Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

QUASAR
Architektur bei Capgemini

Uni Tübingen,
14.01.2015,
Holger Cermann, Christian Nicu

People matter, results count.

Hinweis: Forschung und Lehre leben vom Gedankenaustausch, dabei helfen klare Vereinbarungen:

Haftungsausschluss
Auch wenn diese Präsentation und die damit zusammenhängenden Ergebnisse nach bestem Wissen und sorgfältig erstellt wurden, übernimmt weder Capgemini, noch der/die Autoren, eine Haftung für deren Verwendung.

Bei Fragen wenden Sie sich bitte an:

Christian Nicu
Capgemini | Stuttgart

e-mail: christian.nicu@capgemini.com

Holger Cermann
Capgemini | Stuttgart

e-mail: holger.cermann@capgemini.com

www.de.capgemini.com

Holger Cermann

Managing Business Architect, Capgemini Stuttgart

Studium, Ausbildung

- Informatikstudium an der Uni Stuttgart (1998 – 2004)
- Vertiefung: Verteilte Systeme und SW-Engineering
- Nebenfach: BWL
- Professional Scrum Master

Capgemini

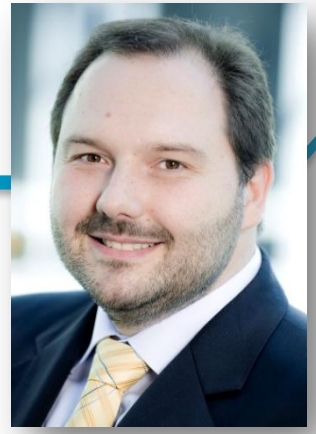
- Bei Capgemini (Custom Solution Development) seit 2004
- Beteiligung bei vielen Projekten im Daimler Vertrieb von Anforderungsanalyse & Spezifikation, über Konstruktion bis hin zur Umsetzung.
  - Sehr viel im Umfeld Global Ordering (GO) tätig gewesen
  - 2009-2012: Chefdesigner für GO und deren Projekte
  - Seit 2013: Beratung @ ConnectedVehicle

… und Privat

- Verheiratet, 1 kleines Kind
- Hobbys: Joggen, Mikrocontroller-Basteleien, SmartHome

Christian Nicu

Managing Solution Architect, Capgemini Stuttgart

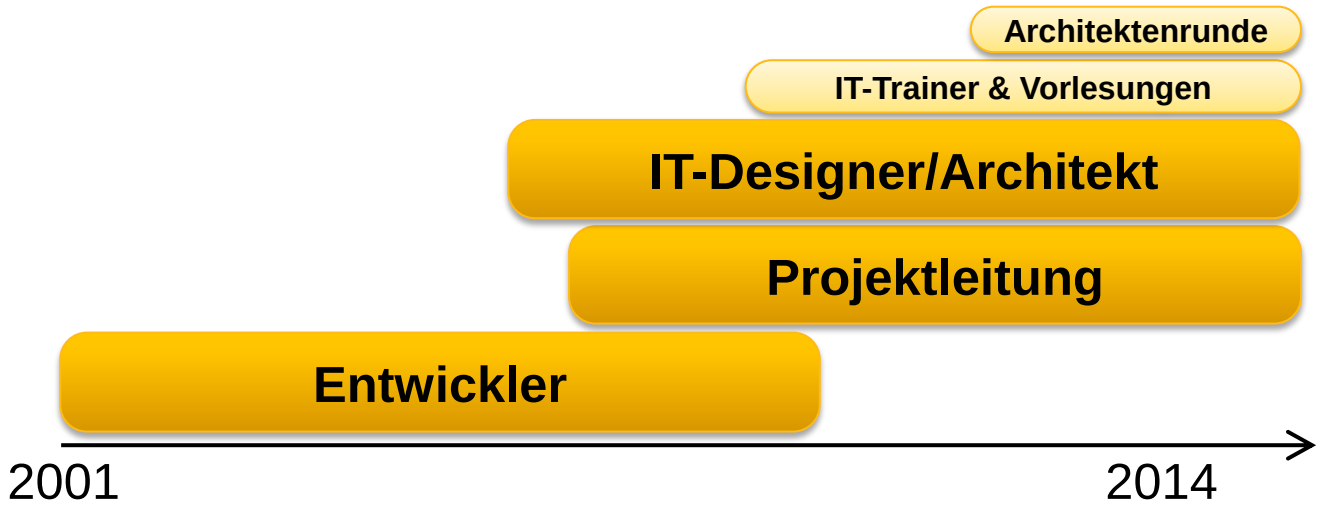


Wer bin ich?

- Individualsoftwareentwicklung
  im Bereich „Sales“, zertifizierter Scrum Master

- Studium: Informatik in Stuttgart
- Ich war erst irregeleitet zu einem anderen Arbeitgeber, bevor ich zur Capgemini kam

- 40 Jahre alt
- Nicht verheiratet, aber so ähnlich
- Keine Kinder



DAIMLER

Mercedes-Benz

schon immer für (aber in unterschiedlichen Projekten)

Java EE

IBM WebSphere

DB/2

PAI

Agenda

- Was ist Software-Architektur?
- Was sind Komponenten?
- Wie finde ich Komponenten?
- Wie viele Schichten dürfen es sein?
- Was sind Architektursichten?
- Abschluss!

Agenda

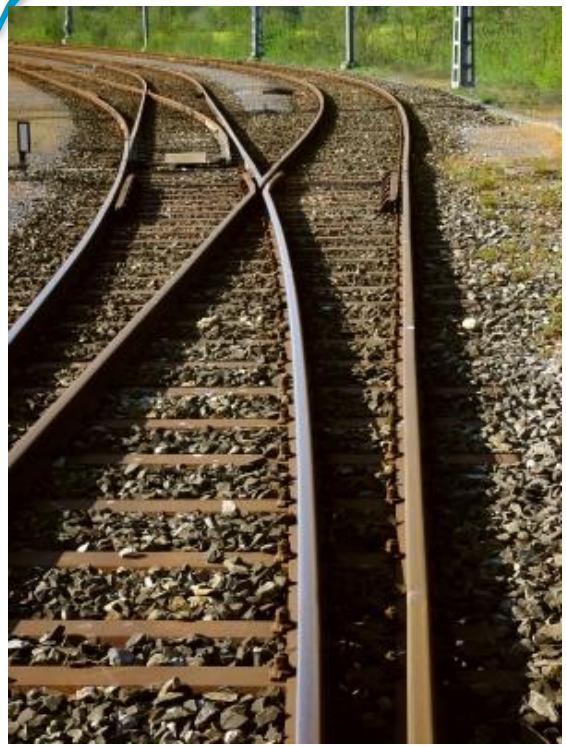
- **Was ist Software-Architektur?**
- Was sind Komponenten?
- Wie finde ich Komponenten?
- Wie viele Schichten dürfen es sein?
- Was sind Architektursichten?
- Abschluss!

Software-Architektur

Wie kommt man von hier…

…nach da?

Software Architektur setzt die Weichen in einem Projekt



- **Kreatives Denken:** Strukturieren der Anforderungen und diese dann auf konkrete Funktionen und Komponenten abbilden. So schwindet Komplexität!
- Trägt bei zum **Erreichen von Nichtfunktionalen Anforderungen**
- Bestimmt den **Rahmen des Projektes**:
  - Planung der Arbeitspakete
  - Teamstruktur
  - Testorganisation
- Bildet die **Basis zum Verständnis** des Systems

Wenn die Architektur nachträglich grundlegend modifiziert werden muss, wird dies sehr, sehr teuer!

Um was geht es bei Softwarearchitektur?

Spezifikation

- **WAS** soll das Softwaresystem leisten?
- **Außensicht** des Systems
- Funktionale und nicht-funktionale **Anforderungen**
- Zielgruppen: **Anwender**, Management, Architekten, Entwickler
- Modelle: **Anwendungsfälle**, Dialogentwürfe, fachliches Datenmodell

Konstruktion

- **WIE** soll das Softwaresystem die Anforderungen erfüllen?
- **Innensicht** des Systems,
- fachliche und technische **Struktur**
- Zielgruppen: **Entwickler**, Architekten, Betrieb, (Management, …)
- Modelle: **Sichten**, Komponenten-, Klassen-, Sequenz-, ER-Diagramme, …

Architektur: Alles wichtige, außer was das System eigentlich tut …

Quasar beinhaltet das gesammelte Wissen von Capgemini

Unterschiedliche Blickwinkel auf Architektur

Grob — Detaillierungsgrad — *Fein*

Anwendungs-landschaft
- Komplette Anwendungen
- Bebauungspläne
- Einordnung in Kategorien

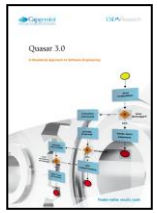
Systemdesign/ -architektur
- Schichten
- Komponenten
- Trennung von Fachlichkeit und Technik
- „Blutgruppen“

Komponenten-design
- Bestimmen der Schnittstellen
- Einsatz von Entwurfs-mustern
- Entitätsklassen

Methoden-design
- Bestimmen der Algorithmen
- Einsatz von Mustern
- Richtlinien

Fokus von Quasar Enterprise

Fokus von Quasar



Quality **S**oftware **Ar**chitecture
besteht aus
Regeln, Leitlinien und Prinzipien

http://www.de.capgemini.com/ressourcen/quasar-30

Die Software-Architektur muss wichtige Anforderungen von Betrieblichen Informationssystemen (BIS) erfüllen

Wichtige Qualitätsmerkmale

- **Größe und Komplexität**
- **Geforderte Lebensdauer**
- **Wandel während der Lebensdauer**
- **Wartbarkeit**
- **Betreibbarkeit**

Notwendige Eigenschaften der Software-Architektur

- Struktur: Teile und herrsche!
- Teilteams
- Stufen
- Verstehbar
- Änderbar
- Wiedererkennbar
- gleichförmige Lösungen
- konsistente Modellierung
- Regeln
- Konventionen
- Infrastruktur
- Austausch/Upgrade von Basissystemen (Middleware, Hardware, Datenbank)

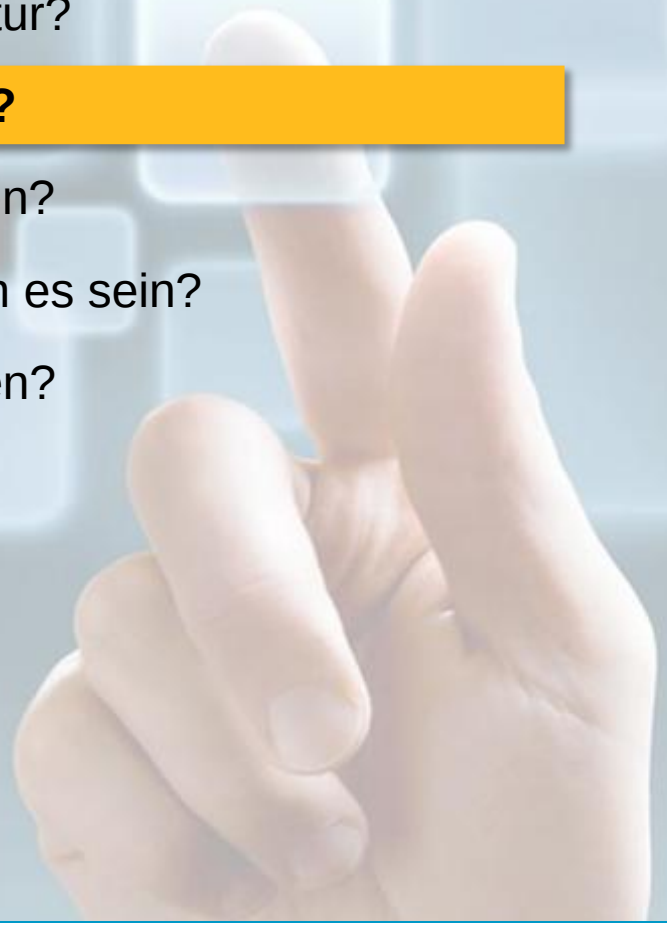
Unterstützung der ISO 9126 Qualitätsmerkmale durch die einzelnen Architekturprinzipien

