



# iSAQB Foundation Level

Beschreibung und Kommunikation von  
Softwarearchitekturen



# Agenda

---

1. LZ 3-1: Qualitätsmerkmale technischer Dokumentation erläutern und berücksichtigen (R1)
2. LZ 3-2: Softwarearchitekturen beschreiben und kommunizieren (R1)
3. LZ 3-3: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2 – R3)
4. LZ 3-4: Architektursichten erläutern und anwenden (R1)
5. LZ 3-5: Kontextabgrenzung von Systemen erläutern und anwenden (R1)
6. LZ 3-6: Querschnittskonzepte dokumentieren und kommunizieren (R2)
7. LZ 3-7: Schnittstellen beschreiben (R1)
8. LZ 3-8: Architekturentscheidungen erläutern und dokumentieren (R1 - R2)
9. LZ 3-9: Dokumentation als schriftliche Kommunikation nutzen (R2)
10. LZ 3-10: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)

# **Beschreibung und Kommunikation von Softwarearchitekturen**

---

- Dauer: 180 Min. Übungszeit: 60 Min.
- Wesentliche Begriffe:
  - (Architektur-)Sichten; Strukturen; (technische) Konzepte; Dokumentation; Kommunikation; Beschreibung; zielgruppen- oder stakeholdergerecht; Meta-Strukturen und Templates zur Beschreibung und Kommunikation; Kontextabgrenzung; Bausteine; Bausteinsicht; Laufzeitsicht; Verteilungssicht; Knoten; Kanal; Verteilungsartefakte; Mapping von Bausteinen auf Verteilungsartefakte; Beschreibung von Schnittstellen und Entwurfsentscheidungen; UML; Werkzeuge zur Dokumentation

# **LZ 3-1: Qualitätsmerkmale technischer Dokumentation erläutern und berücksichtigen (R1)**

- Softwarearchitekt:innen kennen die wesentlichen Qualitätsmerkmale technischer Dokumentation und können diese bei der Dokumentation von Systemen berücksichtigen bzw. erfüllen:
  - Verständlichkeit, Korrektheit, Effizienz, Angemessenheit, Wartbarkeit
  - Orientierung von Form, Inhalt und Detailgrad an Zielgruppe der Dokumentation
- Sie wissen, dass Verständlichkeit technischer Dokumentation nur von deren Zielgruppen beurteilt werden kann.

# Warum Softwarearchitekturen dokumentieren?

---

- Szenario
  - Sie kommen in ein neues Projekt oder neue Entwickler:innen kommen zu Ihnen ins Projekt.
- Typische Probleme
  - Wie sieht die Toolchain aus?
  - Wie sieht der Systemkontext aus?
  - Wie baue ich die Software?
  - Wie finde ich einen Einstieg? Gehören diese Packages zusammen? Habt ihr das irgendwo visualisiert?
  - Wie wird Authentifizierung und Autorisierung durchgeführt?
  - Ich habe hier eine Kleinigkeit verändert. Wieso sind nun alle Tests rot?

# Gängige Argumente gegen Dokumentation

---

***Der Code ist die Dokumentation...***

***In der Zeit können Features implementiert werden...***

***Erstellung von Diagrammen ist overhead...***

***Dokumentation ist nicht agil...***

***Dokumentation veraltet schnell...***

***Dokumentieren ist langweilig...***

# Warum Softwarearchitekturen dokumentieren?

---

- Dokumentierte Lösungen fördern die Einhaltung von **konzeptioneller Integrität**
- Dokumentation eignet sich als **Kommunikationsmittel**. Sie soll die Architekturarbeit unterstützen und für Architekten ein Instrument sein, um die Entwicklung zu begleiten
- Ermöglicht über einen langen Zeitraum den Überblick zu bewahren und auftretenden Fehlern und Problemen besser zu begegnen
- Der Code ist die Ist-Architektur. Ohne Dokumentation geht das Zielbild (Soll-Architektur) verloren. Dokumentation ermöglicht eine Soll-/Ist-Architekturbewertung

# Qualitätsmerkmale einer Dokumentation

---

- Dokumentation muss einige **Qualitätsmerkmale** aufweisen, um von der **Zielgruppe** akzeptiert zu werden.
- Nur dann wird eine Dokumentation gelesen und kann die **Architekturarbeit unterstützen** und somit **Nutzen** stiften.
- Ca. **30% bis 50%** der Wartungs- und Änderungskosten verursacht das **Verstehen** von bestehendem Code, Konzepten und Strukturen.
  - Dokumentation sollte dies verbessern.
  - Dies ist aber nur möglich wenn notwendige, qualitative Eigenschaften erfüllt werden.

# Qualitätsmerkmale einer Dokumentation

---

## Zielgruppenadäquat

- Bedürfnisse unterschiedlicher Zielgruppen (Product Owner / Entwicklung) müssen abgedeckt sein

## Korrektheit

- Veraltete und fehlerhafte Dokumentation findet keine Akzeptanz und führt zu Fehlern in der Umsetzung

## Verständlichkeit

- Leser / Zielgruppe abholen und deren Sprache verwenden
- Begriffsklarheit und einheitliche Begriffsverwendung (Glossar)  
Mehrdeutigkeit vermeiden
- Abbildungen und Diagramme einsetzen

# Qualitätsmerkmale einer Dokumentation

---

## Wartbarkeit

- Hierarchisch organisiert
- Modular aufgebaut
- Redundanzfrei

## Angemessenheit und Effizienz

- Kurzgefasst
- Durchsuchbarkeit
- Navigierbarkeit
- Nur so viel wie nötig und sinnvoll

# Qualitätsmerkmale einer Dokumentation

---

## Weitere Grundprinzipien

- Dokumentiere die **Gründe**, die zu einer Entwurfsentscheidung geführt haben (**WARUM**)
- Erkläre die eingesetzte **Notation** oder verweise auf den Standard
- Verwende **Legenden**, um Abbildungen unmissverständlich zu beschreiben
- Erkläre wichtige Elemente im Diagramm und dessen Beziehungen
- **7 +/- 2 Elemente Regel** für Diagramme
- **Standardisierte Strukturen** für die Dokumentation verwenden

# Ob die Dokumentation verständlich und sinnvoll ist, entscheidet der Leser

---

- Leser:innen (= Stakeholder) sollten Teil der Dokumentation sein (z.B. Benutzer im Systemkontext)
- Leser:innen steuern Dokumentationsinhalte bei, wie z.B. Qualitätsszenarien (LZ 4) oder geben Feedback  
Feedback vom Tester bezüglich Systemverhalten, Feedback vom Nutzer zum Thema Usability und Performance, Feedback vom Entwickler bezüglich geplanter Lösungsansätze
- Die wichtigste Regel: **Schreibe aus Sicht des Lesers!**  
Nur wenn auf die unterschiedlichen Zielgruppen aktiv eingegangen wird, wird die Dokumentation gelesen
- Ziel ist es, das Systems so zu beschreiben, als sei es bereits fertig  
Die Wörter *wird* oder *soll* vermeiden

*Tipp: [Zörner 2015] beschreibt den Produktkarton als Dokumentationsmittel des Systemziels.*

# LZ 3-2: Softwarearchitekturen beschreiben und kommunizieren (R1)

---

- Softwarearchitekt:innen:
  - können Architekturen stakeholdergerecht dokumentieren und kommunizieren und dadurch unterschiedliche Zielgruppen adressieren, z. B. Management, Entwicklungsteams, QS, andere Softwarearchitekt:innen sowie möglicherweise zusätzliche Stakeholder
  - können die Beiträge unterschiedlicher Autorengruppen stilistisch und inhaltlich konsolidieren und harmonisieren
  - kennen den Nutzen von Template-basierter Dokumentation.
- Softwarearchitekt:innen wissen, dass verschiedene Eigenschaften der Dokumentation von Spezifika des Systems, seinen Anforderungen, Risiken, dem Entwicklungsvorgehen, der Organisation oder anderen Faktoren abhängen.

# **Stakeholderidentifikation zum Projektbeginn ist auch für die Dokumentation relevant**

---

- Die Intention der Zielgruppe, d.h. das Interesse des Lesers an der Dokumentation, müssen sich die Autor:innen bewusst machen
- Die Zielgruppe zu verstehen, ermöglicht erst das Schreiben aus Sicht der Leser:innen

# Interessen der Stakeholder

| Zielgruppe         | Interesse an der Dokumentation   | Ziel   |
|--------------------|--|--|
| Architekturteam    | Lösungen erarbeiten, festhalten, dokumentieren   | Architekturarbeit unterstützen                   |
| Entwickler:innen   | Orientierung finden, Vorgaben und Prinzipien entnehmen, Freiheitsgrade verstehen, Ansprechpartner identifizieren                                 | Umsetzung und Weiterentwicklung                  |
| Auftraggeber:innen | Das große Ganze der Lösung begreifen, Einflussfaktoren überprüfen, Nachvollziehbarkeit von Entscheidungen, Szenarien definieren und priorisieren | Architektur nachvollziehbar und bewertbar machen |
| Betrieb            | Systembestandteile verstehen, bei Betriebsstörung entscheidende Komponenten kennen, Ansprechpartner kennen                                       | System betreiben                                 |

# Dokumentationstemplates

---

**Template-/schablonenbasierte Dokumentation** hilft, ein Dokument zielgruppengerecht zu strukturieren

Verwendung von Templates / Schablonen

- Adressiert die Qualitätsmerkmale **Verständlichkeit** und **Navigierbarkeit, Wartbarkeit** sowie das **Auffinden von Informationen**
- Ist eine **Best Practice**, die den **Arbeitsaufwand reduziert** und das **Erstellen** der Dokumentation **vereinfacht**
- Z.B.  
Dokumentenvorlage für die Architektdokumentation,  
Schablonenbeschreibungen für Sichten, Schnittstellen oder  
technische Konzepte

# Mehrwert eines Dokumentationstemplates

Templates enthalten Vorschläge für

- Kategorien
- Struktur
- Reihenfolge
- Format

Fachlich

Querschnitt

Details

Der Autor kann sich voll auf den Inhalt konzentrieren,  
statt auf die Form.

Technisch

Zielgruppen werden besser angesprochen.



# **LZ 3-3: Notations-/Modellierungsmittel für Beschreibung von Softwarearchitektur erläutern und anwenden (R2 – R3)**

- Softwarearchitekt:innen kennen mindestens folgende UML-Diagramme zur Notation von Architektsichten:
  - Klassen-, Paket-, Komponenten- (jeweils R2) und Kompositionsstrukturdiagramme (R3)
  - Verteilungsdiagramme (R2)
  - Sequenz- und Aktivitätsdiagramme (R2)
  - Zustandsdiagramme (R3)
- Softwarearchitekt:innen kennen Alternativen zu UML, beispielsweise (R3):
  - ArchiMate, siehe [ArchiMate]
  - für Laufzeitsichten beispielsweise Flussdiagramme, nummerierte Listen oder Business-Process-Modelling-Notation (BPMN).

# Modellierungssprache und andere Mittel Entwürfe und Konzepte zu dokumentieren

---

## Modellierungssprachen

- **Unified Modelling Language (UML)**
- Entity Relationship Model (ER-Model)
- Business Process Model and Notation (BPMN)
- Systems Modeling Language (SysML)
- ArchiMate der Open Group



## Alternativen

- Textbasierte Modelle
- Textuelle Beschreibungen z.B. mit Schablonen
- Nummerierte Listen zur Beschreibung von Abläufen
- Domain-specific Languages (DSL)

# Unified Modelling Language

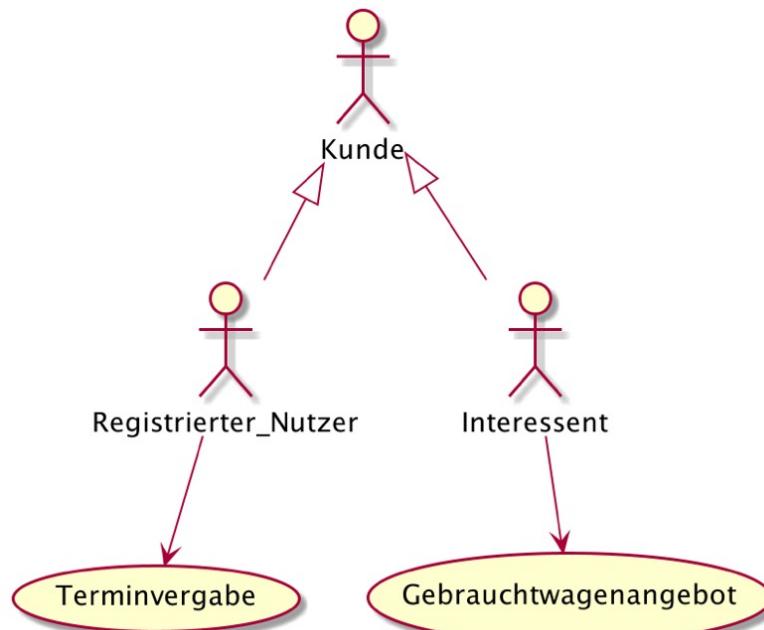
| Strukturdiagramme            | Verhaltensdiagramme           |
|------------------------------|-------------------------------|
| Klassendiagramm              | Aktivitätsdiagramm            |
| Kompositionsstrukturdiagramm | Anwendungsfalldiagramm        |
| Komponentendiagramm          | Interaktionsübersichtdiagramm |
| Verteilungsdiagramm          | Kommunikationsdiagramm        |
| Objektdiagramm               | Sequenzdiagramm               |
| Paketdiagramm                | Zeitverlaufsdiagramm          |
| Profildiagramm               | Zustandsdiagramm              |



Aus [Gharbi 2018]

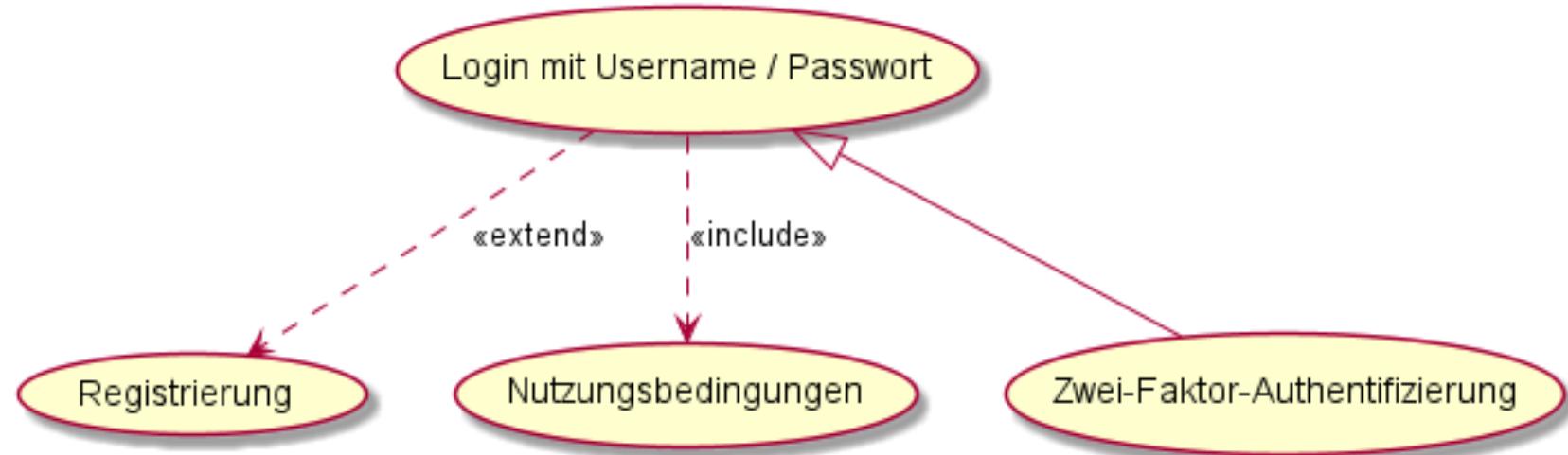
# Anwendungsfalldiagramm - 1/2

---



# Anwendungsfalldiagramm - 2/2

---



# UML Anwendungsfalldiagramm

---



Zum Nachlesen

- Englisch: Use case diagram
- Herleitung der Anforderungen in einer Use Case Analyse und grafische Visualisierung und Dokumentation mit Hilfe des Anwendungsfalldiagramms

# Inhalt

---



Zum Nachlesen

## Dargestellt werden

- die Systemgrenze
- die Anwendungsfälle des Systems
- Akteure, die mit dem System interagieren und Anwendungsfälle durchführen
- Nachbarsysteme, die für die Ausführung von Anwendungsfällen benötigt werden

## Beziehungen von Anwendungsfälle

- Import (include): Der Anwendungsfall wird fester Bestandteil des Wurzel Anwendungsfalls
- Erweiterung (extend): Unter bestimmte Bedingungen wird der Anwendungsfall Bestandteil des Wurzel Anwendungsfall
- Spezialisierung / Generalisierung: Spezialisierter Anwendungsfall erbt das Verhalten des generellen Anwendungsfalls

