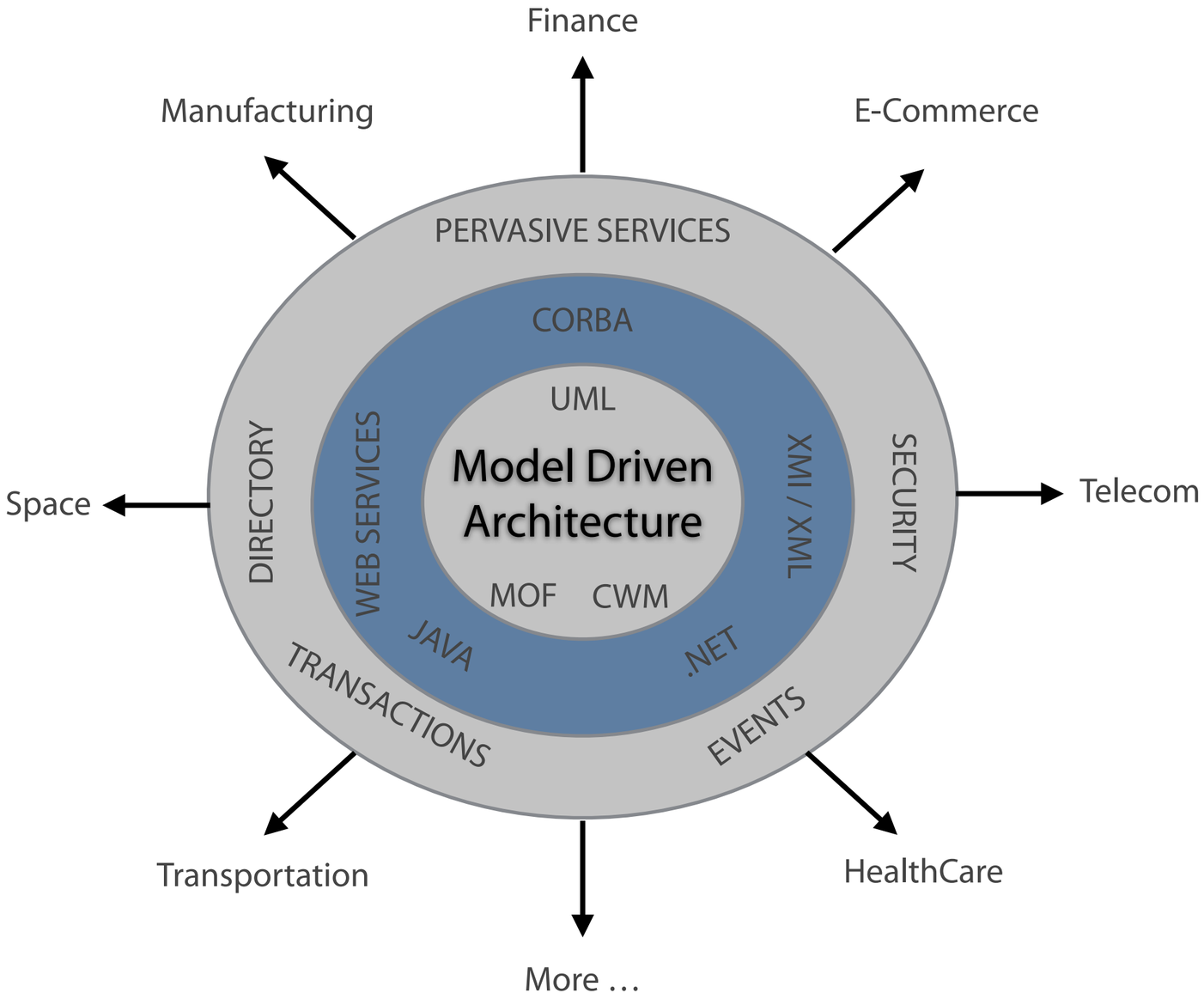
UML (Unified Modeling Language)

- MOF (Meta Object Facility) = Metamodellierungsprache, Modellierungssprache zur Modellierung von Modellierungssprachen
  - Definition von Schnittstellen
  - Manipulation von Schnittstellen

CWM (Common Warehouse Model)

- Metamodell zur Modellierung von Data Warenhauses
- Gesamter Lebenszyklus eines DW
- Entwicklung bis Wartung
- Modellierung von Datenschemata für DW-Applikationen

Das Konzept der Model Driven Architecture



CORBA, J2EE, .NET

- Aktuelle Softwareplattformen

XML/XMI

- Sind für den Export und die Speicherung des Modells zuständig

Es gibt keine geschlossene Menge, sondern nur aktuell favorisierte.

FORTRAN und COBOL-Systeme können genauso modellgetrieben entwickelt werden.

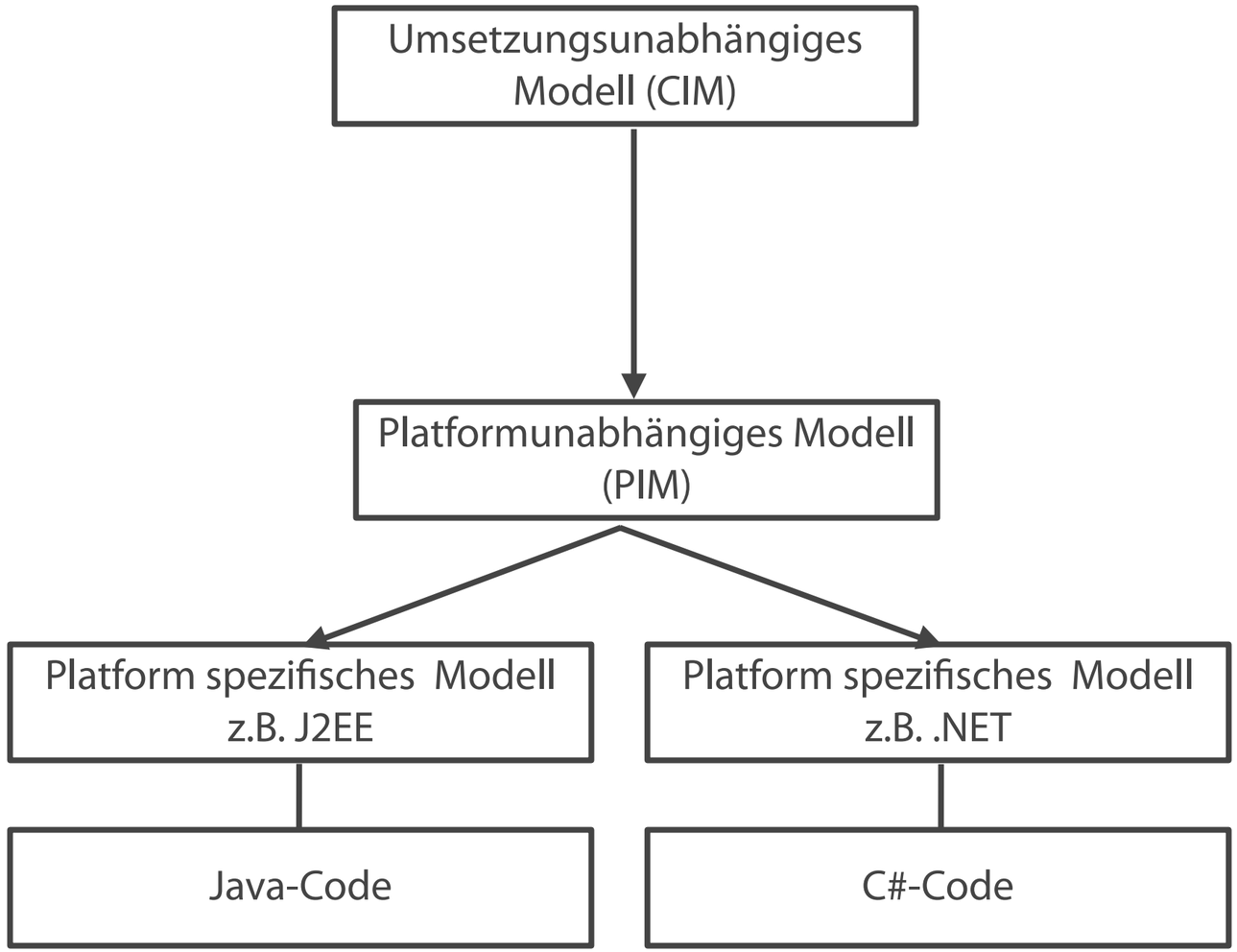
Quelle: Object Management Group 2022b

Modellgetriebene Architektur

- Beschreibung eines Softwaresystems auf fachlicher Ebene
- Legt fest, was ein Softwaresystem leistet
- Wird auch als Domänenmodell oder Geschäftsmodell bezeichnet
- Definiert Anforderungen
- Beschreibt das System und seine Umwelt