Architekturprinzip:		Zuverlässigkeit	Effizienz	Änderbarkeit	Übertragbarkeit
P1	Trennung von Zuständigkeiten	++	+	++	++
P2	Minimierung von Abhängigkeiten	+		++	++
P3	Homogenität	+		++	
P4	Geheimnisprinzip	++	-	++	++
P5	Redundanzfreiheit	++	+	++	+
P6	Software-Kategorien	+	+	++	+
P7	Schichtenbildung	+	-	++	+
P8	Vertragsbasierter Entwurf	++	-	++	+
P9	Datenhoheit	++	-	++	+
P10	Wiederverwendung	+		+	+

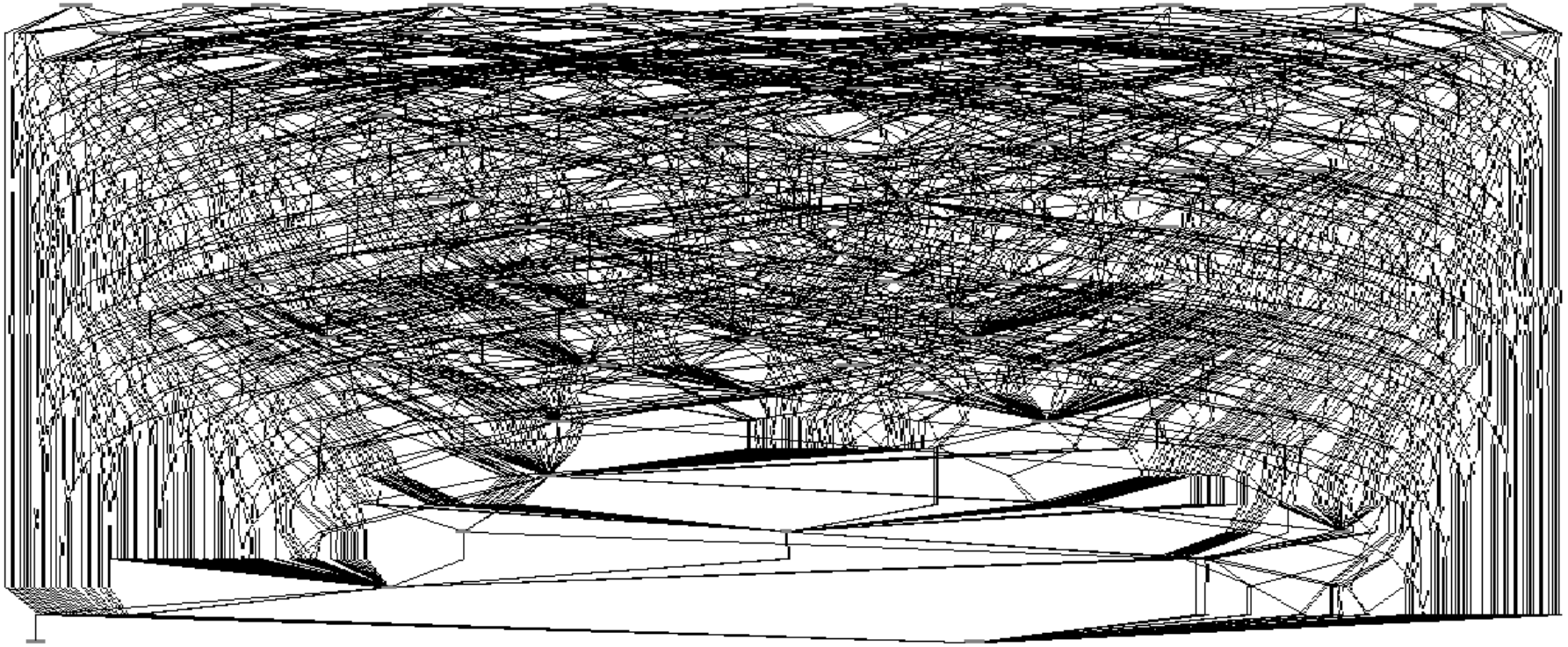
++	unterstützt sehr
+	unterstützt
	hat keinen Einfluss
-	hat eher negativen Einfluss

Agenda

- Was ist Software-Architektur?
- **Was sind Komponenten?**
- Wie finde ich Komponenten?
- Wie viele Schichten dürfen es sein?
- Was sind Architektursichten?
- Abschluss!



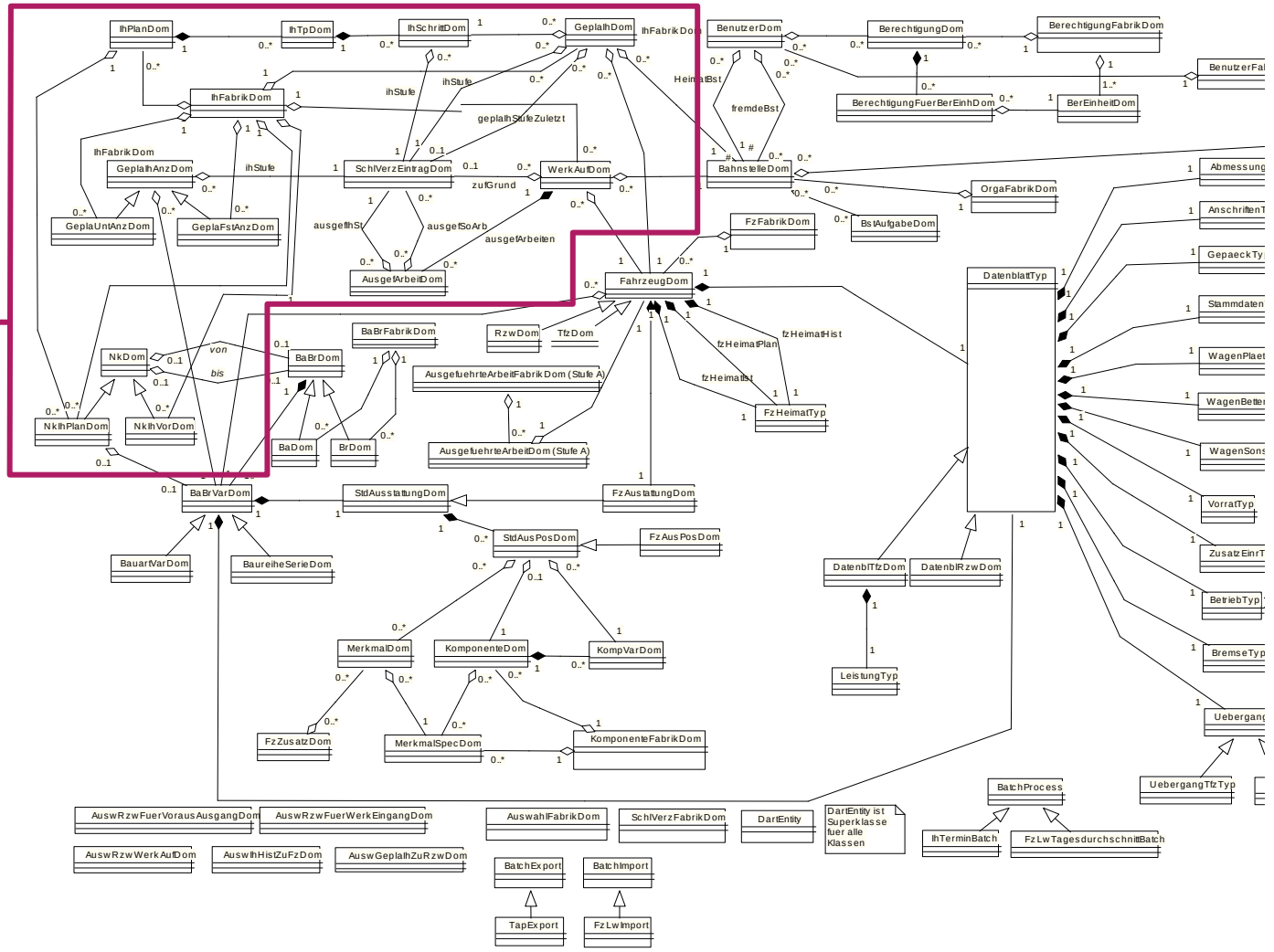
Abhängigkeitsgraph im OO-Betriebssystem Taligent



Erich Gamma, "100 OO Frameworks, Pitfalls and Lessons Learned", 1997

Ich muss eine neue Funktionalität in die Bestandsführung einbauen! Welche Klassen muss ich dafür anfassen?

Dieser Teil ist von der Änderung betroffen



Schlussfolgerung:

Klassen sind als Einheit des Software-Entwurfs viel zu klein und daher ungeeignet!

Lokalität von Änderungen macht Software-Systeme beherrschbarer

Lokalitätsprinzip

Anwendung

Änderung an einem Modul

Änderung beeinflusst andere Module

Komplett-Auslieferung der Anwendung

Komponenten-basierte Anwendung

Änderung an einer Komponente

Keine anderen Komponenten beeinflusst

Auslieferung der geänderten Komponente

Quelle: Business Component Factory; Herzum, Sims

Sechs Merkmale einer Komponente (Definition aus Quasar)

1. Klare Definition (inkl. Semantik) der angebotenen Dienste (**Export-Schnittstellen**)

2. Klare Definition der Abhängigkeiten von Diensten anderer Komponenten (**Import-Schnittstellen**). Sie ist erst lauffähig, wenn alle importierten Schnittstellen zur Verfügung stehen, und dies ist Aufgabe der Konfiguration

3. Jede Komponente **versteckt ihre Implementierung** und kann daher durch eine andere Komponente ersetzt werden, die dieselbe Schnittstelle exportiert