# Klassen-diagramm

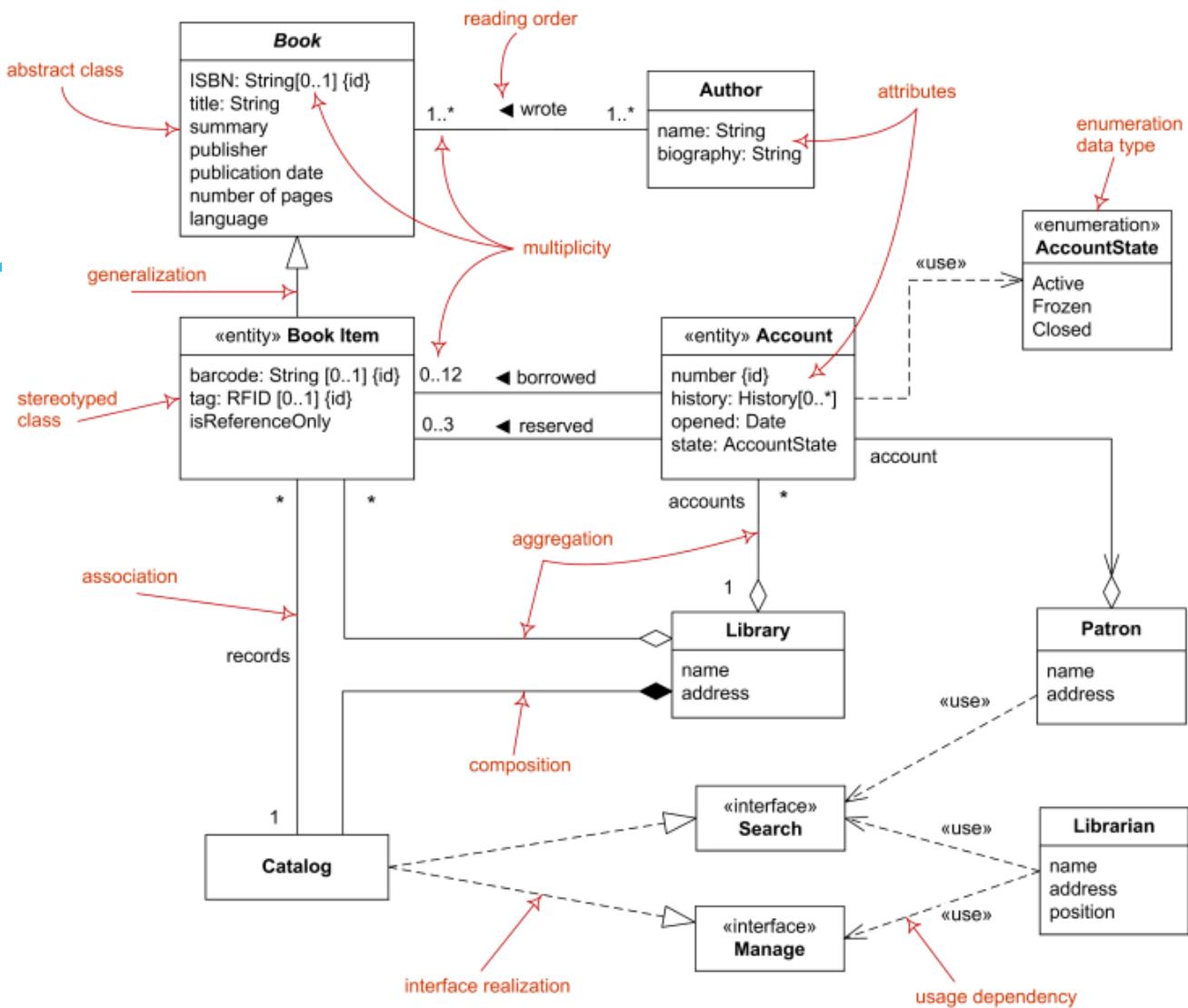


Abbildung aus  
[www.uml-diagrams.org]

# UML Klassendiagramm

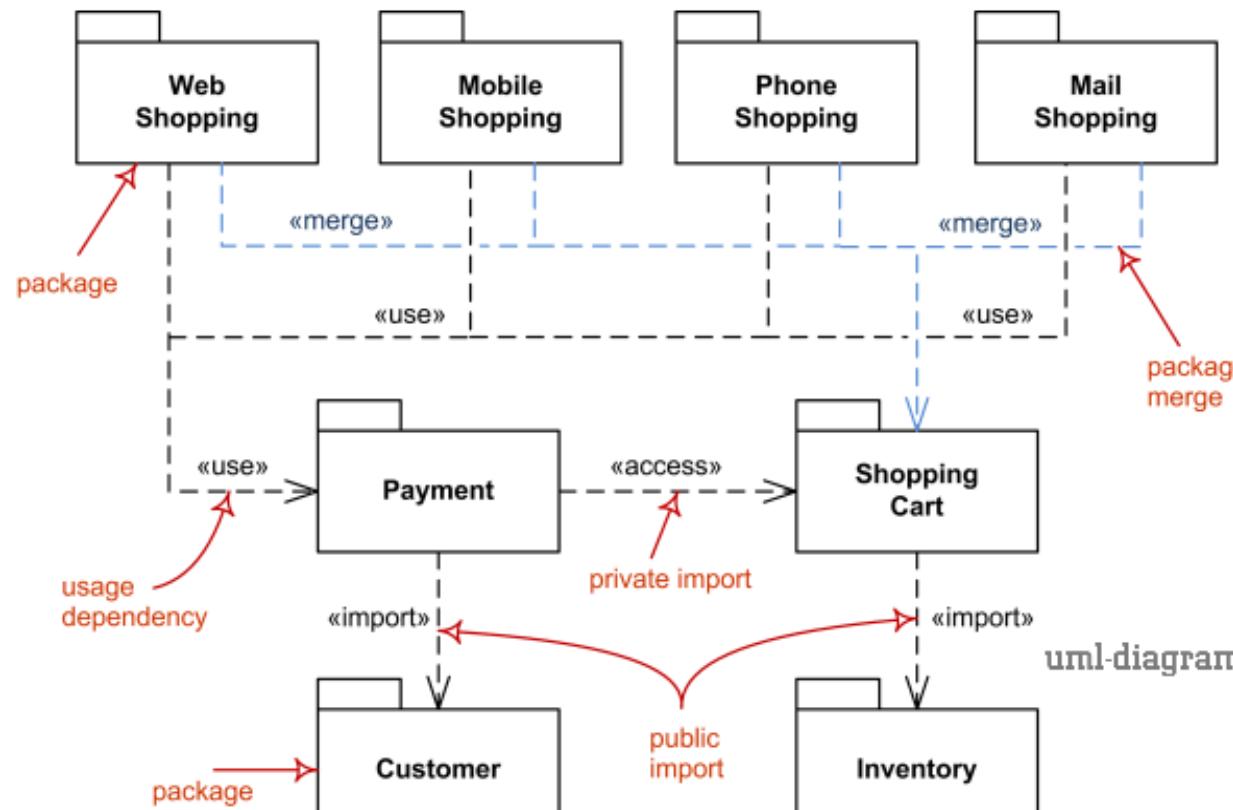
---



Zum Nachlesen

- Englisch: Class diagram
- Zeigt die statische Struktur von Klassen und deren Beziehungen untereinander
- Abstraktionen und Vererbungsstrukturen
- Einfache Assoziationen sowie Spezialformen der Assoziation (Komposition und Aggregation)
- Beschreibung von Methoden, Attributen, Sichtbarkeiten und Parametern

# Paketdiagramm



[uml-diagrams.org](http://uml-diagrams.org)

Abbildung aus  
[[www.uml-diagrams.org](http://www.uml-diagrams.org)]

# UML Paketdiagramm

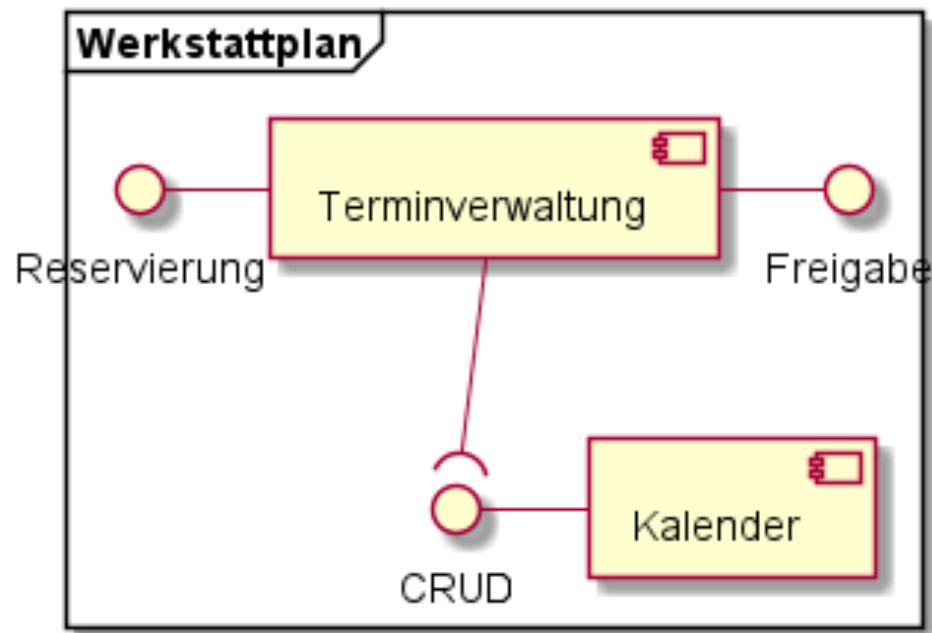
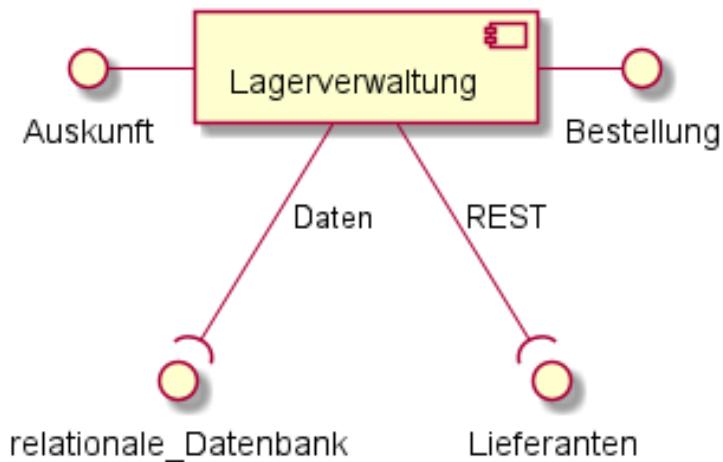
---



Zum Nachlesen

- Englisch: Package diagram
- Zerlegung des Systems in Subsysteme und Bausteine dargestellt durch Pakete (Schichten)
- Logische Zusammenfassung von Klassen und Sub-Paketen zu einer Einheit (Module)
- Beschreibung der Art der Abhängigkeiten zwischen Paketen im Hinblick auf das Paket-Konzept von Programmiersprachen

# Komponentendiagramm



# UML Komponentendiagramm

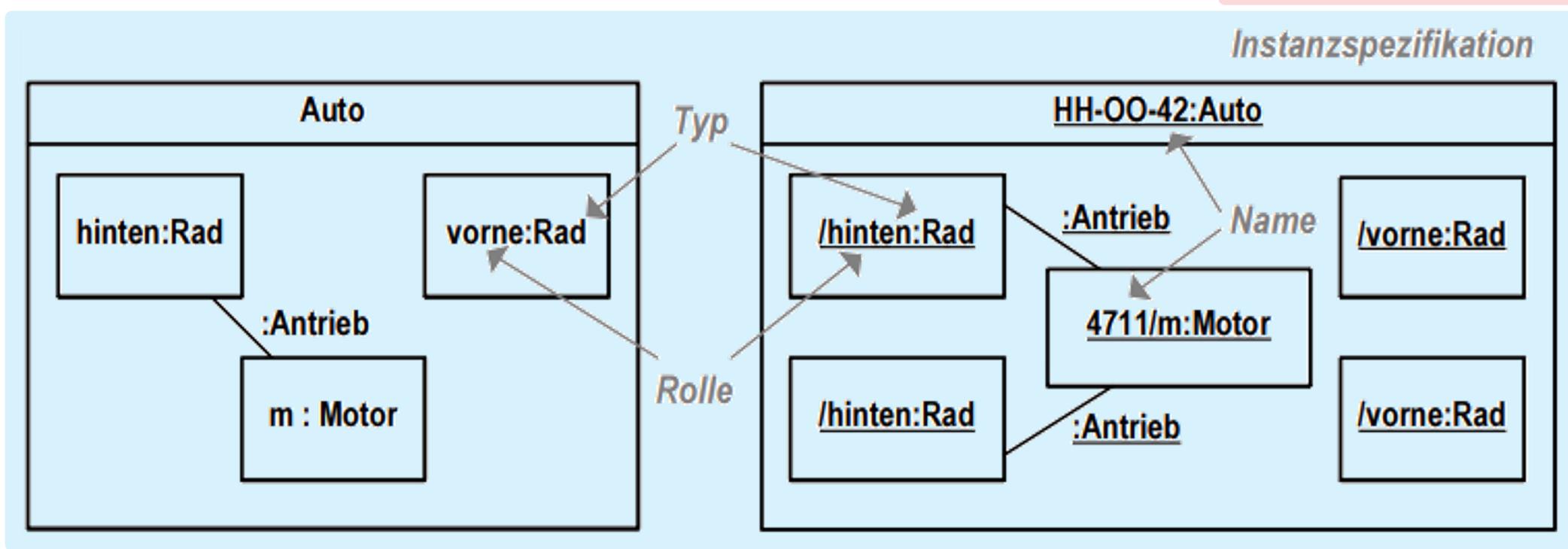
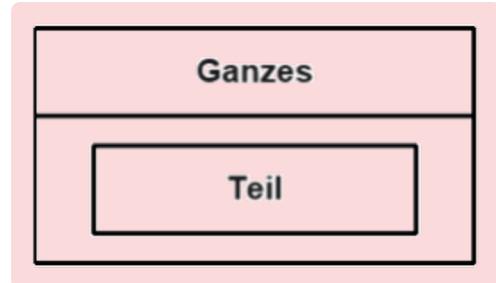
---



Zum Nachlesen

- Englisch: Component diagram
- Zerlegung des Systems in Subsysteme und Bausteine dargestellt durch Komponenten
- Logische Zusammenfassung von Klassen und Sub-Komponenten zu Einheiten, die nach außen über Schnittstellen angeboten werden
- Definition der Beziehungen zwischen Komponenten sowie deren Ein- und Ausgabeschnittstellen
- Funktionale Anforderungen und Cross-Cutting Concerns im Fokus

# Kompositionssstrukturdiagramm



<https://www.oose.de/wp-content/uploads/2012/05/UML-Notations%C3%BCbersicht-2.5.pdf>

# UML Kompositionssstrukturdiagramm

---

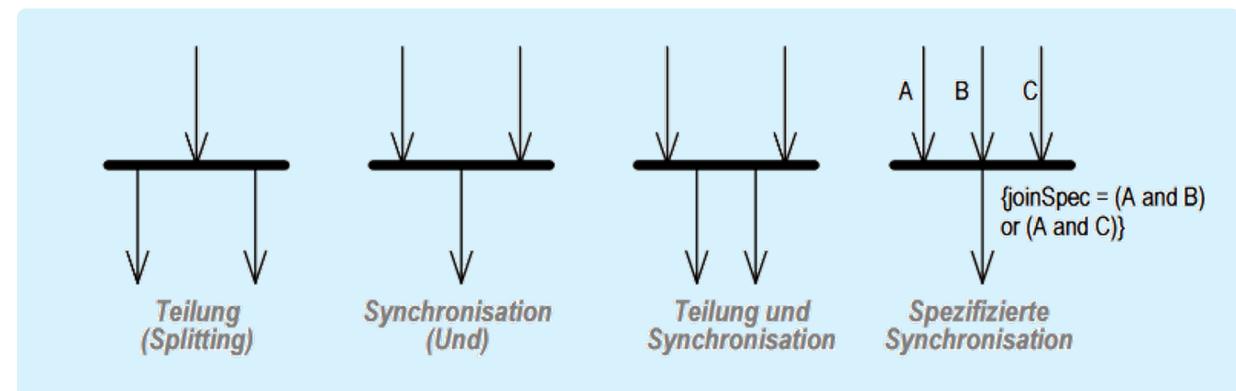
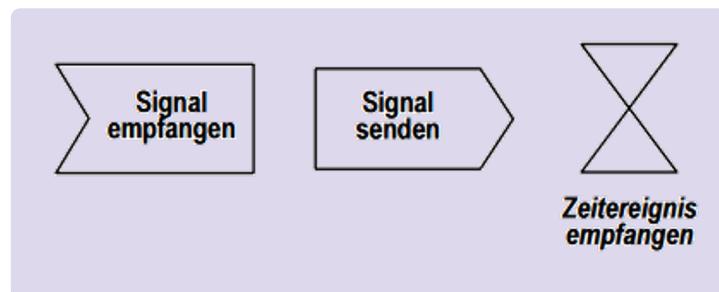
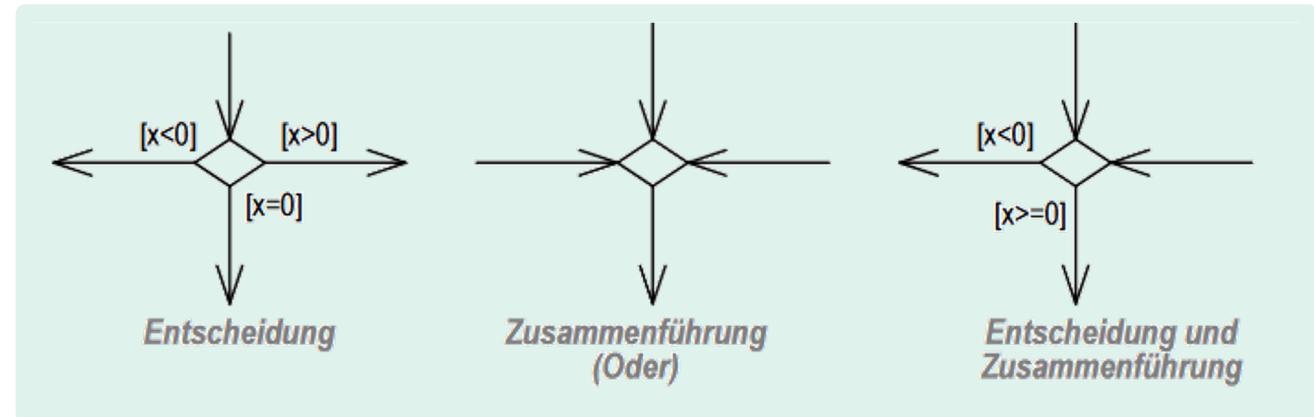
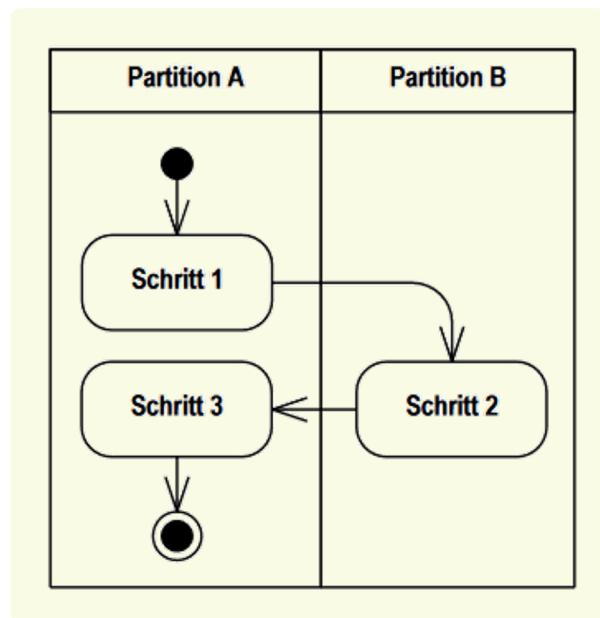


Zum Nachlesen

- Englisch: Composite structure diagram
- Abbildung von Teilen, die zu einem Ganzen zusammengefügt werden
- Strukturell dynamische Kompositionssstrukturen zur Modellierung des statischen Zusammenwirkens von Rollen innerhalb eines Anwendungsfalls
- Strukturell statische Kompositionssstruktur zur Abbildung von Hierarchien (z.B. Dekomposition des Systems in Subsysteme wie ein Auto in seine Einzelteile)

# Aktivitätsdiagramm

<https://www.oose.de/wp-content/uploads/2012/05/UML-Notations%C3%BCbersicht-2.5.pdf>