- Beschreibt die Funktionalität eines System, ohne die Zielplattform zu kennen
- resistent gegen Technologieveränderung

- Umsetzung eines PIM für eine konkrete Plattform
- Nutzt die plattformspezifischen Schnittstellen



Schrittweises Transformieren der verschiedenen Modelle

Quelle: Kempa 2005, S. 300

Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- **Bewertungskriterien**

Einführung in Architekturmuster

- Schichtenarchitekturen
- Integrationsarchitekturen
- Weitere Architekturmuster

ATAM - Architecture Trade-off Analysis Method

Beschreibung

- Szenariobasierter Ansatz zur Bewertung der Architekturentscheidungen, der einen klaren Schwerpunkt auf die Anforderungen an Qualitätsmerkmale legt.
- Entwurfsentscheidungen beeinflussen mehrere Qualitätsmerkmale
- Verbesserung eines Qualitätsmerkmals führt zur Verschlechterung eines anderen

Vorteile

- Eindeutige Korrelation der Anforderungen an Qualitätsattribute und der zugrunde liegenden Geschäftstreiber mit den Merkmalen einer Architektur.
- Früherkennung von Risiken oder Fehlentscheidungen
- Unterstützung der Architekten bei der Architekturentwicklung
- Unterstützung bei der Auswirkungsanalyse in Situationen mit veränderten Anforderungen an Qualitätsmerkmale.

ATAM - Vorgehensmodell

Phase 1: Präsentation

- Vorstellen von ATAM an die Beteiligten durch den Evaluierungsgruppenleiter
- Vorstellen der Geschäftsziele, Motivation für Entwicklungsaufwand und primäre Architektur-Qualitätsziele
- Vorstellen der entsprechenden Architektur für das Softwaresystem im Hinblick auf die genannten Geschäftsziele

Phase 2: Erhebung & Analyse

- Identifizieren der architektonischen Ansätze: Architekt kennzeichnet architektonische AnsätzeErstellen des Qualitätsattributbaumes: Auflisten und Strukturieren der Qualitätsattribute (wie Performanz, Verfügbarkeit, Sicherheit, Modifizierbarkeit)
- Analyse der architektonischen Ansätze: Identifizieren von architektonischen Risiken, empfindlichen Punkten und Zielkonfliktpunkten

Phase 3: Testen

- Brainstorming und Gewichtung der Szenarien
- Analyse der architektonischen Ansätze: Fokus auf hoch gewichtete Szenarien, die im vorigen Schritt hervorgegangen sind

Phase 4: Reporterstellung

- Präsentieren der Ergebnisse: Aus gesammelten Informationen wird Report erstellt und die Ergebnisse übersichtlich zusammengefasst

CBAM - Cost Benefit Analysis Method

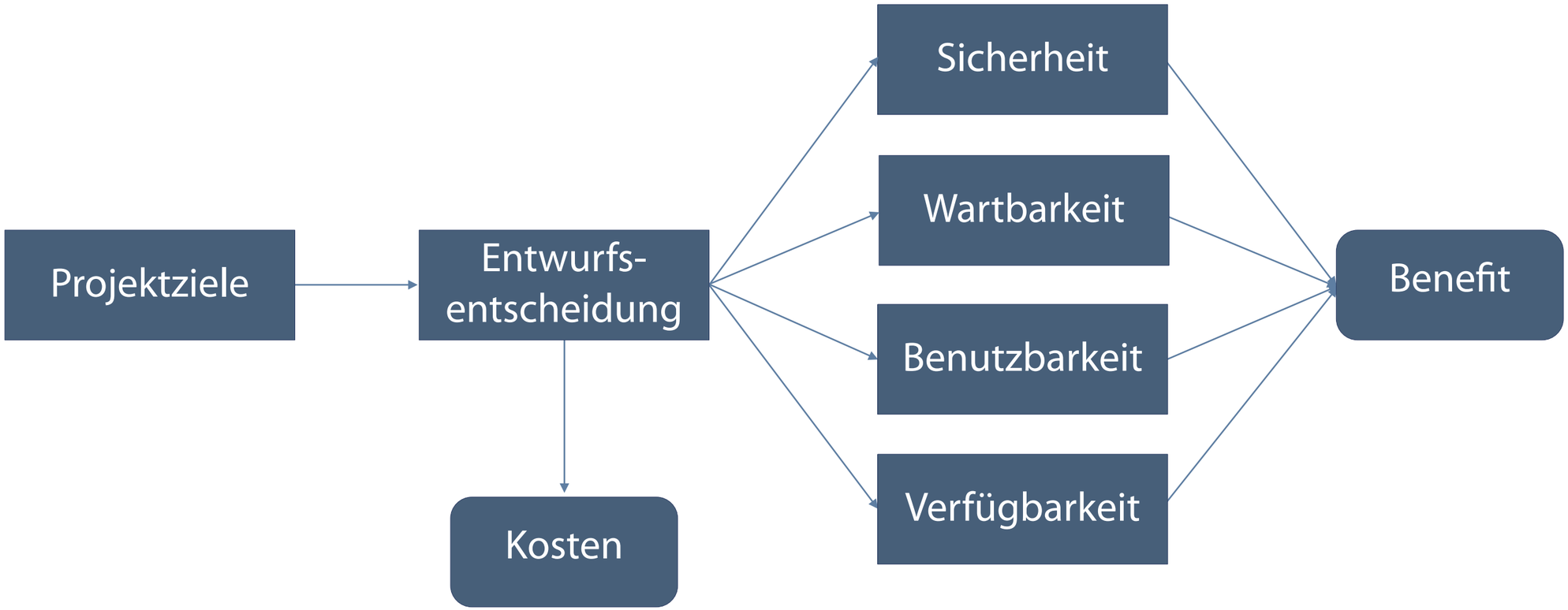
Beschreibung

- Ökonomische Konsequenzen der Entwurfsentscheidung
- Analyse der Qualitätsattribute hinsichtlich Kosten und Leistungen

Vorteile

- Einfache Anwendung
- Verschafft Klarheit in unvorhersehbaren Situationen
- Identifizieren und bewerten von Ausgaben
- Gibt an, inwiefern der Nutzen die Kosten überwiegt

Kontext und Ziele der CBAM



Quelle: Reussner 2006, S. 291

CBAM - Vorgehensmodell

Phase 1

- Ermitteln der „Scenarios of Concern“: es werden die Szenarien, welche für die Stakeholder am meisten von Interesse sind mit den entspr. architektonischen Strategien ermittelt

Phase 2

- Schätzung der Qualitätsattribut-Leistung: Manager, die in das Geschäft involviert sind schätzen die Leistungen, welche aus dem Systemverhalten/Qualität resultieren

Phase 3

- Quantifizieren der Leistungen für versch. architektonische Strategien: Architekten ermitteln Strategien anhand des Qualität-Niveaus, das mittels der architektonischen Strategie erreicht werden kann (bzw. wie und mit welchen Mitteln)

Phase 4

- Quantifizierung die aus der Architektur resultierenden Kosten und Terminpläne - Zurückgreifen auf dokumentierte Erfahrungen zu bestehenden Architekturen und Projekten