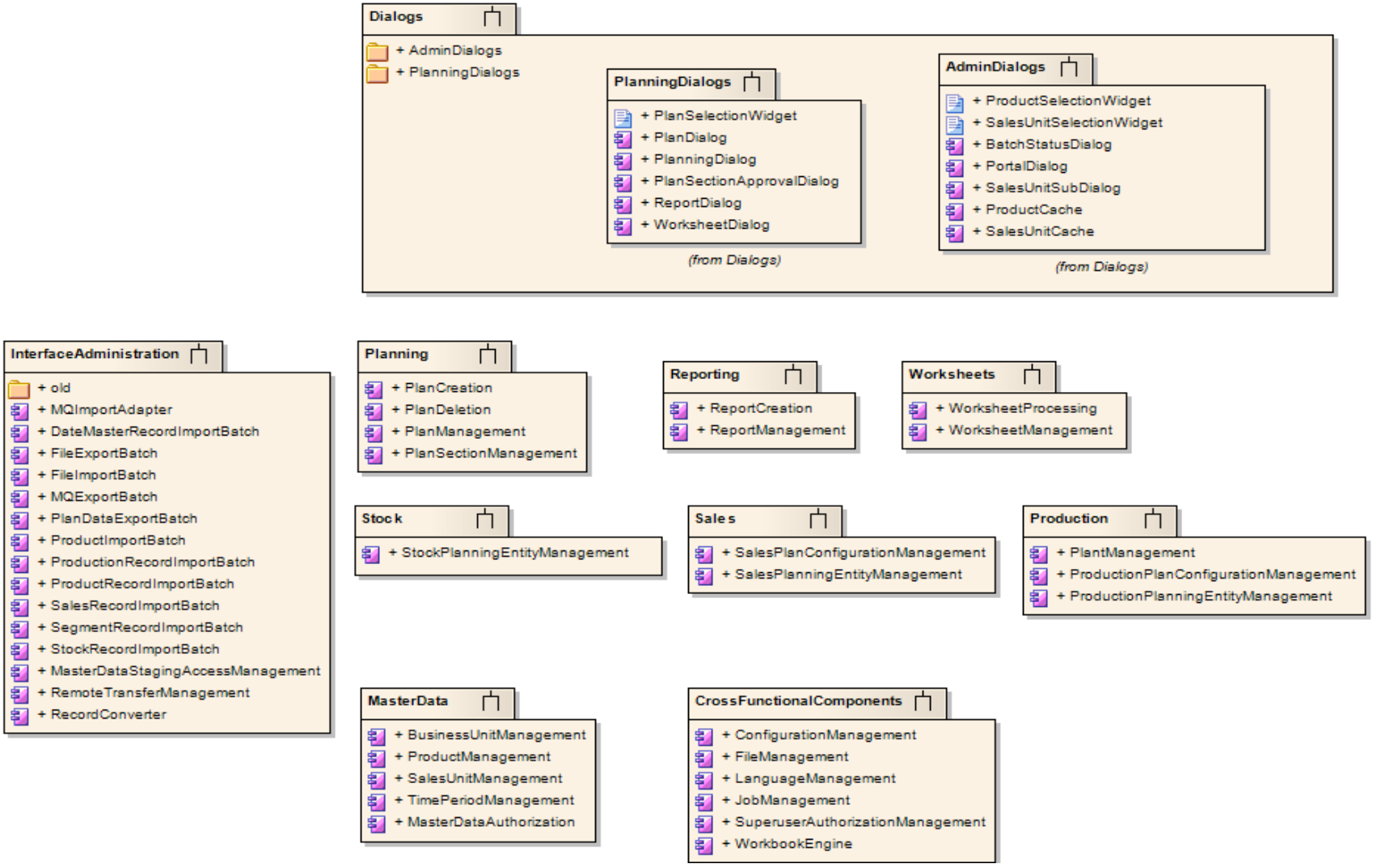
4. Geeignet als **Einheit der Wiederverwendung**, denn eine Komponente kennt nicht die Umgebung, in der sie läuft, sondern macht darüber nur minimale Annahmen

5. **Komponenten können andere Komponenten enthalten**:
Man kann neue Komponenten aus vorhandenen Komponenten zusammensetzen (oder komponieren), und dies über beliebig viele Stufen

6. Die Komponente die wesentliche **Einheit des Entwurfs, der Implementierung und damit der Planung**

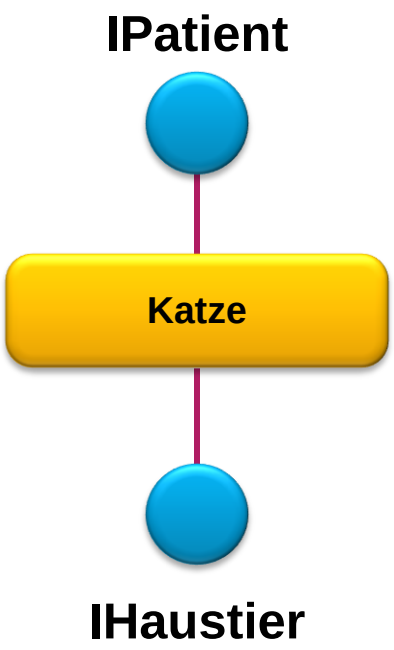
"Vergleichbar mit einem Sachbearbeiter, der seinen Abschnitt des Formulars ausfüllt."
(vgl. Denert, Software-Engineering, 1992)

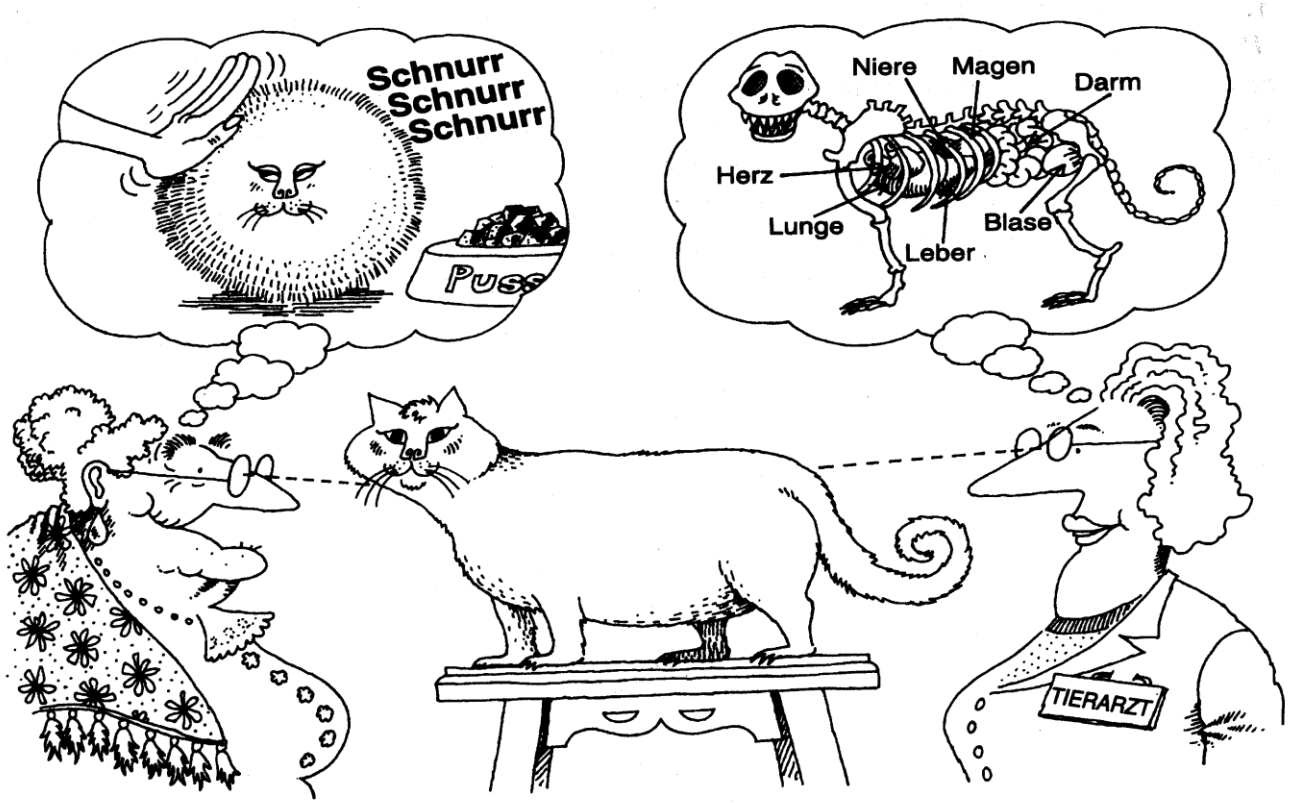
Beispiel: eine Vielzahl von Komponenten, die in Subsystemen fachlich gruppiert sind.



Schnittstellen definieren Sichten auf Objekte

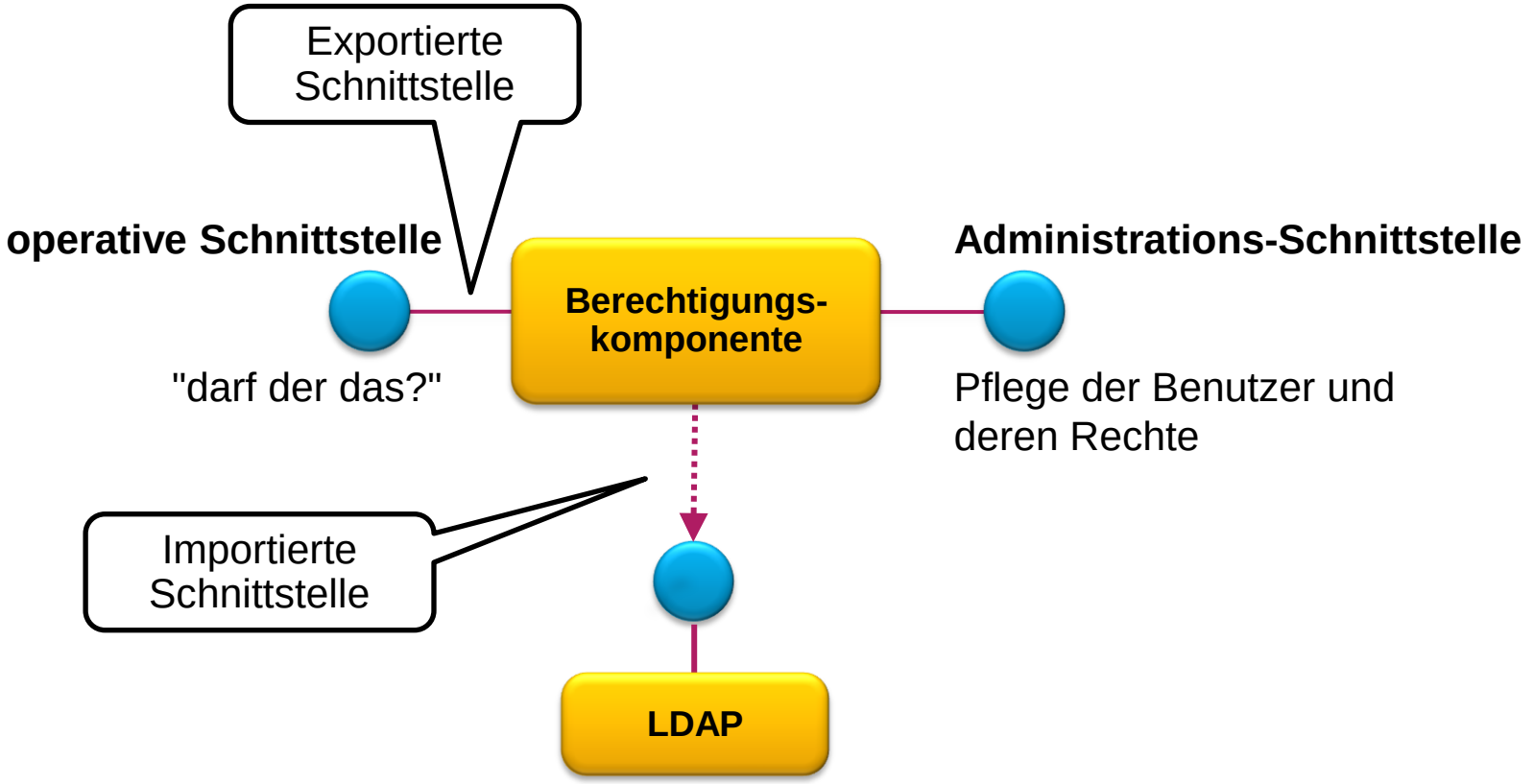
Meine Katze ist auch objektorientiert [Roger King]






```
class Katze implements
IPatient, IHaustier {…}
```