# UML Aktivitätsdiagramm

---

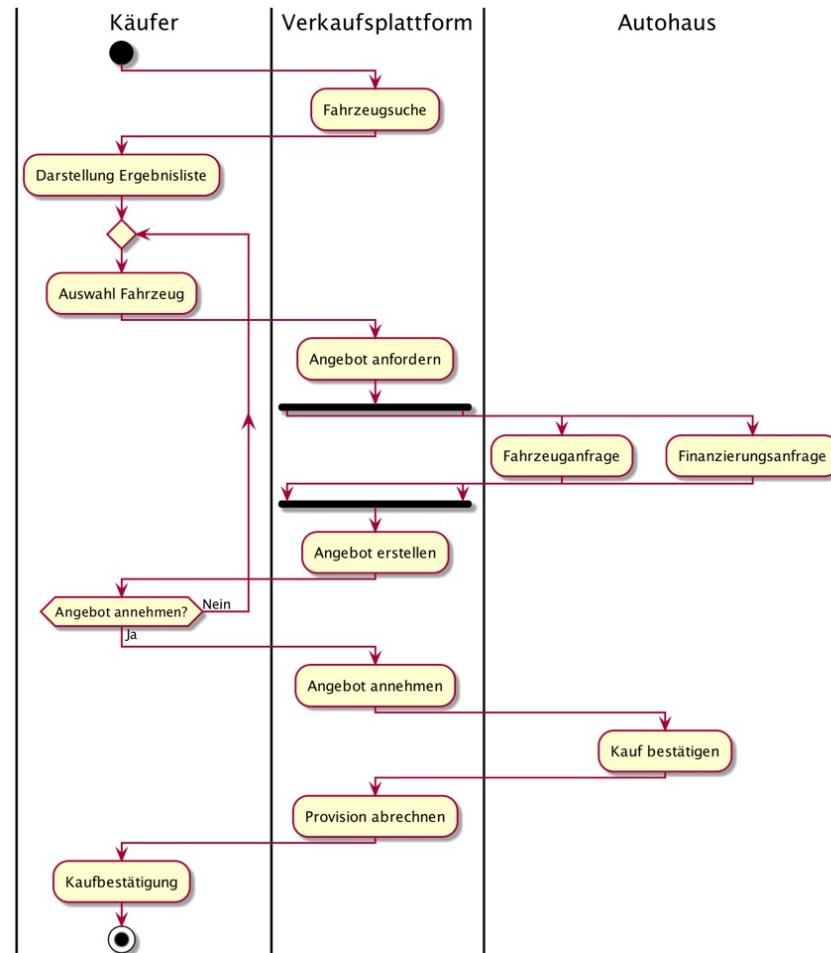


Zum Nachlesen

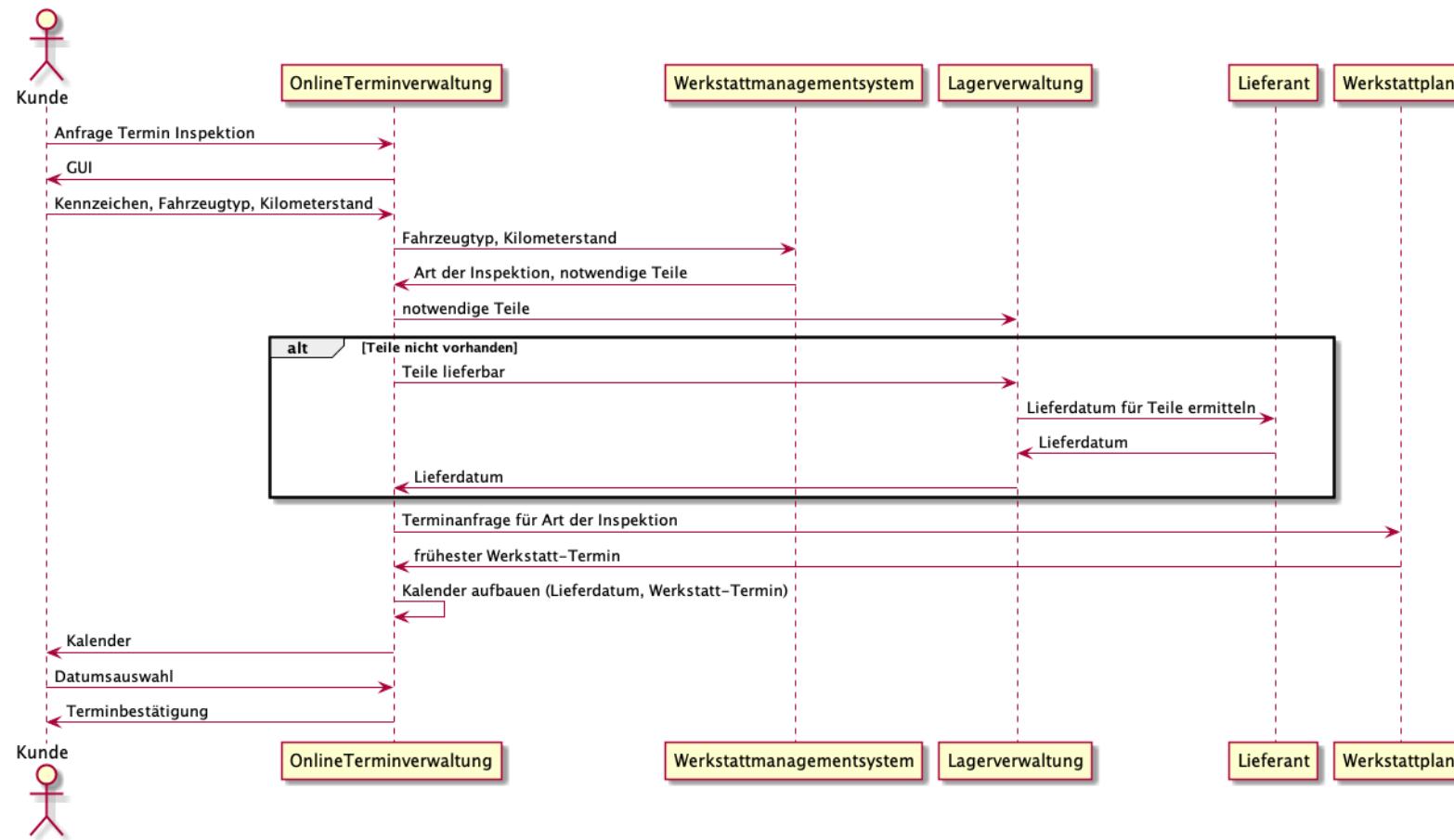
- Englisch: Activity diagram
- Visualisierung von Abläufen innerhalb von Systembestandteilen wie Komponenten, Klassen oder Anwendungsfällen
- Beschreibung von Algorithmen, Daten und Kontrollflüssen
- Ein Ablauf besteht aus Start, Aktivitäten, Verzweigungen, Ereignissen und Ende

# Beispiel

Fahrzeugkauf über eine  
Mobile Plattform



# Sequenzdiagramm



# UML Sequenzdiagramm

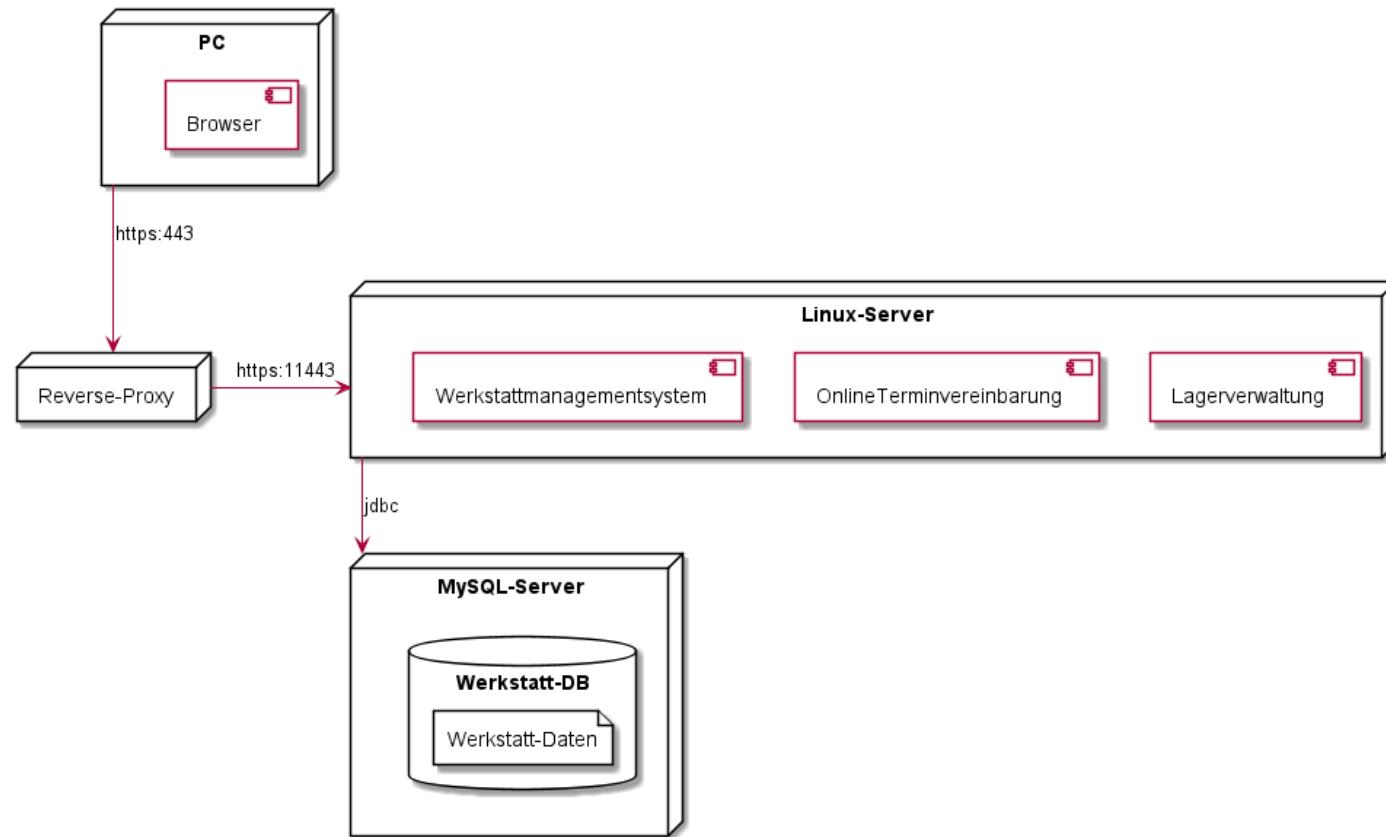
---



Zum Nachlesen

- Englisch: Sequence diagram
- Visualisierung der Interaktion (Nachrichtenaustausch) zwischen Instanzen von Bausteinen / Klassen, die zur Laufzeit stattfinden
- Detaillierte Beschreibung mit Schleifen, Bedingungsprüfungen und Fehlerbehandlung möglich

# Verteilungsdiagramm



# UML Verteilungsdiagramm

---

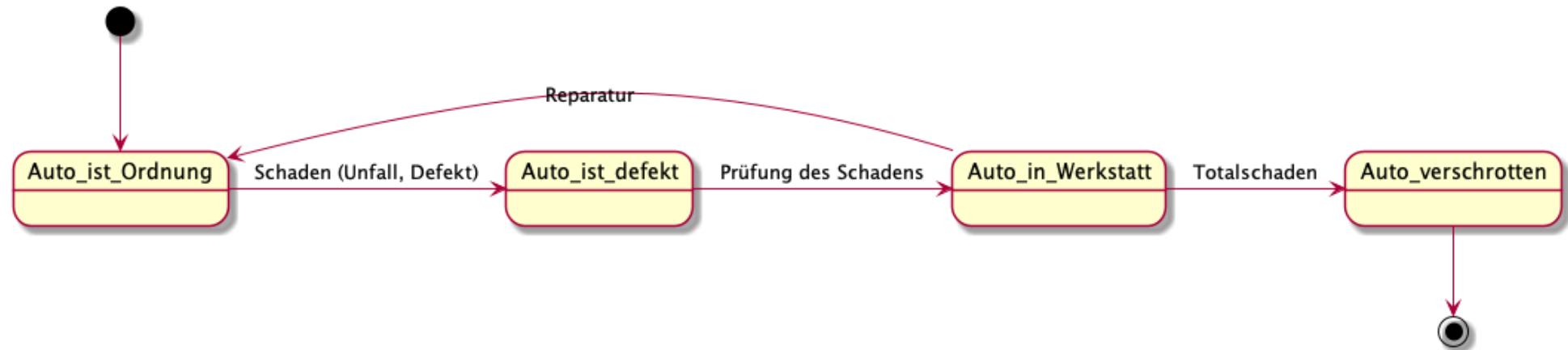


Zum Nachlesen

- Englisch: Deployment diagram
- Zeigt die Infrastruktur und ihre Abhängigkeiten
- Beschreibt die Verteilung der Artefakte auf die Hardware

# Zustandsdiagramm

---



# UML Zustandsdiagramm

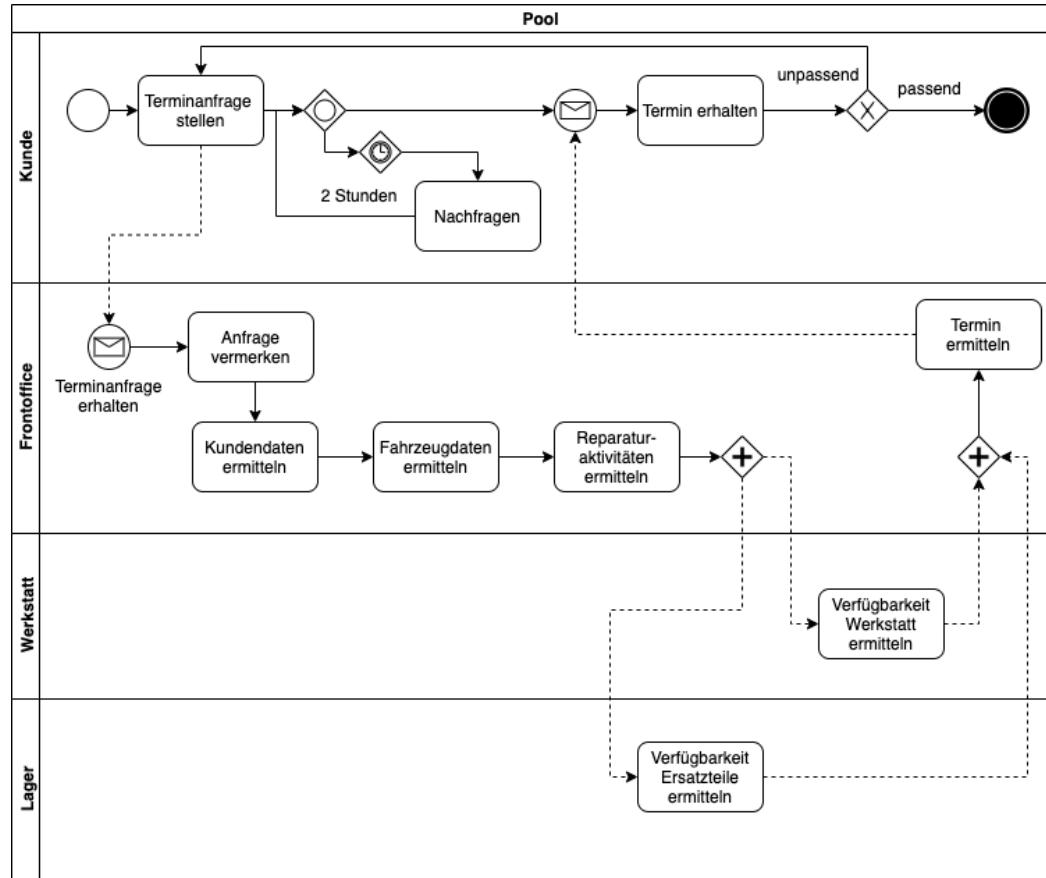
---



Zum Nachlesen

- Englisch: State machine diagram
- Zeigt die möglichen Zustände eines Systembestandteils sowie die erlaubten Zustandsübergänge
- Verknüpft Aktivitäten mit Zuständen und Übergängen

# Modellierungssprache: BPMN



[http://www.bpmn.de/images/BPMN2\\_0\\_Poster\\_DE.pdf](http://www.bpmn.de/images/BPMN2_0_Poster_DE.pdf)

# BPMN & ArchiMate

---



Zum Nachlesen

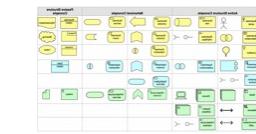
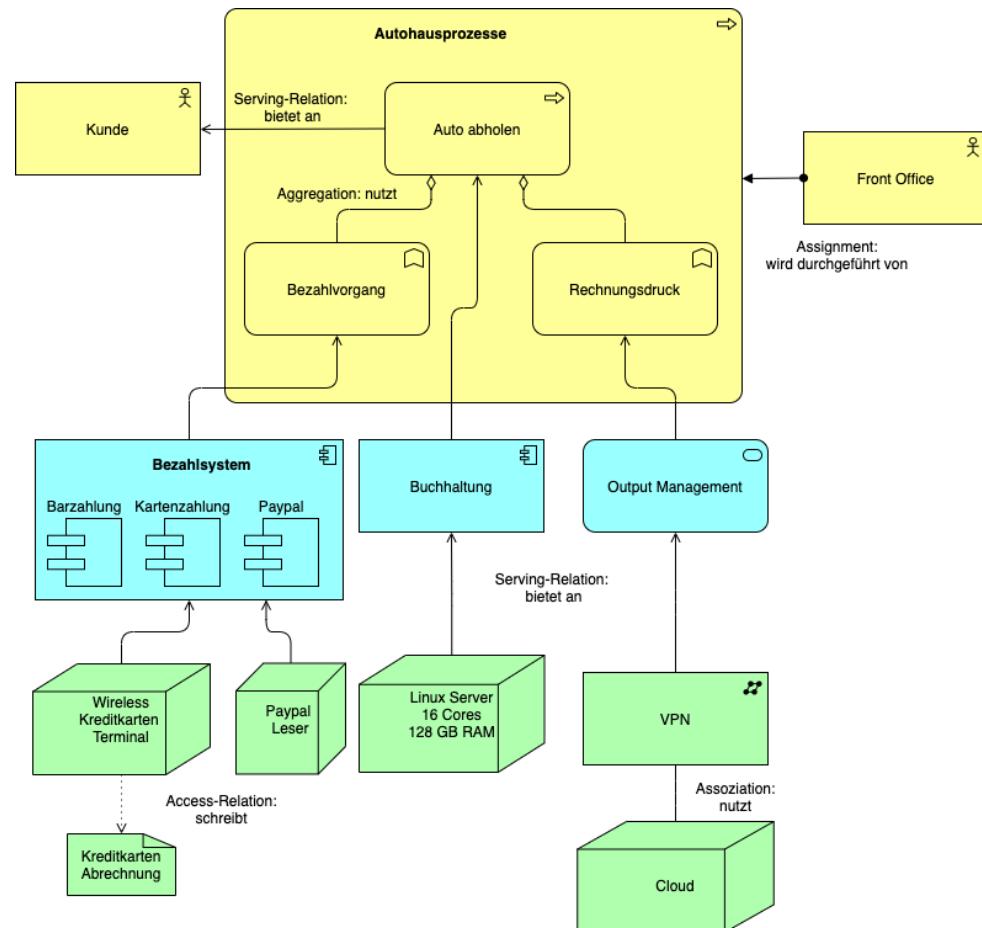
## BPMN

- Offener Standard zur Geschäftsprozessmodellierung
- Zur Kommunikation mit nicht-technischen Stakeholdern
- Automatisierbar in (teils kommerziellen) Process Engines

## ArchiMate

- Standard der Open Group
- Modellierungssprache für Enterprise Architektur
- Fokus auf TOGAF-konforme Architekturendokumentation
- Unterschiedliche Diagrammtypen (Business-, Anwendungs- und Technologiearchitektur)

# Modellierungssprache: ArchiMate



<https://pubs.opengroup.org/architecture/archimate3-doc/>

# Modellierung & Dokumentation

---

Obgleich zur Modellierung und Dokumentation teilweise die gleichen Tools verwendet werden, gibt es Unterschiede:

## Modellierung

- Vor / während der Implementierung
- Zum **Ermitteln** einer Problemlösung
- Experimentiell, inkrementell & iterativ

## Dokumentation

- Während / nach der Implementierung
- **Vermitteln** der Problemlösung
- Direkte Darstellung der Gegebenheiten

# Tipps für die Modellierung & Dokumentation

---

- **Modellieren im Team**

Skizzen / Modelle im Team zur Förderung des gemeinsamen Verständnisses entwickeln [Toth 2015] beschreibt in diesem Zusammenhang das Prinzip „Analog modellieren“ z.B. auf Flipcharts und Whiteboards, diese Entwürfe als Foto zu sichern und direkt oder nach Überführung in ein Modellierungstool in die Dokumentation einzufügen.