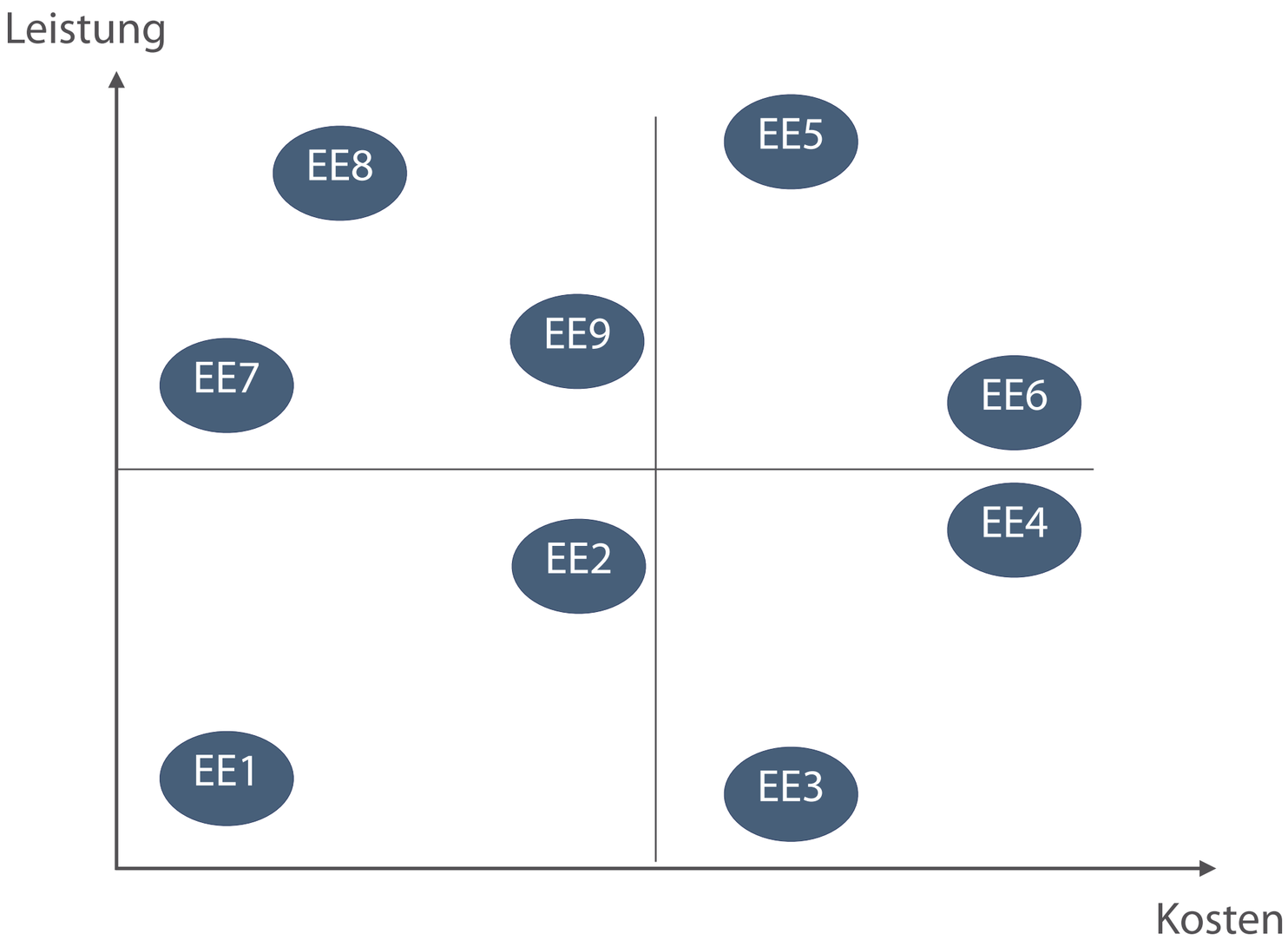
Phase 5

- Kalkulation des (wirtschaftlichen) Attraktivität: ausgehend von Schritt 3 und 4 wird zu jeder der ermittelten architektonischen Strategien das Verhältnis zwischen deren Leistung und Kosten berechnet

Phase 6

- Entscheidungsfindung für architektonische Strategie basierend auf der Kalkulation aus Schritt 5

Nutzen-Kosten-Portfolio



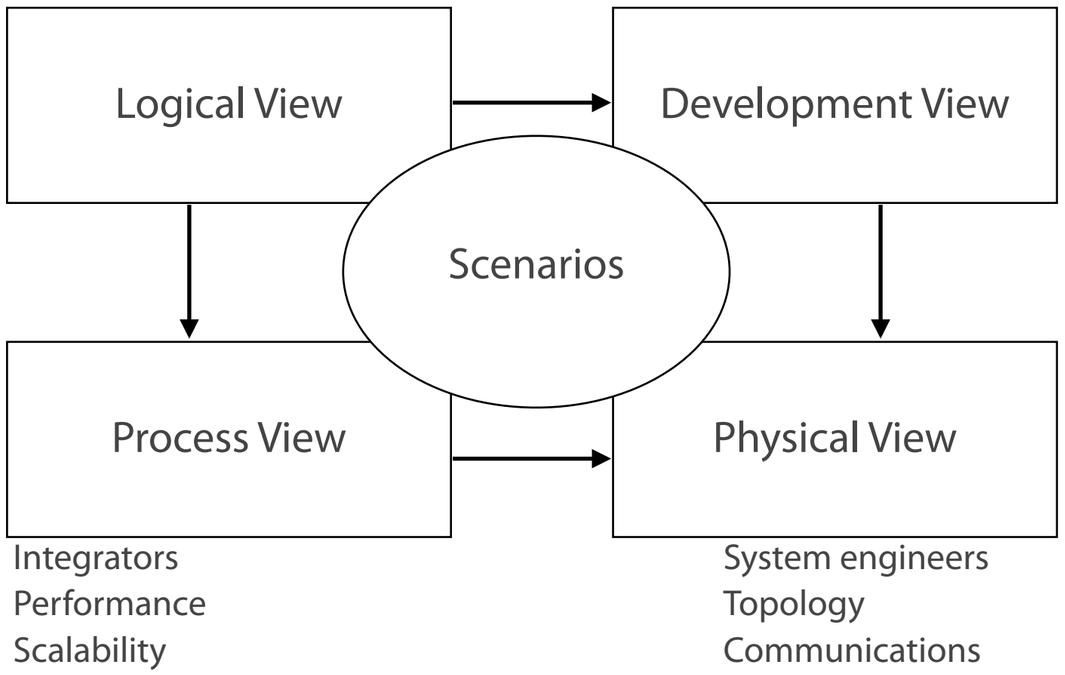
Quelle: Reussner 2006, S. 291

ALMA - Architecture-Level Modifiability Analysis

Beschreibung

- Entwickelt und getestet für Business Information Systems (BIS) aber auch geeignet für Embedded Systems
- Adressiert Modifizierbarkeit
- Vorhersage der Kosten zukünftiger Änderungen
- Identifizieren von Systeminflexibilität
- Input: „4+1" Modell von Kruchten + Architekturbeschreibung in UML

4+1-Modell von Kruchten



Quelle: Kruchten 1995, S. 43

ALMA - Vorgehensmodell

Phase 1

- Setzen des Analyseziels: mit ALMA lassen sich unterschiedliche Ziele setzen: Risikobeurteilung, Wartungs- und Kostenvorhersage, Software-Architekturselektion

Phase 2

- Beschreibung der Architekturen: Zerlegung des Systems in einzelne Systemkomponenten, Beschreibung der Beziehungen zwischen den Komponenten und der Beziehung des Systems zur Umgebung

Phase 3

- Ermitteln von Change-Szenarien: Szenarien finden, die eine Rolle für die Architektur Modifizierbarkeit spielen; Interviews mit Stakeholdern und anschließende Dokumentation der resultierenden Change-Szenarien

Phase 4

- Evaluierung der Change-Szenarien: ALMA Analyst kommuniziert mit Architekten u. Entwicklern, um die Auswirkung der Change-Szenarien festzustellen u. das Resultat in einer verwendbaren und messbaren Weise für das Analyseziel auszudrücken

Phase 5

- Interpretation der Resultate: Deutung der Resultate in Übereinstimmung mit den Zielen der Analyse und Überprüfung dieser gegen die Systemanforderungen

Überblick: Qualitätseigenschaften und Bewertungstechniken

	ATAM	CBAM	ALMA
Sicherheit			
Verfügbarkeit	X	O/X	
Zuverlässigkeit	O/X		
Wartbarkeit	X	O/X	
Performanz	X	O/X	
Sicherheit	O/X		
Modifizierbarkeit	X	O/X	X
Erweiterbarkeit	X	O/X	
Ökonomie		X	
Portierbarkeit	X	O/X	