Die Schnittstellen bieten verschiedene Sichten auf eine Komponente

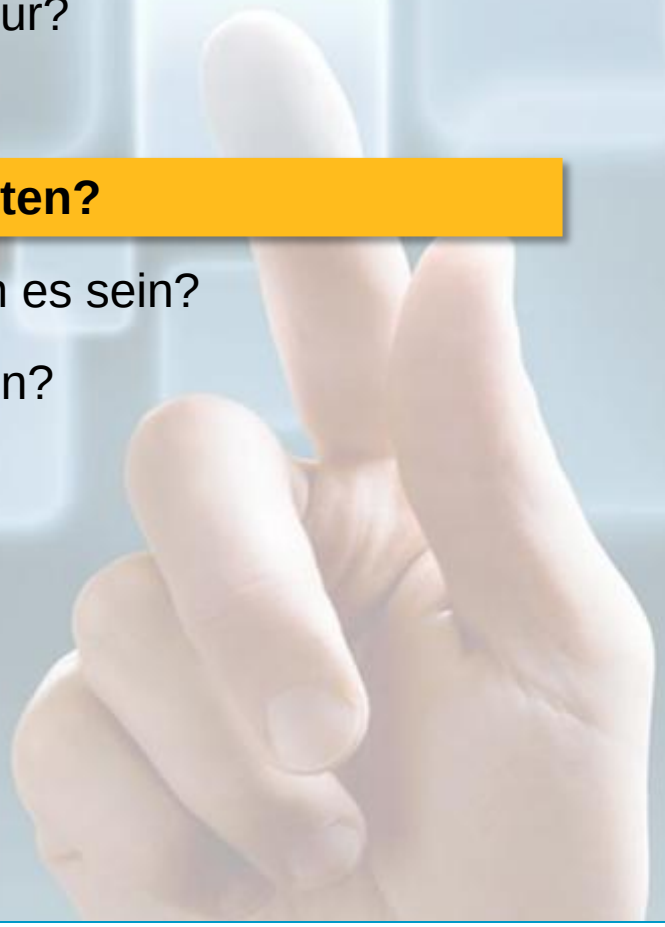


Eine Komponente wird nur über ihre Schnittstellen benutzt!

Die exportierten Schnittstellen einer Komponente definieren die Komponente

Agenda

- Was ist Software-Architektur?
- Was sind Komponenten?
- **Wie finde ich Komponenten?**
- Wie viele Schichten dürfen es sein?
- Was sind Architektursichten?
- Abschluss!



Beim Finden und Schneiden von Komponenten geht es um die Trennung von Zuständigkeiten und die Kontrolle der Abhängigkeiten

Maximen zum Komponentenschnitt

- **Trenne Zuständigkeiten:**
  Nutze die Softwarekategorien als Leitlinie für die Zerlegung in Komponenten
- **Kontrolliere die Abhängigkeiten:**
  Achte auf hohen inneren Zusammenhang und lose Kopplung nach außen

Softwarekategorien helfen beim Untergliedern des Systems

Idee

Wissensgebiete

- Unterteile die Software nach erforderlichen “Fachwissen”
- “Fachwissen” = ein Aspekt (technisch, fachlich) des Problems
- Wissensgebiet = Softwarekategorie

Iterative Verfeinerung

- Schrittweise Verfeinerung der Kategorien
- Stärkere Verfeinerung = enger gestecktes Wissensgebiet

Strukturierungsmittel

- Kategorien aus Verfeinerung dienen als Strukturierung der Komponenten
- Komponenten sollen Kategorieren nicht vermischen

Vorgehen: Erst die Softwarekategorien finden und aus diesen dann Komponenten des Systems ableiten

Das Trennen von Zuständigkeiten ist ein wichtiges Prinzip von Quasar, als Analogie dienen die Blutgruppen

Kategorien (Blutgruppen)	Das heißt…	Kombinationen
O-Software	**Unabhängig** von Fachlichkeit und Technik; Ideal **wiederverwendbar**; **Beispiel:** Klassenbibliothek für Container	**A + 0 = A** **T + 0 = T** **A + T = AT**
A-Software	Bestimmt durch die **Fachlichkeit**; **Unabhängig von Technik**; Meist der größte Teil des Systems; **Beispiel:** "Mitarbeiter", "Buchung"	
T-Software	**Unabhängig** von der **Fachlichkeit**; **Technische Komponenten** – gut wiederverwendbar **Beispiel:** Zugriffsschicht auf Datenbank	
~~**AT-Software**~~	Vermischt **Technik und Fachlichkeit**; Zu **vermeiden**: schwer zu warten; widersetzt sich Änderungen; Wiederverwendung sehr schwierig!	

Der Komponentenschnitt soll die Zuständigkeiten definieren und die Abhängigkeiten minimieren

Ziele

- Trennung technischer Dienste
- Eindeutige Verantwortlichkeiten/ Zuständigkeiten
- Parallele Entwicklung
- Abschottung Nachbarsysteme

Leitlinien

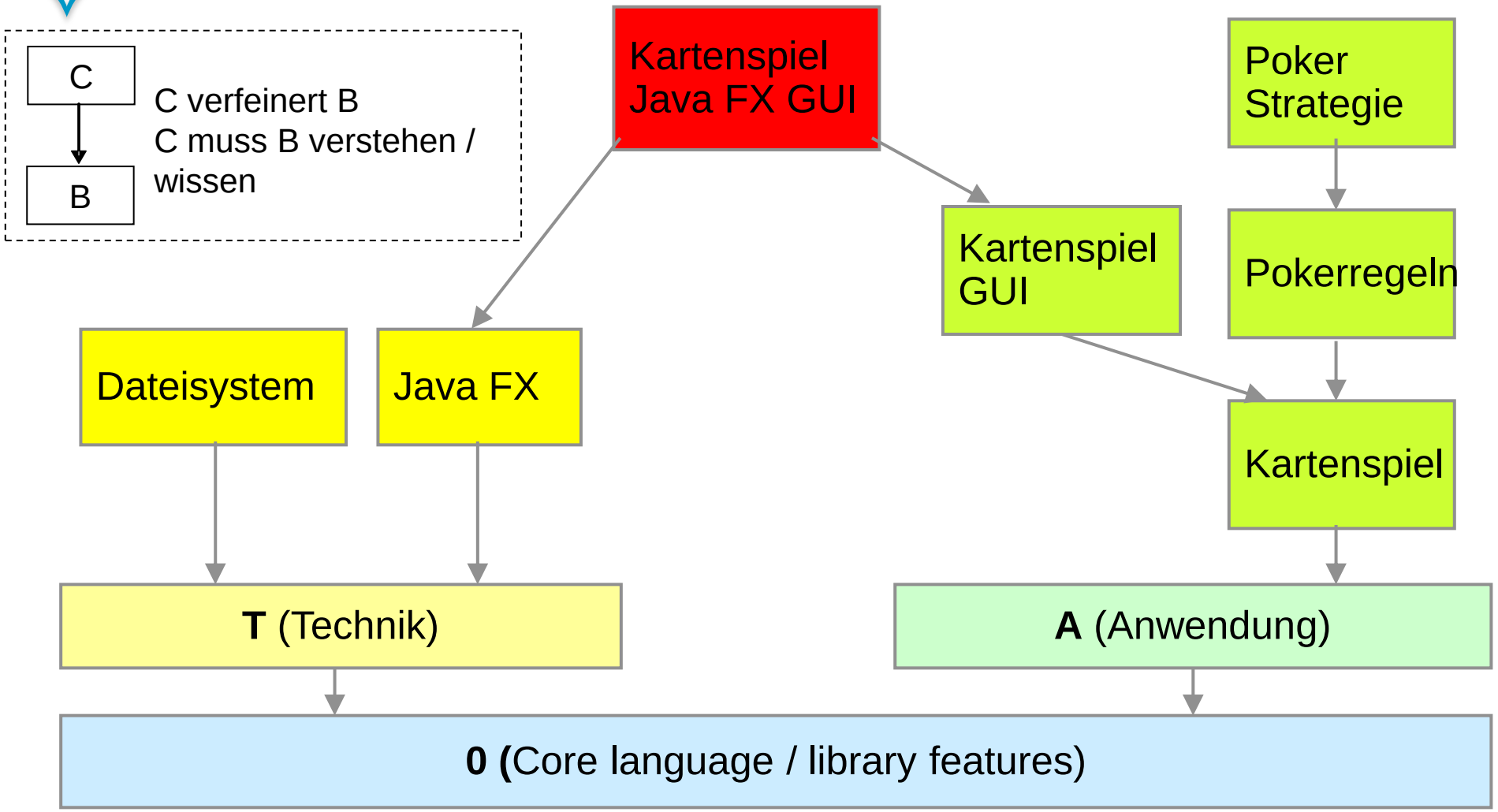
- Trennung von Fachlichkeit (A) und Technik (T)
- Anwendungskern (A-Software) ist technikunabhängig
- T-Abhängigkeiten kapseln

- Mehr als eine Zuständigkeit ist schlecht
- Vermeide verteilte Zuständigkeiten
- Dokumentiere Zuständigkeiten (Was ist das Geheimnis der Komponente?)
- **Komponenten gehören zu genau einer Softwarekategorie**

- Schneide Komponenten so, dass sie getrennt entwickelt werden können
- Minimiere die Schnittstellen zwischen den Komponenten

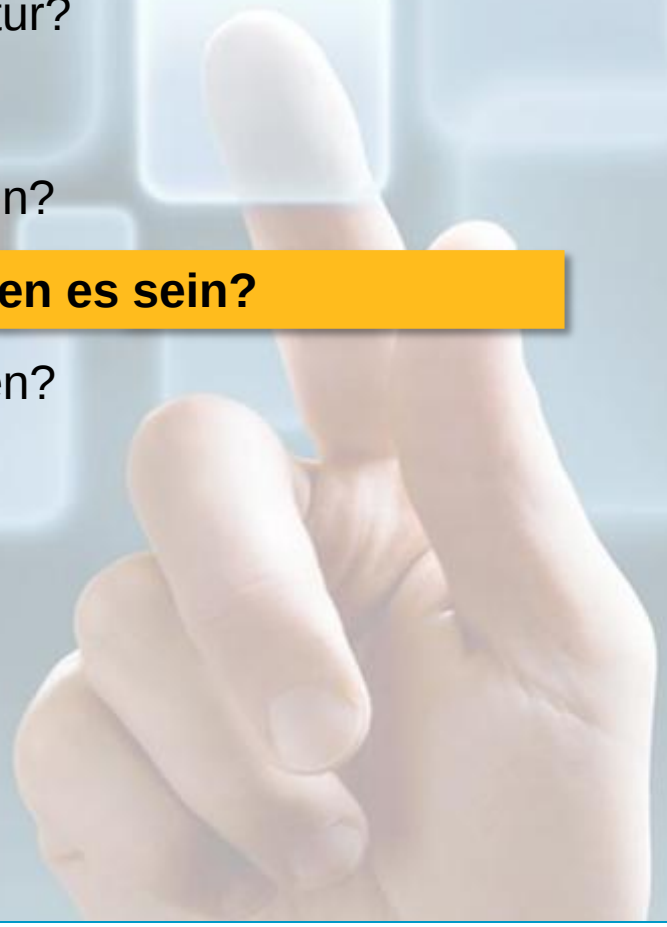
- Kapsle Nachbarsysteme
- Möglichst keine enge Kopplung

