

Softwaretechnik

Softwarearchitektur

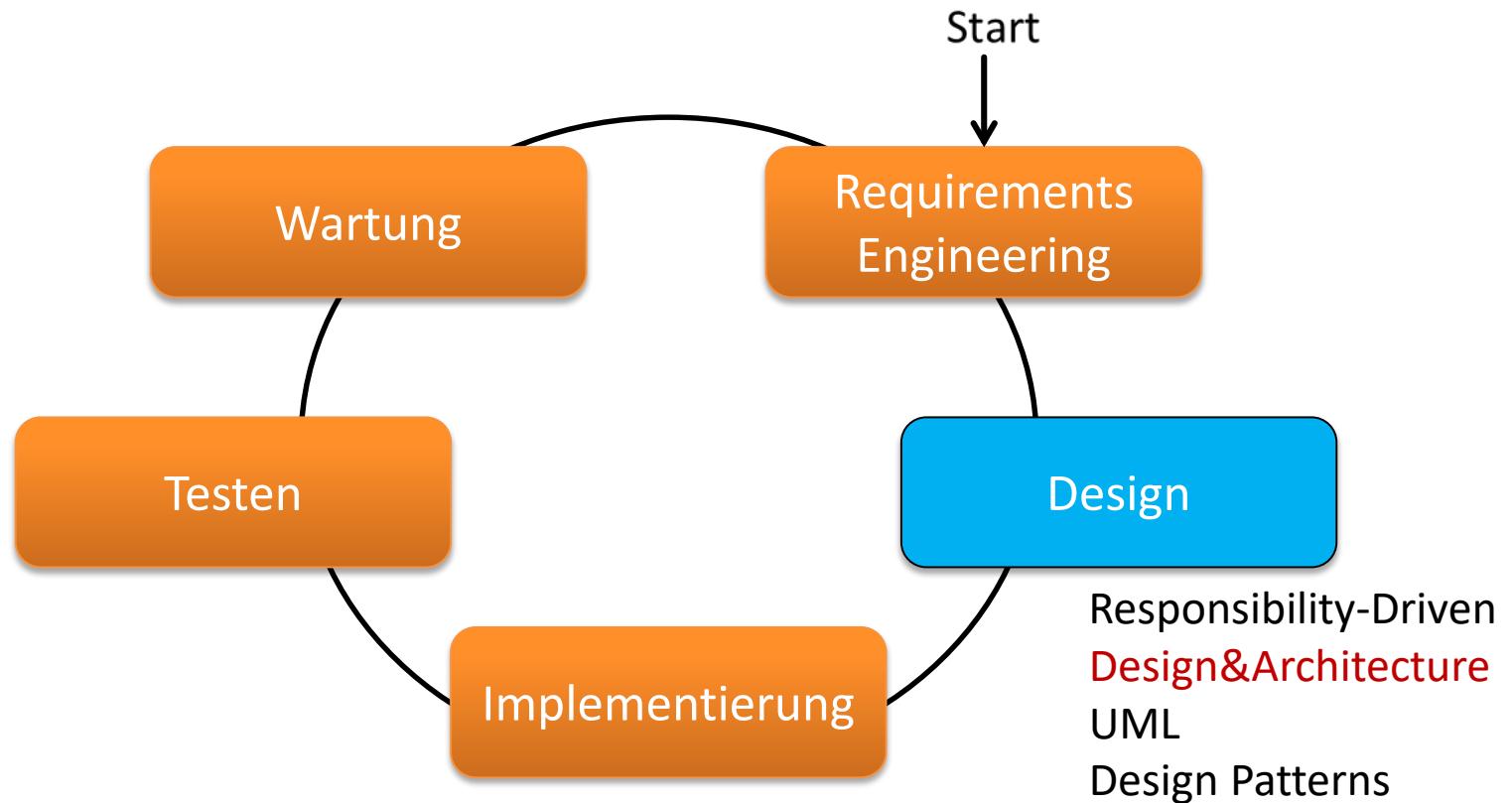


Prof. Dr.-Ing. Norbert Siegmund
Software Systems



UNIVERSITÄT
LEIPZIG

Einordnung



Lernziele

- Arten von Software-Architekturen kennen
- Zusammenhang zwischen Design und Architektur einordnen
- Konkrete Architekturmuster wissen

Was ist Software Architektur?



Was ist Software Architektur?

A neat-looking drawing of some boxes, circles, and lines, laid out nicely in Powerpoint or Word, does not constitute an architecture.

— D'Souza & Wills

Was ist (wirklich) Software Architektur?

Die Architektur eines Systems besteht aus:

- der *Struktur(en) ihrer Teile*
 - einschließlich Design-, Test und Laufzeit Hardware und Software Teile
- den *extern sichtbaren Eigenschaften* dieser Teile
 - Module mit Interfaces, Hardware-Einheiten und Objekte
- den *Beziehungen und Bedingungen* zwischen ihnen

In anderen Worten:

Eine Menge von Design-Entscheidungen über ein (Sub)System, die die Entwicklerinnen davon abhält sinnlos herumzuexperimentieren.

Design vs. Architektur

- Design beschreibt Aufbau von Subsystemen und Komponenten (fein granular)
 - Welche Klassen gibt es (in Modul X) und wie interagieren sie?
 - Siehe Responsibility-Driven Design / objektorientiertes Design
- Architektur beschreibt den groben Aufbau eines Systems (welche Komponenten gibt es?)
 - Welche Komponenten / Subsysteme / Module gibt es und wie interagieren sie?

Sub-systeme, Module und Komponenten

- Ein Sub-system ist selbst ein System, dessen Operation *unabhängig* von den Leistungen und Funktionen anderer Sub-systeme ist.
- Ein Modul ist eine Systemkomponente, die *Dienstleistungen / Funktionen anbietet*, welche andere Komponenten benötigen, aber welche nicht als komplett separates System angesehen werden.
- Eine Komponente ist eine *unabhängig auslieferbare Einheit* von Software, die ihr Design und Implementierung eingeschlossen hat (hiding) und ihr Interface zur Außenwelt anbietet, so dass sie mit anderen Komponenten zusammengefasst werden kann, um ein größeres System zu bilden.



Workarounds für schnelle Lösungen

Architekturprobleme

Fehlende Abstimmung zwischen Teams



Fehlende gesamtheitliche Modellierung



Zentralisierte Komponenten



Zu viele kleine Komponenten (zu viel Kommunikation erforderlich)

Modul skaliert
nicht mit Rest des
Systems



Interfaces passen nicht zusammen
(falsche Annahmen und fehlende Validierung)



Falsche / unklare Kompositions-
bzw. Ausführungsreihenfolge

Arten von SW Architektur



Parallelen zur (echten) Architektur

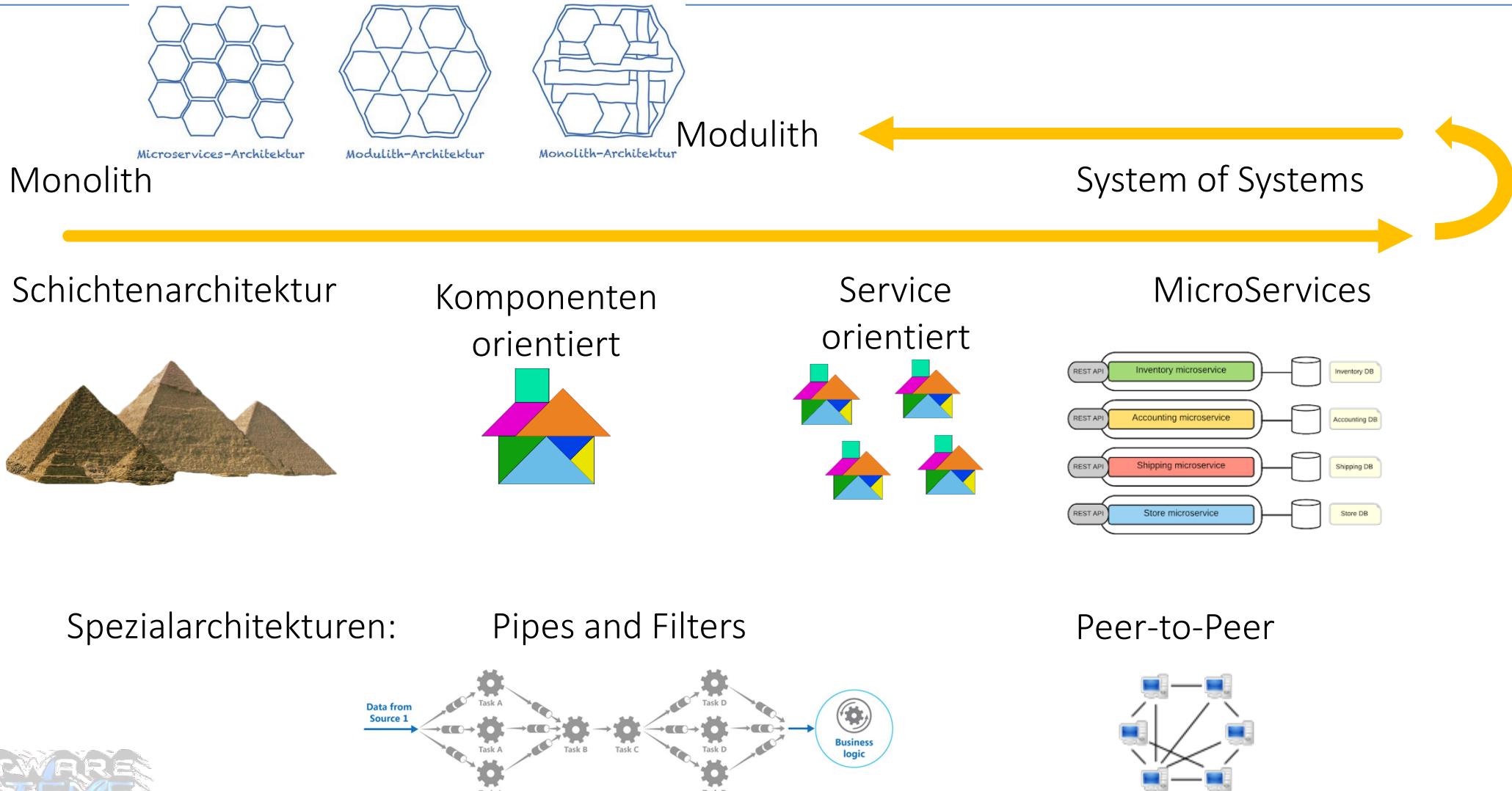
- Architekten sind die *Schnittstelle* zwischen den Kunden und den Auftragnehmern, die das System/Gebäude bauen
- Eine schlechte Architektur für ein Gebäude *kann nicht mehr durch gute Konstruktion gerettet werden* – gleiches gilt für Software
- Es gibt *spezielle Typen* von Gebäuden und Software-Architekten.
- Es gibt *Schulen oder Styles* des Bauens und der Software –Architektur.

Architectural Styles

An architectural style defines a *family of systems* in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of *components* and *connector* types, and a set of *constraints* on how they can be combined.

— Shaw and Garlan

SW-Architekturen für große Softwaresysteme



Welche Architektur?

amazon.de Prime

Koffer, Rucksäcke & Taschen

DE Hallo, Jonas Mein Konto Mein Prime Meine Listen Einkaufswagen

Alle Kategorien Jonas' Amazon Angebote Gutscheine Verkaufen Hilfe

Amazon Fashion DAMEN HERREN KINDER & BABY GEPÄCK MARKEN SALE

KOSTENLOSER RÜCKVERSAND Innerhalb von 30 Tagen Mehr Informationen

Koffer, Rucksäcke & Taschen > Taschen > Umhängetaschen

 Für größere Ansicht Maus über das Bild ziehen

AmazonBasics Tasche für Laptop / Tablet mit Bildschirmdiagonale 15,6 Zoll / 39,6 cm

von AmazonBasics 616 Kundenrezensionen | 64 beantwortete Fragen

Bestseller Nr. 1 in Notebook-Akketaschen

Preis: EUR 19,49 ✓Prime Alle Preisangaben inkl. USt

Auf Lager.

Verkauf und Versand durch Amazon. Geschenkverpackung verfügbar.

Größe: 15,6 Zoll / 39,6 cm

11,6 Zoll / 29,5 cm EUR 13,99 ✓Prime 14,1 Zoll / 35,8 cm EUR 14,99 ✓Prime 15,6 Zoll / 39,6 cm EUR 19,49 ✓Prime 17,3 Zoll / 44 cm EUR 19,99 ✓Prime

- Schlanke und kompakte Tasche, perfekt für Laptops bis 29,5 cm (15,6 Zoll), ohne unnötigen Ballast
- Zubehörtaschen für Maus, iPod, Handy und Stifte
- Einschließlich gepolstertem Schulterriemen

Weitere Produktdetails

Falsche Produktinformationen melden

Möchten Sie Ihr Elektro- und Elektronik-Gerät kostenlos recyceln? (Erfahren Sie mehr.)

Teilen

In den Einkaufswagen

Jetzt mit 1-Click® kaufen

Innerhalb von 3h 9min bestellen und Lieferung erfolgt am: Samstag, 21 Jan (GRATIS Premiumversand)

Lieferort: Jonas Hecht- Weimar - 99423

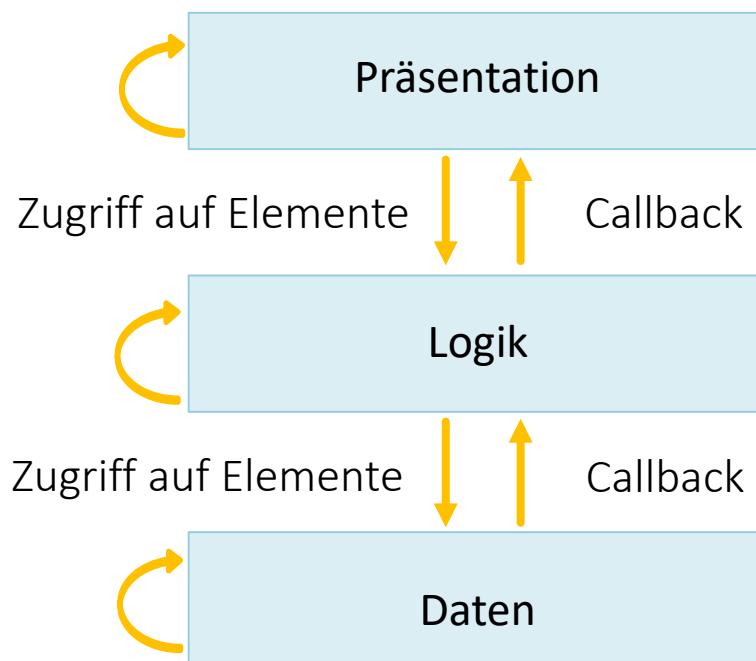
Dies ist ein Geschenk

Auf die Liste

Möchten Sie verkaufen? Bei Amazon verkaufen

Schichtenarchitekturen

- Eine Schichtenarchitektur organisiert ein System in eine Menge von Schichten, wobei jede Schicht eine Menge von Leistungen / Funktionen für die Schicht “darüber” anbietet.