X - Wird unterstützt
X/O - Kann unterstützt werden

Quelle: Reussner 2006, S. 286

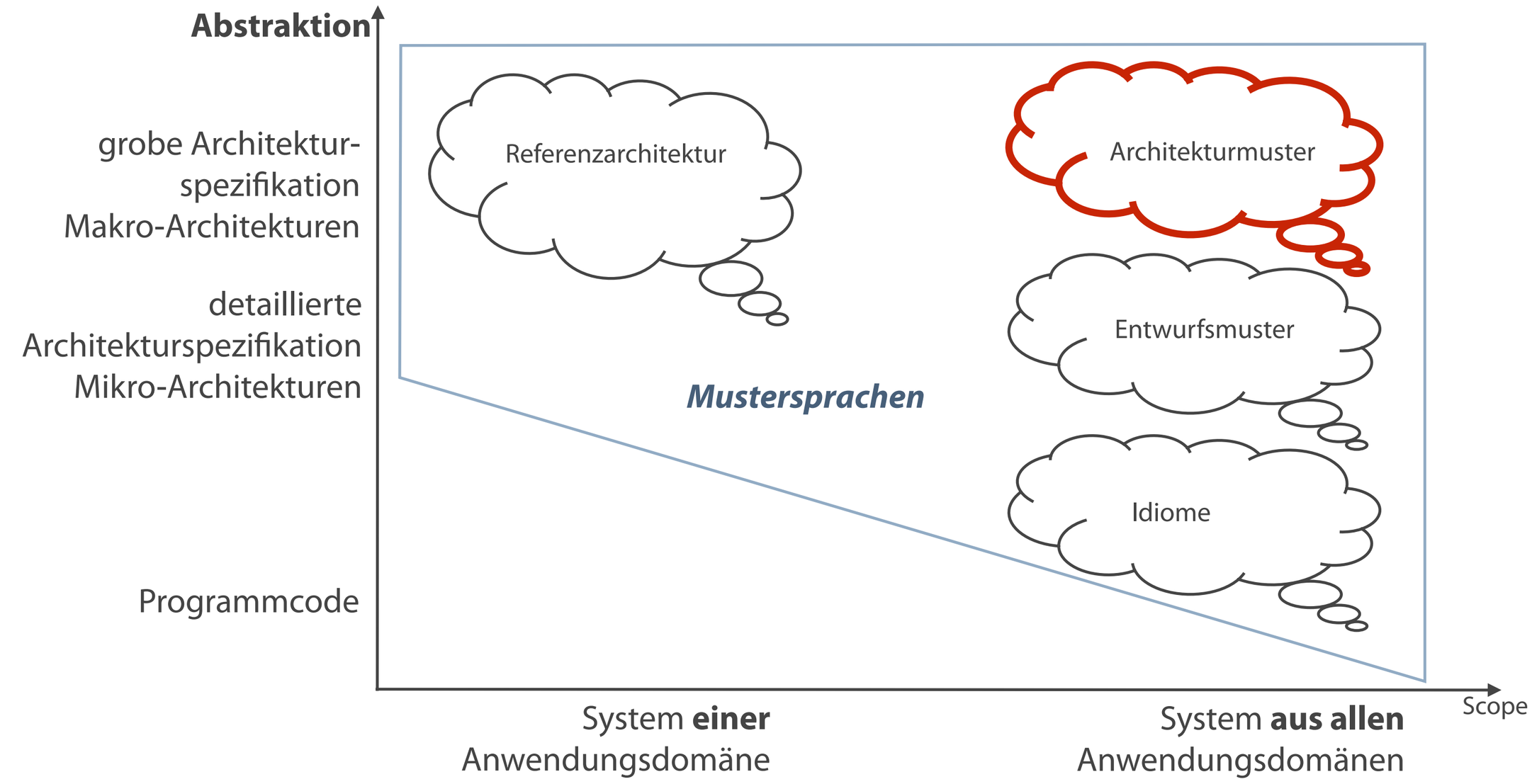
Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- Bewertungskriterien

Einführung in Architekturmuster

- Schichtenarchitekturen
- Integrationsarchitekturen
- Weitere Architekturmuster

Architekturstil - Architekturmuster - Entwurfsmuster



Ein Architekturstil stellt das Alphabet für die Mustersprachen zur Verfügung.

Quelle: Reussner 2008, S. 330

Musterkategorien

Architekturmuster

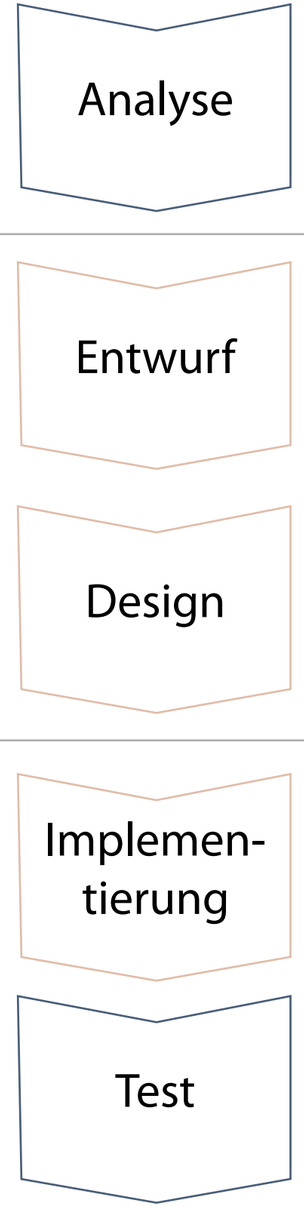
- Templates für konkrete Softwarearchitekturen
- Systemweite Spezifikation von Eigenschaften einer Anwendung
- Beeinflussung der Architektur der Subsysteme

Entwurfsmuster

- Schema zur Entwicklung von Subsystemen bzw. Komponenten von Systemen
- Geringere Reichweite und Auswirkungen als Architekturmuster
- Unabhängig von Programmiersprachen oder Programmierparadigmen

Idiome

- „Implementierungsmuster“
- Abhängig von Programmiersprachen
- Lösen konkrete Probleme in einer konkreten Programmiersprache



Musterkategorien orientieren sich am Entwicklungszyklus von Software

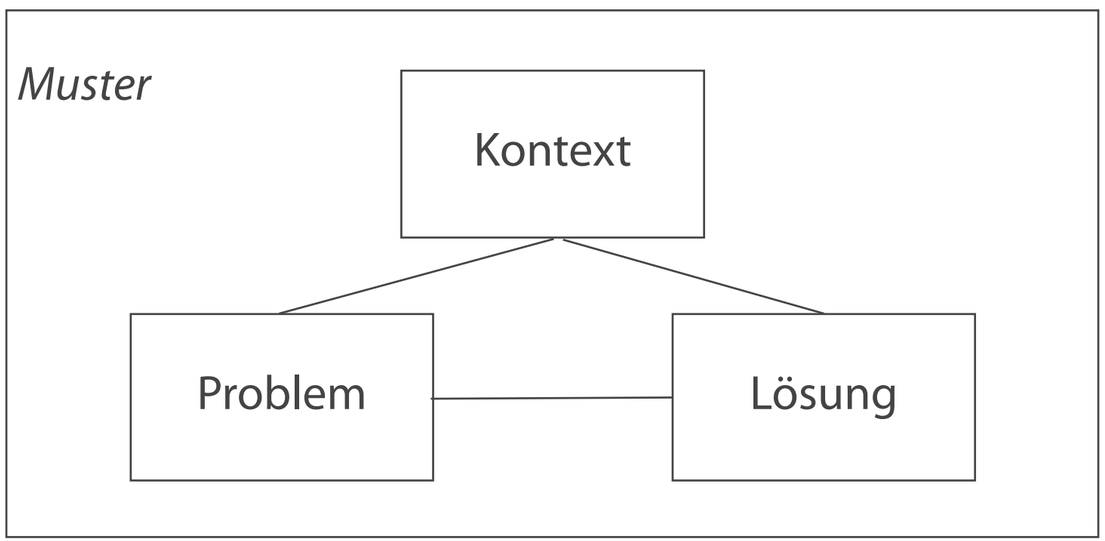
Quelle: Buschmann 1996, S. 12-14

Zuordnung von Anwendungsproblemen zu Mustern

	Architekturmuster	Entwurfsmuster	Idiome
Vom Untergeordneten zur Struktur	▪ Schichten ▪ Pipes & Filters ▪ Blackboard		
Verteilte Systeme	▪ Broker ▪ Pipes & Filters ▪ Microkernel		
Interaktive Systeme	▪ Model View Controller ▪ PAC		
Anpassungsfähige Systeme	▪ Microkernel ▪ Reflection		
Strukturelle Dekomposition		▪ Whole-Part	
Arbeitsorganisation		▪ Master-Slave	
Zugriffskontrolle		▪ Proxy	
Management		▪ Command Processor ▪ View Handler	
Kommunikation		▪ Publisher-Subscriber ▪ Forwarder-Receiver ▪ Client-Dispatcher-Server	
Resource Handling			▪ Counted Pointer