Beispiel: Softwarekategorien bei einem Pokerspiel

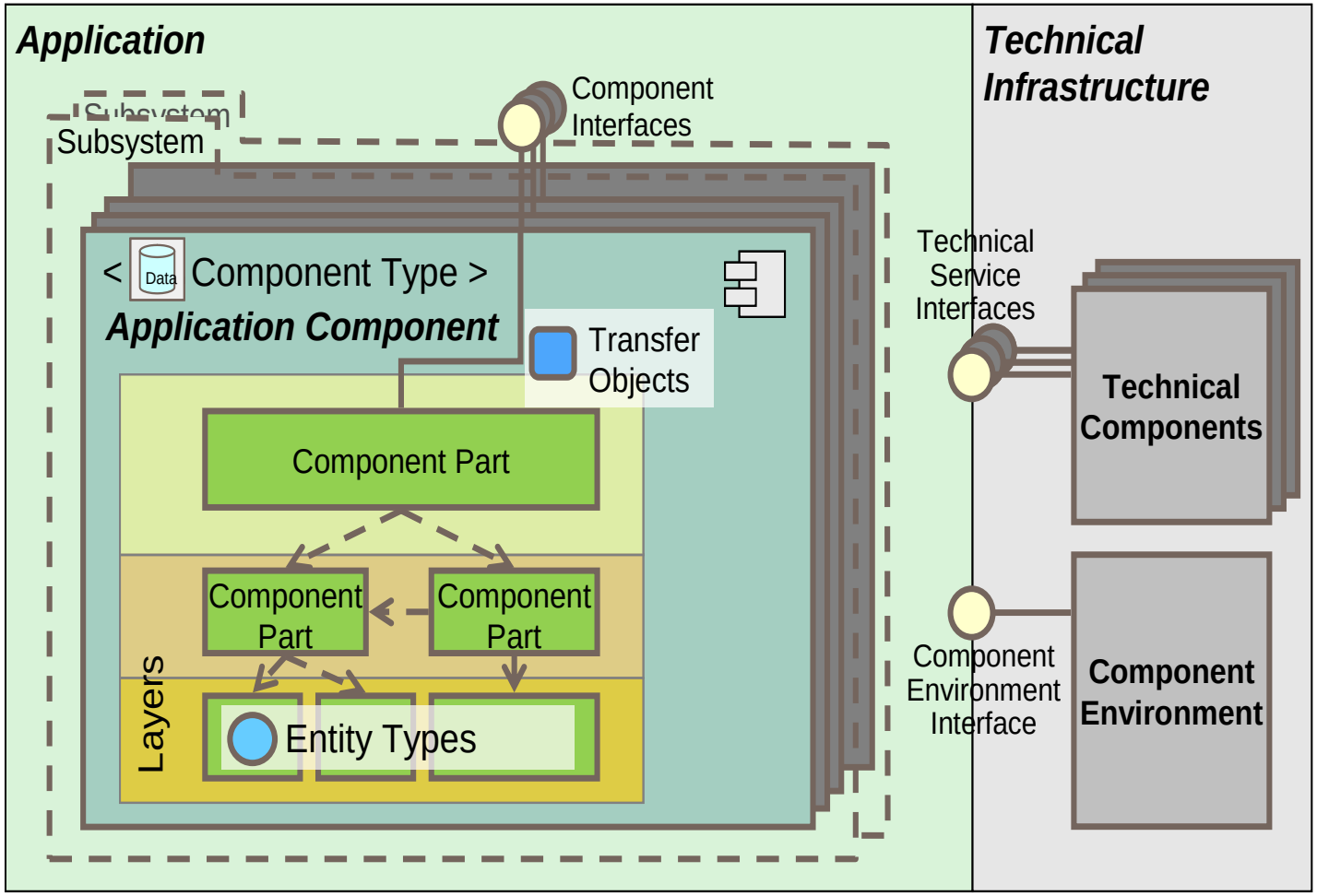


Agenda

- Was ist Software-Architektur?
- Was sind Komponenten?
- Wie finde ich Komponenten?
- **Wie viele Schichten dürfen es sein?**
- Was sind Architektursichten?
- Abschluss!



Das Konstruktionsmodell (schematisch)



Das **Konstruktionsmodell** dient dazu, die Beziehungen zwischen den Artefakten der Konstruktion aufzuzeigen und ein konsistentes Begriffsgebäude für die Konstruktion aufzubauen

Das Konstruktionsmodell

Das **Konstruktionsmodell** dient dazu, die Beziehungen zwischen den Artefakten der Konstruktion aufzuzeigen und ein konsistentes Begriffsgebäude für die Konstruktion aufzubauen.

Die **Subsysteme** dienen der (groben) Strukturierung der Fachlichkeit. Subsysteme werden selbst nicht implementiert, sie gruppieren nur Anwendungskomponenten, die die gesamte Fachlichkeit umsetzen. Jede Schnittstelle, die ein Subsystem anbietet, wird von einer Anwendungskomponente des Subsystems bereitgestellt. Subsysteme sind in einem Projekt organisatorisch wichtig: Sie können zum Schneiden von Teams, oder als Basis für X-shoring dienen oder eine Zuordnung in der fachlichen Verantwortung kann sich nach Subsystemen richten.

Eine Anwendung besteht aus einem oder mehreren Subsystemen.

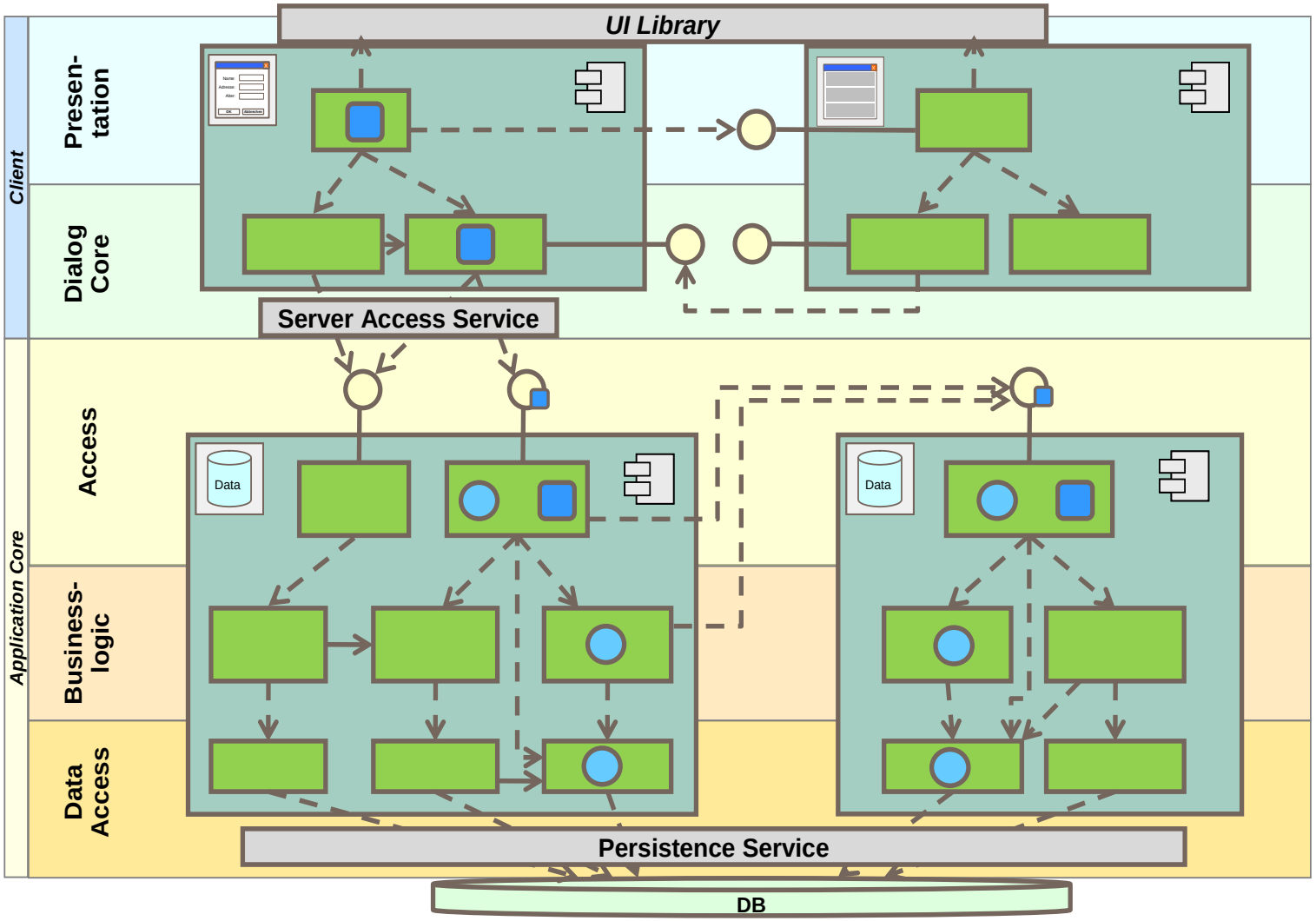
Ein **technischer Dienst** (technical Service) stellt einen technischen Aspekt der Infrastruktur möglichst technikneutral (wie in [Sie04] beschrieben) als Schnittstelle zur Verfügung.

Beispiele für Technische Dienste sind:

- Persistenzdienst (persistiert Objekte auf der Datenbank),
- Transaktionsdienst (erlaubt Zugriff auf die technische Transaktionssteuerung),
- Server-Zugangsdienst (erlaubt Clients den Aufruf von Schnittstellen von serverseitigen Komponenten).

Die **Komponentenumgebung** (Component Environment) verwaltet und erzeugt Instanzen von Anwendungskomponenten. Die Komponentenumgebung ist idealerweise 0-Code. Die Komponentenumgebung muss normalerweise konfiguriert werden, damit sie die Anwendungskomponenten erzeugen kann. Dies geschieht mit Hilfe der Komponentenkonfiguration.