Vorteile:

- Inkrementelle Entwicklung von Subsystemen
- Wenn ein Interface einer Schicht sich ändert, *sind nur benachbarte Schichten betroffen*
- Modularität, Kohäsion

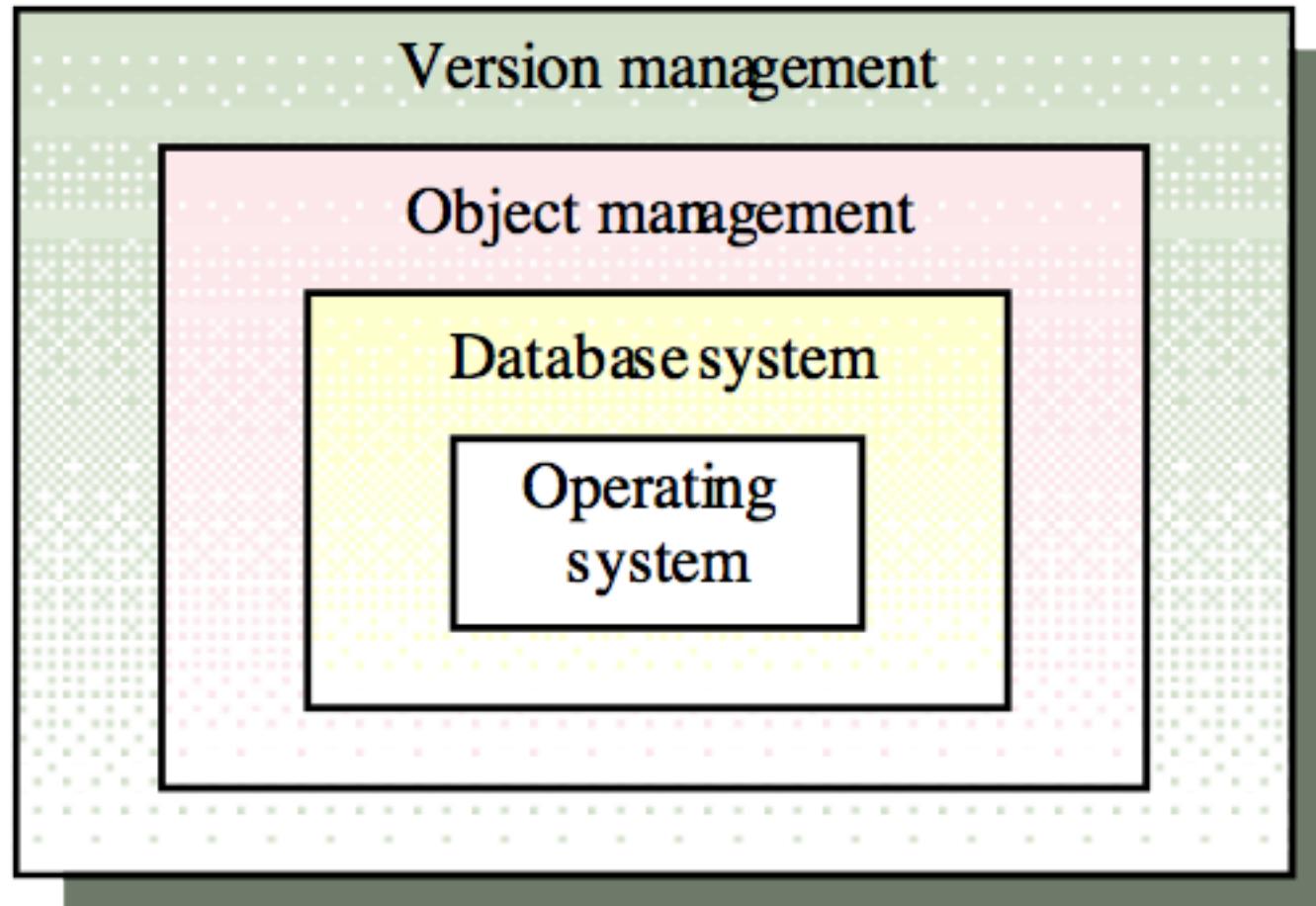
Layered Architectures

Eine Schichtenarchitektur organisiert ein System in eine Menge von Schichten, wobei jede

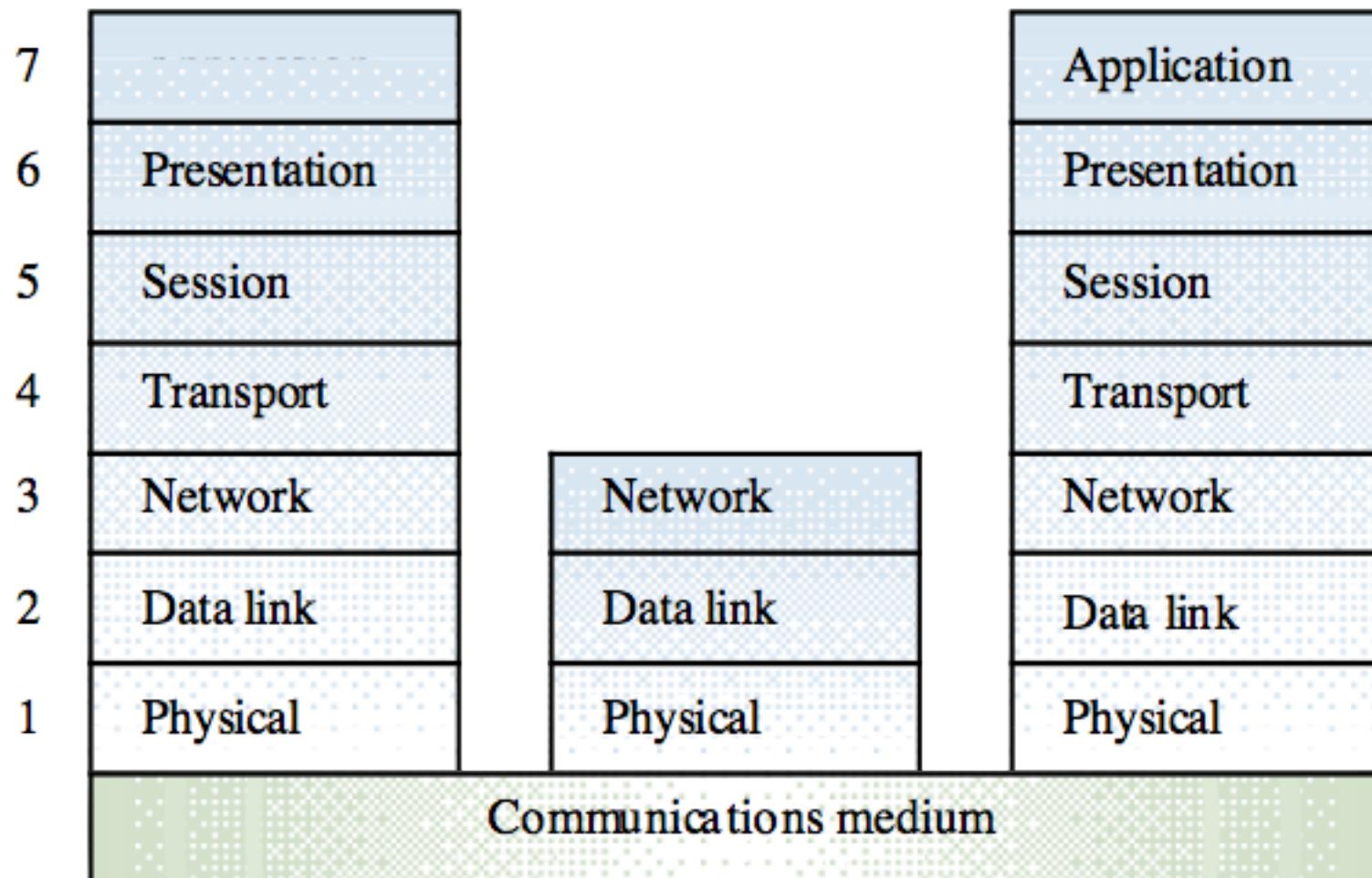
Schicht eine Menge von Leistungen / Funktionen für die Schicht “darüber” anbietet.

- Schichten sind normalerweise *beschränkt*, so dass Elemente nur
 - Andere Element in der gleichen Schicht oder
 - Elemente von der Schicht darunter sehen können
- *Callbacks* können verwendet werden, um mit höheren Schichten zu kommunizieren
- Unterstützt die *inkrementelle Entwicklung* von Sub-systemen in unterschiedlichen Schichten
 - Wenn ein Interface einer Schicht sich ändert, *sind nur benachbarte Schichten betroffen*.

Version Management System



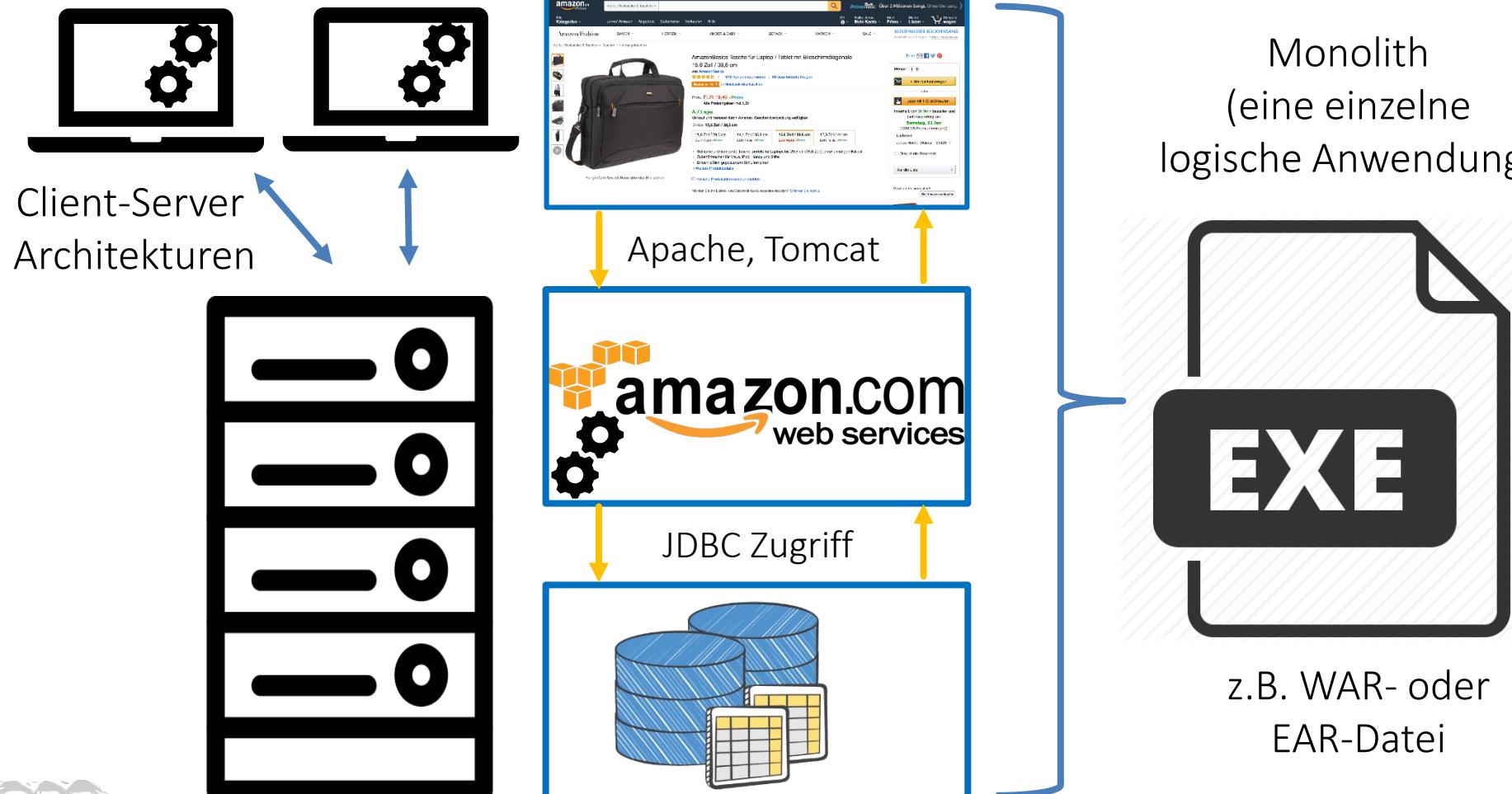
OSI Reference Model



Beispiel: 3-Layer Architektur



Client-Server & Monolith



Client-Server Architektur

Eine Client-Server Architektur *verteilt Applikationslogik und Funktionalität* zu einer Anzahl von Klienten (clients) und Server-Subsystemen, wobei jede potentiell auf einer unterschiedlichen Maschine läuft und über das Netzwerk kommuniziert.

Vorteile:

- Einfache *Datenverteilung*
- Effektive *Hardwareauslastung*
- Einfaches *Hinzufügen* neuer Server

Nachteile:

- Kein *geteiltes Datenmodell*
- *Redundante* Verwaltung
- Evtl. *zentrale Registrierung* erforderlich
(welcher Server stellt welche Dienstleistung zur Verfügung?)



Und ohne Web?

Wie würde Sie die Architektur entwerfen, wenn wir eine Desktop-Anwendung schreiben würden?

Kommunikation / Controller

Apache, Tomcat, JavaScript

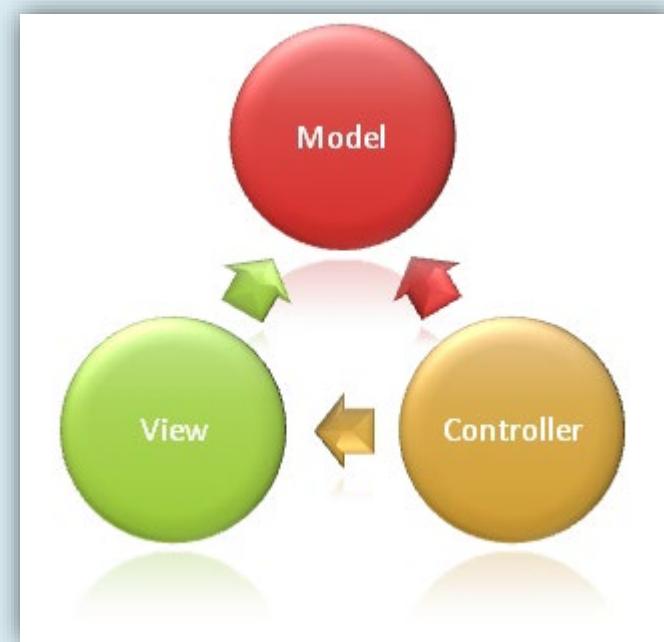
Sicht / View

JavaScript Frontend

Java Implementierung, DB, ...

Model / Business-Logik / Daten

Model-View-Controller

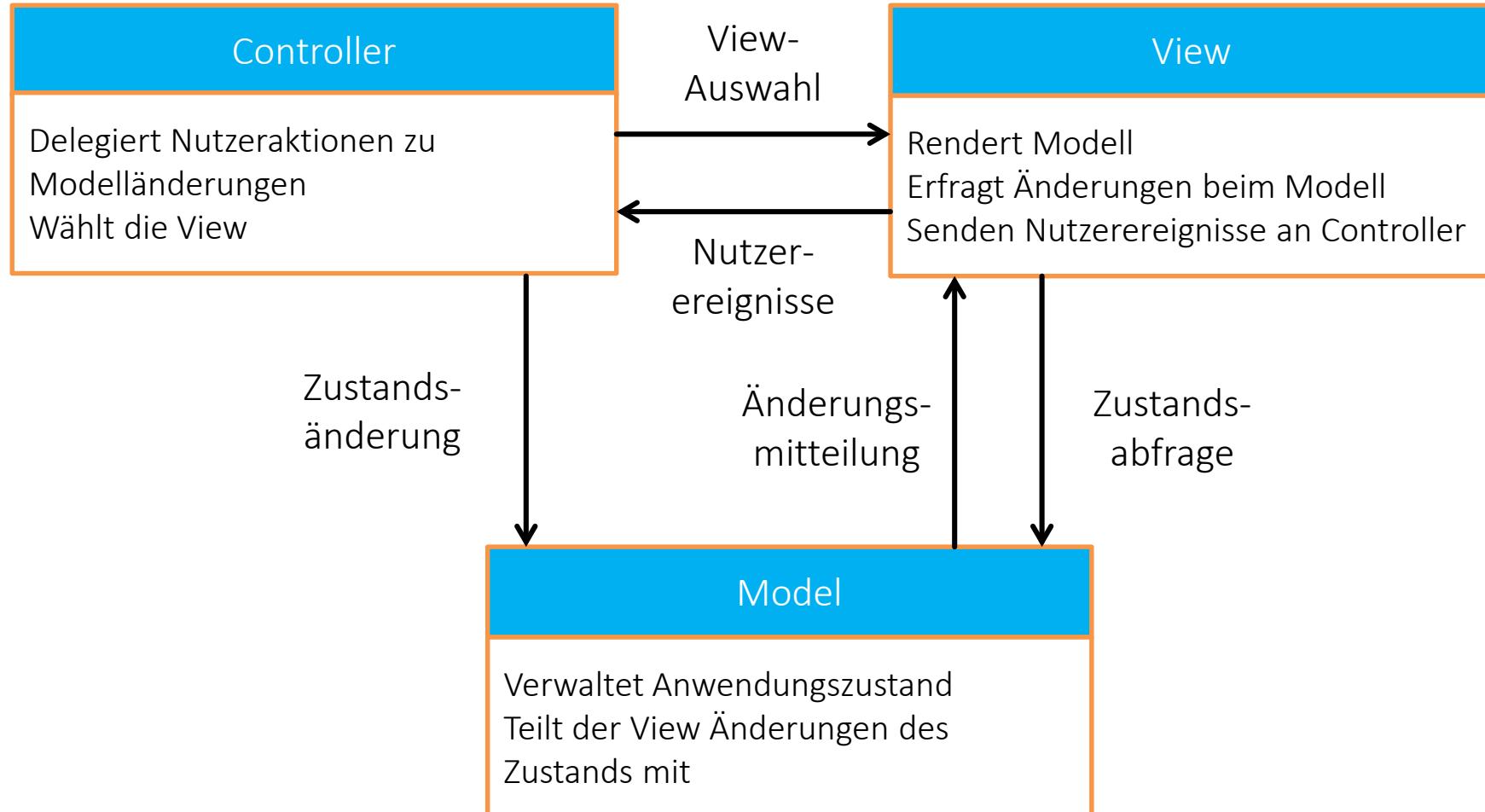


Model-View-Controller (MVC) Architektur

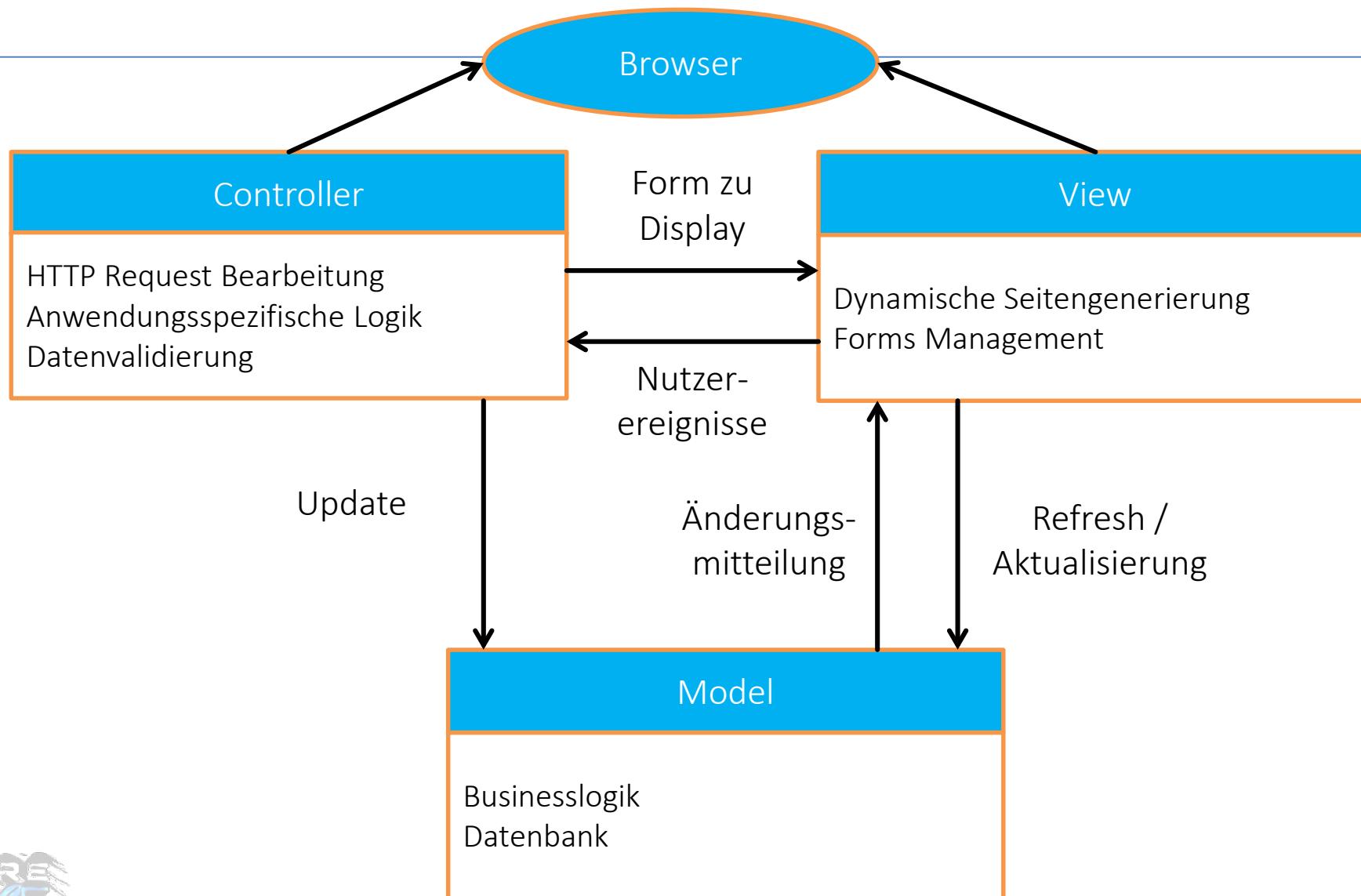
Idee: Sepriere *Präsentation* und *Interaktion* von den *Daten* des Systems