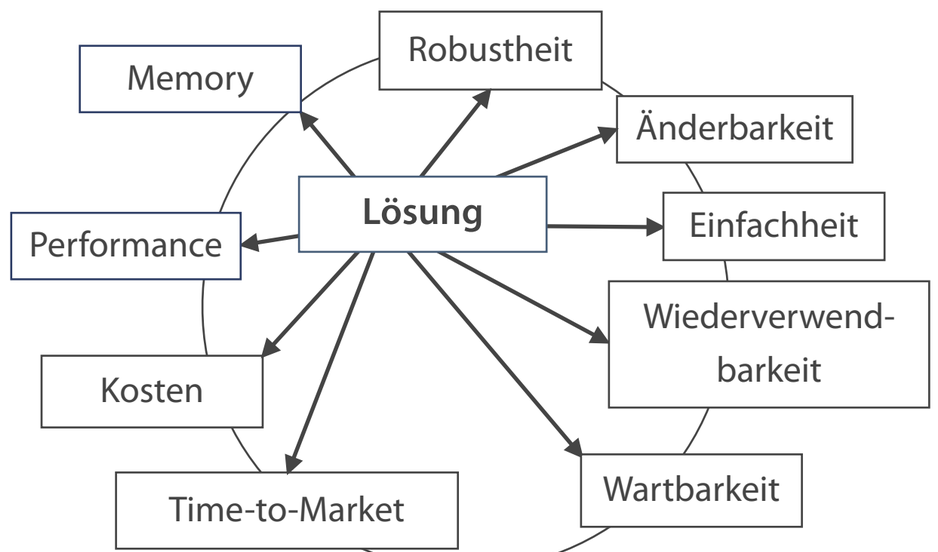
Quelle: Buschmann 1996, S. 366

Definition Softwaremuster



Muster

- Dreiteiliges Regelwerk zur Verdeutlichung von Beziehungen zwischen Kontext, Problem und Lösung



Softwaremuster

- Dreiteilige Regel
- Beziehung zwischen einem bestimmten Kontext, einem bestimmten System an Kräften, die in diesem Kontext wiederkehrend auftreten, und einer bestimmten Software-Konfiguration, die diesen Kräften erlaubt, sich gegenseitig aufzulösen

Quelle: Vogel 2005, S. 173

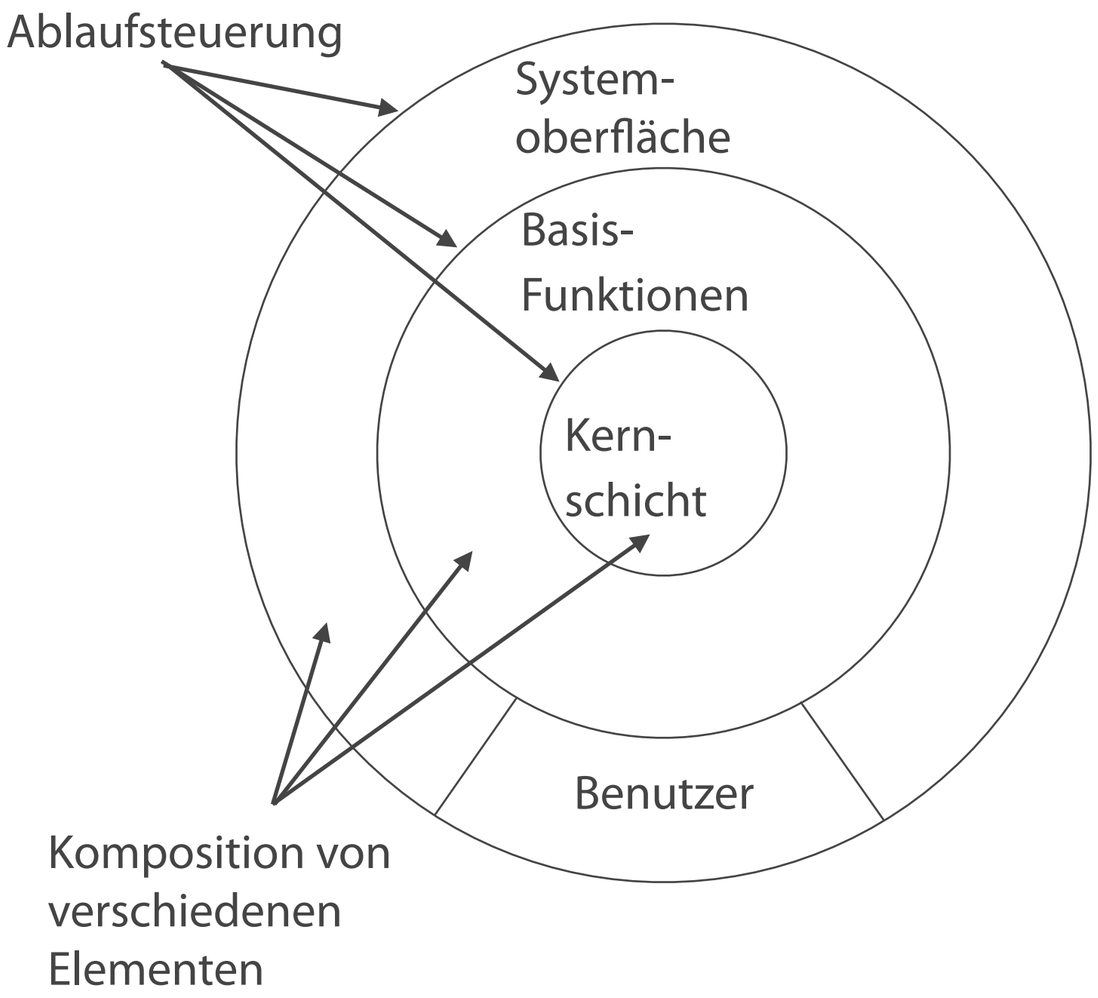
Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- Bewertungskriterien

Einführung in Architekturmuster

- **Schichtenarchitekturen**
- Integrationsarchitekturen
- Weitere Architekturmuster

Schichtenmuster



Stilbeschreibung

- Hierarchische Organisation
- Zur Verfügungstellung von Services für die übergeordnete Schicht
- Benutzung der Services der darunter liegenden Schicht
- Stilkomponenten = Schicht
- Stilkonnektoren = Interaktionsbestimmende Protokolle zwischen den Schichten

Die Drei-Schichten-Architektur ist insbesondere für betriebliche Anwendungssysteme etabliert.

Quelle: Shaw und Garlan 1996, S. 25

Charakter von Schichtenmustern

Zweck und Aufbau

- Mittel der Strukturierung
- Darstellung verschiedener, aufeinander aufbauender Abstraktionslevel
- Schichten können selbst wieder strukturiert sein

Vorteile

- Leicht verständlich
- Technologietrennung und -integration
- Wohl definierte Zugriffskonzepte durch Beschreibung der Protokolle

Übliche Modelle

- 2-Schichten-Modell, z.B. Rich-Client Szenarien
- 3-Schichten-Modell, z.B. Daten-, Geschäftslogik- und Präsentationsschicht
- N-Schichten-Modell, z.B. ISO OSI Referenzmodell für die Kommunikation offener Systeme

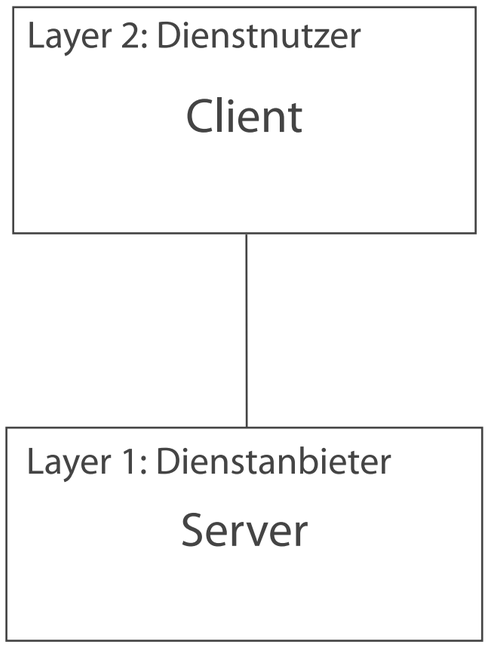
Nachteile

- Notwendigkeit von definierten stabilen Schnittstellen
- Kritische Performance
- Hohe Kosten bei Gesamtsystemveränderung

Schichten-Muster werden häufig in betrieblichen Anwendungssystemen eingesetzt.