Das Konstruktionsmodell (hineingezoomt)

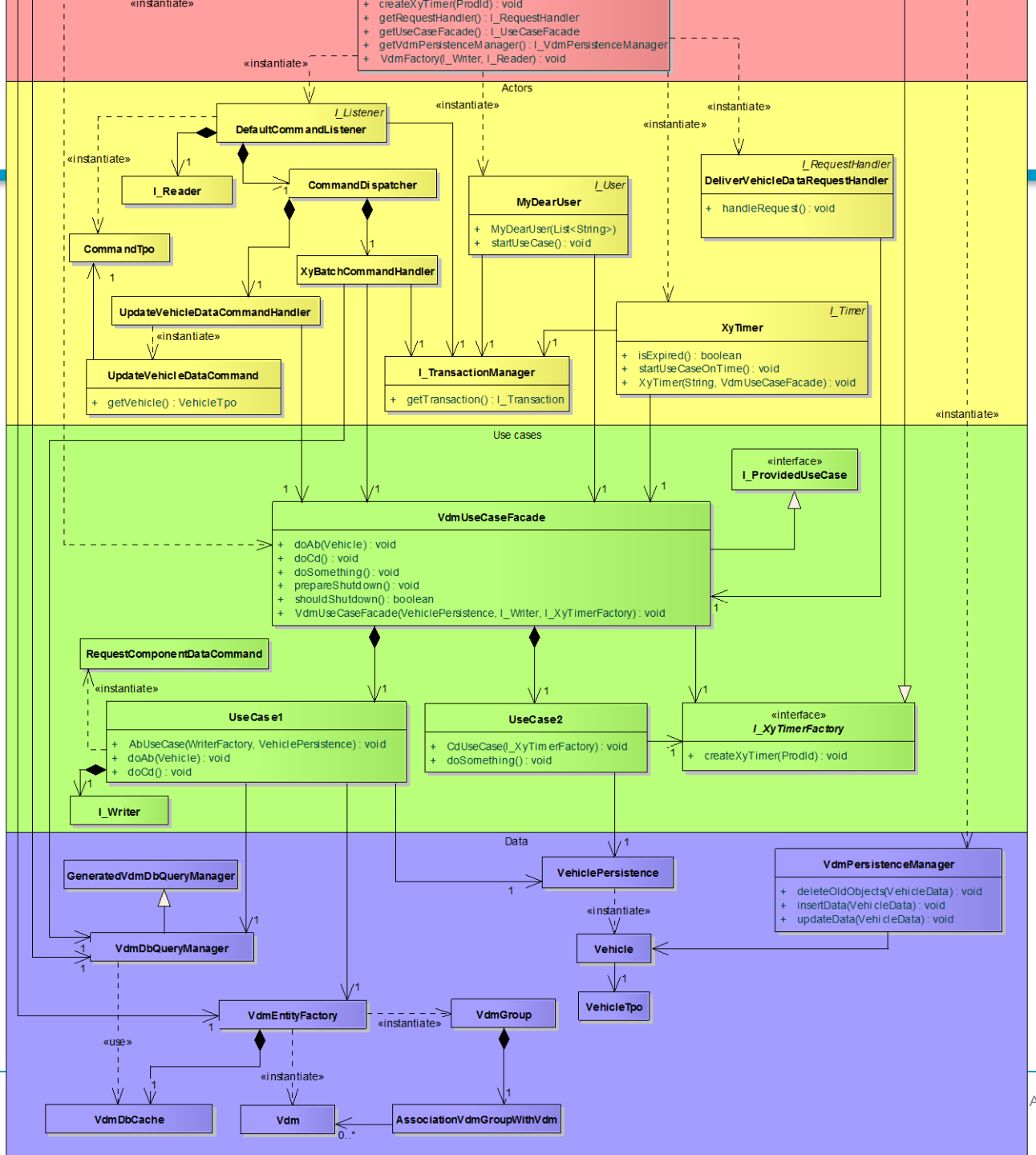


Die Schichten einer Anwendung kapseln die technischen Schritte innerhalb der fachlichen Aufgabe / Komponente

HANDOUT

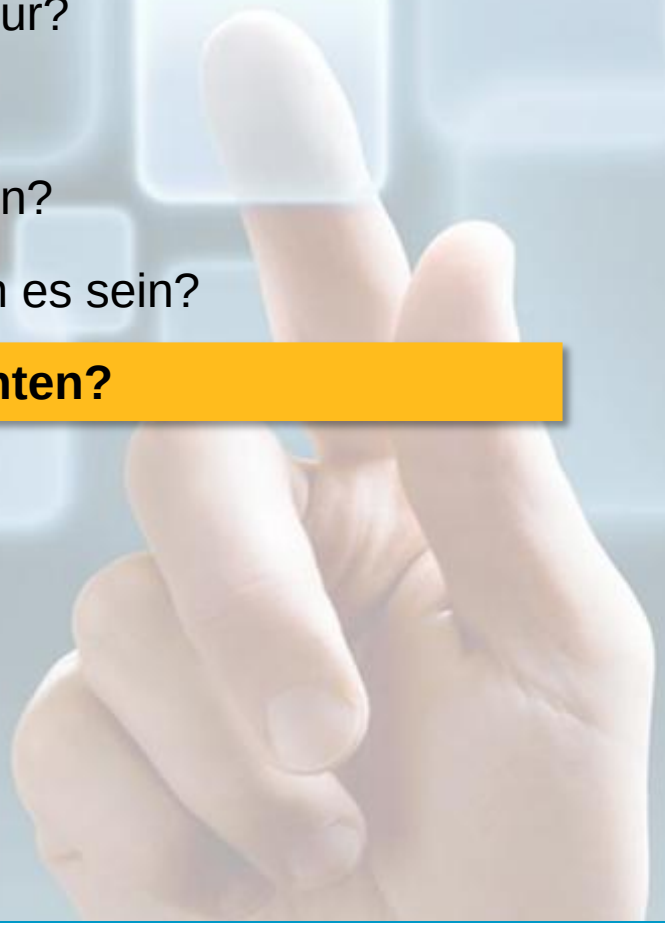
		Aufgabe	Kommunikations-Regeln
Client	**Präsen-tation** (Presentation)	■ Darstellung / Steuerung der Benutzeroberfläche ■ Eingabeunterstützung für den Benutzer ■ Transformation der Dialogkerndaten in Darstellung (bijektiv) ■ Hat Präsentationszustände (Sperren von GUI-Elementen u.ä.)	■ Komponenten-Schnittstellen derselben Schicht ■ Innerhalb der Komponente: – Zur Dialogkern-Schicht – Innerhalb der Schicht
	Dialog-kern (Dialog Core)	■ Halten der fachlichen Daten für Präsentation ■ Fachliche Steuerung (fachliche Aktionen, Workflows, fachlicher Dialog-Zustand) ■ Kapselung Server-Aufrufe	■ Komponenten-Schnittstellen derselben Schicht und von Server-Komponenten (Zugangsschicht) ■ Innerhalb der Komponenten zu Komponenten-Teilen derselben Schicht
Server	**Zugang** (Access)	■ Bietet serviceorientierte Komponentenschnittstellen für den Client und Aufruf außerhalb der Domäne (insbesondere für den Client) an ■ Prüft Eingabedaten auf Konsistenz und Gültigkeit ■ Konvertiert Transfer Objects in Entitätsobjekte ■ Ruft die Business-Logik mit den Entitätstypen auf ■ Technisch: Transaktionsklammer wird hier gesetzt / bestimmt	■ Komponenten-Schnittstellen derselben Schicht ■ Innerhalb der Komponente: – zur Businesslogik-Schicht – zur Zugriffsschicht (aufl. von ET‘s) – Innerhalb der Schicht?
	Business-logik (Business)	■ Implementiert die fachliche Logik (auch Entitätsübergreifend) ■ Bietet evtl. objektorientierte Schnittstellen innerhalb der Domäne an ■ Schnittstellen enthalten Entitätsobjekte	■ Komponenten-Schnittstellen der Zugangs- und Business-Schicht (nur innerhalb einer Domäne) ■ Innerhalb der Komponente: – Zur Zugriffsschicht – Innerhalb der Schicht
	Zugriffs-schicht (Entity)	■ Persistiert Entitätsobjekte auf der DB oder in Daten o.ä. ■ Definiert benötigt Selects u.ä. für die DB ■ Hat Hoheit über bestimmte Entitäten ■ Schnittstellen enthalten Entitätsobjekte	■ Innerhalb der Komponente zu Komponententeilen derselben Schicht

Beispiel für eine detaillierte Konstruktion mit **Schichten**



Agenda

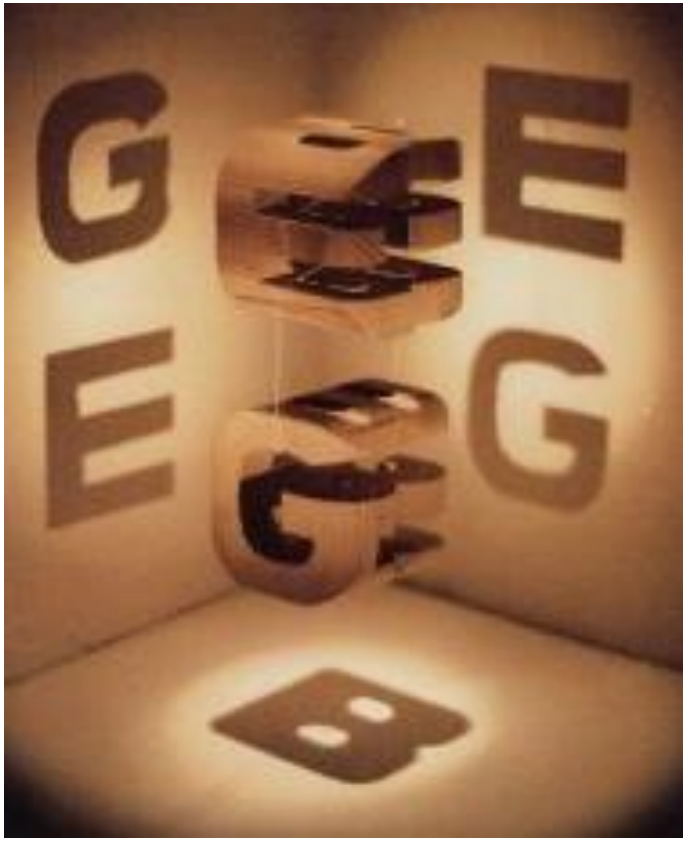
- Was ist Software-Architektur?
- Was sind Komponenten?
- Wie finde ich Komponenten?
- Wie viele Schichten dürfen es sein?
- **Was sind Architektursichten?**
- Abschluss!



Die Architektursichten sind elementare Bausteine, um die Software-Architektur darzustellen

- Die **Software-Architektur** definiert den Aufbau und die Struktur eines Informationssystems:
  - die **statischen Aspekte**, wie z.B. Komponenten, ihre Schnittstellen und Beziehungen untereinander
  - die **dynamischen Aspekte**, wie z.B. die Kommunikation zwischen den Komponenten
- bestimmt Wiederverwendbarkeit und ist zum Teil projektübergreifend nutzbar
- ist das Ergebnis des Software-Designs
- gibt früh Sicherheit über Qualitätsmerkmale und die Einhaltung nicht-funktionaler Eigenschaften

- ***Architektursichten*** reduzieren die Komplexität durch Beschränkung auf spezifische Aspekte: Quasar z.B.:
  - A-Architektur
  - T-Architektur
  - TI-Architektur



Die Quasar Architektursichten liefern eine A/T Trennung auf der Architekturebene

Softwarearchitektur: Komponenten und Schnittstellen

A-Architektur
Anwendungsarchitektur

- frei von technischen, produktbezogenen Sachzwängen
- wird für jedes Projekt neu entwickelt
- strukturiert die Software aus der Sicht der Anwendung
- enthält fachliche Klassen wie "Mitarbeiter" oder "Konto"

A-Komponenten

T-Architektur
Technische Architektur

- beschreibt die "virtuelle Maschine", auf der die mit der A-Architektur entworfene Software läuft (Container für A-Komponenten)
- Stellt die Blaupause für die Umsetzung der A-Komponenten bereit
- Wiederverwendung möglich
- verbindet A- und TI- Architektur
- Enthält die technischen Komponenten (T-Komponenten)