- Das System ist strukturiert in drei Komponenten:
 - *Model*: verwaltet Systemdaten und Operationen auf den Daten
 - *View*: Präsentiert die Daten zum Nutzer
 - *Controller*: händelt Nutzerinteraktion; schickt Informationen zur View und zum Model
- Nützlich, wenn es mehrere Wege gibt auf Daten zuzugreifen
- Ermöglicht das Ändern der Daten unabhängig von deren Repräsentation
- Unterstützt unterschiedliche Präsentationen der selben Daten

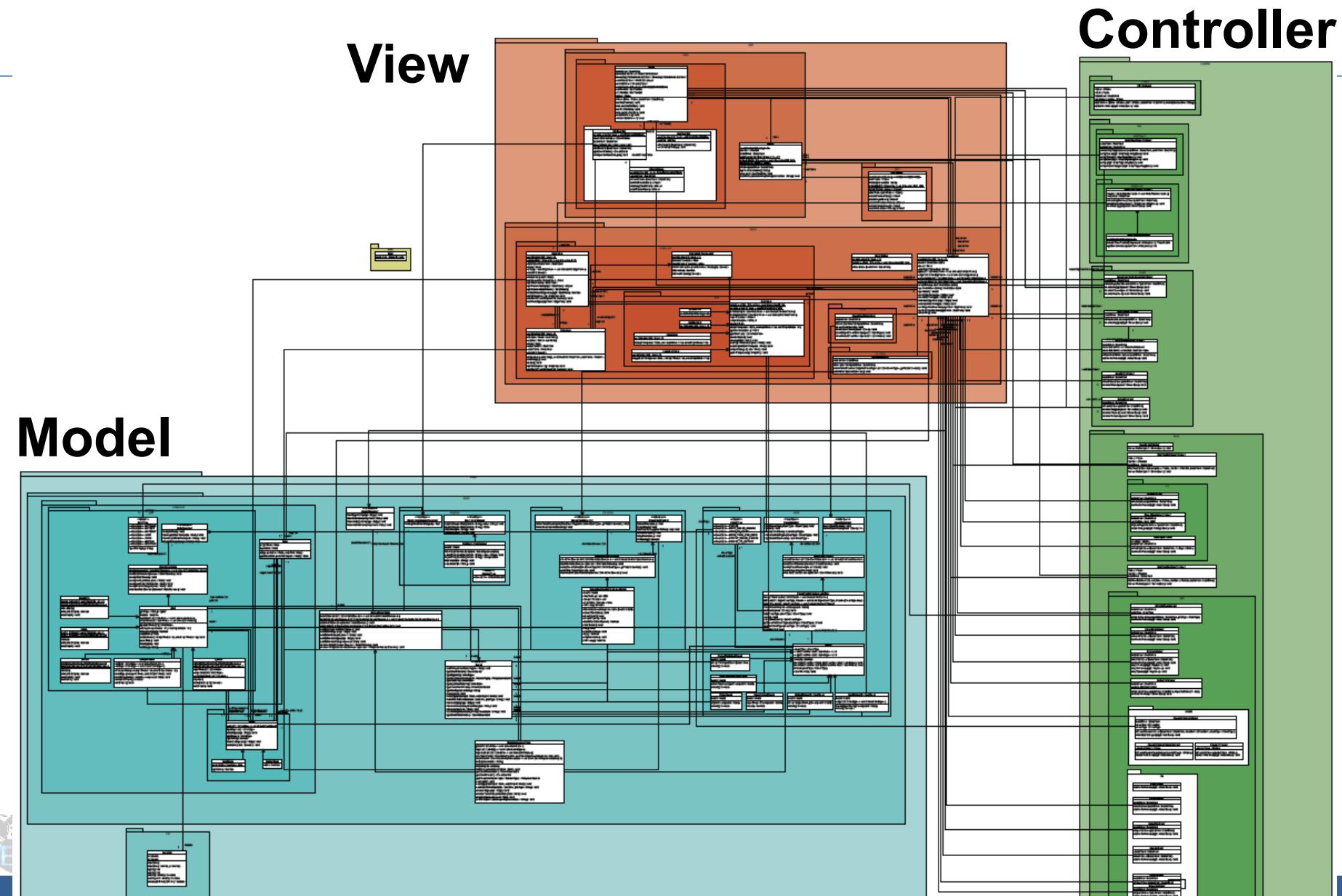
MVC Übersicht



MVC Beispiel

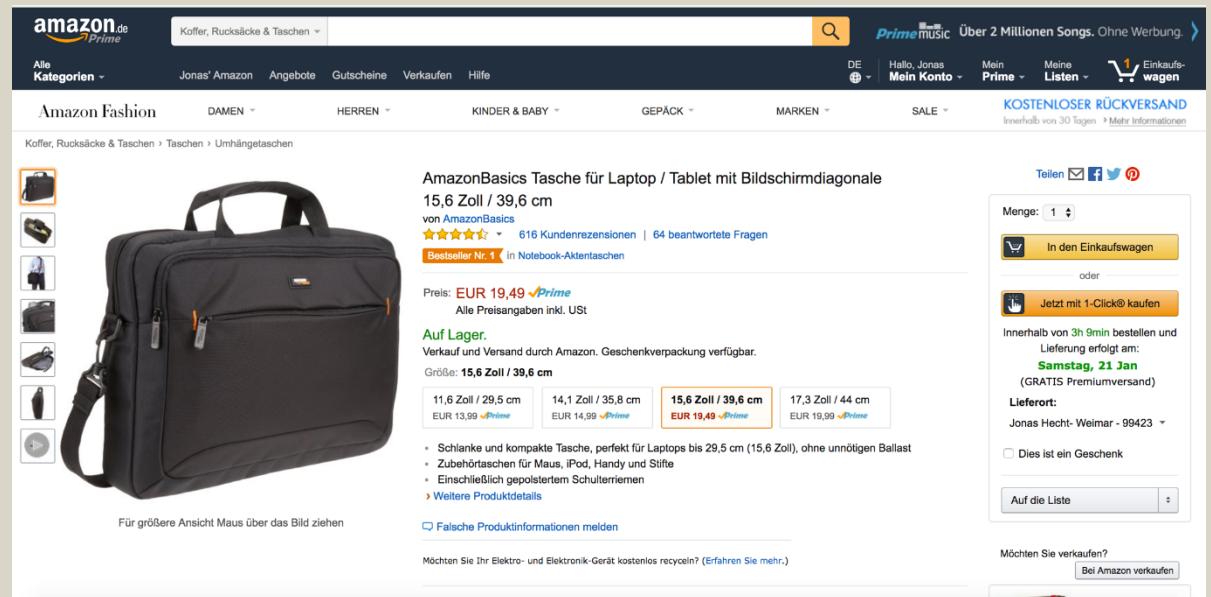


MVC in Action (Circuit Simulation)



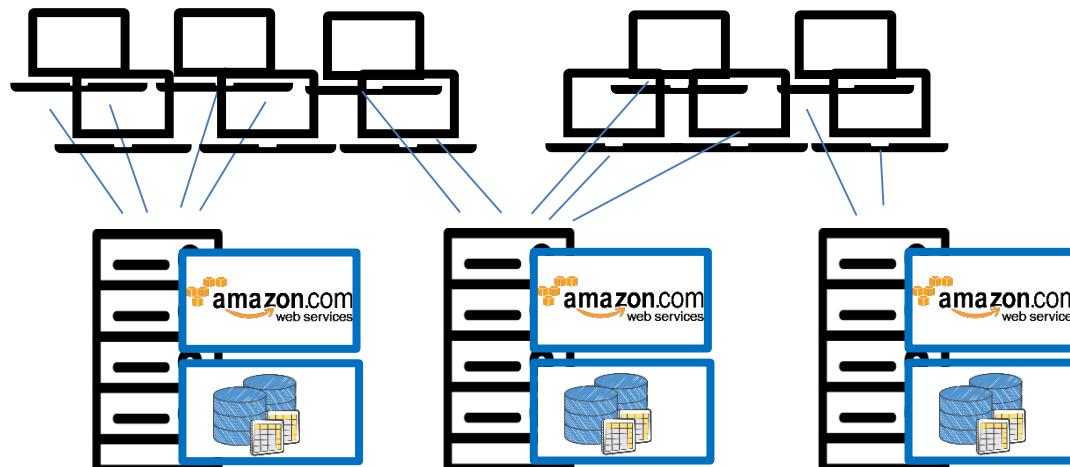
Probleme der Architekturen

- Welche Probleme erwarten Sie, wenn wir diese Architekturen in der Praxis für unser Beispiel einsetzen?
- Hinweise:
 - Großes Softwaresystem
 - Hohes Nutzeraufkommen



Probleme traditioneller Architekturmuster für große Softwaresysteme

- Häufige Änderungen
 - Monolithisches System muss komplett neu gebaut werden
 - Abhängigkeiten zwischen Subsystemen erschweren und verzögern Änderungen
- Skalierbarkeit der Hardware
 - Alle Architekturen sind schwer skalierbar, selbst Client-Server



- Neue Probleme:
- Verteilte, *replizierte* Daten
 - *Konsistenz* und Synchronität?
 - *Gesamtes* System repliziert, obwohl nur Subsysteme ausgelastet sind

Probleme traditioneller Architekturmuster für große Softwaresysteme

- Was passiert bei Ausfällen und Fehlern von Subsystemen?
 - Oft zu starke Kopplung der Subsysteme und kaum Möglichkeit des „Fail-overs“
 - Alternative Views bei MVC möglich, aber alternative Modelle?
- Wie wird das laufende System geupdated?
 - Herunterfahren der Server ist keine Option
- Wie kann die Entwicklung von Subsystemen parallelisiert werden?
 - Schichtenarchitektur und MVC oft zu grob-granular
 - Komponenten und Services nicht gut bei querschneidenden Belangen und erfordern zu viel Glue-Code / Kommunikation
- Wie vereinfache ich das Testen? Usw.

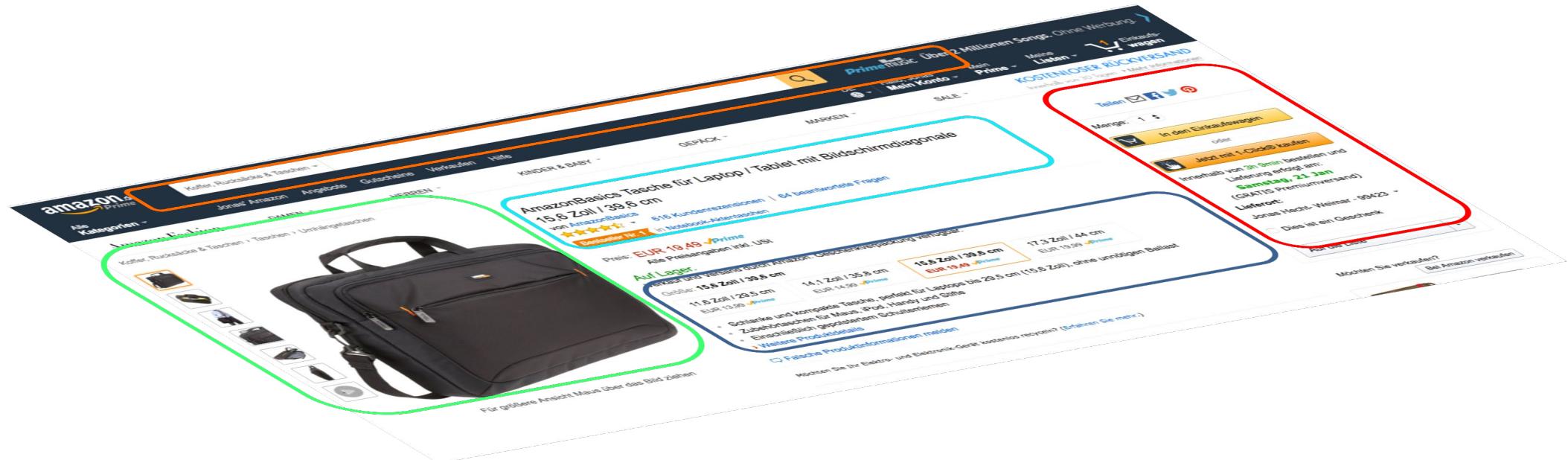
Gewünschte Eigenschaften

- Schneller Austausch von Subsystemen ohne, dass das gesamte System betroffen ist (lose Kopplung + hohe Modularität)
- Skalierbarkeit bei großen Lasten von einzelnen Subsystemen (Beispiel: Black Friday)
- Weniger Abstimmungsaufwand innerhalb der Unternehmensorganisation (bei Entwicklerinnen und Sachverständigen)
- Parallelisierung in der Entwicklung sowie Anwendbarkeit von agilen Methoden der Softwareentwicklung
- Einfache Testbarkeit von Subsystemen

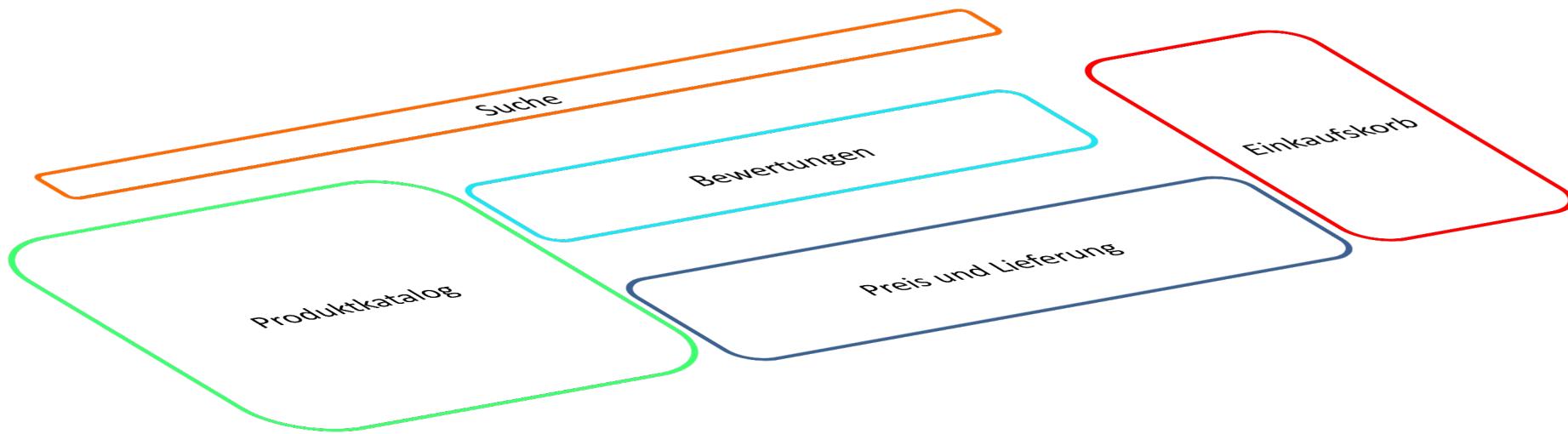
Welche Architektur?