- **Gerade gut genug**

(Überflüssige) Details ignorieren

- **Document Continuously**

Dokumentation und Entwicklung verzahnen / In jeder Iteration dokumentieren

- **Document Late**

Gerade rechtzeitig dokumentieren, so dass zum Zeitpunkt der Dokumentation alle Informationen vorliegen

- **Diagramm-Plus-Text**

Diagramme benötigen (kurze) textuelle Beschreibungen

- **Redundanzfreie Modelle** zur Sicherung der Wartbarkeit und Integrität

# LZ 3-4: Architektsichten erläutern und anwenden (R1)

---

- Softwarearchitekt:innen können folgende Architektsichten anwenden:
  - Kontextsicht (auch genannt Kontextabgrenzung)
  - Baustein- oder Komponentensicht (Aufbau des Systems aus Softwarebausteinen)
  - Laufzeitsicht (dynamische Sicht, Zusammenwirken der Softwarebausteine zur Laufzeit, Zustandsmodelle)
  - Verteilungs-/Deployment-Sicht (Hardware und technische Infrastruktur sowie Abbildung von Softwarebausteinen auf diese Infrastruktur)

# **LZ 3-5: Kontextabgrenzung von Systemen erläutern und anwenden (R1)**

---

- Softwarearchitekt:innen können:
  - Kontext von Systemen z.B. in Form von Kontextdiagramm mit Erläuterungen darstellen
  - externe Schnittstellen von Systemen in der Kontextabgrenzung darstellen
  - fachlichen und technischen Kontext differenzieren.

# LZ 3-7: Schnittstellen beschreiben (R1)

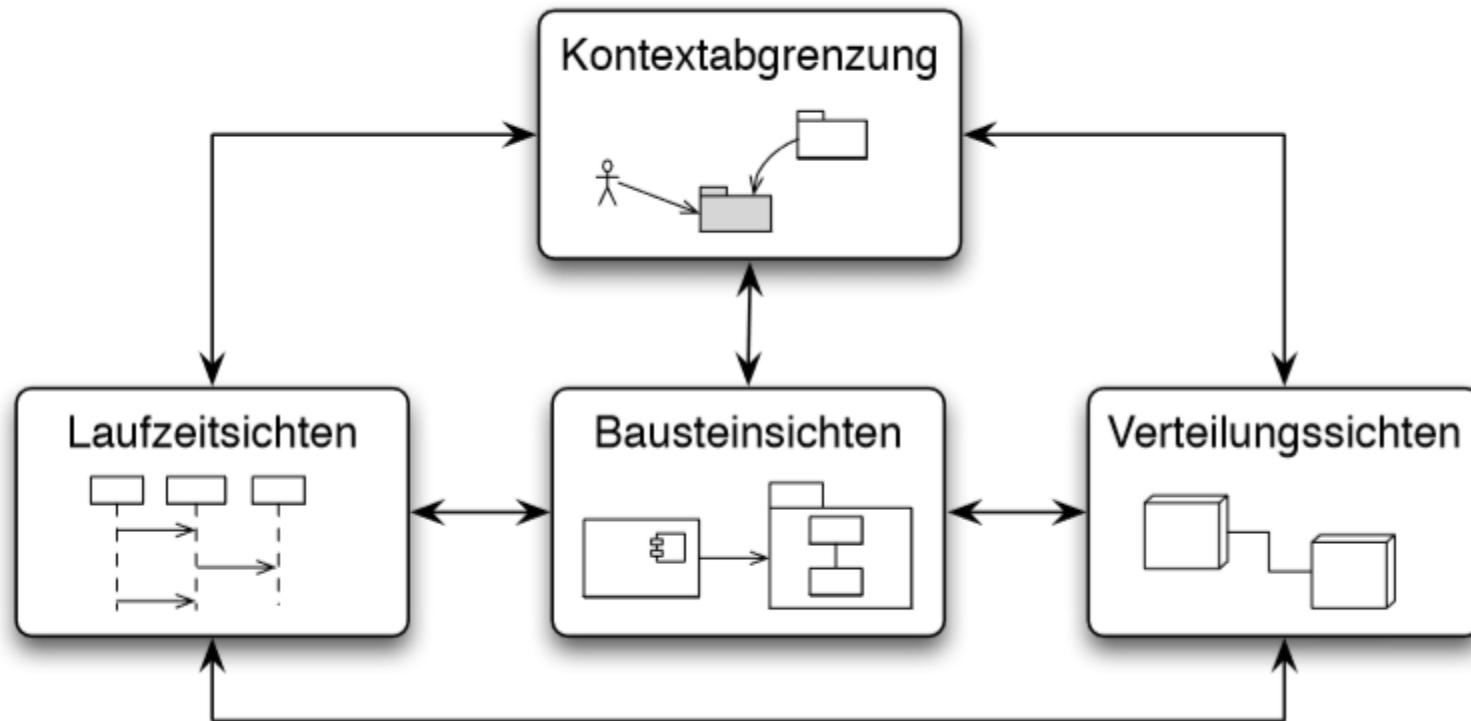
---

- Softwarearchitekt:innen können sowohl interne als auch externe Schnittstellen beschreiben und spezifizieren.

**Achtung!** LZ 3-6 wird hier übersprungen und folgt im Anschluss

- LZ 3-7 passt gut zu LZ 3-4 & 3-5, da mögliche Schnittstellen in den Diagrammen bereits auftauchen und anhand derer beschrieben werden können

# Vier Sichten als Basis für gute Softwarearchitektur



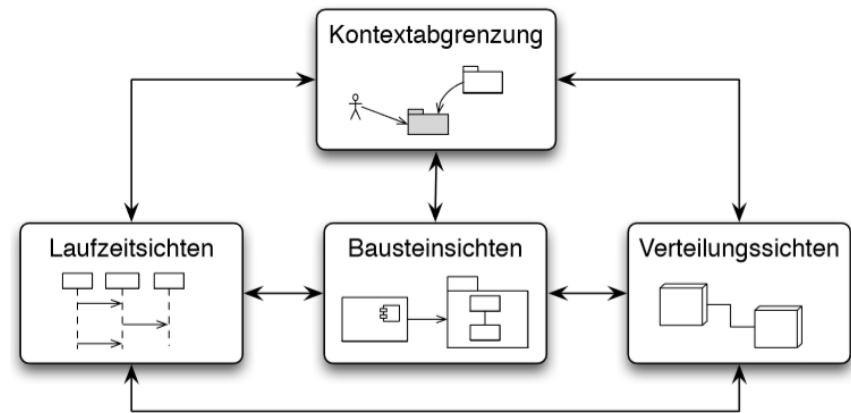
# Faustregeln für die Beschreibung von Architektsichten

---

- Die Beschreibung sollte einer **einheitlichen Gliederung** folgen, bezogen auf die ganze Dokumentation (arc42) und auf jede Sicht (Schablone)
- So wenig Formalismus wie möglich, so viel wie nötig
- Kleinigkeiten und Ungenauigkeiten können oft akzeptiert werden  
Architekten sollten pragmatisch anstatt dogmatisch dokumentieren
- **Umfang der Dokumentation in Abhängigkeit vom Risiko**  
Baustein mit hohem Risiko erfordert detaillierte Dokumentation - Baustein mit wenig Risiko erfordert weniger Details

# Kontextsicht

- **Kontextsicht** zeigt:
  - System als **Blackbox**
  - das Umfeld des Systems
  - die Schnittstellen zur Außenwelt und technischen Systemumgebung
- Mischung aus Komponenten- und Verteilungsdiagramm
- Nicht Teil des UML Standards
- Verständlich für viele Stakeholder mit Feedback-Möglichkeit



# Schablone zur Kontextsicht

---

- 1. Kurzbeschreibung (optional)**
- 2. Diagramm**
- 3. Elementkatalog**  
Beschreibung der Diagrammelemente (Akteure, externe Systeme) und ihrer Interaktionen
- 4. Variabilitäten (optional)**  
Wo wird Flexibilität und Änderbarkeit benötigt, Beschreibung von Konfigurationen, Beschreibungen von Unsicherheiten in Anforderungen, Architektur und Design
- 5. Hintergrundinformationen (optional)**  
Begründungen von Architekturentscheidungen, Ergebnisse von Analyse, Voruntersuchungen und Prototypen, Annahmen über die Systemumgebung, Referenzen auf andere Sichten, Quellen, Beispielcode, etc.

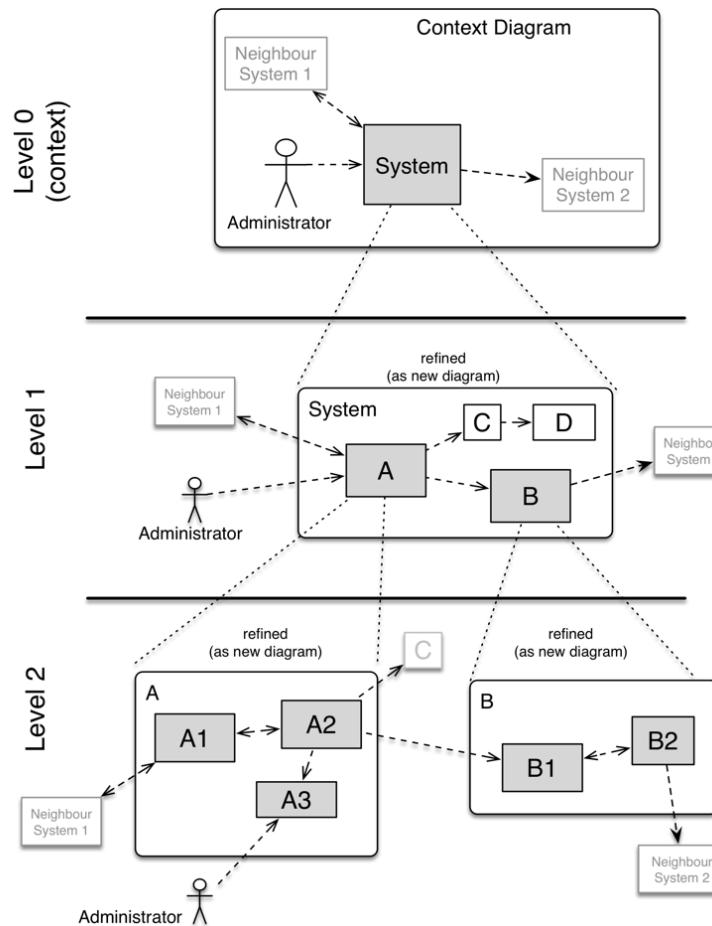
# Inkrementeller & iterativer Sichtenentwurf

## Top-Down Dekomposition

Ausgehend vom Ziel des Systems erfolgt die Zerlegung in Teilprobleme.



54



## Bottom-Up Komposition

Ausgehend von der Teillösung erfolgt der Zusammenbau des Systems

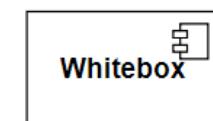


Abbildung von  NOVATEC  
<https://faq.arc42.org/questions/B-10/>

# Fachlicher vs. technischer Systemkontext

- Unterscheidung notwendig
- Management, Entscheider:innen und Auftraggeber:innen bevorzugen i.d.R. den fachlichen Systemkontext
- Entwickler:innen und Betrieb hat mehr Interesse am technischen Systemkontext

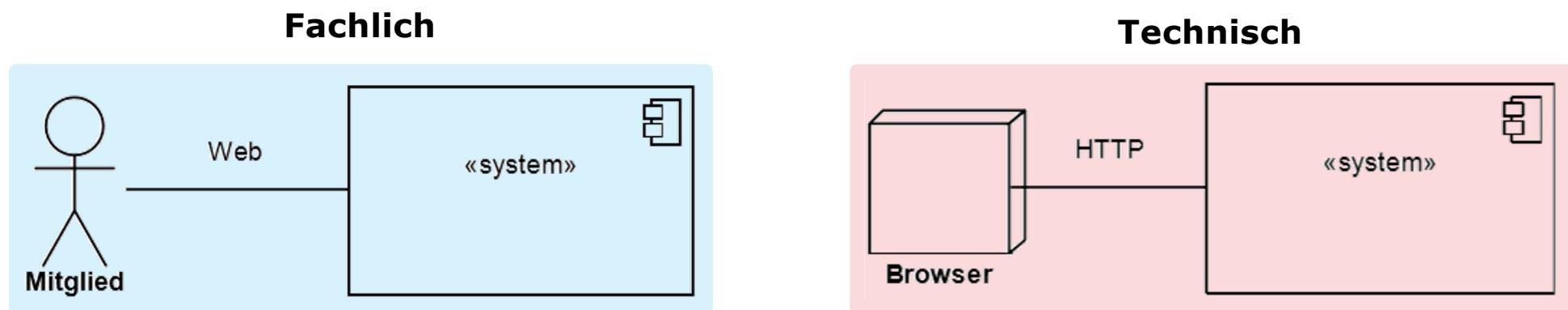
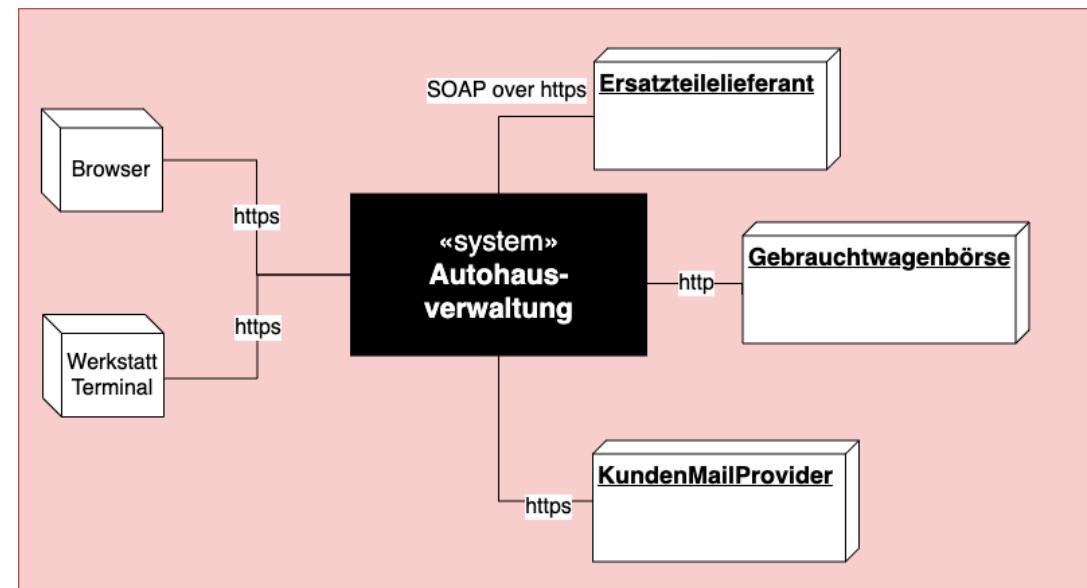
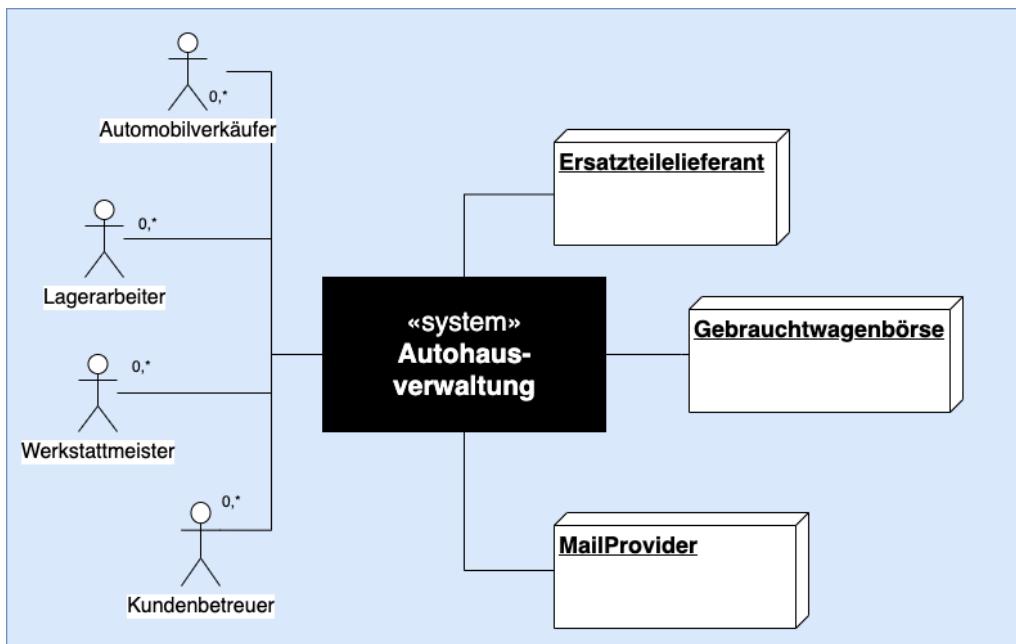