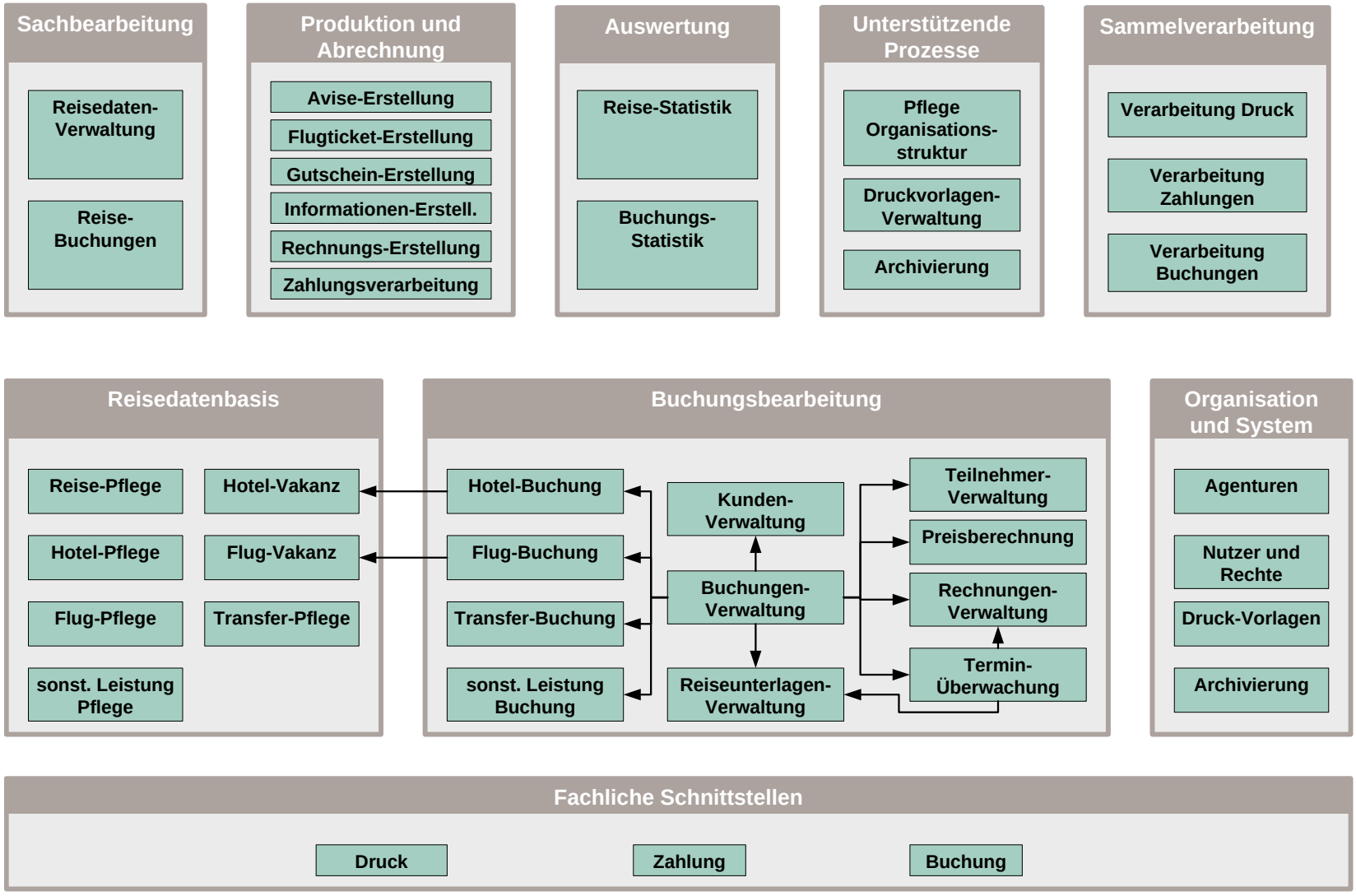
T-Komponenten

TI-Architektur
Technische Infrastruktur

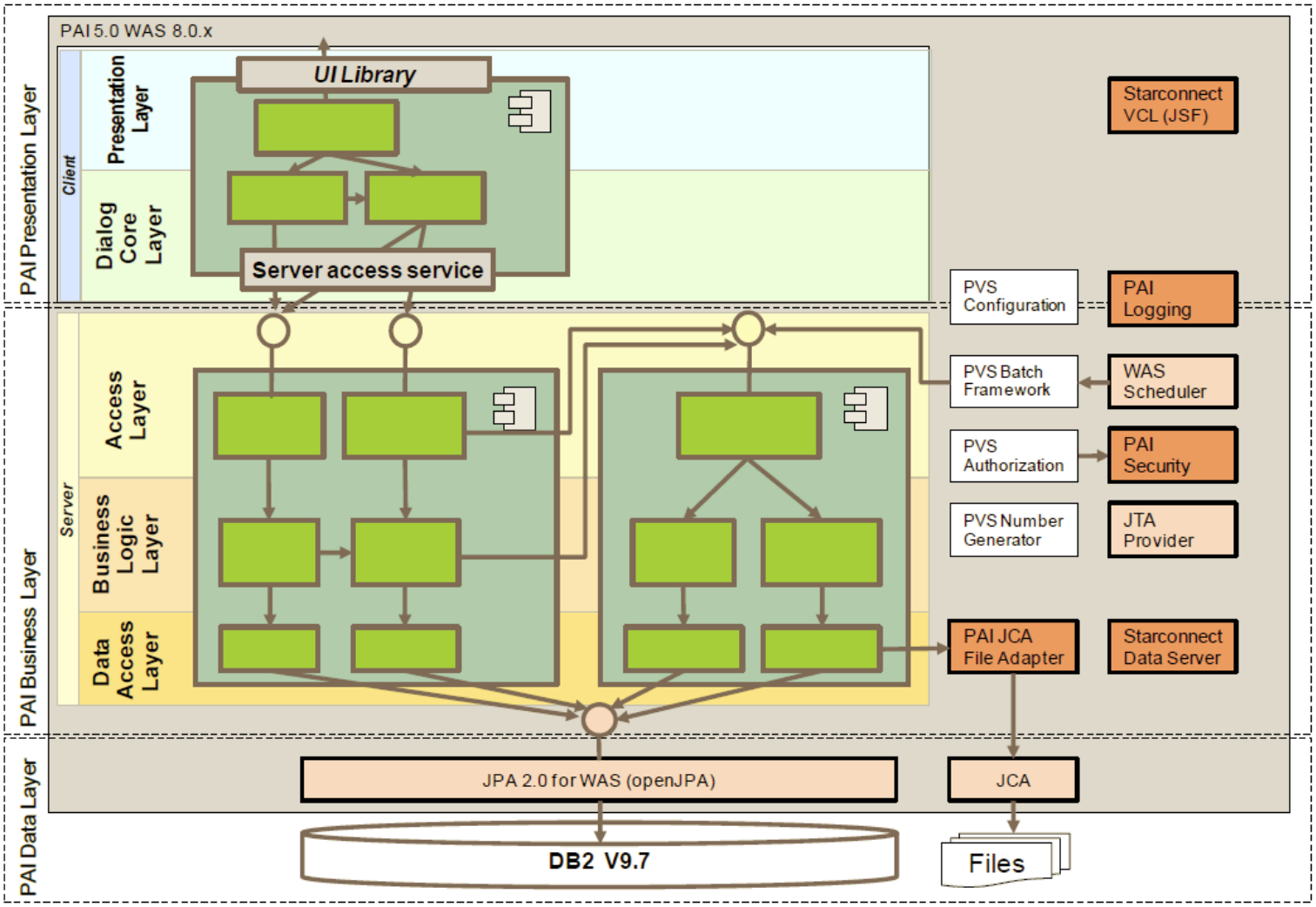
- physischen Geräte (Rechner, Netzleitung, ..)
- System-Software (BS, DBMS, App-Server, ..)
- Zusammenspiel der Hardware mit darauf installierter System-Software
- verwendete Programmier-sprachen

i. d. R. Produkte

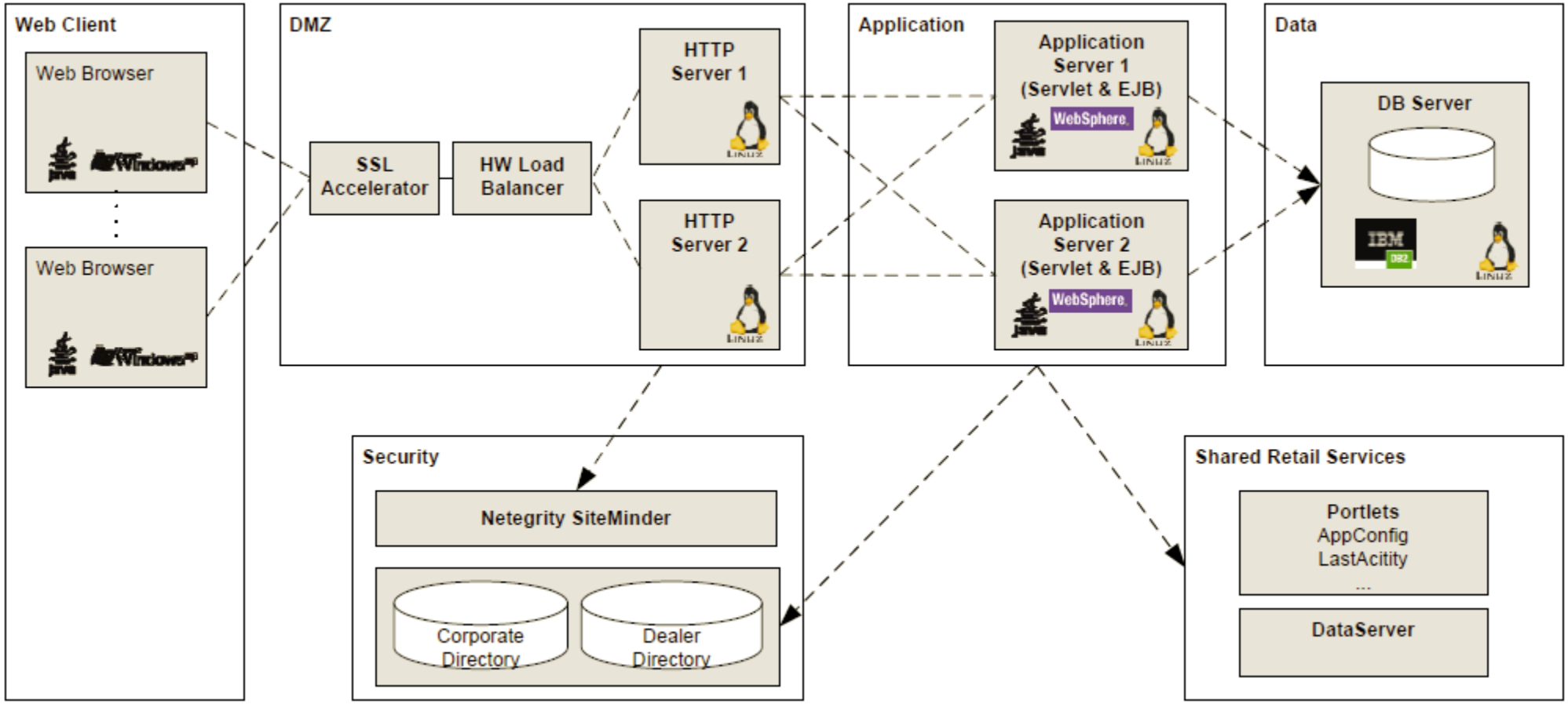
Beispiel einer A-Architektur eines Systems im Touristikumfeld