The screenshot shows an Amazon product page for a laptop bag. The page includes a navigation bar with links like 'Amazon Fashion', 'DAMEN', 'HERREN', 'KINDER & BABY', 'GEPÄCK', 'MARKEN', and 'SALE'. A search bar at the top has an orange border. The main product image is highlighted with a green oval. To the right of the image, product details are shown in a blue-bordered box: 'AmazonBasics Tasche für Laptop / Tablet mit Bildschirmdiagonale 15,6 Zoll / 39,6 cm' by 'AmazonBasics', with a price of 'EUR 19,49 Prime'. Below this, there's a section about availability ('Auf Lager') and size ('Größe: 15,6 Zoll / 39,6 cm'). A red box highlights the purchase options on the right, which include a yellow 'In den Einkaufswagen' button and a blue 'Jetzt mit 1-Click® kaufen' button. The delivery information 'Innerhalb von 3h 9min bestellt und Lieferung erfolgt am: Samstag, 21 Jan (GRATIS Premiumversand)' is also visible.

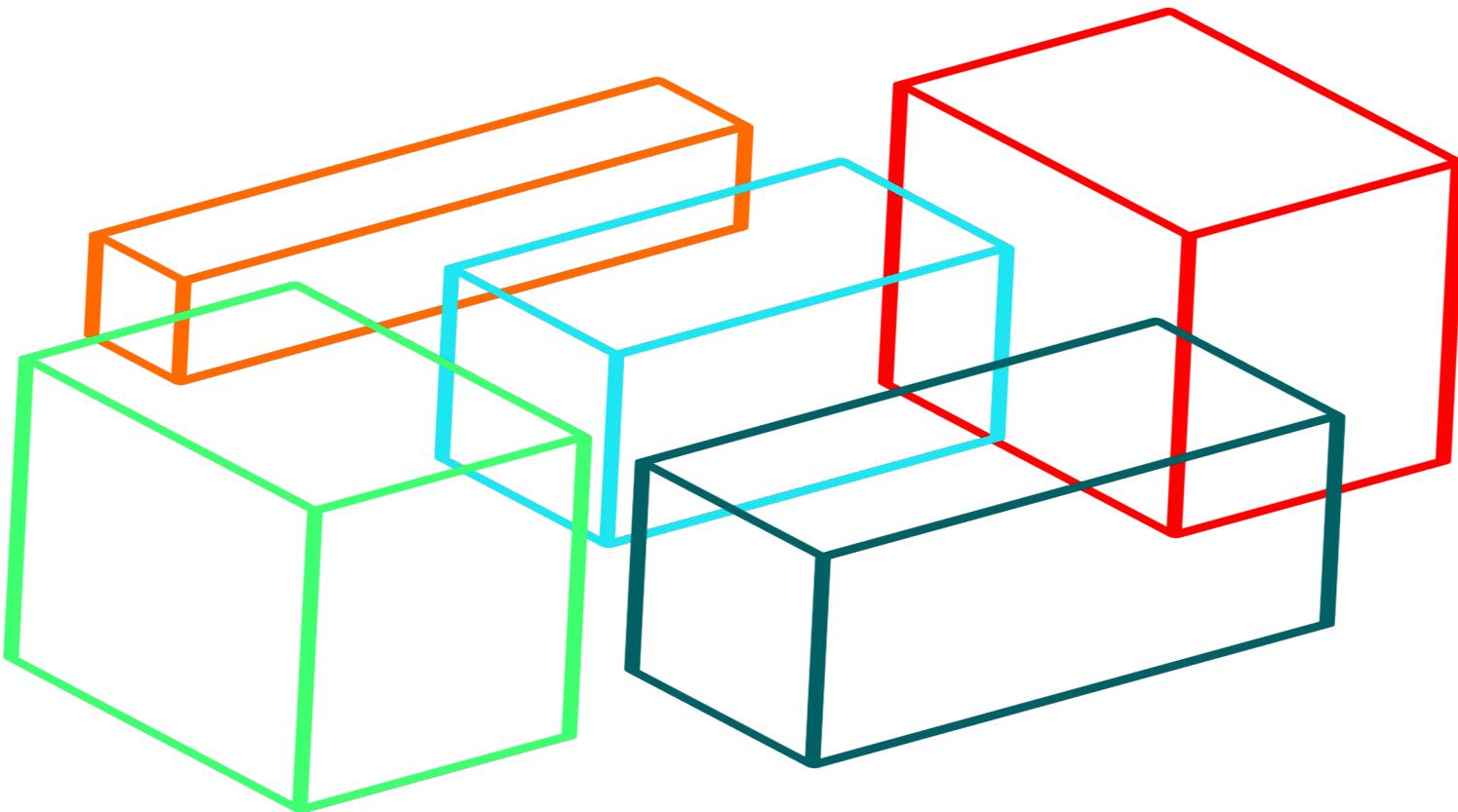
Zerlegung des Systems...



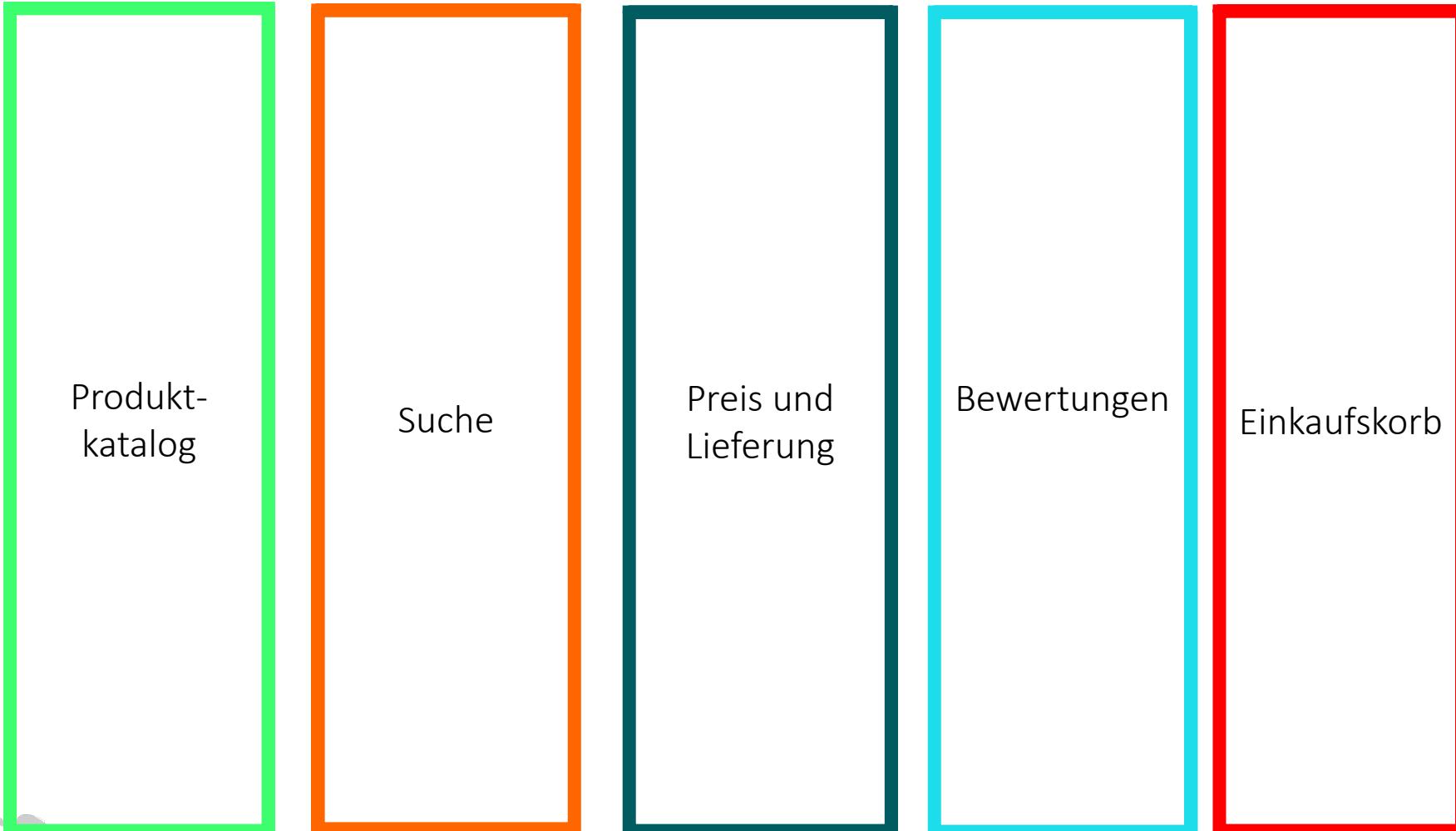
Zerlegung des Systems...



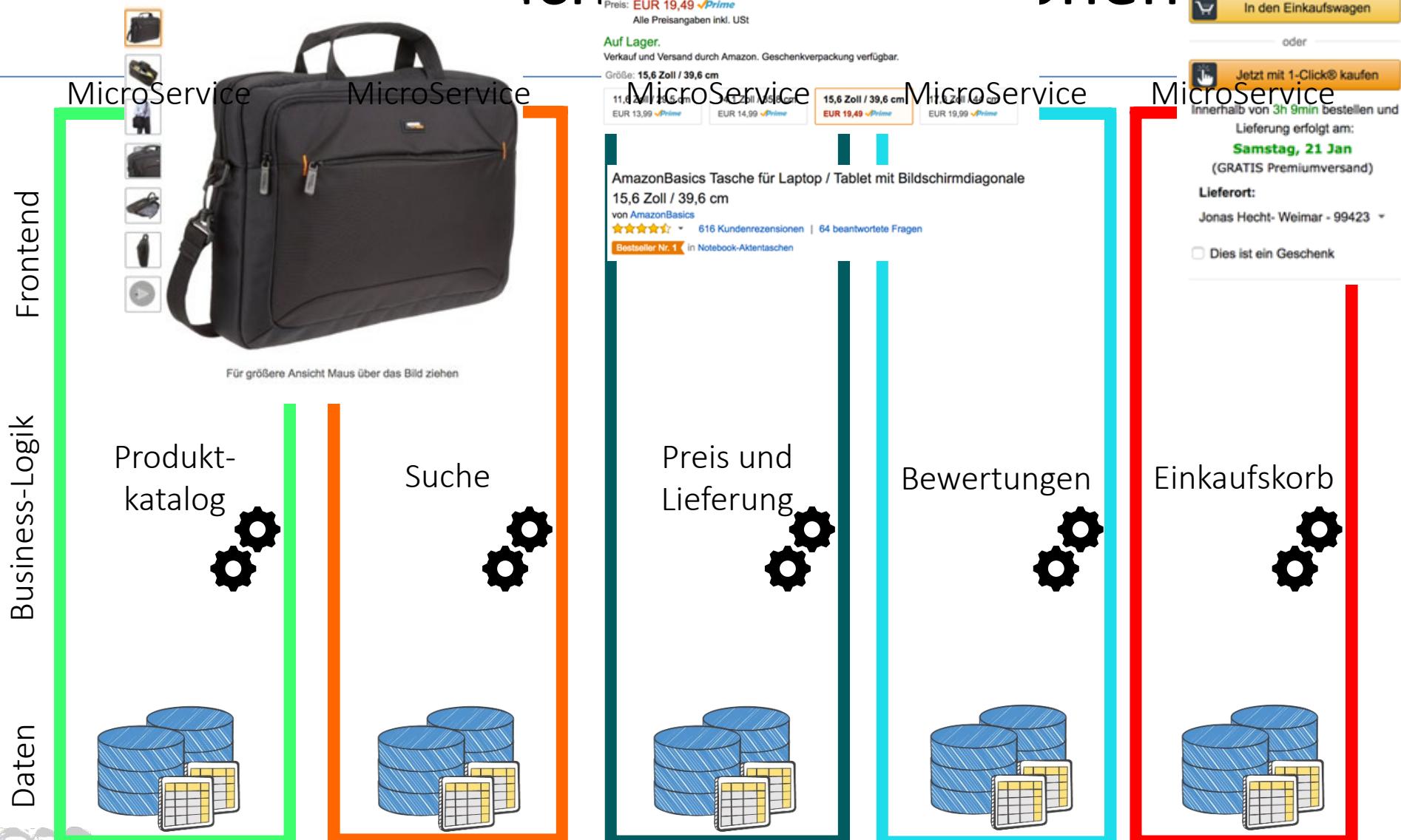
Zerlegung des Systems...



... nach fachlichen Funktionen



nach fachlichen Funktionen



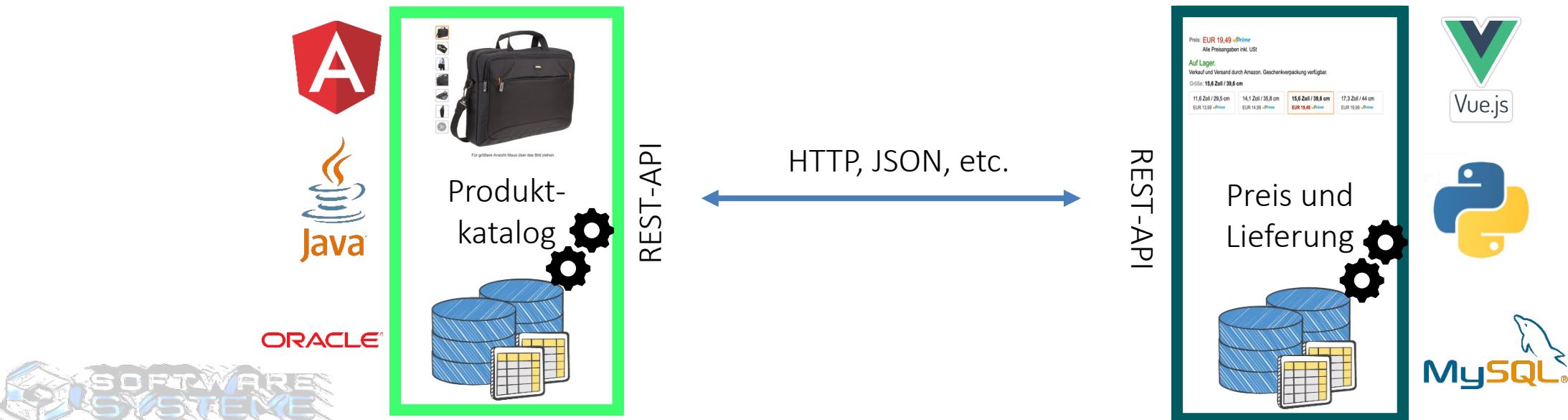
MicroServices: Idee

- Jede *fachliche* Funktion in einen *autonomen, unabhängigen Subsystem* modularisieren
- Jeder Service bietet die *vollständige* Funktionalität für die jeweilige fachliche Aufgabe an
 - Alle *Daten*, die hierfür notwendig sind
 - Gesamte *Business-Logik* für die Aufgabe
 - Alle *Sichten* und *Interaktionsmöglichkeiten*
- Teamzusammensetzung ideal für agile Entwicklung
 - Fachexpertinnen, Entwicklerinnen, Tester, DevOps

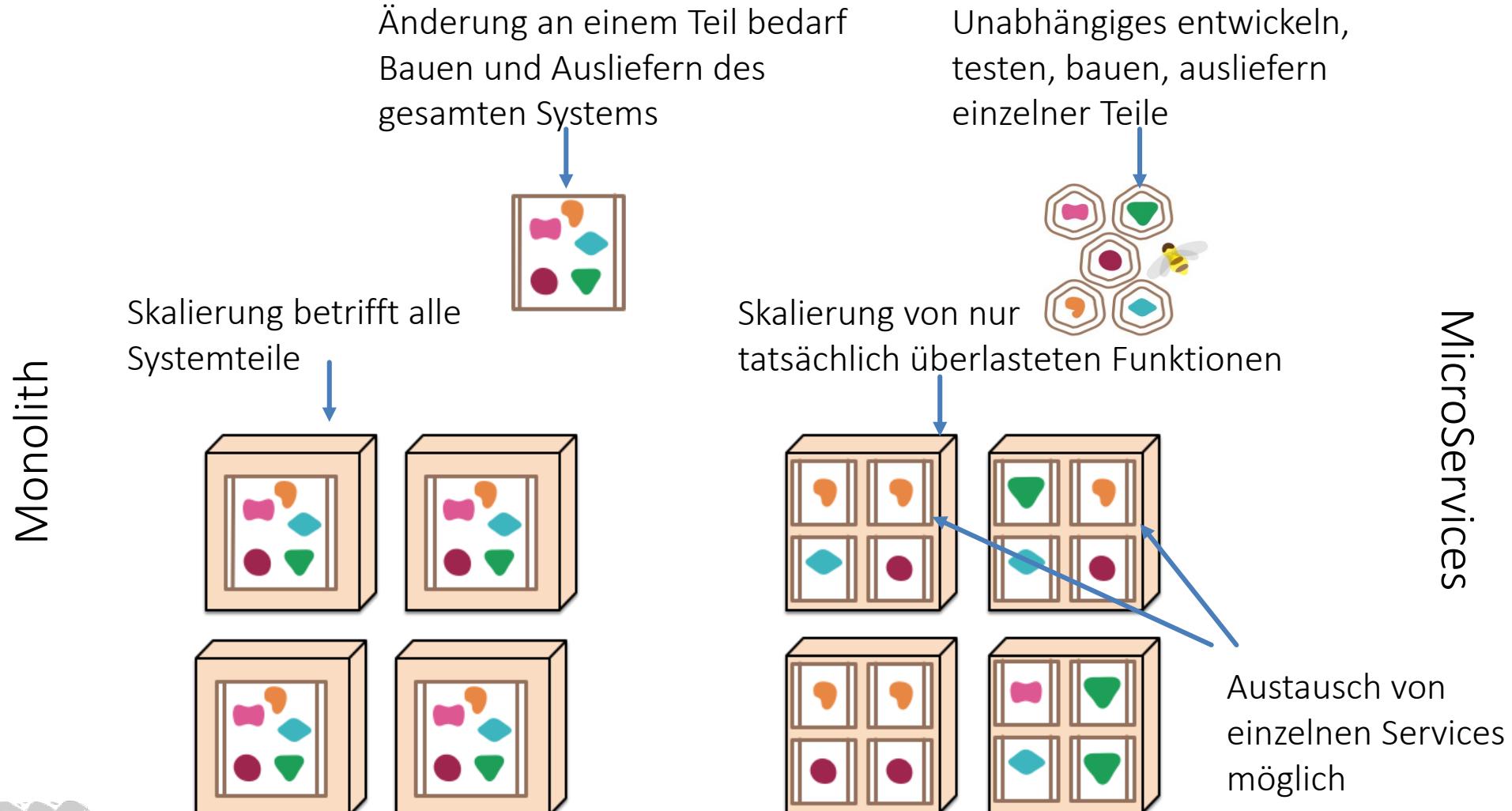
Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. -- Melvyn Conway, 1967