Abbildung aus  
[Zörner 2015]

# Beispiel

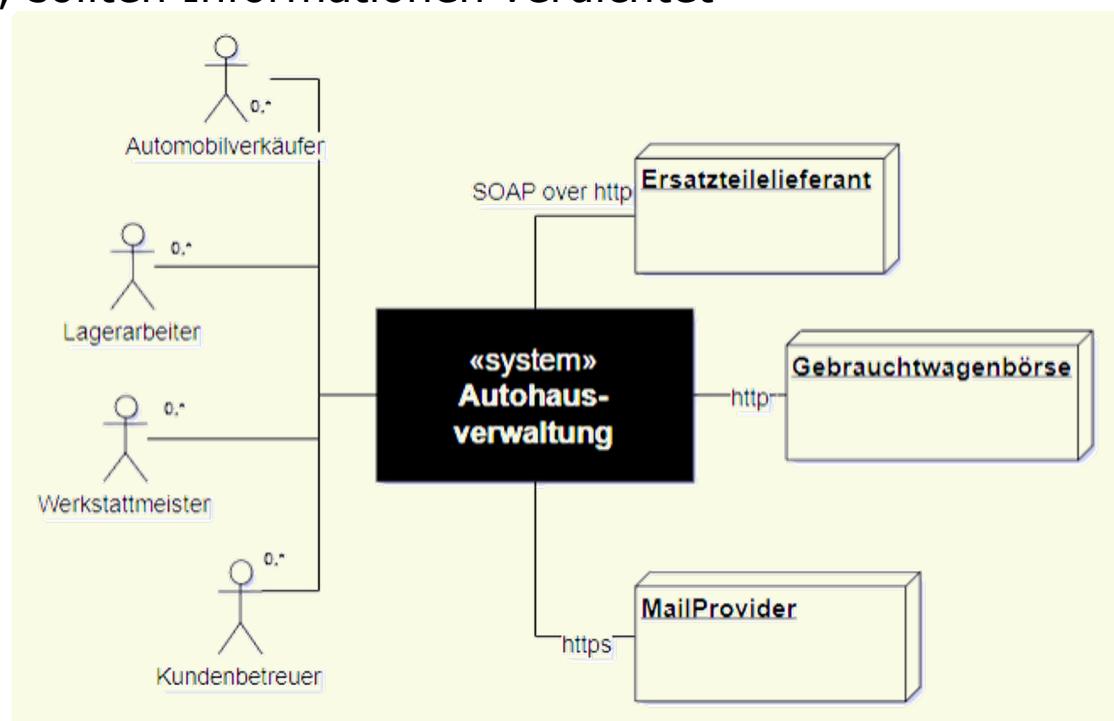
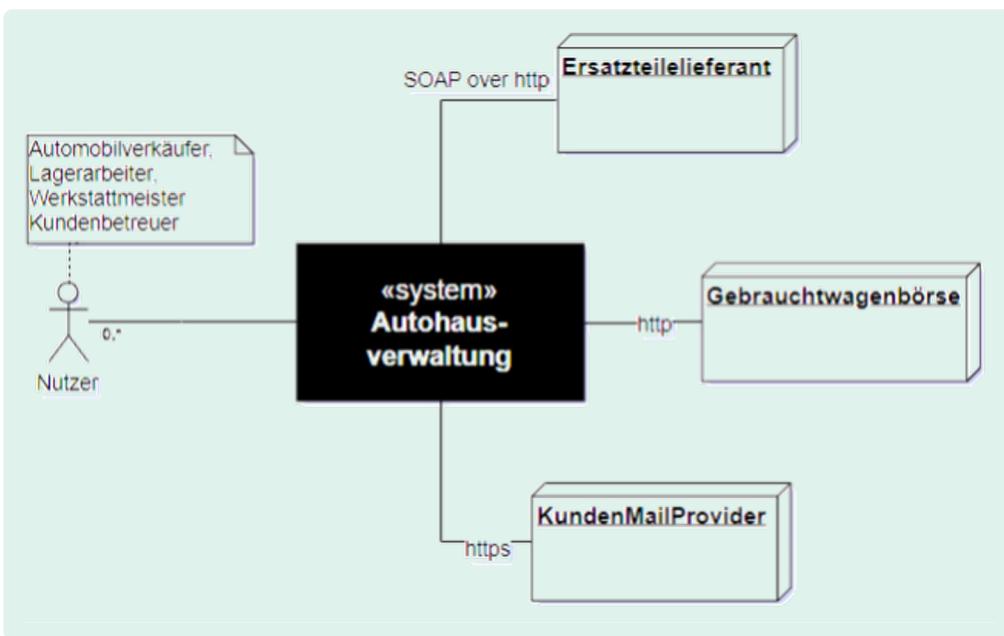
## Fachlich



**Technisch**

# Einer vs. mehrere Akteure

- Bedenke **7 +/- 2 Regel**
- Bei zu vielen Akteuren oder Elementen, sollten Informationen verdichtet werden



# Tipps für die Nutzung der Kontextsicht

---

- Nutzung als Feedback-Instrument für eine Vielzahl an Stakeholder
- Absicherung, dass alle benötigten Nachbarsysteme bekannt sind
- Risikoanalyse der externen Schnittstellen:  
Welche Umsysteme ...
  - haben eine hohe Änderungsrate
  - sind fachlich komplex
  - sind technisch komplex und drohen mit hohen Implementierungsaufwänden
  - haben ein Service Level Agreement
  - haben hohe Qualitätsanforderungen bzgl. Durchsatz, Verfügbarkeit und Sicherheit

# Elementkatalog des Systemkontexts

| Kontextelement   | Beschreibung  | Technische Interoperabilität  |
|--|---|---|
| Ersatzteilelieferant   | WebService des Ersatzteilelieferanten zur Bestellung von Ersatzteilen anhand der Teilenummern des Herstellers.  | <ul style="list-style-type: none"><li>▪ SOAP over HTTPS</li><li>▪ SSL verschlüsselte Verbindung</li><li>▪ Client Zertifikat im SOAP Header</li></ul>  |
| Gebrauchtwagenbörse  | Öffentliche Plattform im Internet, auf der gebrauchte Fahrzeuge von privaten und kommerziellen Händlern zum Verkauf angeboten werden. Über die öffentliche API sollen Fahrzeuge und ihre Verkaufspreise abgefragt werden, um diese intern zu vergleichen. | <ul style="list-style-type: none"><li>▪ JSON over HTTP (REST API)</li><li>▪ Keine weiteren Sicherheitsmaßnahmen notwendig</li></ul>   |
| Nutzer:innen<br>(Automobilverkäufer:innen,<br>Kundenbetreuer:innen,<br>Lagerarbeiter:innen,<br>Werkstattmeister:innen) | Nutzer:innen der Verwaltungssoftware in unterschiedlichen Rollen  | <ul style="list-style-type: none"><li>▪ Zugriff erfolgt durch den Web-Browser und das Werkstatt Terminal</li><li>▪ SSL verschlüsselte Verbindung</li><li>▪ Authentifizierungstoken im HTTP Header</li></ul> |

# Stakeholder der Kontextsicht

---

- Projektleitung
- Anforderungsanalyst:innen
- Systemanalyst:innen
- Fach- oder Domänenexpert:innen
- Unternehmensarchitekt:innen
- Softwarearchitekt:innen
- Entwickler:innen
- Betrieb
- Tester:innen
- Management

# Übung

---

- Erstellen Sie den Systemkontext der Anwendung  
VerSicher24mal7

---

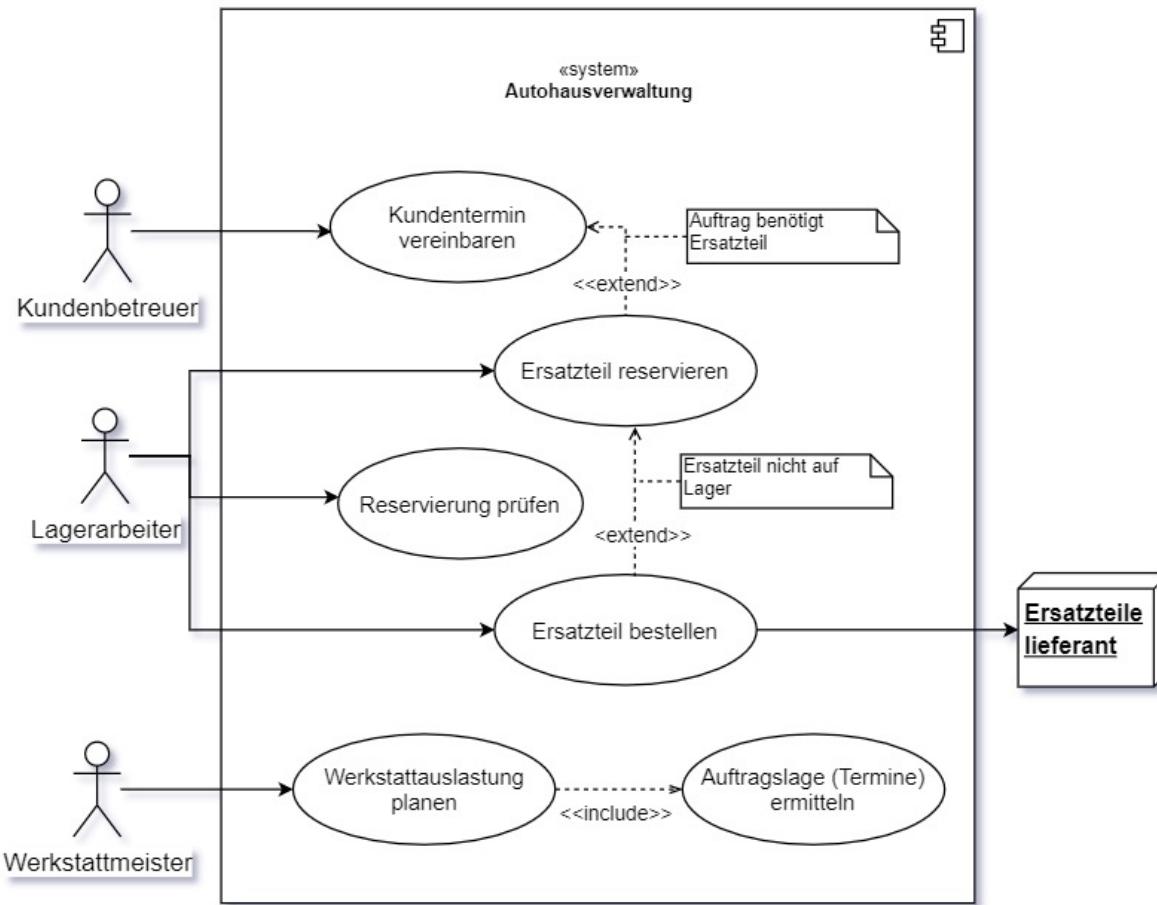
# Autohausverwaltung

# Anwendungsfallmodell

---

- Als **Kundenbetreuer:in** möchte ich einen Termin abgestimmt auf Werkstattkapazität und Verfügbarkeit von Ersatzteilen über das System erstellen, während ich im Gespräch mit den Kunden bin.
- Als **Lagermitarbeiter:in** möchte ich Wartungsteile automatisiert bestellen. Ein Eingriff soll nur notwendig sein, wenn die Verfügbarkeit automatisiert nicht sichergestellt werden kann.
- Als **Werkstattmeister:in** möchte ich die Auslastung meiner Werkstattmitarbeiter:innen abgestimmt mit der Auftragslage planen können.

# Anwendungsfallmodell



# Anwendungsfallmodell

---

- **Termin vereinbaren** mit automatisierter **Ersatzteilreservierung**
- Die **Ersatzteilreservierung** muss zusätzlich für den Lagerarbeiter zugänglich sein
- Bei der **Werkstattplanung** muss die aktuelle Auftragslage für den anstehenden Planungszeitraum ermittelt werden
- **Reservierungsanfrage**, die nicht automatisiert bearbeitet werden können, müssen vom Lagerarbeiter **geprüft** werden

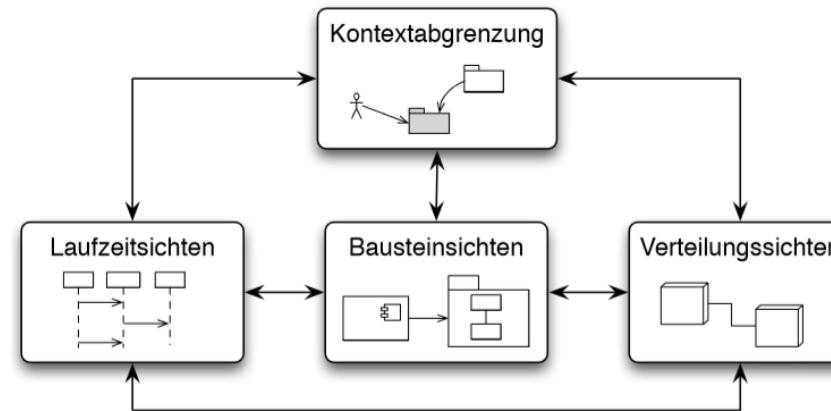
# Weitere Details für das Beispieldaten

---

- Die Kunde vereinbart einen Termin für eine Wartungsarbeit z.B. Inspektion 1
- Dafür wird ein Ölfilter, Öl, Mikrofilter und ein Luftfilter (Ersatzteile) benötigt
- Wartungsteile werden immer vom gleichen Lieferanten bestellt
- Wartungsteile sind nur in Ausnahmefällen nicht über Nacht lieferbar
- Eine voll automatisierte Reservierungs- und Bestellverarbeitung ist zu 90% ohne manuelle Eingriffe möglich
- Für 10% der Bestellungen ist eine manuelle Bearbeitung durch den Sach- und / oder Lagermitarbeiter notwendig
- Der Kunde wünscht eine Terminbestätigung via E-Mail

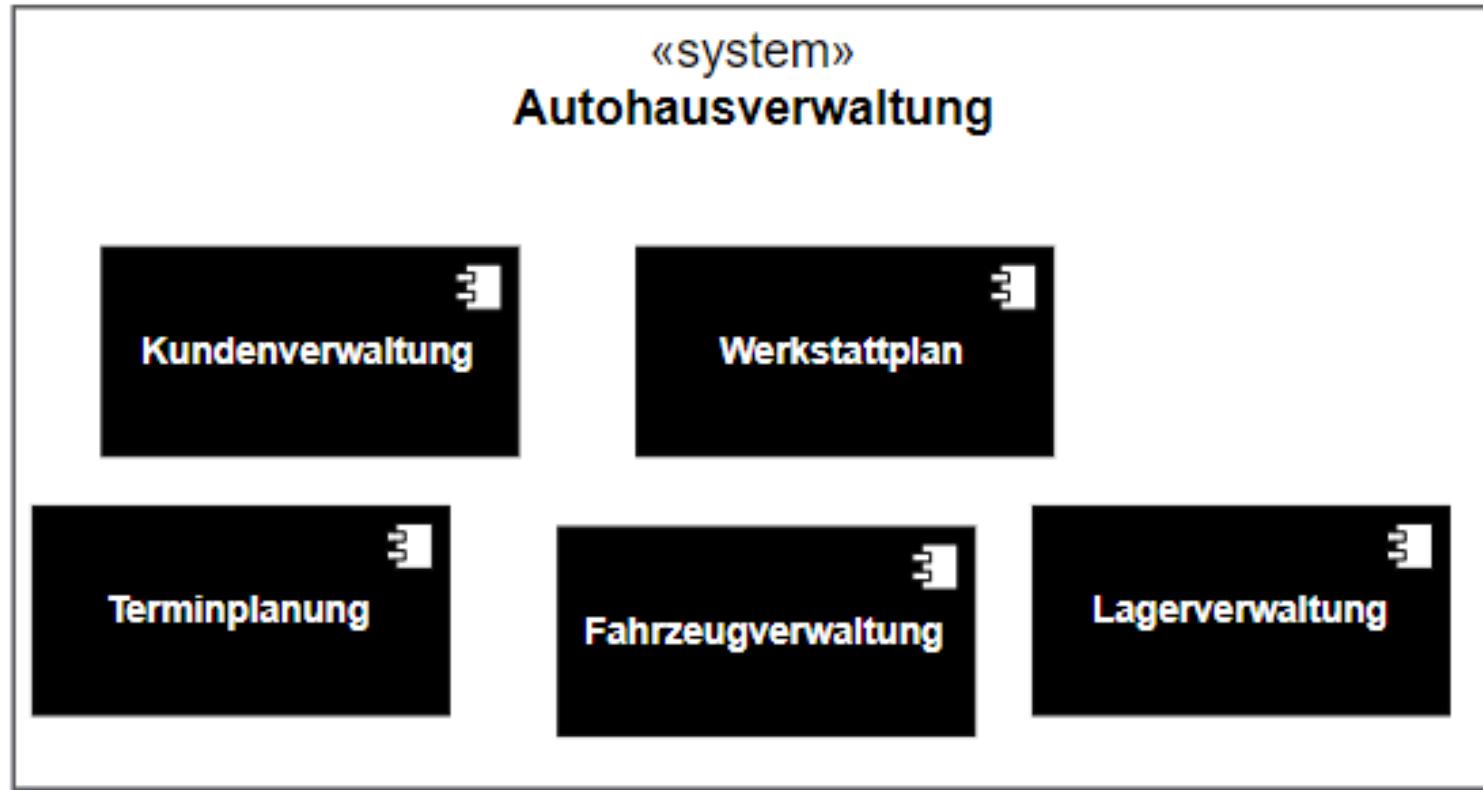
# Bausteinsicht

- Statische Strukturen des Systems bestehend aus Bausteinen und deren Beziehungen
- Beschreibung der Schnittstellen zwischen den Bausteinen
- Bestimmung in zwei Schritten:
  1. Identifikation der abstrakten Bausteine
  2. Abhängigkeiten analysieren, Beziehungen definieren und Bausteine strukturieren