Quelle: Buschmann 1996, S. 31-51

Anwendung des Schichten-Musters in betrieblichen Anwendungssystemen
Client-Server-Architekturen



Enthält die Anwendungslogik, häufig umgesetzt als Fat-Client

Stellt für die Bearbeitung der Funktionen dem Client die notwendigen Daten zur Verfügung

Eigenschaften

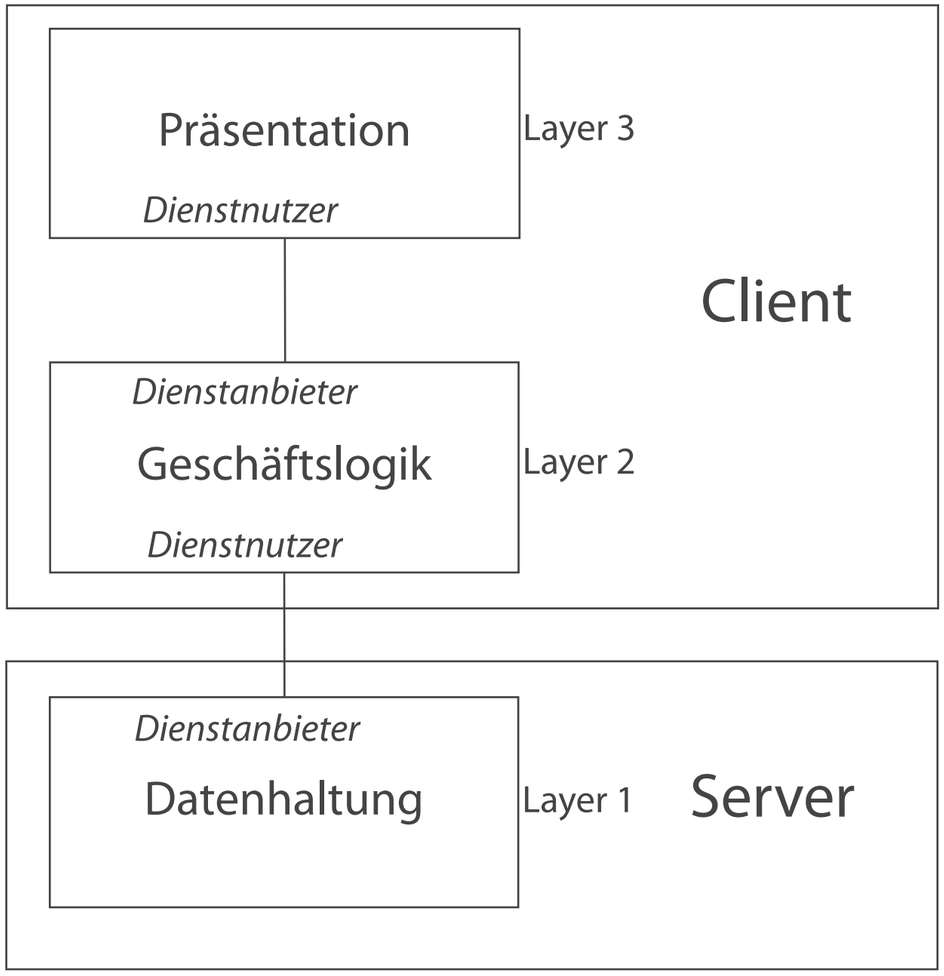
- Trennung von Datenhaltung und Datenverarbeitung
- Keine Verantwortung für die Datenspeicherung und sämtlicher damit verbundener Konzeptionen der Client-Komponente
- Client-Komponente nutzt die zugesicherten Dienste der Server-Komponente

Die grundlegendste Anwendung des Schichten-Musters ist die Client/Server-Architektur.

Quelle: Hasselbring 2006

Anwendung des Schichten-Musters in betrieblichen Anwendungssystemen

Drei-Schichten-Architektur



Eigenschaften

- Weitere Dekomposition der Client-Komponente
- Businesslogik wird getrennt von der Präsentation
- Variante 1: Geschäftslogik ist Teil des Servers (Thin-Client)
- Variante 2: Geschäftslogik ist Teil des Client (FAT-Client)

Die Client-Server-Architektur lässt sich weiter unterteilen.

Quelle: Hasselbring 2006

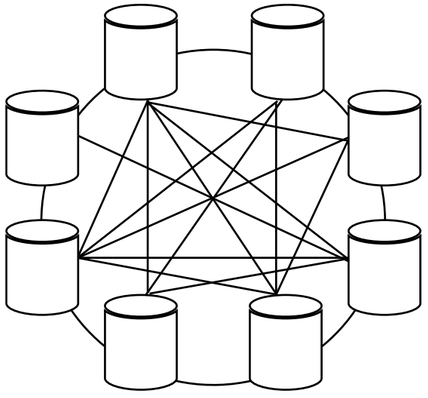
Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- Bewertungskriterien

Einführung in Architekturmuster

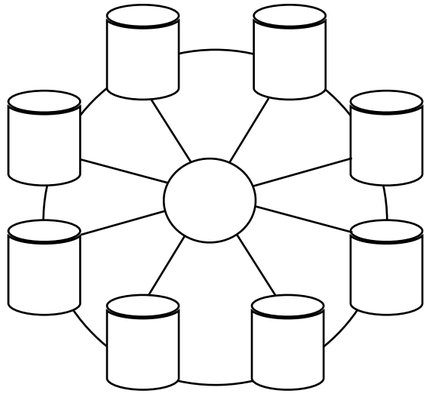
- Schichtenarchitekturen
- **Integrationsarchitekturen**
- Weitere Architekturmuster

Prinzipien von Integrationsarchitekturen



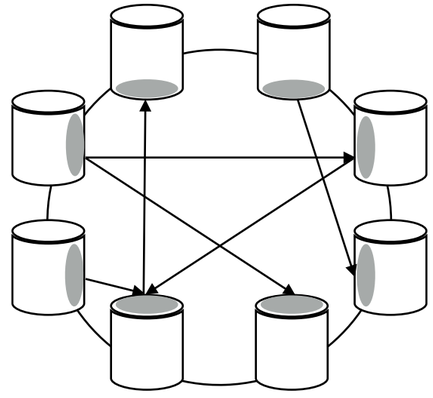
Punkt zu Punkt-Architektur (P2P)

- Individuelle Anpassung der Schnittstellen
- Dezentraler Aufbau der Systemlandschaft
- Feste Kopplung



Nabe-Speiche-Architektur (Hub and Spoke)

- Datenaustausch über eine zentrale Integrationsplattform



Service-orientierte-Architektur (SOA)

- Dezentraler Aufbau der Systemlandschaft
- Standardisierte und wiederverwendbare Schnittstellen
- Lose Kopplung von Systemen

Quelle: Schönherr 2005, S. 27

Punkt zu Punkt-Architektur (P2P)

Funktionsweise

- Verbindung einzelner Systeme zueinander mittels Schnittstellen

Vorteile

- Geringer Aufwand bei wenigen Systemen

Nachteile

- Komplexe Abhängigkeiten erschweren Weiterentwicklung und Anpassung
- Skalierbarkeit bei Einführung neuer Systeme nicht gegeben, da alle bisherigen Schnittstellen aktualisiert werden müssen

Quelle: Gessa 2007

Nabe-Speicher Architektur (Hub and Spoke)

Funktionsweise

- Zentraler Knotenpunkt leitet den gesamten Verkehr zu den System
- Jedes System tauscht Nachrichten nur mit dem zentralen Knotenpunkt unter Verwendung eines spezifischen Protokolls aus
- Direkte Kommunikation zwischen den Systemen ist nicht möglich

Vorteile

- Prozess der Extraktion, Transformation und Laden von Daten kann separiert werden
- Reduzierung der Verbindungen der einzelnen Systemen

Nachteile

- Zentraler Knotenpunkt als Single Point of Failure
- Zentraler Knotenpunkt als Performance-Engpass

Quelle: Gessa 2007

Service-orientierte-Architektur (SOA)

Funktionsweise

- Unterteilung der Systeme in Dienste, welche loose gekoppelt und eigenständig sind
- Enterprise-Service Bus als zentrale Kommunikationsplattform der Dienste untereinander