Von MicroService zum Softwaresystem

- Anwendung besteht aus einer Menge an MicroServices
 - Jeder hat eigene Prozesse und Daten
 - Leichtgewichtige Kommunikation (meist REST-API)
 - Unabhängig voneinander auslieferbar, testbar, entwickelbar
- Maximale Unabhängigkeit der MicroServices

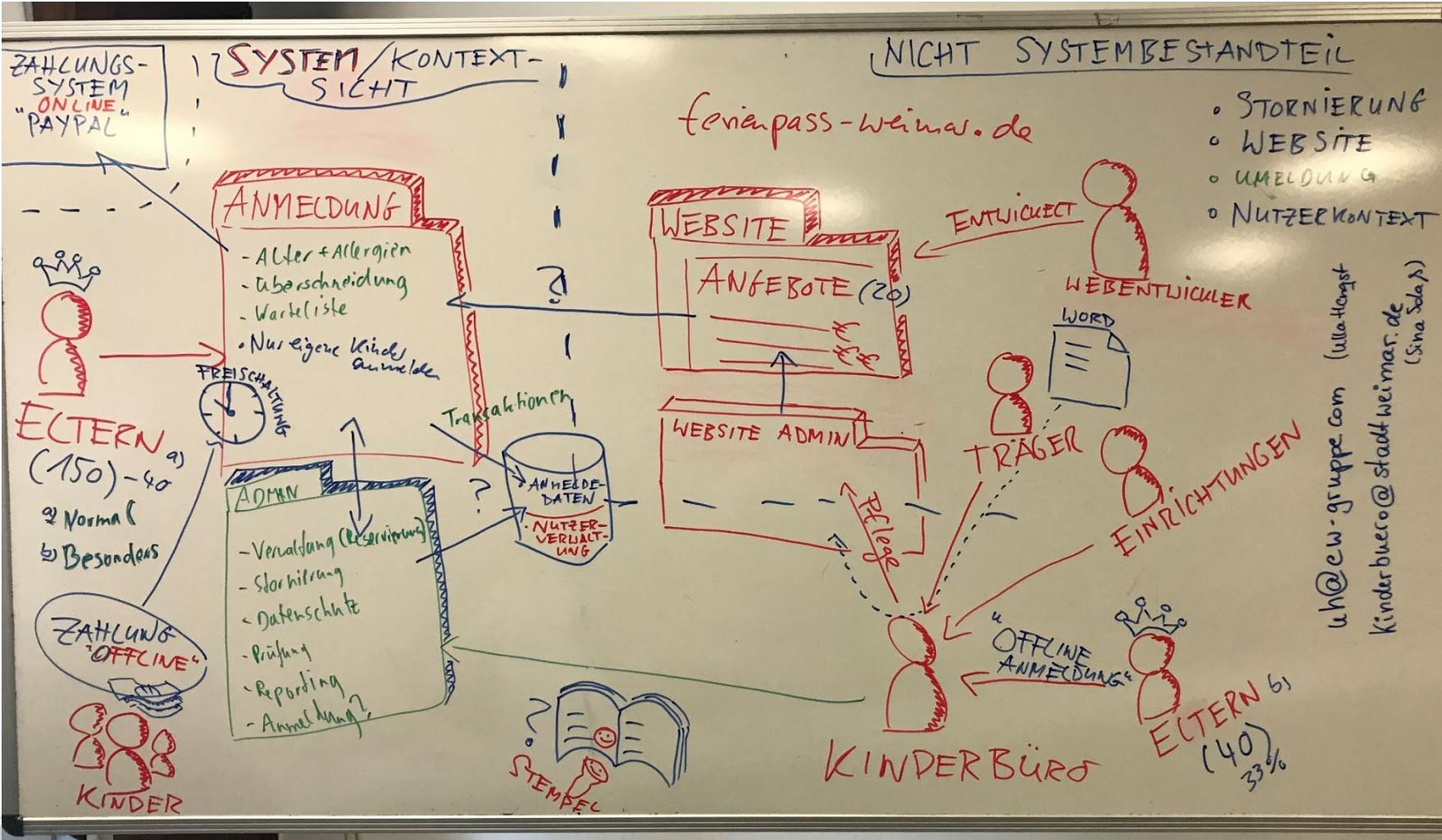


Monolith vs. MicroServices



by James Lewis & Martin Fowler

Erfahrungen Ihrer VorgängerInnen



Und in der Praxis?



NETFLIX



UBER



ebay

Kritische Diskussion

- Was sind mögliche Nachteile einer MicroService Architektur?
- Welche Besonderheiten und Voraussetzungen sollten beim Einsatz dieser Architektur betrachtet werden?
 - MicroServices für Word oder Virenscanner?

Nachteile von MicroServices

- Benötigt richtigen „Mindset“ der Entwicklerinnen und klare Definition von eines MicroServices (z.B. Mittels Domain-Driven Design Entwurfsmethodik)
- Immer noch hoher Kommunikationsaufwand zwischen Teams
- Moderne DevOps Pipeline erforderlich, Stichwort: Continuous Integration and Delivery
- Technologien entwickeln sich schnell weiter
- Architektur resultiert in ein verteiltes System mit all seinen Vor- und Nachteilen

Kriterien zur Auswahl von Architekturmustern

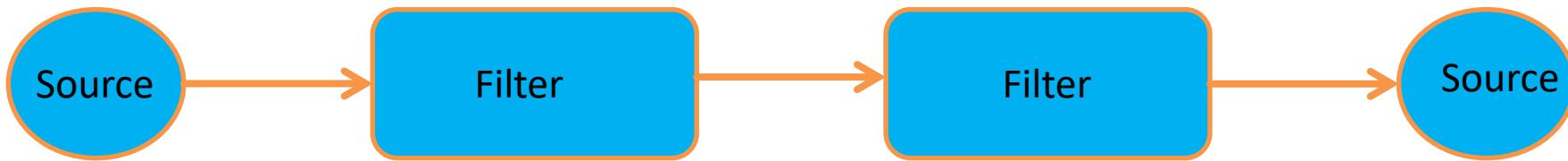
- Welche Struktur des Softwaresystems?
 - Komponenten-orientiert, monolithisch, Schichten, Pipes and Filters
- Wie kommunizieren die Sub-Systeme?
 - Ereignis-basiert, Publish-Subscribe , ansynchrone Nachrichten
- Wie sind die Sub-Systeme verteilt?
 - Client-Server, shared nothing, Peer2Peer, service-orientiert, Cloud

Pipes and Filters



Aufbau

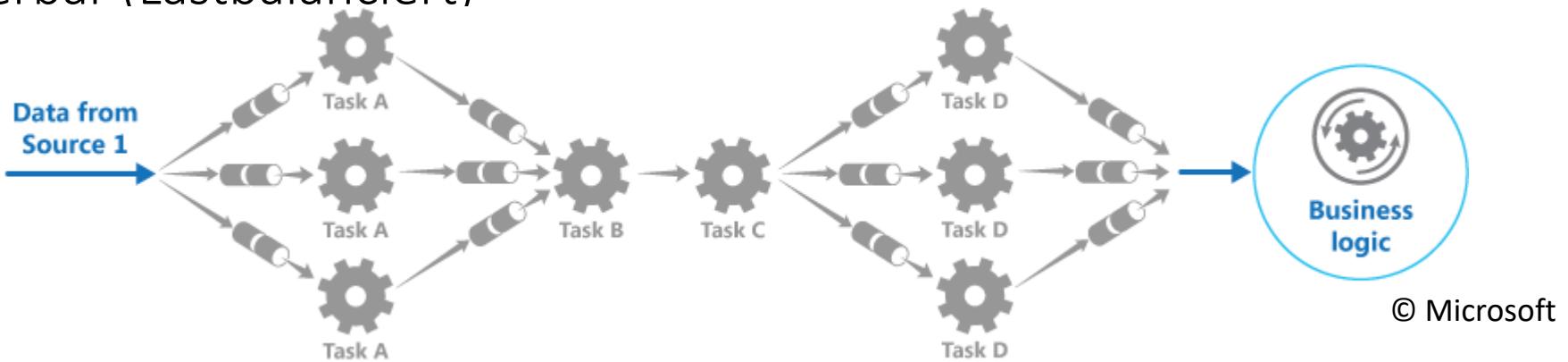
- Pipe: Verbindungsglied, welches Daten von einem Filter zu einem anderen weiterleitet
- Filter: Transformiert Daten, die es durch eine Pipe bekommen hat



- Vorteile:
 - Filter können einfach hinzugefügt und herausgenommen werden
 - Robust, performant, skalierbar, gute Wartbarkeit!

Vor- und Nachteile

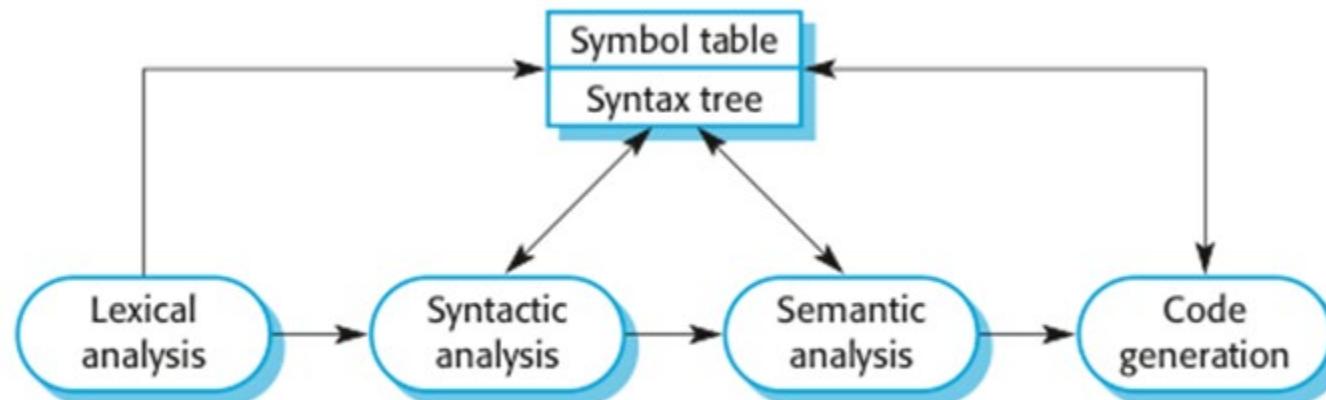
- Parallelisierbar (Lastbalanciert)



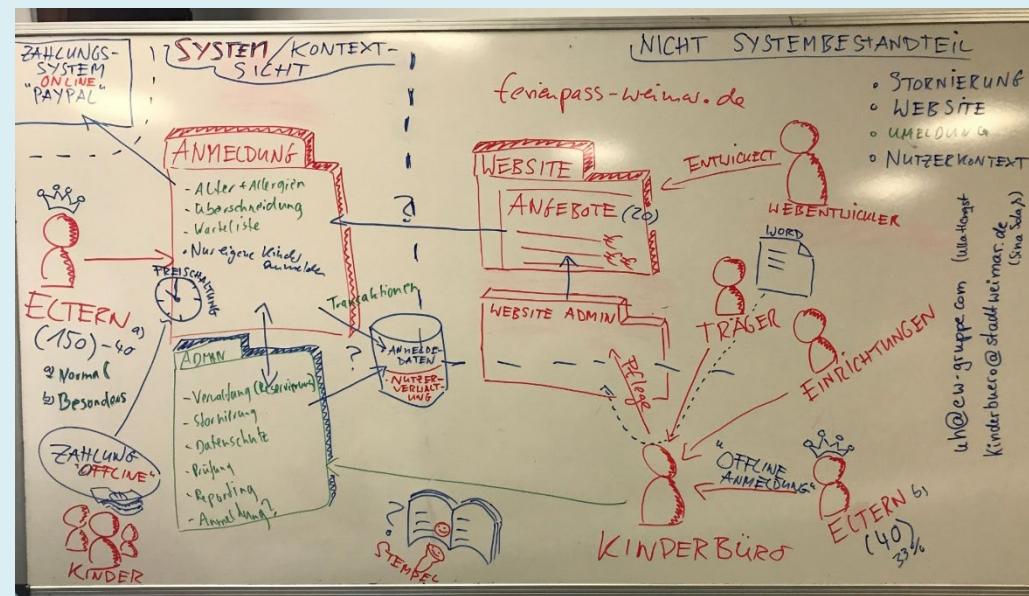
- Probleme:
 - Komplexität steigt
 - Bufferoverflow möglich
- Wann anwendbar?
 - Anwendung kann in mehrere unabhängige Teile zerlegt werden
 - Wenn viele Transformationen auf Daten nötig sind
 - Wenn Flexibilität notwendig ist (z.B. in der Cloud)

Beispiele

<i>Domain</i>	<i>Data source</i>	<i>Filter</i>	<i>Data sink</i>
Unix	<code>tar cf - .</code>	<code>gzip -9</code>	<code>rsh picasso dd</code>
CGI	HTML Form	CGI Script	generated HTML page

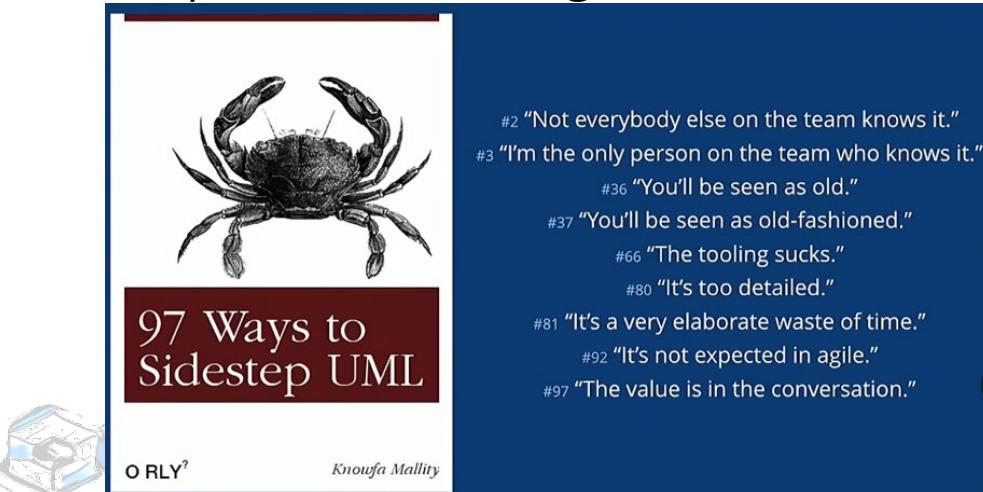


Modellierung von Softwarearchitektur

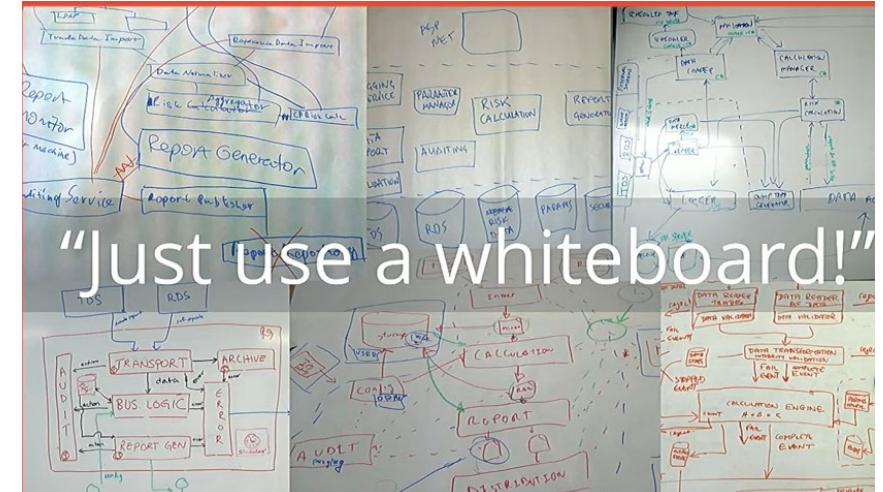


Unterschiedliche Sichten auf Softwarearchitektur

- Ziel: Modellierung des Verhaltens, der Struktur, der Logik und der Infrastruktur eines Systems, so dass alle Anforderungen erfüllt und keine Bedingungen verletzt werden.
- Architektur überbrückt Anforderungen zur Implementierung
 - Aber: Wie machen?
- Verwende eine sinnvolle Modellierungssprache für die unterschiedlichen Sichten auf das System und integriere sie zu einem gesamtheitlichen Bild