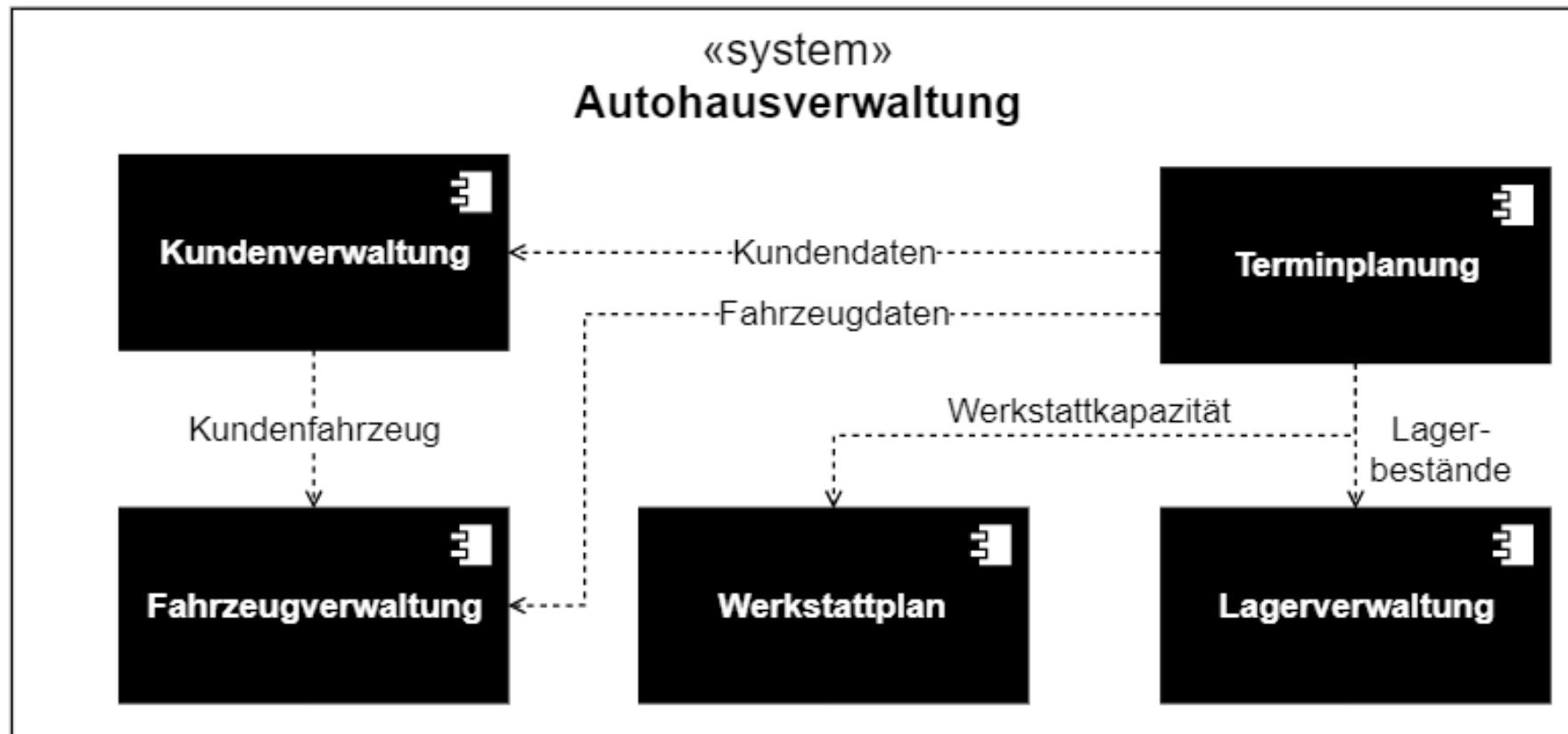


# 1. Schritt: Identifikation

---



## 2. Schritt: Beziehungen



# Schablone zur Bausteinsicht

---

## 1. Kurzbeschreibung

## 2. Diagramm

## 3. Schnittstellen

Beschreibung der bereitgestellten Schnittstellen und der genutzten Schnittstellen bzw. Verweise auf separate Dokumente

## 4. Elementkatalog (nur Whitebox-Sicht)

Beschreibung der Diagrammelemente (Akteure, externe Systeme) und ihrer Interaktionen sowie deren Schnittstellen

## 5. Variabilitäten (optional)

Wo wird Flexibilität und Änderbarkeit benötigt, Beschreibung von Konfigurationen, Beschreibungen von Unsicherheiten in Anforderungen, Architektur und Design

## 6. Hintergrundinformationen (optional)

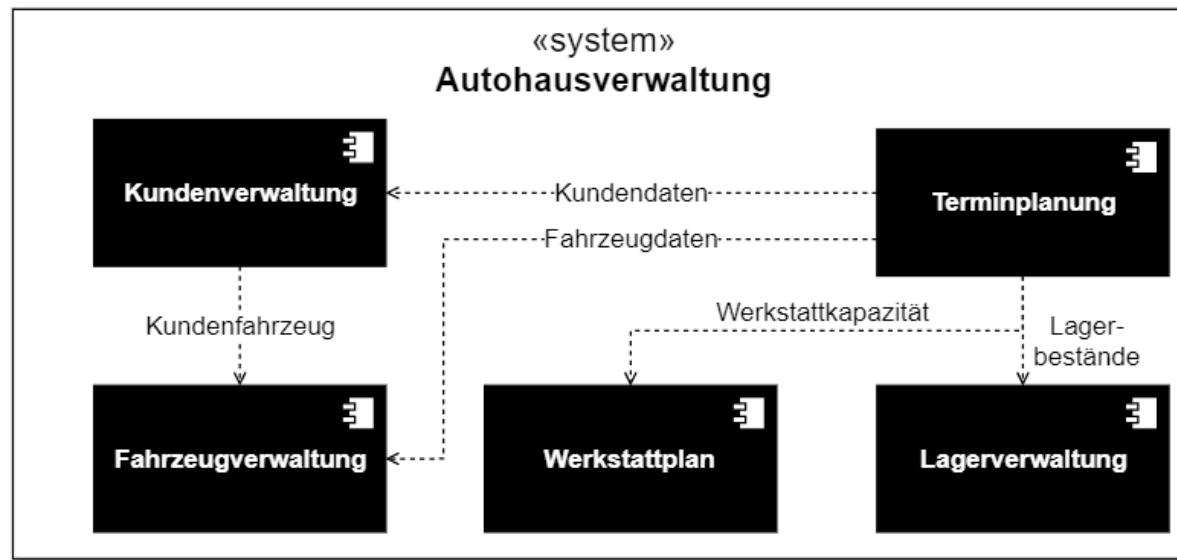
Begründungen von Architekturentscheidungen, Ergebnisse von Analyse, Voruntersuchungen und Prototypen, Annahmen über die Systemumgebung, Referenzen auf andere Sichten, Quellen, Beispielcode, etc.

# Sichtenbeschreibung Bausteinsicht

## 1. Kurzbeschreibung

Die Whitebox-Sicht der Autohausverwaltung zeigt die Bausteine des Systems der obersten Abstraktionsebene. Für die Darstellung wird ein UML Komponentendiagramm eingesetzt.

## 2. Diagramm



# Sichtenbeschreibung Bausteinsicht

---

## 3. Schnittstellenbeschreibung (folgt später)

## 4. Elementkatalog

| Element            | Typ                                | Beschreibung  |
|--------------------|------------------------------------|---|
| Kundenverwaltung   | UML Komponentendiagramm / Baustein | Kapsel um die Verwaltung, Anlage und Manipulation von Kundendaten mit Stamm- und Bewegungsdaten.  |
| Terminplanung      | UML Komponentendiagramm / Baustein | Terminplanung bietet die Möglichkeit Termine mit dem Kunden zu verwalten. Außerdem übernimmt die Terminplanung die Abstimmung von Werkstattkapazitäten und Lagerbestände für Ersatzteile. |
| Fahrzeugverwaltung | UML Komponentendiagramm / Baustein | Kapsel um die Verwaltung, Anlage und Manipulation von Fahrzeugdaten mit Stamm- und Bewegungsdaten.  |

# Sichtenbeschreibung Bausteinsicht

---

## 5. Variabilitäten

Nicht jeder Nutzer des Systems hat die gleichen Berechtigungen für den angebotenen Funktionsumfang.

## 6. Hintergrundinformationen

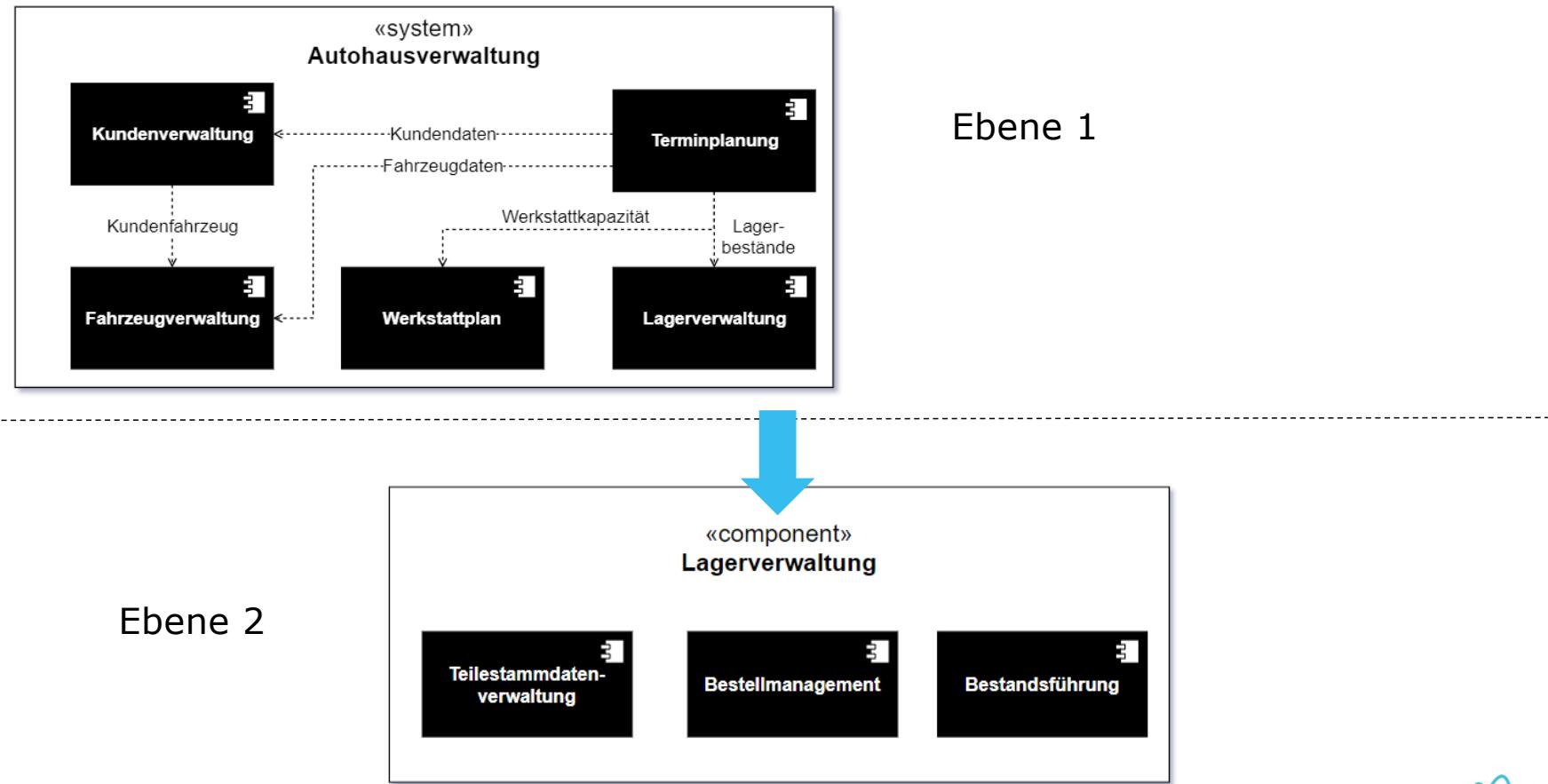
### 1. Verweise

1. Entwurfsentscheidung Kunde-Fahrzeug Zuordnung
2. Fachliches Rollen und Rechte Modell
3. Verweis auf Lösungsstrategie Rollen und Rechte

### 2. Annahmen

1. Es wird davon ausgegangen, dass eine Kaufkomponente für die Fahrzeugverwaltung eingesetzt wird, sofern diese alle Anforderungen erfüllt.

# Die Lagerverwaltung



# Stakeholder der Bausteinsicht

---

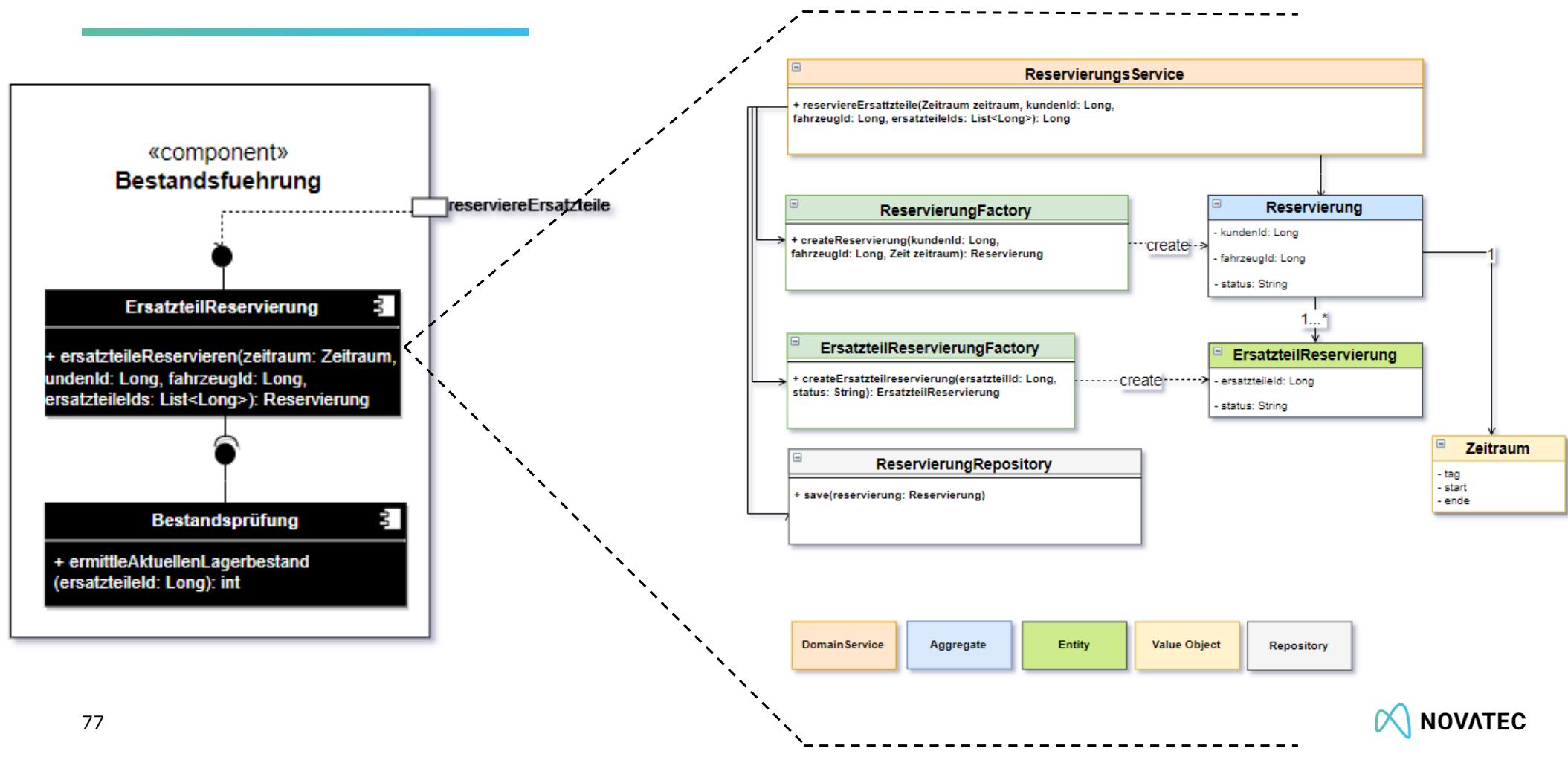
- Softwarearchitekt:innen
- Entwickler:innen
- Tester:innen
- Qualitätssicherung
- Projektleitung  
(hilft bei Erstellung von Arbeitspaketen und Aktivitäten)
- Bewerter:innen im Rahmen einer Architekturbewertung

# Tipps für die Modellierung der Bausteinsicht

---

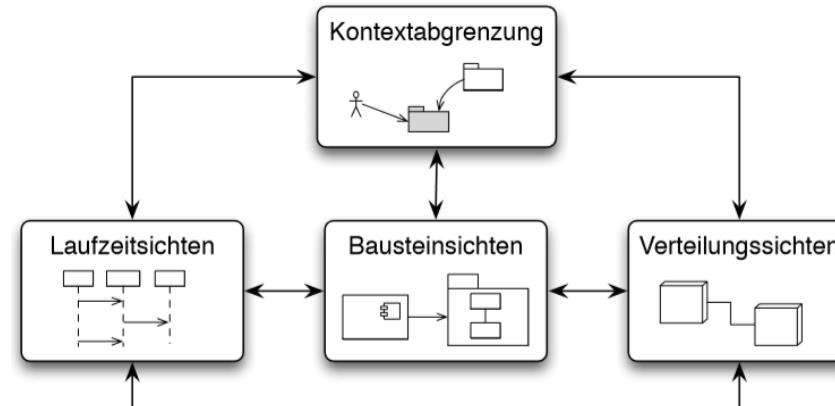
- Die höchste Abstraktionsstufe des Systems sollte mindestens beschrieben sein (Level 1 in arc42), um das System im Großen zu verstehen
- Bei der Verfeinerung der Bausteine Level 1 sollten interessante, besonders kritische, riskante oder komplexe Bausteine im Vordergrund stehen.
- Interne Schnittstellen können durch Unit-Tests und Quellcode veranschaulicht werden
- Dokumentation und Modellierung bis Ebene 3 oder 4 bei großen Systemen ist in der Regel ausreichend
- Keine Level überspringen – Jeder Baustein muss bis Ebene 1 nachverfolgbar sein