Vorteile

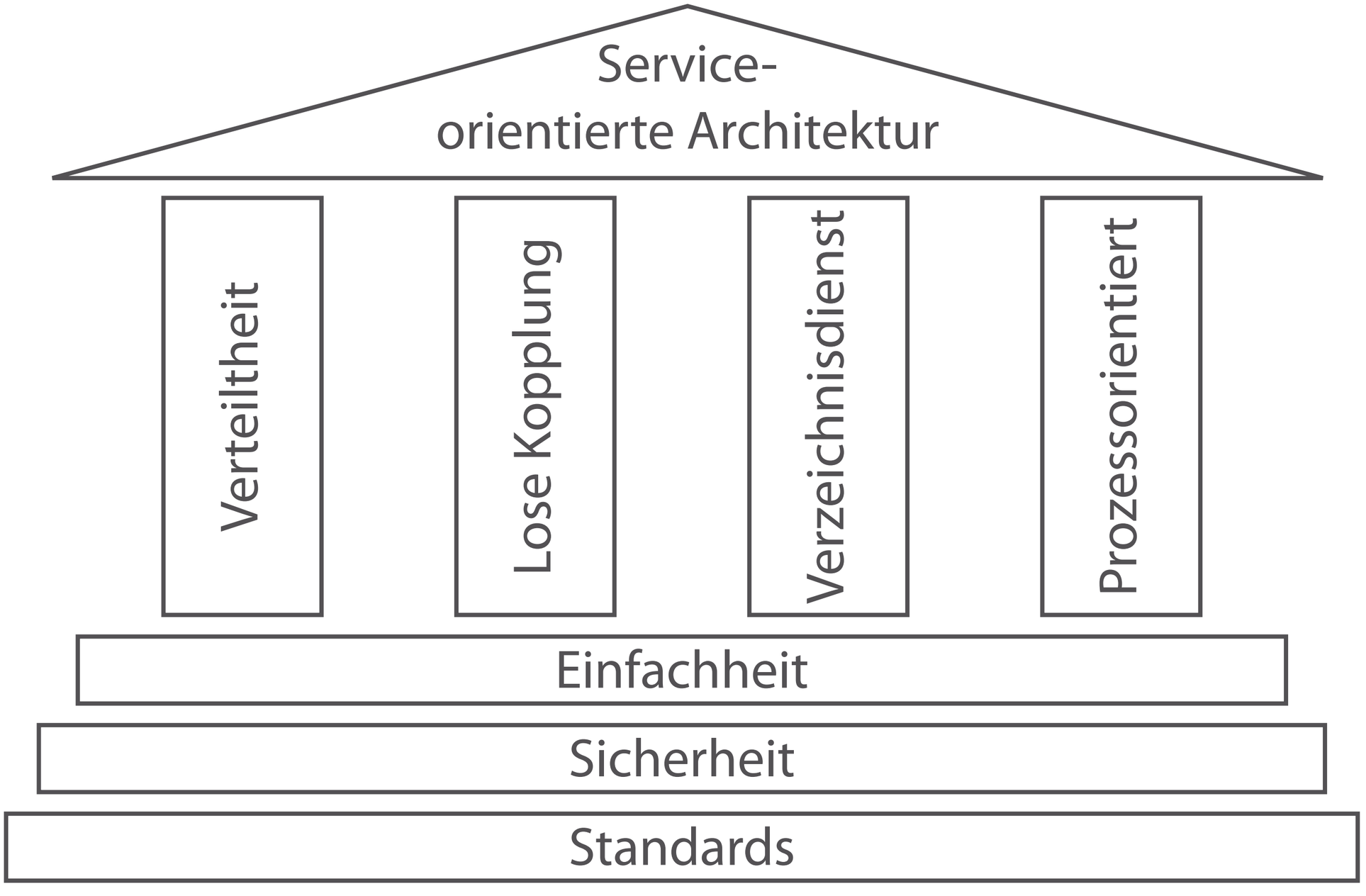
- Nutzung offener, bereits verbreiteter Standards
- Plattform- und sprachunabhängig
- Erhöhte Code-Wiederverwendung
- Bessere und schnellere Anpassungen an veränderte Rahmenbedingungen
- Einfache Entwicklung, Bereitstellung und Wartung von Anwendungen
- Einfache Integration von Services auf verschiedenen Granularitätsstufen

Nachteile

- Aufwändige Neugestaltung
- Dynamische Architektur und somit ggf. Instabilität
- Inhomogene IT Umgebung
- Keine Echtzeit
- Zusammenspiel mit externen Services nicht unbedingt nötig

Quelle: Gessa 2007

SOA Tempel



Quelle: Melzer 2010

Grundlegende Merkmale einer SOA

Lose Kopplung

- Dynamisches Finden und Einbinden von Diensten
- Bindung zur Laufzeit

Schnittstellen zur Kommunikation

- Verwendung von (möglichst) offenen Standards fördert die Akzeptanz
- Maschinenlesbar

Dienstsuchverzeichnis

- Ermöglicht das Finden relevanter Dienste
- Zugriff auf Instanzen des Diensteverzeichnisses
- Wissen über die Kategorie in welcher der Dienst eingebunden ist
- Anmeldung des Dienstes im Verzeichnis

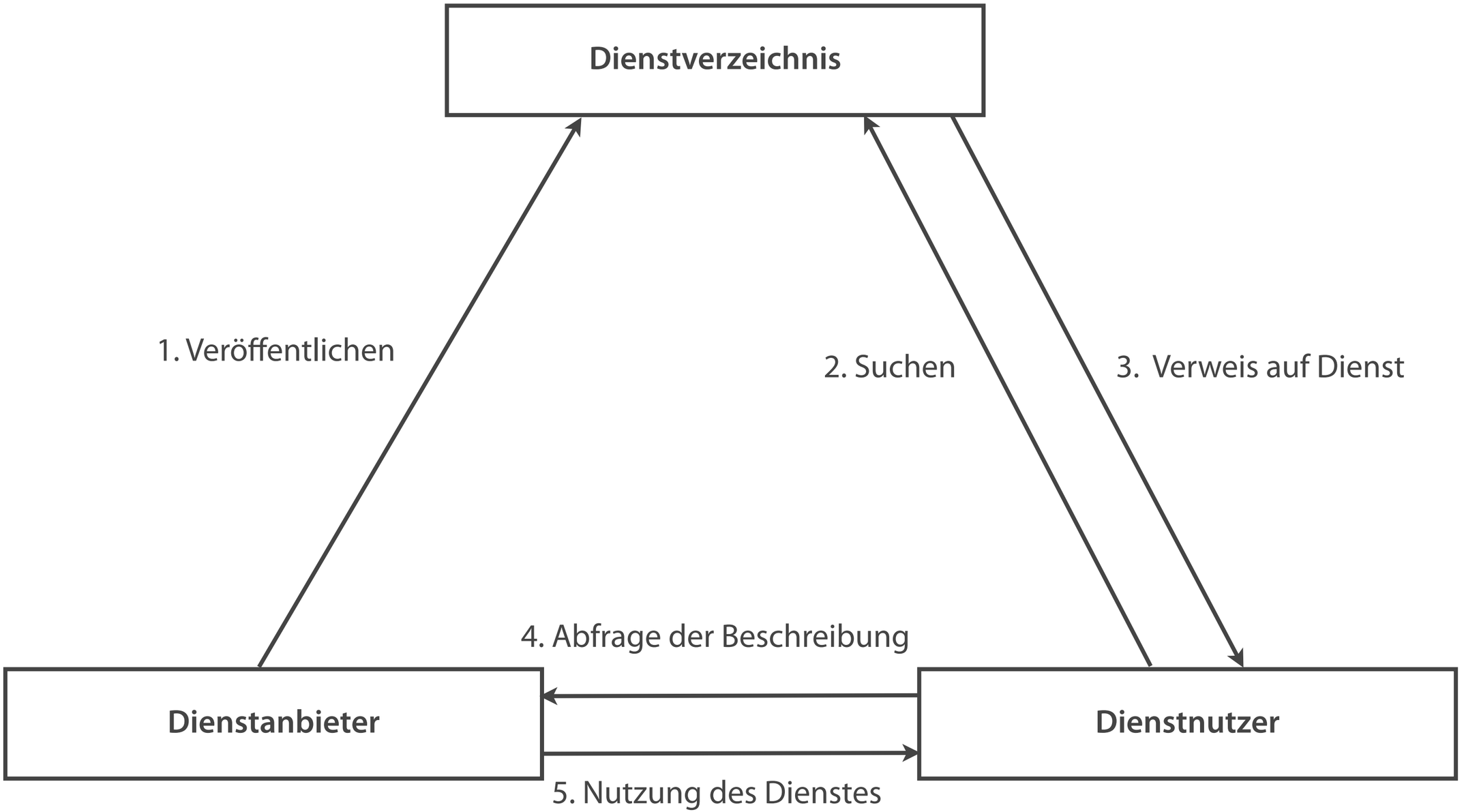
Wiederverwendung

- Gekapselte Dienste
- Mehrfach verwendbar in verschiedenen Umgebungen ohne Aufwand