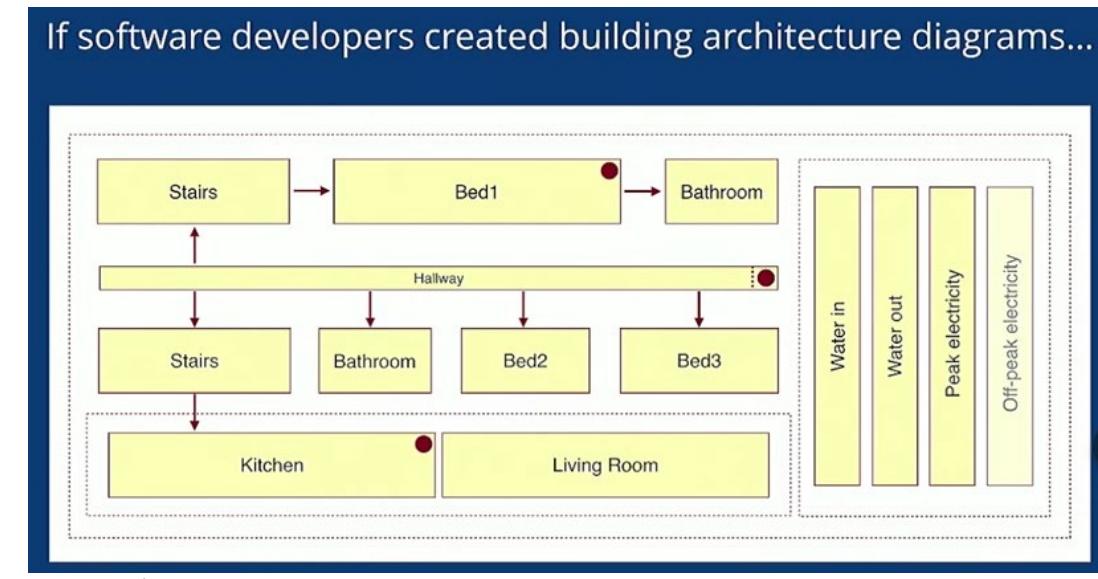
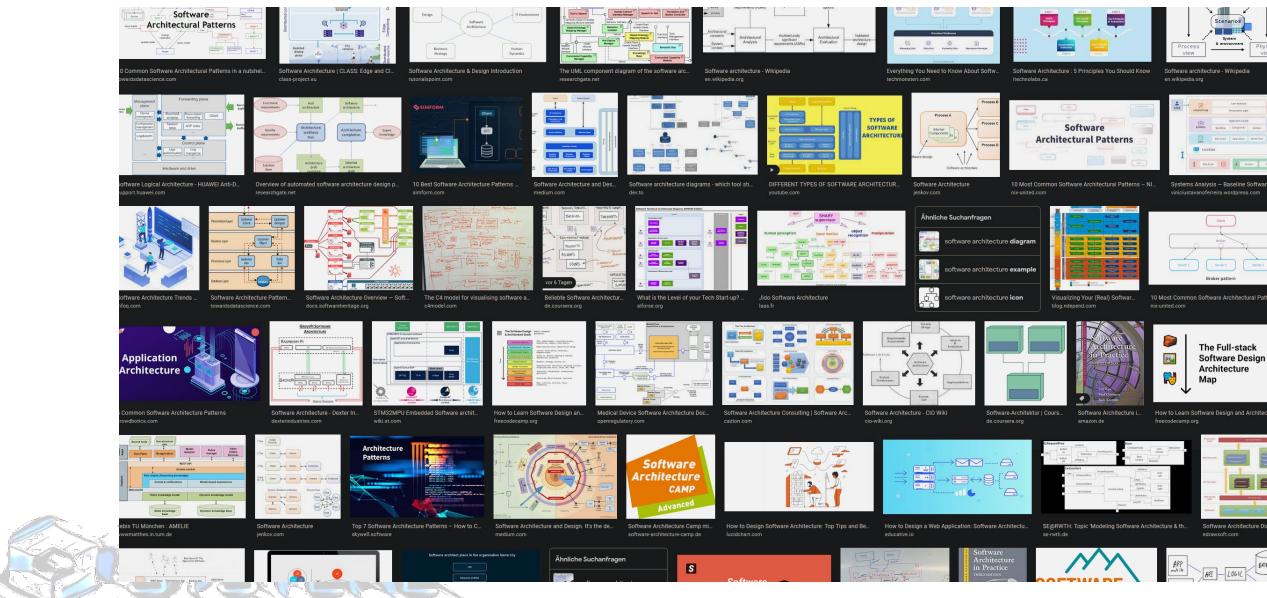


Beide Extreme funktionieren nicht allgemein in der Praxis.
Aber was dann machen?



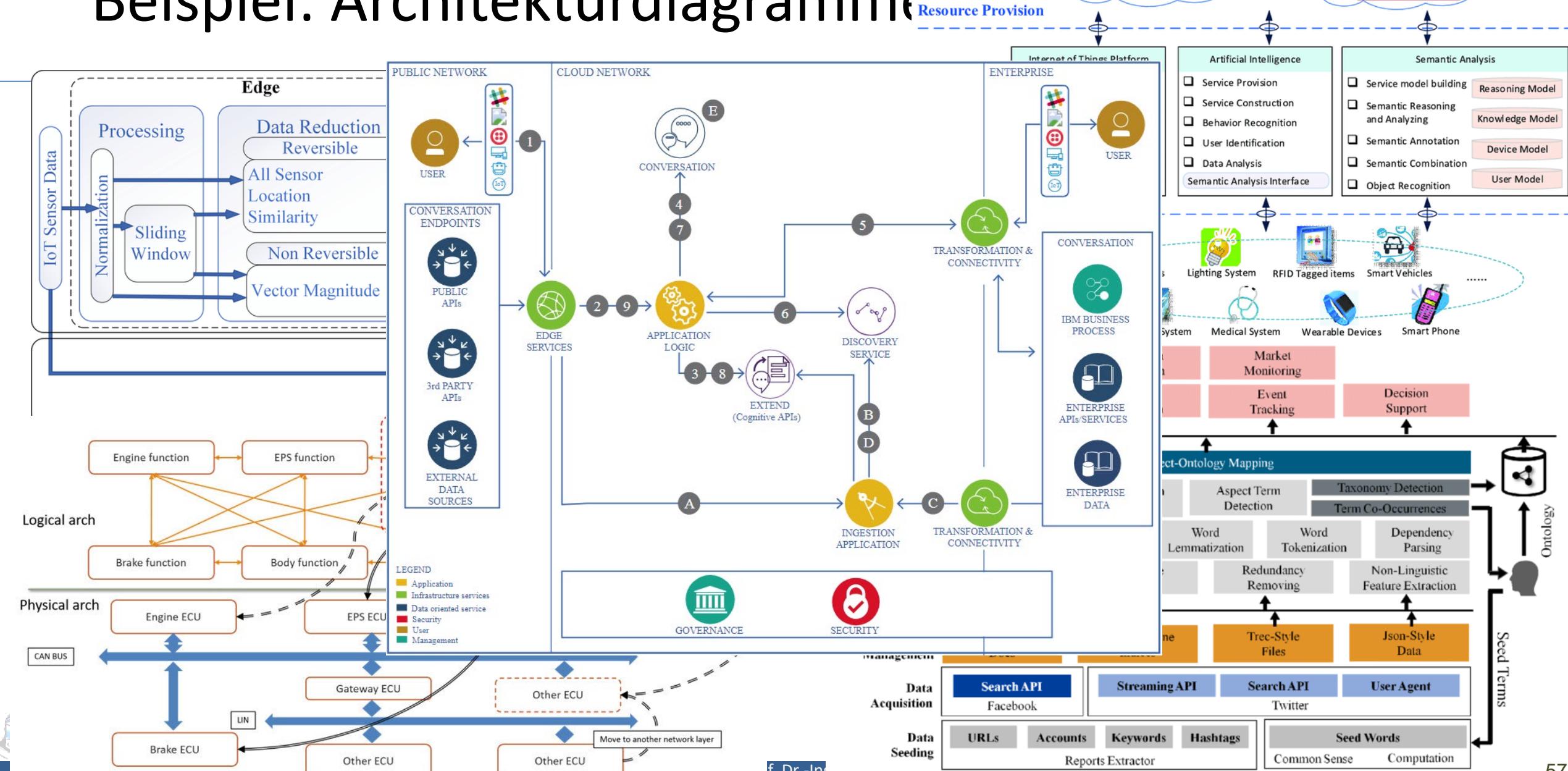
Probleme von Architekturdiagrammen

- Boxen: Unterschiedliche Formen, Größen, Farben, Kanten, Rahmen, etc.
- Linien: Formen, Pfeile, Farben, Muster, Größe, Dicke, Verbindungen
- Symbole und Formen: Unterschiedliche Bedeutungen, unklare Bilder, Akronyme
- Fehlende Elemente: Unverbundene Boxen, fehlende Akteure, Legende?
- Arrangement: Verschachtelung, Layout



Source: Simon Brown

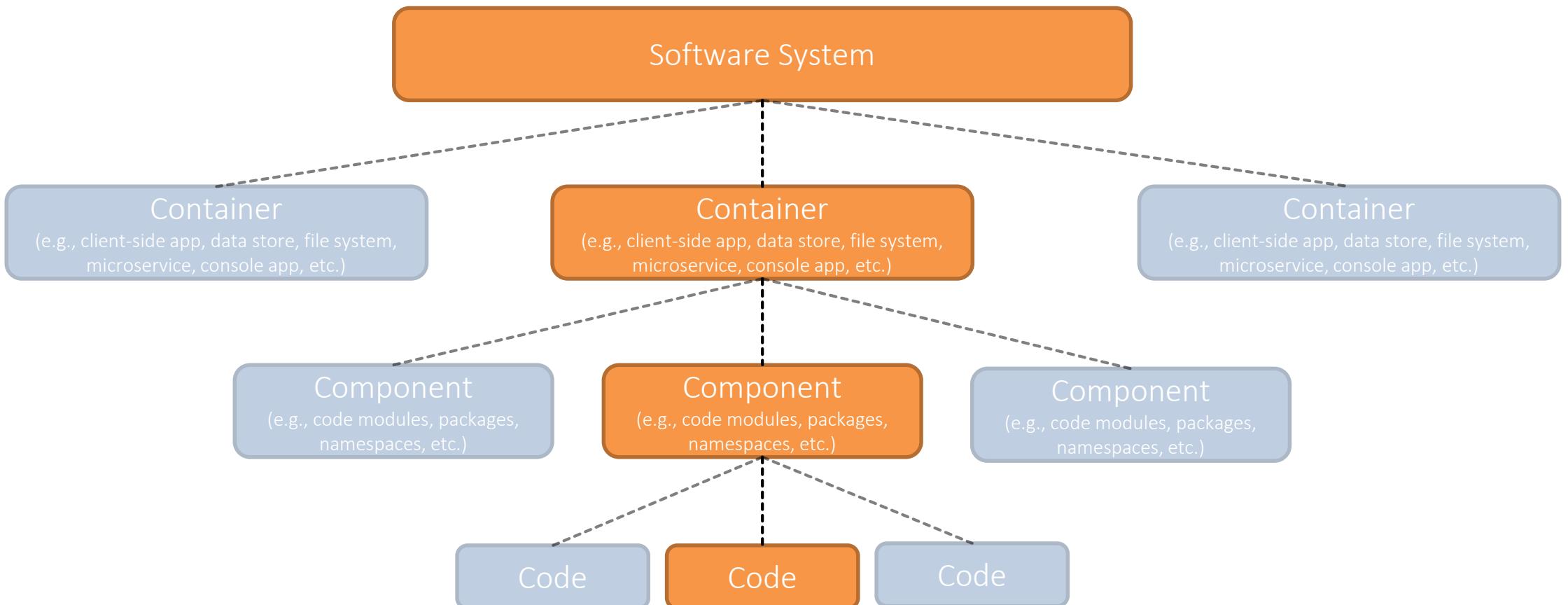
Beispiel: Architekturdiagramm



Einsicht: Abstraktionen elementar; Notationen zweitrangig

- Notationen sind komplex und können unterschiedliche Dinge für unterschiedliche Personen bedeuten
- Notationen sind oft fehlerhaft verwendet, falsch interpretiert, ad-hoc erfunden oder vergessen
- Abstraktionen sind das Schlüsselement eines jeden Systems
- Egal was für eine Notation man verwendet, Abstraktionen sind vorhanden

Modellierung der Architektur mit C4



C4 Model Overview

- Idee: Beschreibung des Softwaresystems in unterschiedlichen Detaillevel.
- Kein Zwang zur Modellierungsreihenfolge (button-up oder top-down).
- Kein Zwang alle Levels zu modellieren.
- C4 = Context, Containers, Components, Code

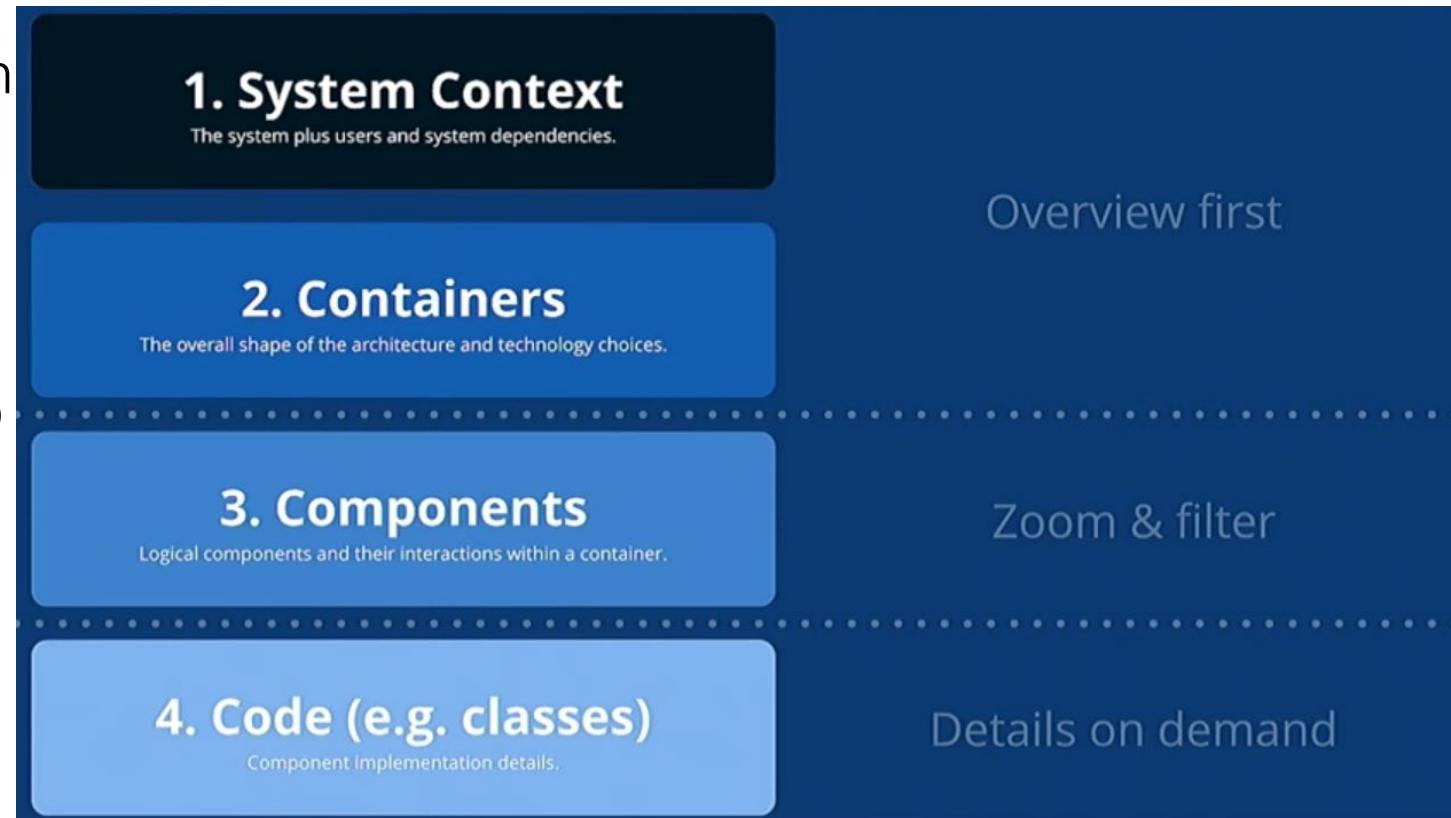


Diagramm == Karte für Entwicklerinnen

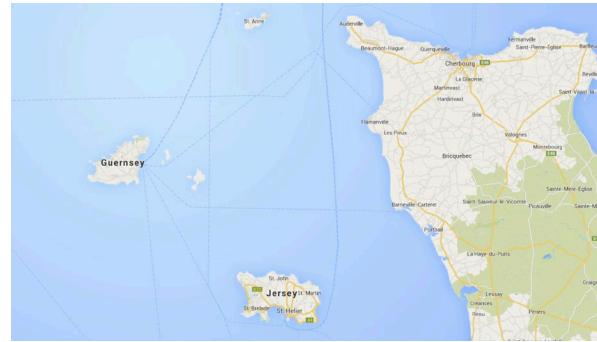
Detailliert; Realwelt Aussehen, aber wo bin ich?