# Ebene 3 / 4 der Bausteinsicht Bestandsführung

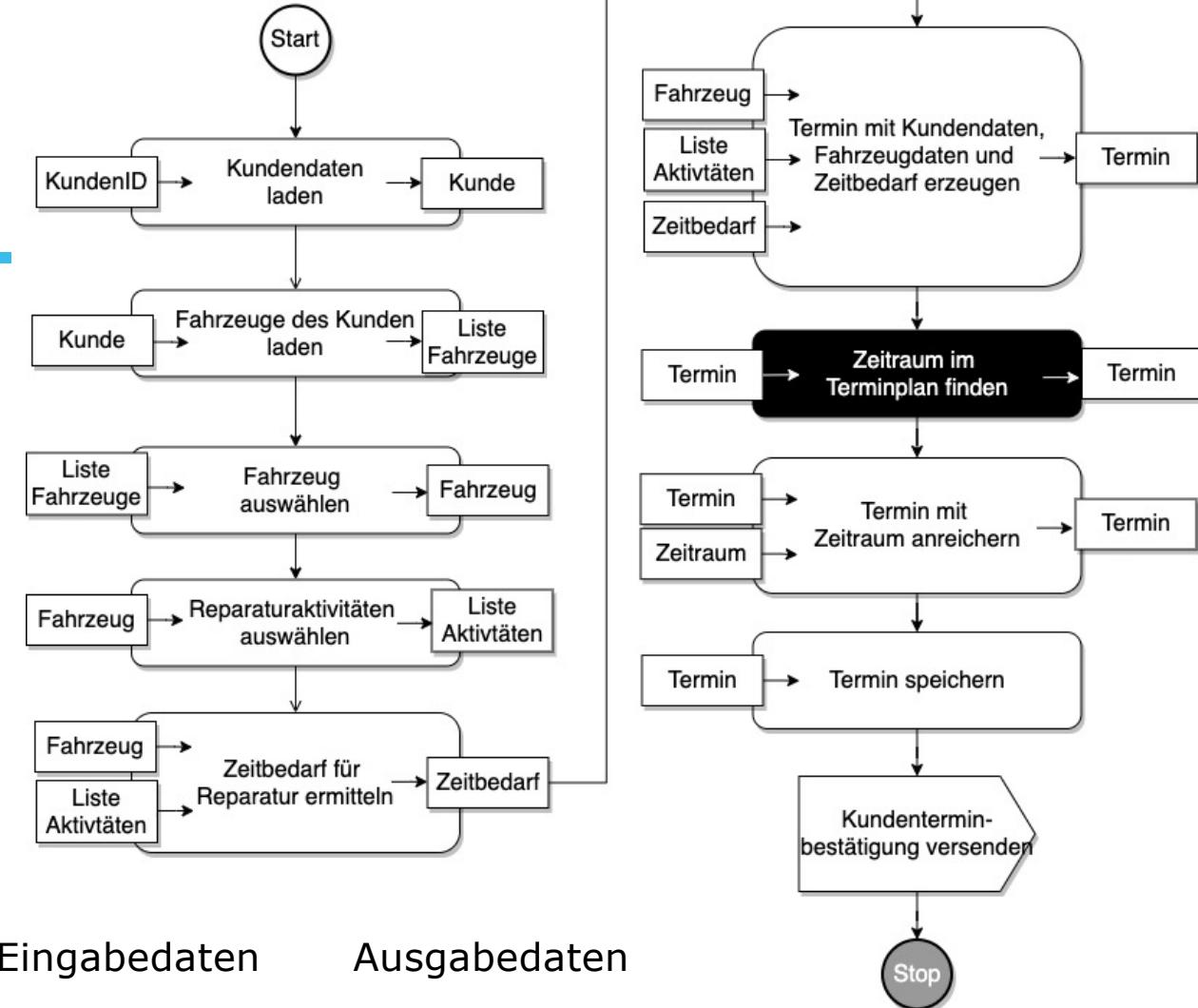


# Laufzeitsicht

- Welche Bausteine interagieren zur Laufzeit miteinander?
- Welche Informationen müssen die Bausteine austauschen?
- Durch die Modellierung von Abläufen werden Erkenntnisse über die **benötigten Schnittstellen** von Bausteinen sowie Informationen über ineffiziente Abläufe und Algorithmen gewonnen
- Als Beispiel wird hier der Workflow **Automatische Terminfindung** mit Blackbox-Teilprozess **Zeitraum im Terminplan finden** verwendet

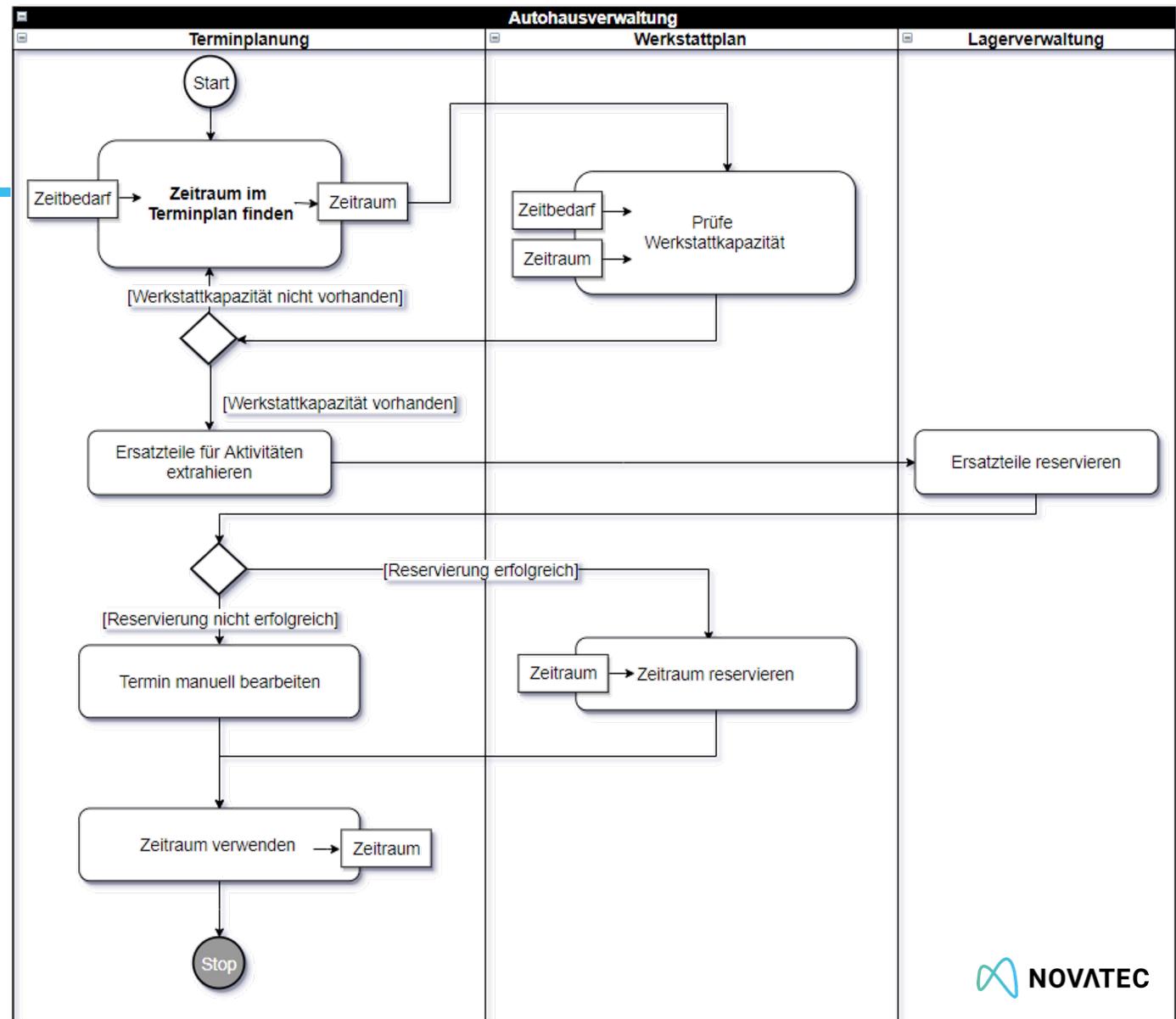


# Beispiel: Laufzeitsicht



# Laufzeitsicht

Whitebox-Sicht **Zeitraum im Terminplan finden**



# **Stakeholder der Laufzeitsicht**

---

- Softwarearchitekt:innen und Entwickler:innen
- Tester:innen und Qualitätssicherer:innen
- Bewerter:innen im Rahmen einer Architekturbewertung

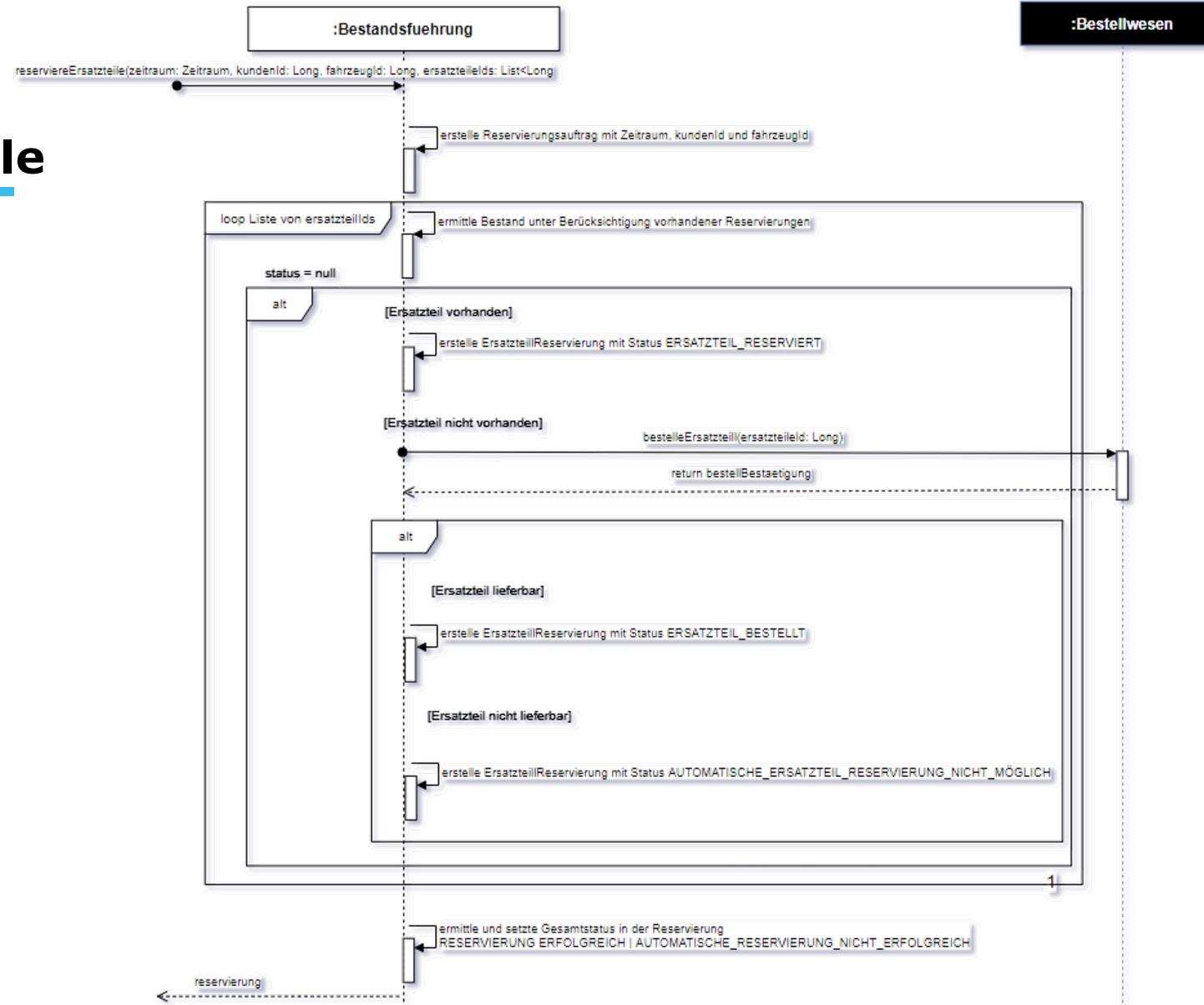
# Tipps für die Modellierung der Laufzeitsicht

---

- Fokussierung auf wenige Szenarien
- Priorisierung nach besonderen, risikoreichen Abläufen mit Relevanz für das Systemverständnis
- Abläufe können alternativ auch mit Listen oder Pseudocode beschrieben werden
- Sequenz- und Aktivitätsdiagramme sind schwer wartbar und sollten mit Bedacht eingesetzt werden
- Sequenzdiagramme eignen sich gut, um das Verhalten von Schnittstellen zu beschreiben

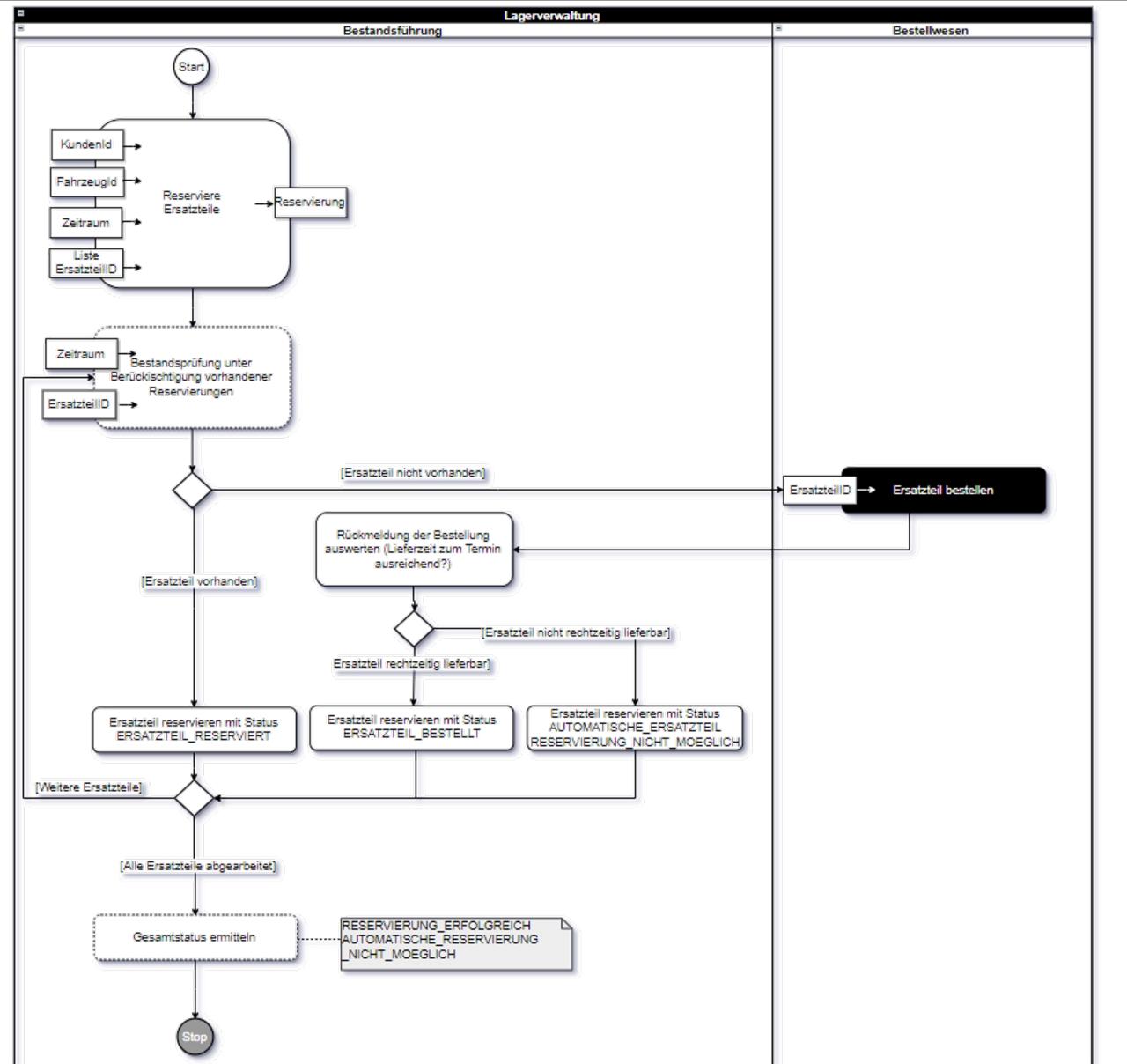
# Laufzeitsicht reserviereErsatzteile

- Sequenzdiagramm



# Laufzeitsicht reserviereErsatzteile

- Aktivitätsdiagramm



# Dokumentation von Schnittstellen

---

- Die **Dokumentation** von **Schnittstellen** erfolgt **während des Sichtenentwurfs**, insbesondere bei Entwurf der Baustein- und Laufzeitsicht
- Eigenständige Schnittstellendokumentation können für bestimmte Stakeholder relevant sein
  - Falls eine API nach außen bereitgestellt wird, kann hierfür beispielsweise OpenAPI Specification / Swagger (<https://swagger.io/>) genutzt werden
  - Weitergehende Merkmale einer Schnittstelle, wie Service Level Agreements oder das Fehlerverhalten, sind für Entwickler, Softwarearchitekten, Enterprise Architekten, Projektleiter und Manager interessant
- Die Nutzung einer Schnittstelle lässt sich gut mit Beispielcode verdeutlichen

# Externe Schnittstellen

---

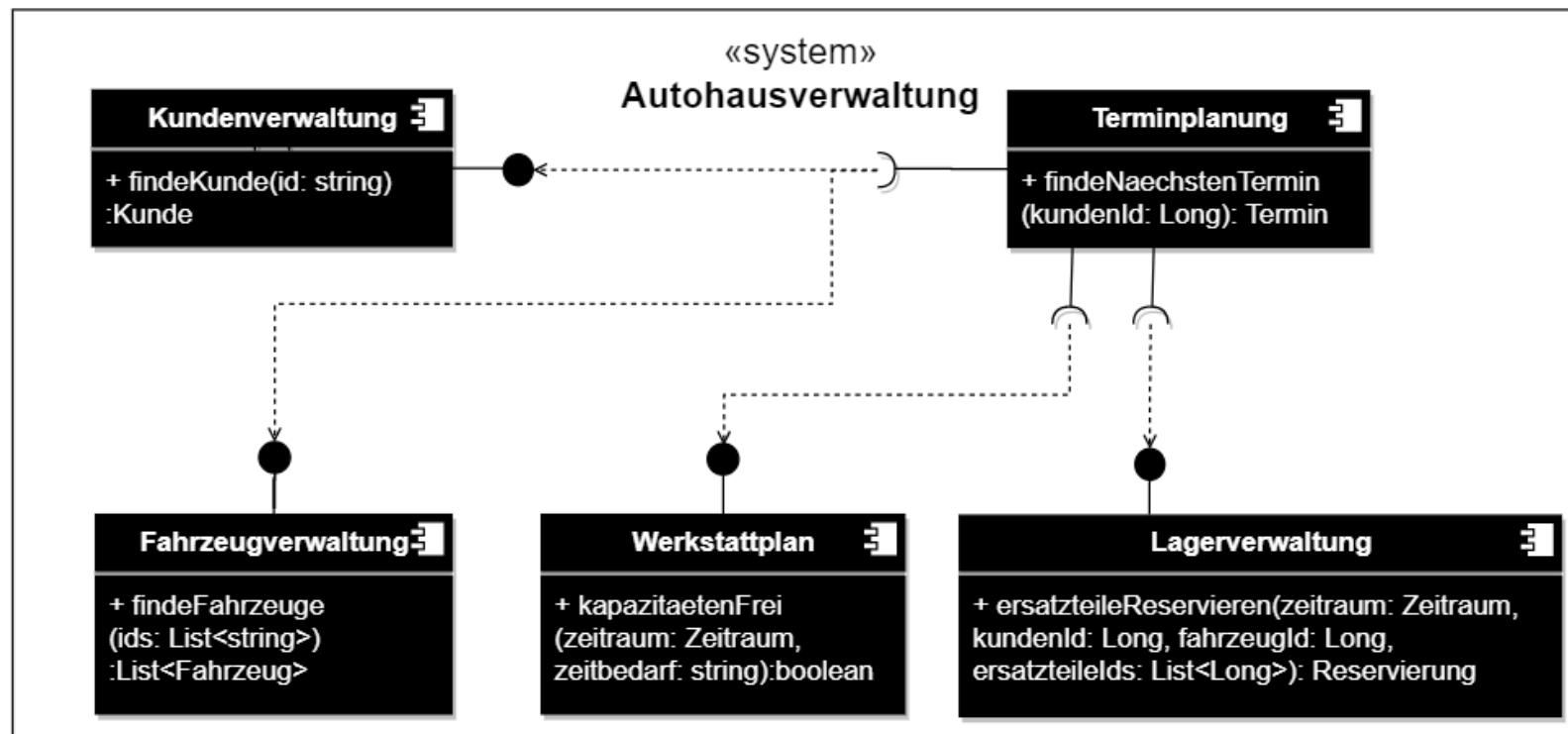
- **Externe Schnittstellen**  
Schnittstellen, die nach außen gerichtet sind und von anderen Systemen genutzt werden
- Externe Schnittstellen benötigen i.d.R. eine gesonderte Schnittstellendokumentation
- Die Qualität von externen Schnittstellen entscheidet über das erfolgreiche Zusammenwirken des Systems mit seinem Kontext