Komponentenorientierung ist durch die Trennung von Schnittstelle und Implementierung gegeben

Quelle: Melzer 2010

Grundprinzipien einer Service-orientierten Architektur (SOA)



Quelle: Melzer 2010

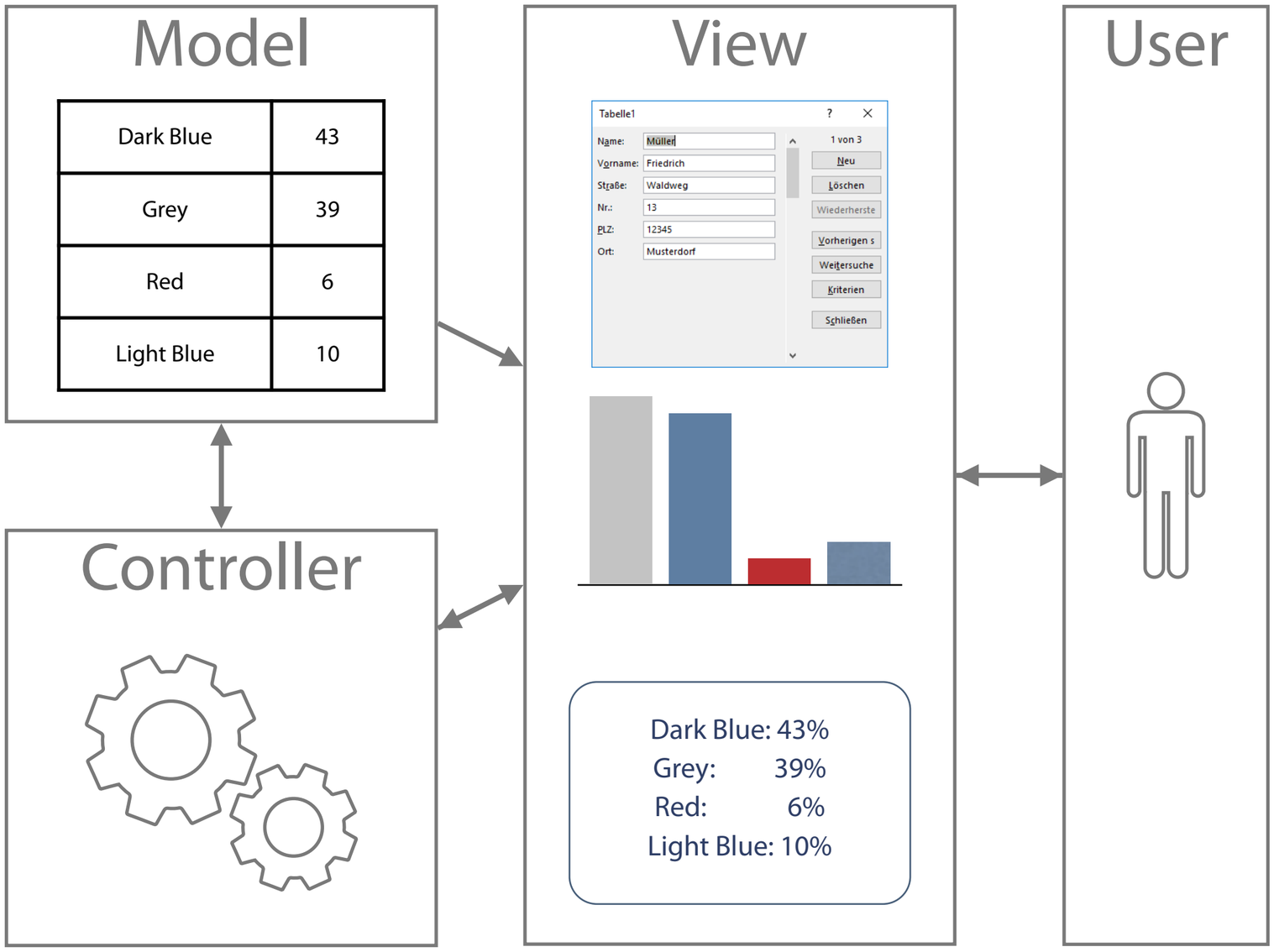
Einführung von Software-Architekturen

- Aufbau von Softwarearchitekturen
- Einführung in die Model Driven Architecture
- Bewertungskriterien

Einführung in Architekturmuster

- Schichtenarchitekturen
- Integrationsarchitekturen
- **Weitere Architekturmuster**

Model-View-Controller (MVC)-Muster



Aufteilung des Systems in drei Bereiche

- Model: Kernfunktionalität und Daten
- View: Anzeige der Informationen
- Controller: Verarbeitung der Nutzereingaben

} *Benutzungs-schnittstelle*

Dieses Muster ist für interaktive Systeme anwendbar.

Quelle: Buschmann 1996, S. 125

Zusammenfassung
Schichtenmuster, MVC und SOA

Schichtenmuster

- Dient der Strukturierung durch Dekomposition
- Bereitstellung eines Services für die obere Schicht
- Nutzung von Services der unteren Schicht
- Schichtenmuster ist etablierter Standard für betriebliche Anwendungssysteme

Service-orientierte Architektur

- Trennung von Schnittstelle und Implementierung
- Dynamisches Finden und Einbinden von Diensten durch lose Kopplung
- Offene Standards, maschinenlesbar
- Wiederverwendbarkeit durch Kapselung
- Dienstsuchverzeichnis

Model-View-Controler

- Anwendungsbereich sind interaktive Systeme
- Model für Daten und Funktionalität
- View zur Darstellung der Informationen
- Controller zum Abfangen und weiterleiten der Benutzereingaben
- Wichtig für Web-Architekturen, deren Funktion in der Interaktion mit dem Benutzer liegt.

Literatur

Bison AG. (2020). Website, [online] https://www.bison-group.com/downloads?tx_bisondownload_download%5Baction%5D=list&tx_bisondownload_download%5Bcategory%5D=44&tx_bisondownload_download%5Bcontroller%5D=Download&cHash=60a94ca5b17af3a127ee1b8cad410eb4 (Abgerufen am 31.07.2020).

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996). Pattern-oriented Software Architecture. A System of Pattern. 1. Auflage. West Sussex: John Wiley & Sons.

Gessa, N. (2007). An ontology-based approach to define and manage B2B interoperability (Doktorarbeit, alma).

Getty Museum (2022). The getty center turns 15. [online]. https://blogs.getty.edu/iris/the-getty-center-turns-15/ (Abgerufen am 06.12.2022).

Hasselbring, W. (2006): Aktuelles Schlagwort - Software-Architektur, in: Informatik Spektrum (29) 1/2006. Springer Verlag, Heidelberg 2006.

Hruschka, P., und Starke, G. (2006). Praktische Architekturdokumentation: Wie wenig ist genau richtig. OBJEKTspektrum, 1, 53-57.

Kempa, Martin; Mann, Zoltán Ádám: Model Driven Architecture. In: Informatik Sprektrum August 2005

Kruchten, P. B. (1995). The 4+ 1 view model of architecture. IEEE software, 12(6), S. 42-50.

Melzer, I. (2010) Serviceorientierte Architekturen mit Web Services - Konzepte, Standards, Praxis. München

Object Management Group 2022a. MOF: Object Management Group, Meta-Object Facility, [online] http://www.omg.org/mof/, (Abgerufen am 06.12.2022).

Object Management Group 2022b. MDA: Object Management Group, Model Driven Architecture. [online] http://www.omg.org/mda/ .(Abgerufen am 06.12.2022).

Schönherr, M. (2005). Enterprise Application Integration (EAI und Middleware). Grundlagenarchitekturen und Auswahlkriterien. ERP Management 1/2005, Heft 1, S. 25-29.

Shaw, M., Garlan, D. (1996). Software architecture. Perspectives on an emerging discipline. 101. Ausgabe. New York: Prentice-Hall.

Starke, Gernot: Effektive Software-Architekturen. Hanser, 2017.

Vogel, O. et al. (2005): Software-Architektur, Grundlagen - Konzepte - Praxis, München 2005.

Vogel, O., Arnold, I., Chughtai, A., Ihler, E., Kehrer, T., Mehlig, U., Zdun, U. (2009). Software-Architektur. Grundlagen - Konzepte - Praxis. 2. Auflage. Heidelberg: Spektrum Akademischer Verlag.