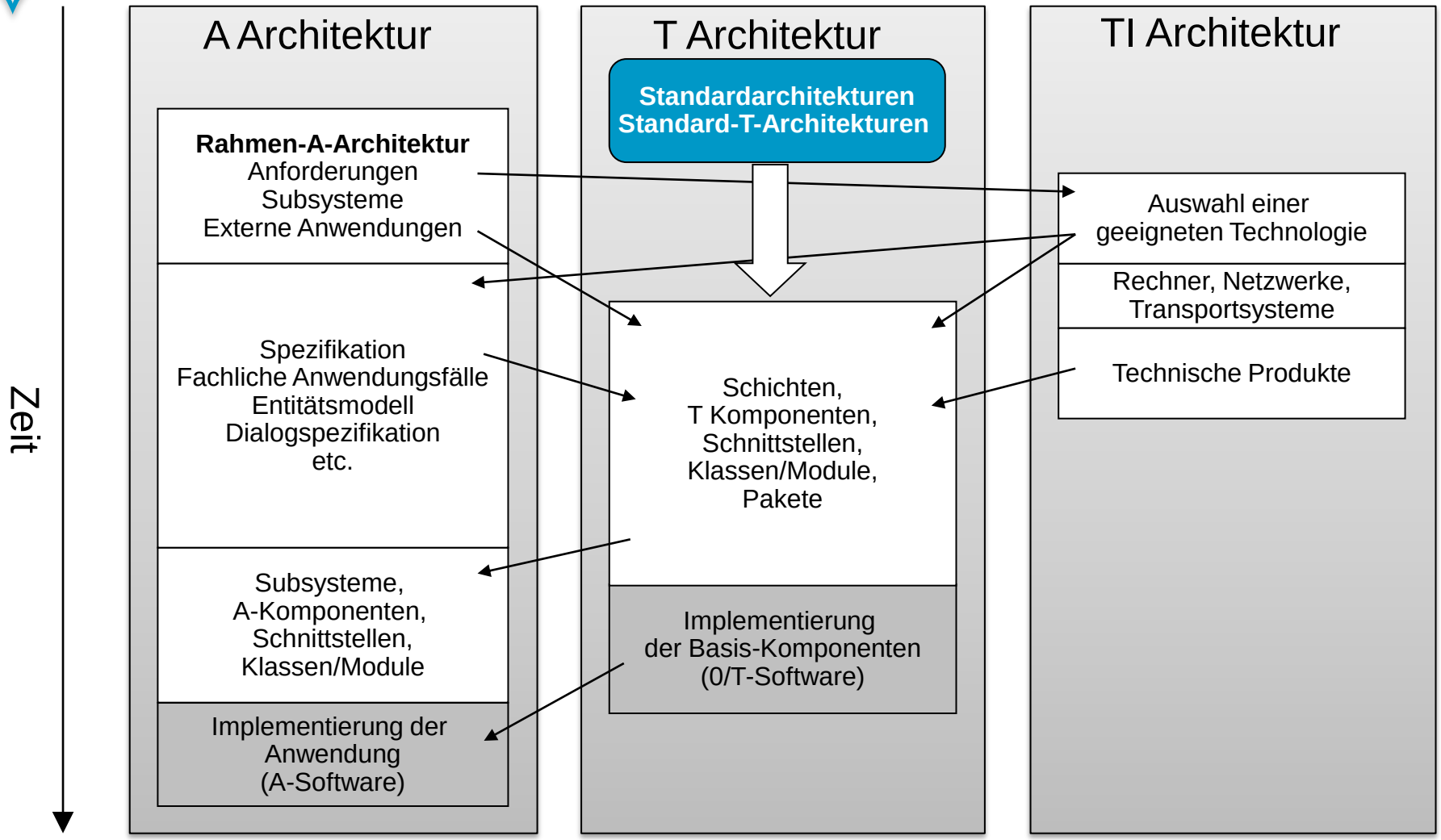
T-Architektur-Beispiel eines realen Kundenprojektes



TI-Architektur eines realen Projektes

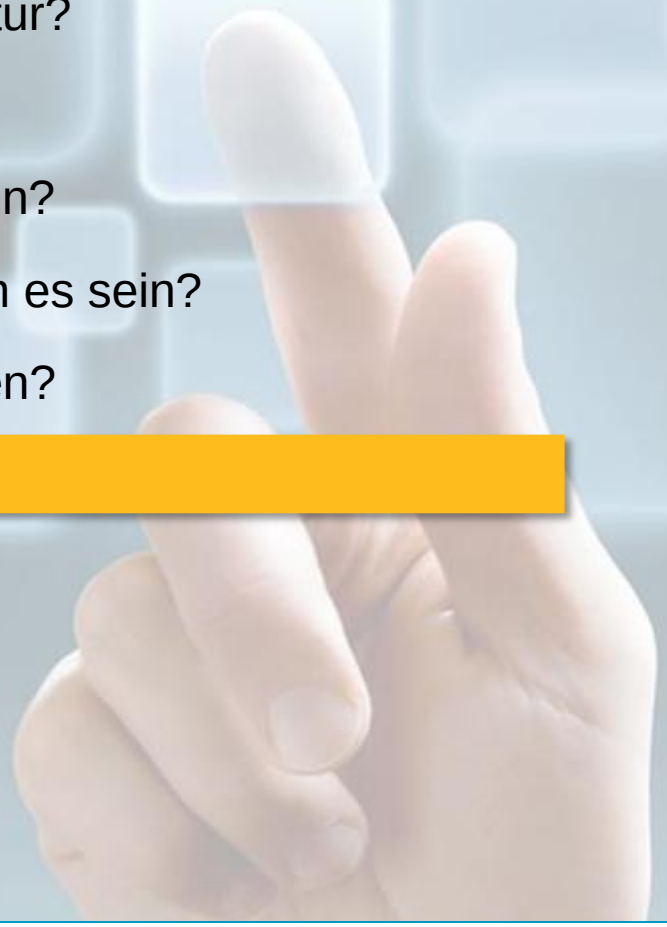


Entwicklung der verschiedenen Architekturaspekte über die Zeit

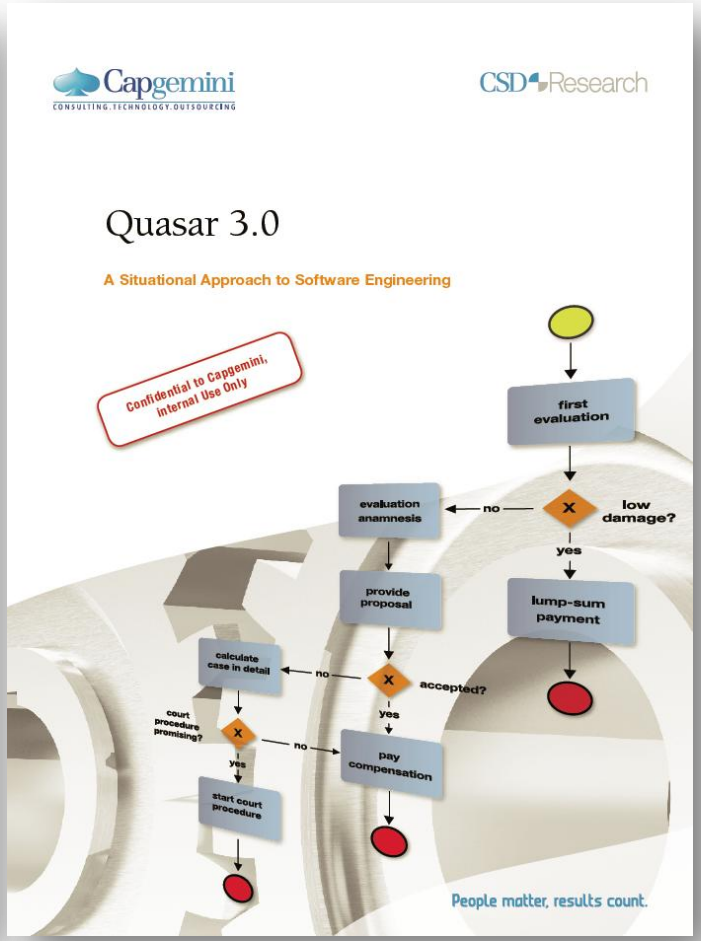


Agenda

- Was ist Software-Architektur?
- Was sind Komponenten?
- Wie finde ich Komponenten?
- Wie viele Schichten dürfen es sein?
- Was sind Architektursichten?
- **Abschluss!**



Qualitätssoftwarearchitekturen sind ein Markenzeichen von Capgemini



Die wichtigen Dinge zum Schluss:
Gute Software zeichnet sich durch ihre Einfachheit aus!

Antoine de Saint-Exupéry

Perfektion erreicht man nicht etwa dann, wenn sich nichts mehr hinzufügen lässt, sondern dann, wenn sich nichts mehr entfernen lässt ...

KISS
Keep it simple, stupid!
Keep it short and simple
…

Kelly Johnson

Wer das als nächstes anfasst,

- **muss ein Produktionsproblem lösen,**
- **hat nur halb so viel Ahnung wie Du,**
- **nur halb so viel Zeit wie nötig, und keine Chance, anständig zu testen,**
- **während sein Chef und der Kunde dreimal anrufen, ob es noch nicht fertig ist.**

Hilf ihm, seinen Job zu behalten – denn es könnte auch Dir passieren!

Fragen?

Kontaktinformationen

Holger Cermann
Manging Business Architect
holger.cermann@capgemini.com

Tel. (0711) 50505-733
Mobil (0160) 477 28 09

Capgemini
Loeffelstrasse 46

70597 Stuttgart

Christian Nicu
Managing Solution Architect
christian.nicu@capgemini.com

Tel. (0711) 50505-456
Mobil (0151) 402 515 66

Capgemini
Loeffelstrasse 46

70597 Stuttgart

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

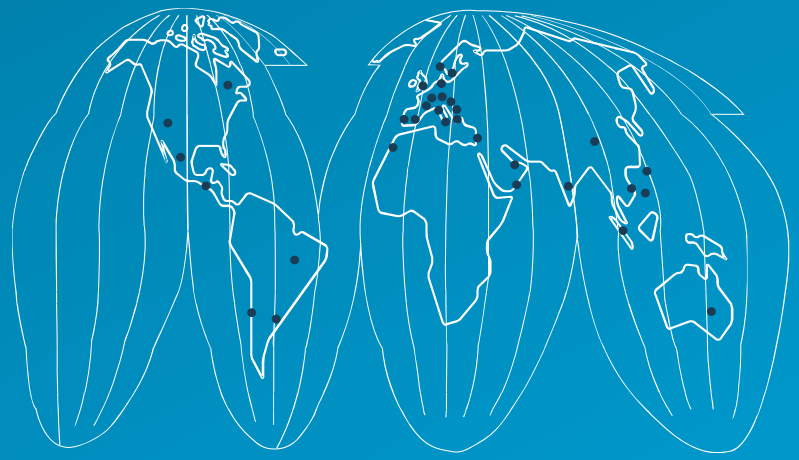
People matter, results count.

About Capgemini

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion.

Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience™, and draws on Rightshore®, its worldwide delivery model.

Rightshore® is a trademark belonging to Capgemini



www.capgemini.com