Zooming out; Wir sind auf einer Insel, aber auf welcher?



Wir sind auf Jersey, neben einer Insel / Kontinent?

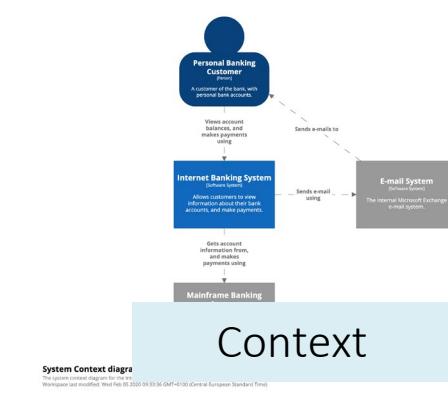
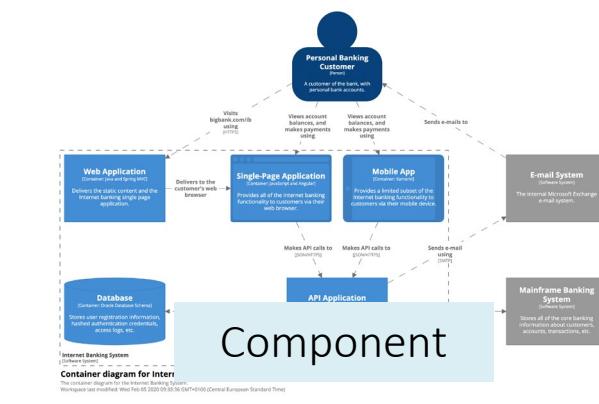
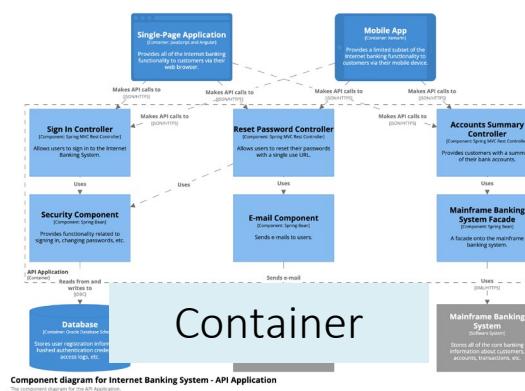
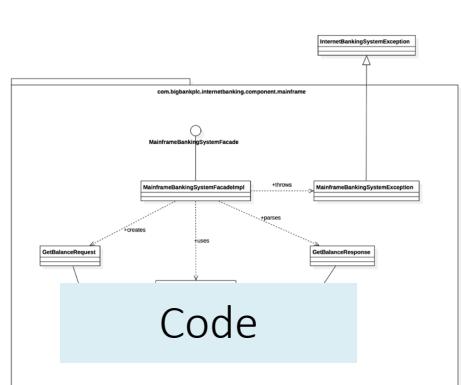


Wir sind nah an Frankreich!

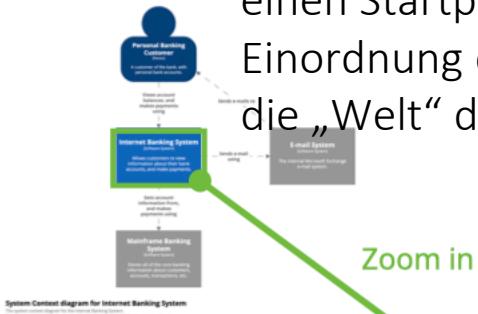


Zoom-Levels können unterschiedliche Geschichten erzählen und verschiedene Aspekte zu unterschiedlichen Adressaten des Systems hervorheben.

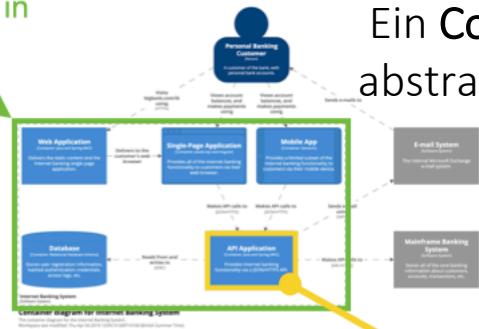
In der SW Entwicklung haben wir unterschiedliche Adressaten (Nutzende, Kunden, ArchitektInnen, Testende, Data Scientists, etc.). Also, sollten wir auch unterschiedliche Aspekte beleuchten.



Das System Context-Diagramm stellt einen Startpunkt her, der die Einordnung des Softwaresystems in die „Welt“ darstellt.



Zoom in



Zoom in

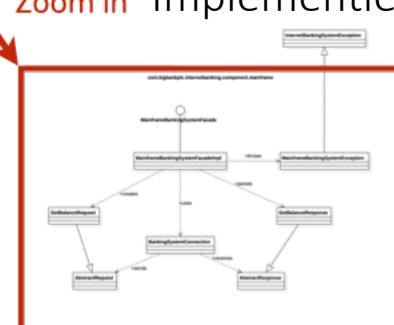
Ein Container-Diagramm zeigt die abstrakten Bausteine des Systems.

Ein Component-Diagramm zeigt die Komponenten / Module eines Containers.



Zoom in

Ein Code (z.B. UML Klassen-)Diagramm zoomt in eine individuelle Komponente, um deren Implementierung darzustellen.



Level 1
Context

Level 2
Containers

Level 3
Components

Level 4
Code

Abstraktionen: Elementare Elemente

Person: Nutzer eines Systems (Aktoren, Rolen, etc.)

Softwaresystem: Software, die Wert für den Kunden liefert (schließt nicht-Menschen mit ein, wie z.B. andere Systeme die darauf angewiesen sind bzw. vice versa).

Container: Anwendungen oder Datenhaltung, die ausgeführt werden muss, so dass das gesamte System operieren kann. Auslieferbare/Ausführbare Einheit oder Laufzeitumgebung.

Component: Gruppe von verwandter Funktionalität hinter einem wohl-definierten Interface (Schnittstelle). Komponenten im selben Container werden im selben Prozessraum ausgeführt.

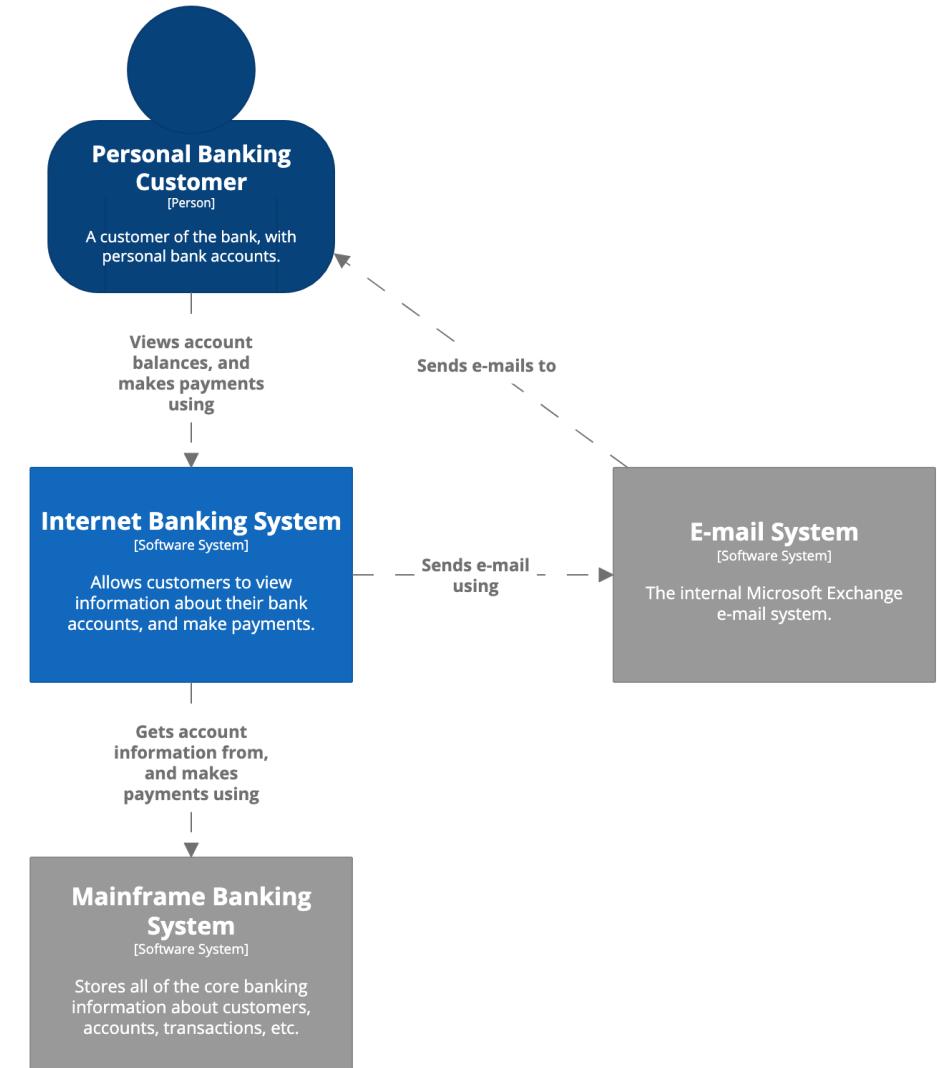


C1: Context-Diagramm

Globale Sicht auf das System und wie es in der aktuellen Systemlandschaft, einschließlich möglichen Nutzern und anderen Systemen sich integriert.

Ideale Sicht, um das System innerhalb anderer Datenverarbeitenden bzw. –produzierenden Systemen zu platzieren und deren Interaktion zu definieren.

Keine Angabe von Technologien, Protokollen und Systemdetails benötigt. Hauptsächlich zur Kommunikation mit Nicht-technischen Personen verwendet.

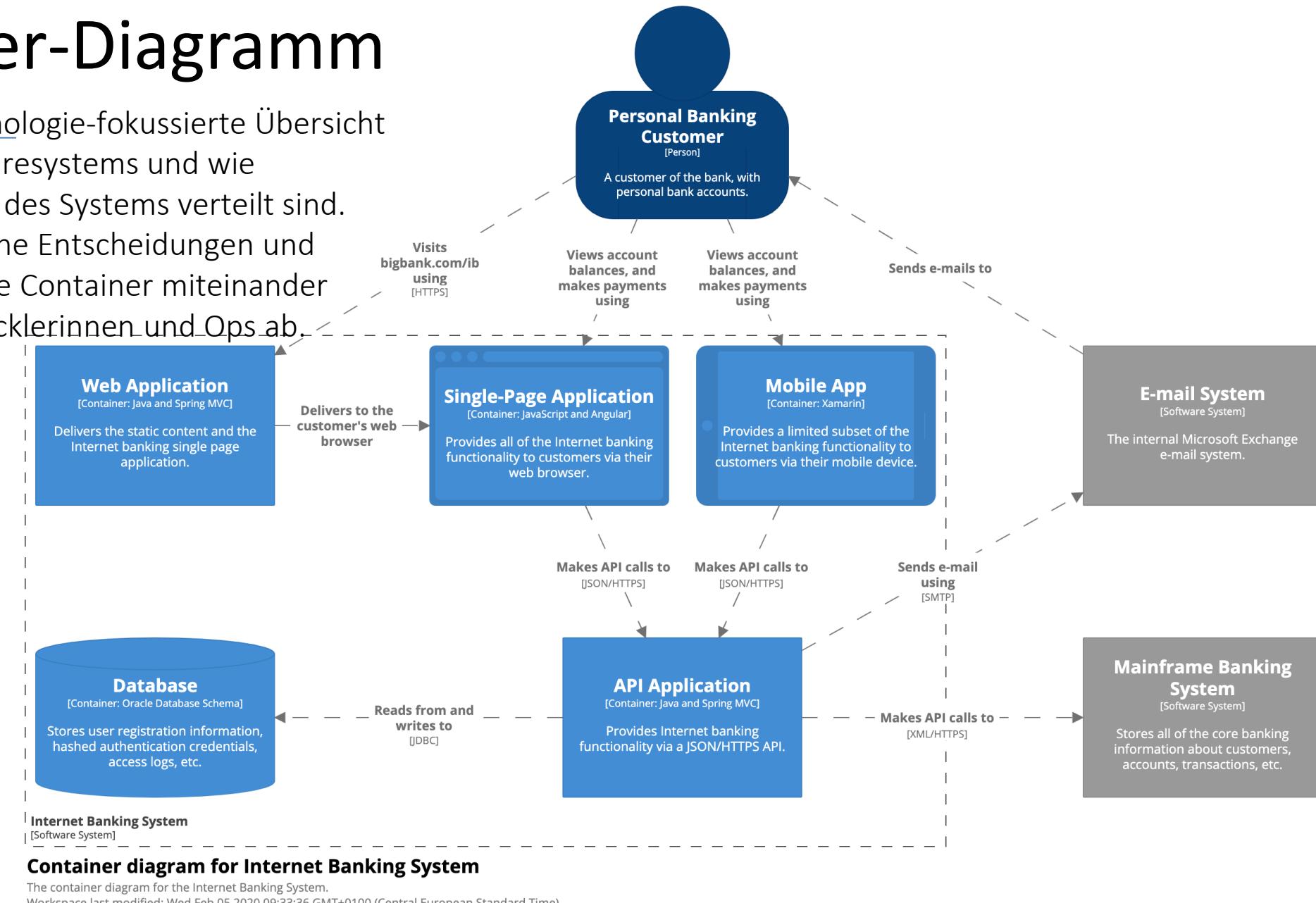


System Context diagram for Internet Banking System

The system context diagram for the Internet Banking System.
Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

C2: Container-Diagramm

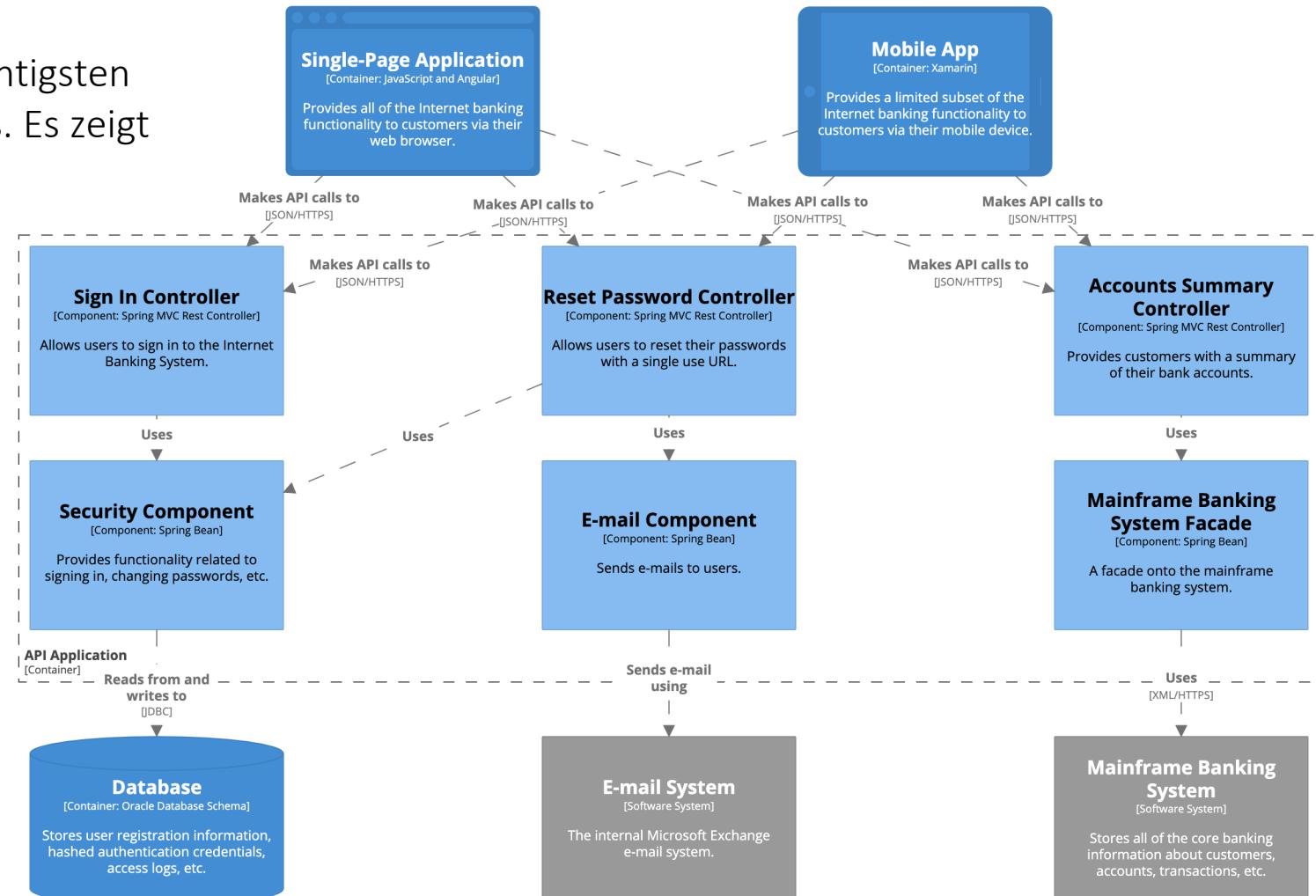
Repräsentiert eine grobe, technologie-fokussierte Übersicht über die Architektur des Softwaresystems und wie Verantwortlichkeiten innerhalb des Systems verteilt sind. Beinhaltet wichtig technologische Entscheidungen und beschreibt, wie unterschiedliche Container miteinander kommunizieren. Zielt auf Entwicklerinnen und Ops ab.



C3: Component-Diagramm

Das Komponentendiagramm visualisiert die wichtigsten Bausteine (also Komponenten) eines Containers. Es zeigt deren Verantwortlichkeiten, Technologien und Implementierungsdetails.

Ziel auf Entwicklerinnen ab.



Component diagram for Internet Banking System - API Application

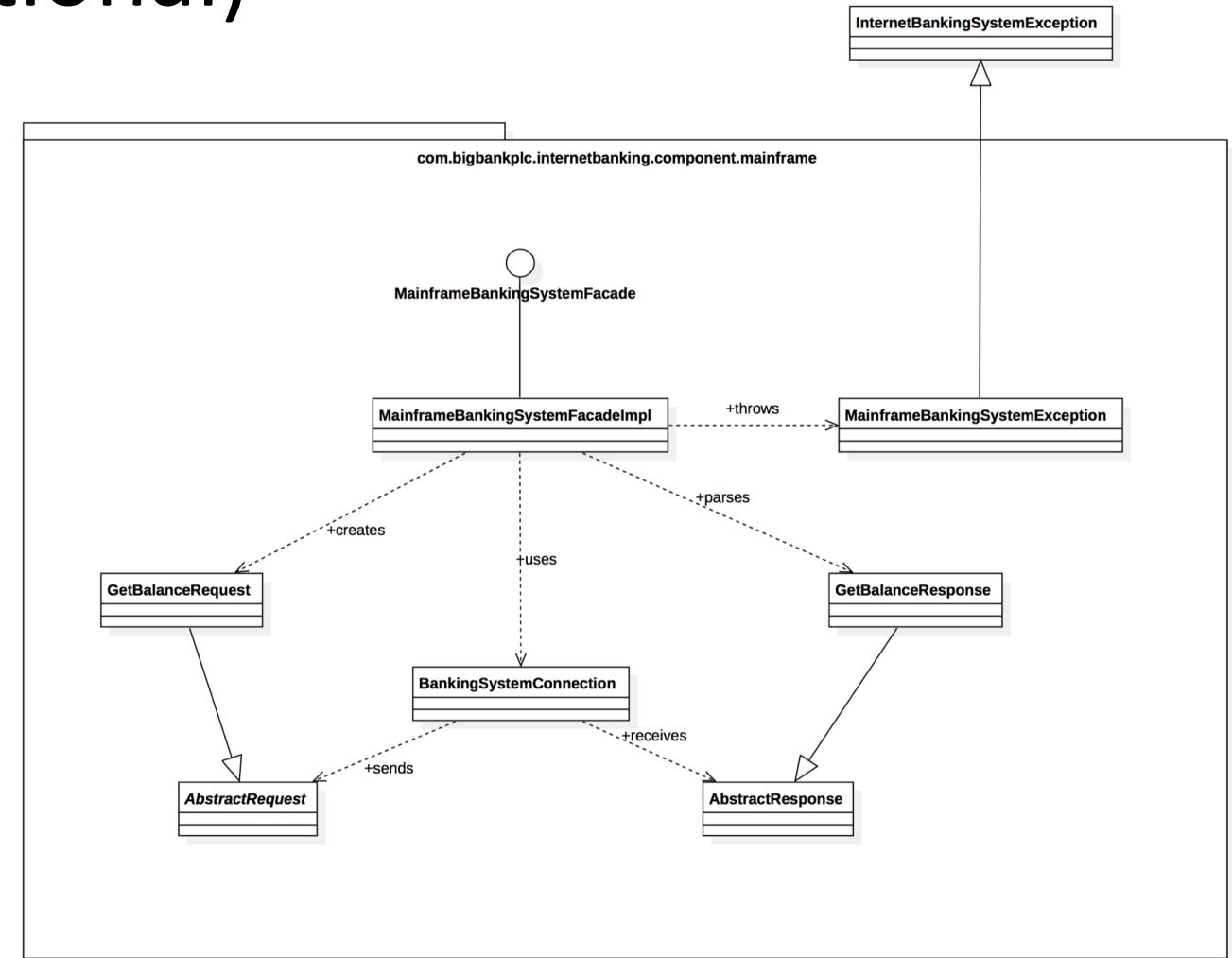
The component diagram for the API Application.

Workspace last modified: Wed Feb 05 2020 09:33:36 GMT+0100 (Central European Standard Time)

C4: Code-Diagramm (optional)

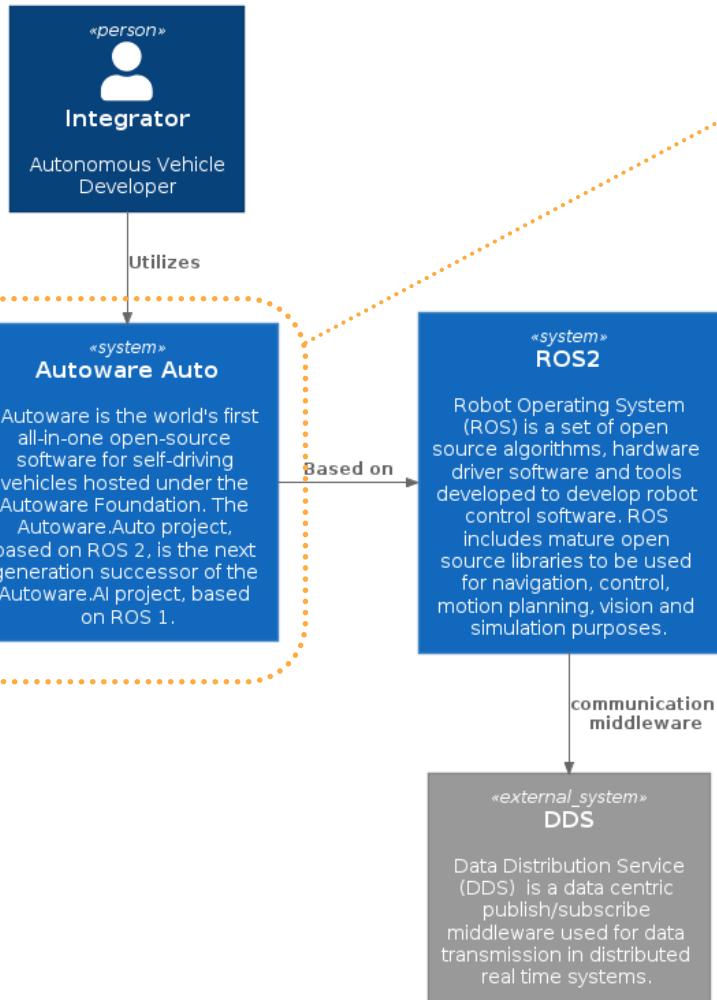
Das Code-Diagramm ist nicht spezifiziert und hängt vom Anwendungsfall ab. Oftmals werden UML-Klassendiagramme, ER-Diagramme oder Zustands- und Sequenzdiagramme verwendet.

Ein solches Diagramm könnte auch von einer IDE generiert werden. Nur für sehr komplexe Komponenten empfohlen.

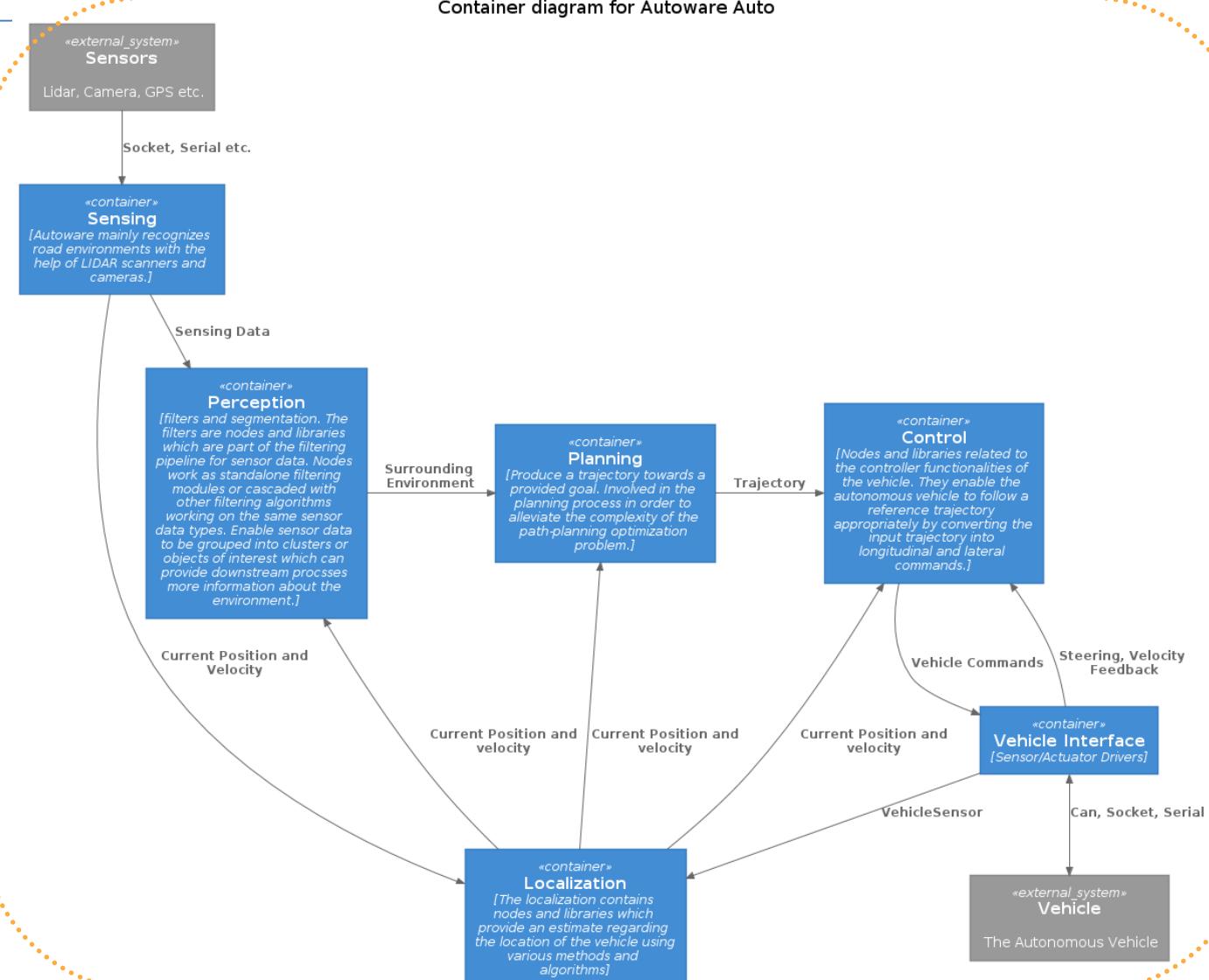


Bispiel: Autonomer Roboter mit C4

System Context diagram for AutoWare Auto and ROS2

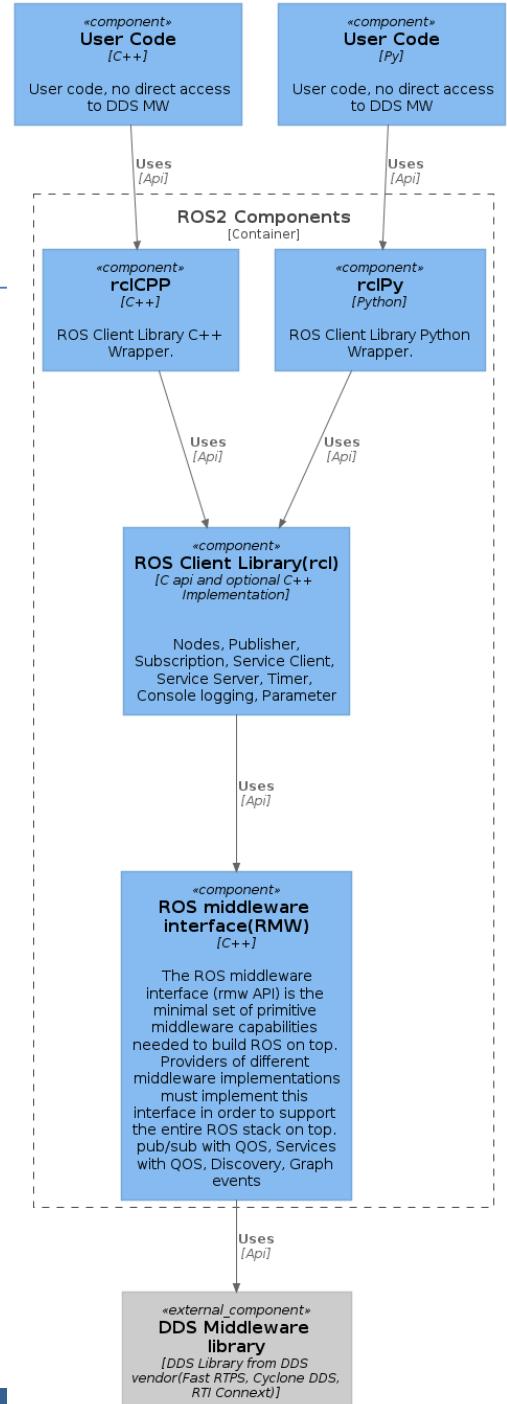


Container diagram for Autoware Auto

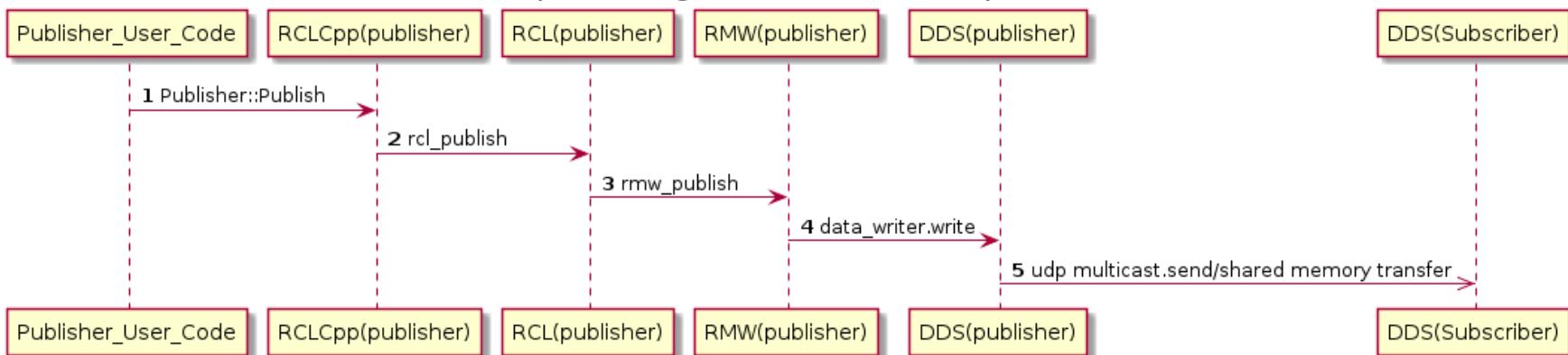


Beispiel: Autonomer Roboter mit C4

Component Diagram for ROS2 components



Sequence Diagram for Ros2 Mw components



Gute Architektur == Team Aufwand

- Vermeide Inselwissen und nehme unterschiedliche Perspektiven ein (Entwicklerin, Data Scientists, Nutzende, etc.) indem das Team sich für eine Architektur entscheidet
- Erfahrene Expertinnen kennen die Werkzeuge und Technologien für die sie Entscheidungen treffen müssen
- Domänenexpertinnen kennen die Schnittstellen und Konsequenzen mit der Umgebung sowie die Anwendungsszenarien
- Managers kennen organisatorische Randbedingungen (Komitees, Firmenpolitik, Ressourcen)



Was Sie mitgenommen haben sollten:

- Inwiefern wirkt sich die Architektur auf das Softwaresystem aus?
- Wie kann die Auswahl einer geeigneten Architektur den Entwurf / Implementierung vereinfachen?
- Was bedeutet Kopplung und Kohäsion auf Architekturebene?
- Was ist ein Architektur-Stil?
- Für welche Szenarien sind MVC oder Pipes and Filters sinnvoll?
- Was sind Limitierungen von monolithischen Systemen?
- Was sollten Schichten nicht auf Elemente in Schichten darüber Zugriff haben?
- Wann macht der Einsatz von MicroServices Sinn und wann nicht?
- Welche Kriterien für den Einsatz bestimmter Architekturmuster gilt es zu beachten?

Was Sie mitgenommen haben sollten:

- How does software architecture constrain a system?
- How does choosing an architecture simplify design?
- What are coupling and cohesion?
- What is an architectural style?
- For which application scenarios is MVC beneficial? For which is pipes and filters?
- Why shouldn't elements in a software layer “see” the layer above?
- What is the role of programming in model-driven architecture?

Literatur

- Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997 [Chapter 3, 4]
- *Software Engineering*, I. Sommerville, 7th Edn., 2004.
- *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999
- *Pattern-Oriented Software Architecture — A System of Patterns*, F. Buschmann, et al., John Wiley, 1996
- *Software Architecture: Perspectives on an Emerging Discipline*, M. Shaw, D. Garlan, Prentice-Hall, 1996

Literatur

- Service-Oriented Architecture (SOA) vs. Component Based Architecture. Helmut Petritsch (http://petritsch.co.at/download/SOA_vs_component_based.pdf)
- Microservices. James Lewis und Martin Fowler.
<https://martinfowler.com/articles/microservices.html>