# Interne Schnittstellen

---

- **Interne Schnittstellen**  
Schnittstellen zwischen Bausteinen des Systems
- Die Begriffe interne - und externe Schnittstelle sind ebenfalls Bestandteil der Blackbox - und Whitebox Sichten
  - Die Blackbox Sicht zeigt die externen Schnittstellen des Bausteins
  - Die Whitebox Sicht zeigt die internen Schnittstellen zwischen Sub-Bausteinen / Klassen eines Bausteins

# Erweiterung der Bausteinsicht um Schnittstellen



# **Schablone zur Schnittstellenbeschreibung aus [Zörner 2015]**

---

## **1. Steckbrief**

- Name der Schnittstelle
- Kurzbeschreibung der Funktionalität
- Kurzbeschreibung der Interaktionspartner (Provider / Consumer)
- Version

## **2. Interaktionen**

- Funktion / Methode / Nachrichten Payload
- Beschreibung von Syntax (technisch) , Semantik (fachlich) und Fehlerbehandlung

## **3. Datentypen und Konstanten**

- Aufrufparameter, Rückgabewerte, Konstanten und Datenformate

# **Schablone zur Schnittstellenbeschreibung aus [Zörner 2015]**

---

## **4. Fehlerbehandlung**

- Generelles zur Fehlerbehandlung
- Detaillierung der Information unter 2

## **5. Einstellungen**

- Konfigurationsmöglichkeiten der Schnittstelle / Baustein

## **6. Qualitätsmerkmale**

- Beschreibung von Qualitätsmerkmalen (z.B. Antwortzeit)
- Referenz zu betroffenen Qualitätsszenarien

# **Schablone zur Schnittstellenbeschreibung aus [Zörner 2015]**

---

## **7. Entscheidungen**

- Fragen, Einflussfaktoren, Annahmen, Alternativen und Begründung für Entwurfsentscheidungen im Zusammenhang mit der Schnittstelle

## **8. Beispielverwendung**

- Pseudocode
- Prototyp
- Beispieldaten

# Schnittstellenbeschreibung für reserviereErsatzteile

---

## 1. Steckbrief

Die Schnittstelle **reserviereErsatzteile** des Bausteins Lagerverwaltung ermöglicht die Reservierung einer Liste von Ersatzteilen anhand ihrer ID.

Version 1

## 2. Interaktionen

- *Reservierung reserviereErsatzteile(Zeitraum zeitraum, Long kundenId, Long fahrzeugId, List<Long> ersatzteileIds)*
- Schnittstelle ist als Java Methode der Klasse *ReservierungsService* implementiert
- Kann eine Reservierung nicht durchgeführt werden, wird dies durch einen Status getrackt. Der Konsument kann anhand des Status selbst entscheiden, wie damit umzugehen ist.

# Schnittstellenbeschreibung für reserviereErsatzteile

---

## 3. Datentypen und Konstanten

- Die Klassen *Reservierung*, *ErsatzteilReservierung* und *Zeitraum* sind im Domänenmodell beschrieben  
([Link zum Domänenmodell im Confluence](#))
- Für die Klasse *ErsatzteilReservierung* kann der Status folgende Werte annehmen
  - ERSATZTEIL\_RESERVIERT
  - ERSATZTEIL\_BESTELLT
  - AUTOMATISCHE\_ERSATZTEIL\_RESERVIERUNG\_NICHT\_MÖGLICH
  - ERSATZTEIL\_ID\_UNBEKANNT
- Für die Klasse *Reservierung* kann der Status folgende Werte annehmen
  - RESERVIERUNG\_ERFOLGREICH
  - AUTOMATISCHE\_RESERVIERUNG\_NICHT\_ERFOLGREICH

# Schnittstellenbeschreibung für reserviereErsatzteile

---

## 4. Fehlerbehandlung

- Während der Erstellung einer Reservierung können ausschließlich RuntimeExceptions auftreten
- Fachlich auftretende Fehler hinsichtlich einer Reservierung werden durch den Status beschrieben (siehe 2 und 3)

## 5. Einstellungen

- Keine weiteren Konfigurationsmöglichkeiten

## 6. Qualitätsmerkmale

- Performanz – Antwort kommt binnen 3 Sekunden
- Verfügbarkeit – 5:00 Uhr bis 23:00 Uhr
- Sicherheit – OAuth2 Authentifizierung
- Siehe [Link zu den Qualitätsmerkmalen im Confluence](#)

# Schnittstellenbeschreibung für reserviereErsatzteile

---

## 7. Entscheidungen

- Die Schnittstelle soll alle Informationen anhand der Parameter erhalten, sodass der Baustein zustandslos arbeiten kann
- Die Reservierung hält eine Referenz auf den Kunden und das Fahrzeug, für das die Ersatzteile benötigt werden
- Die Schnittstelle bietet die Möglichkeit eine Menge von Ersatzteilen zu reservieren. Das Ergebnis der Reservierung wird via Status getrackt und dem Konsumenten zur Auswertung angeboten. Ein Status wird auf Ebene eines Ersatzteils sowie kumuliert im Reservierungsauftrag bereitgestellt

# Schnittstellenbeschreibung für reserviereErsatzteile

---

## 8. Beispielverwendung

```
Long kundenid = 4711L;
```

```
Long fahrzeugId = 1337L;
```

```
Zeitraum zeitraum = new Zeitraum(2020, 2, 12, 60);
```

```
List<Long> ersatzteile = List.of(42L, 4815162342L);
```

```
Reservierung reservierung = new ReservierungsService()
```

```
.reserviereErsatzteile(zeitraum, kundenId, fahrzeugId, ersatzteile);
```

# Sicherheitsaspekte in der Schnittstellenbeschreibung

---

An die vorgestellte Schablone für die Schnittstellenbeschreibung lässt sich der Aspekt **Security** hinzufügen, für z.B.

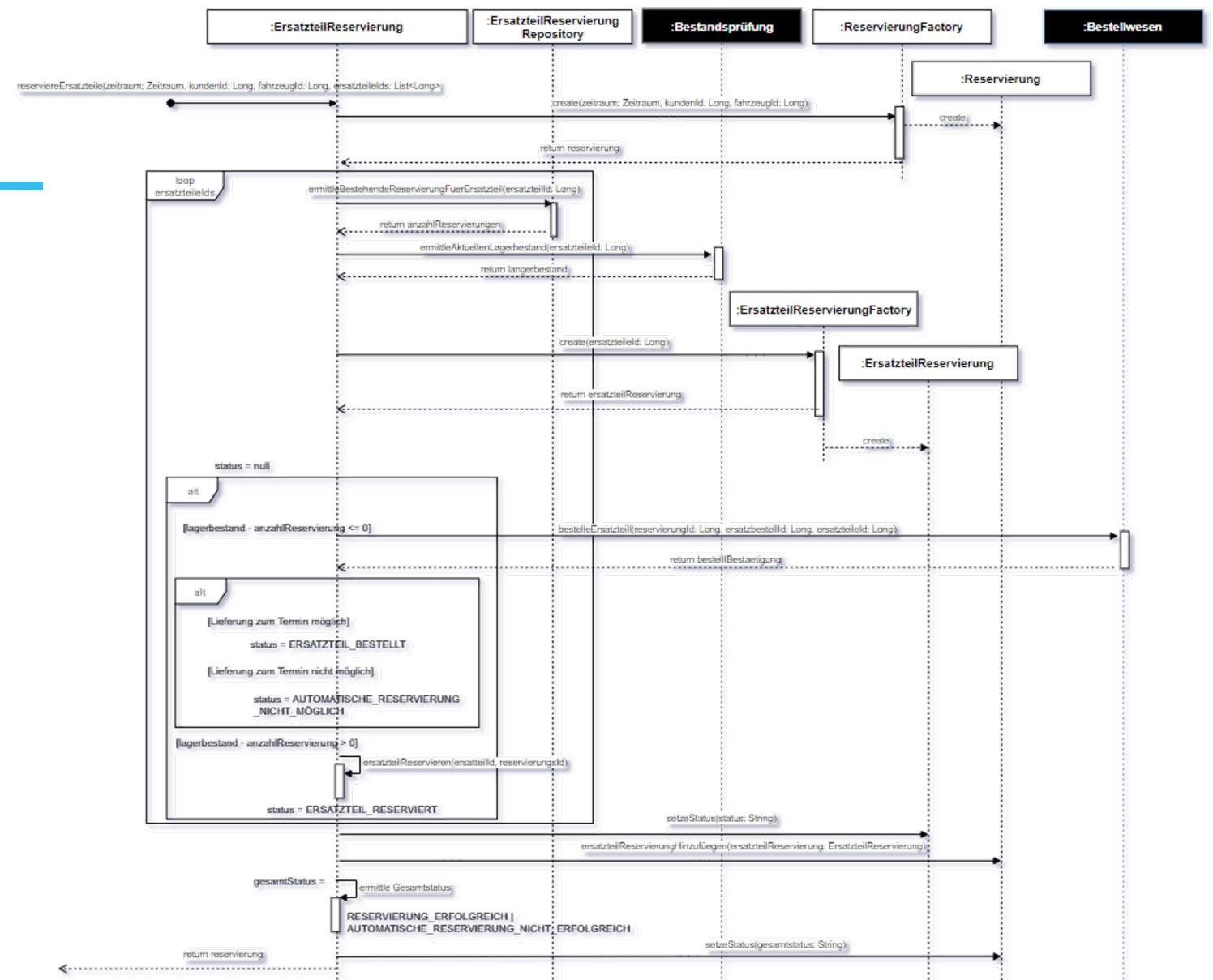
- Benötigte Rollen und Rechte für die Geschäftsfunktionen und Geschäftsobjekte
- Authentifizierungs- und Autorisierungsverfahren (Zertifikate, Login, 2-Factor, etc.)
- Tokens (Json Web Token)
- Protokolle (OAuth, OpenID, SAML)
- Verschlüsselung der Kommunikation und Daten (SSL, HTTPS, Private-Key Public-Key Verfahren)

# Laufzeitsicht

- Detailliertes Sequenzdiagramm



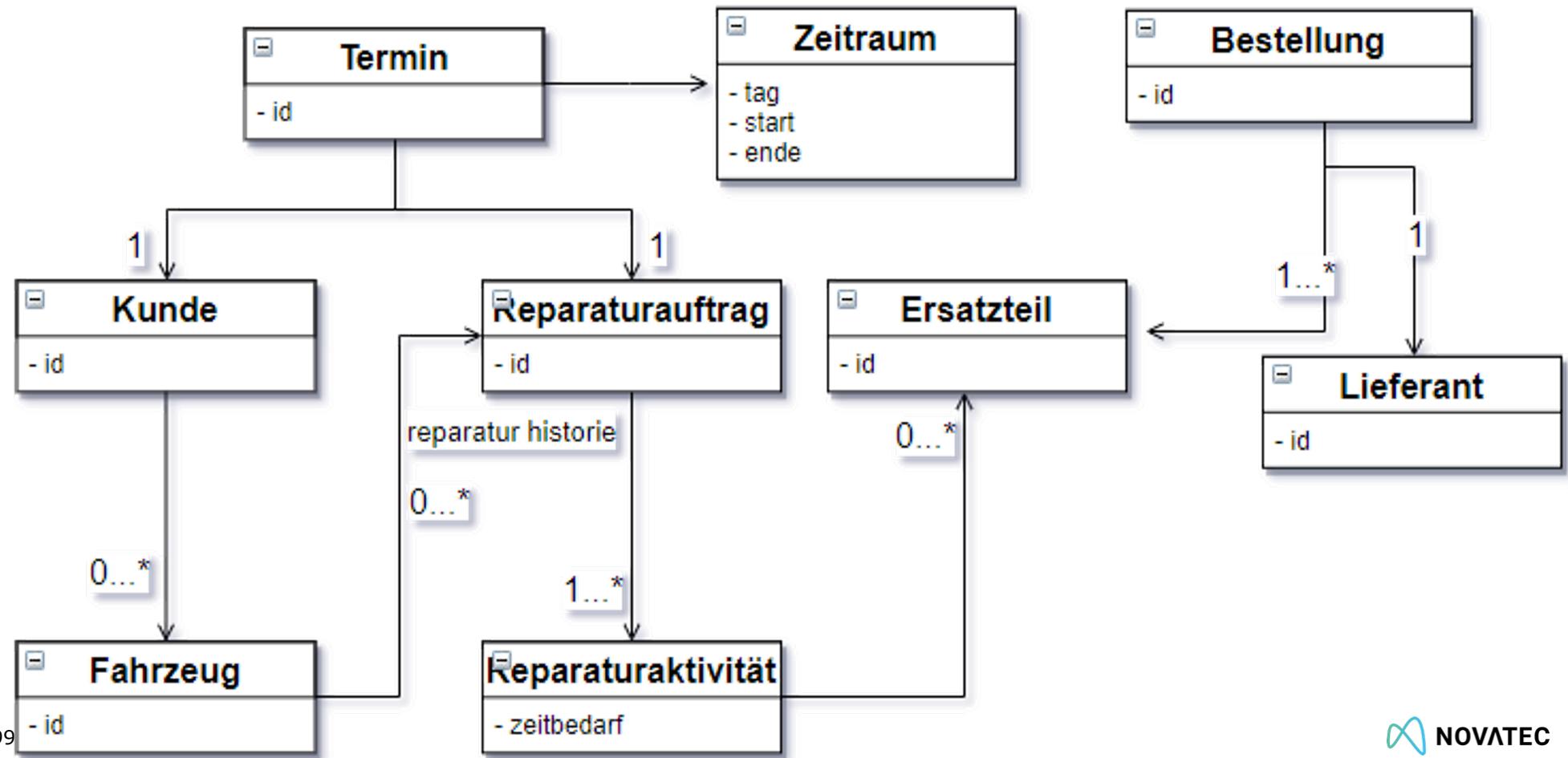
Zum Nachlesen



# Domänenmodell



Zum Nachlesen



# Entwurf Domänenmodell

---



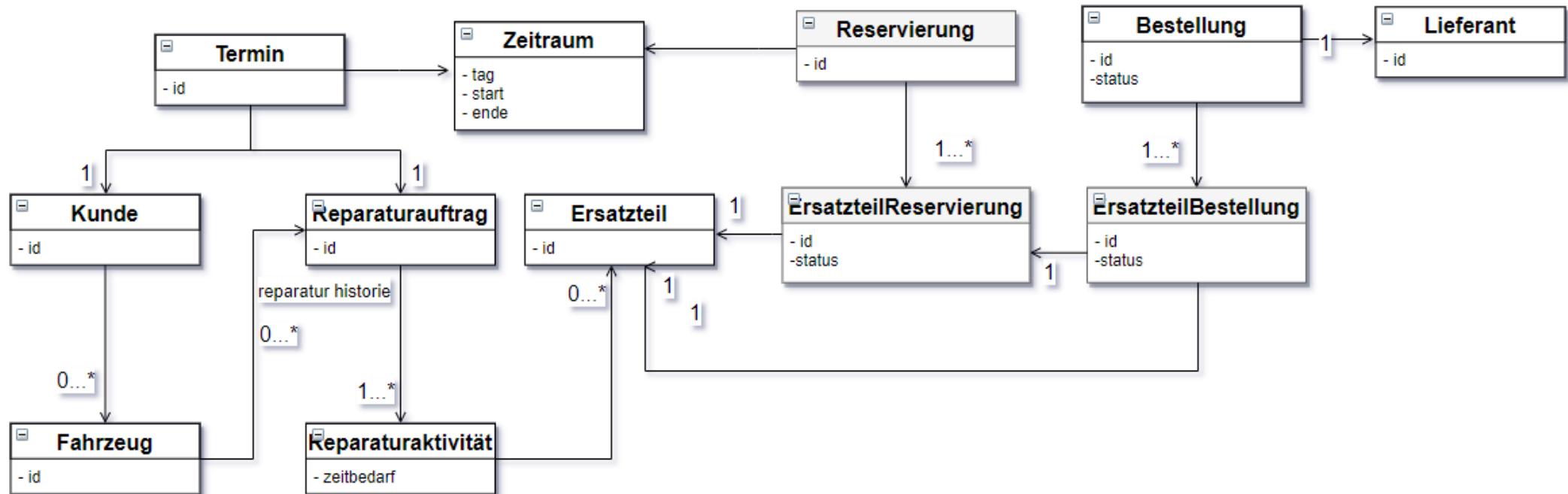
Zum Nachlesen

- Ein erstes Domänenmodell entsteht auf Basis der gewonnenen Information durch Analyse der fachlichen Abläufe
- Ein Bild entsteht durch ein Konzeptmodell, welches sich noch weiter entwickeln wird
- Assoziationen und Attribute werden sich noch ändern und ergänzt werden
- Aggregates und Entitäten werden identifiziert
- Weitere Ergänzung mit Prinzipien, Regeln und Muster für Domänenmodelle und Datenarchitekturmuster

# Wechselwirkung Domänenmodell



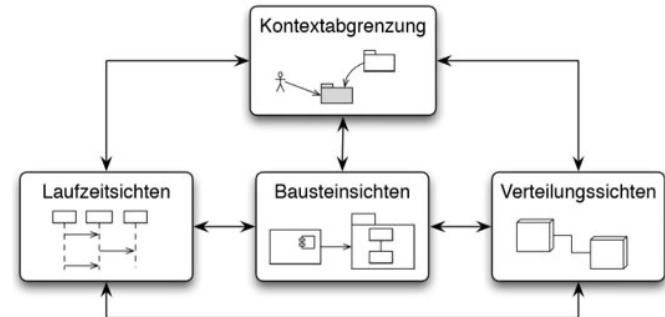
Zum Nachlesen



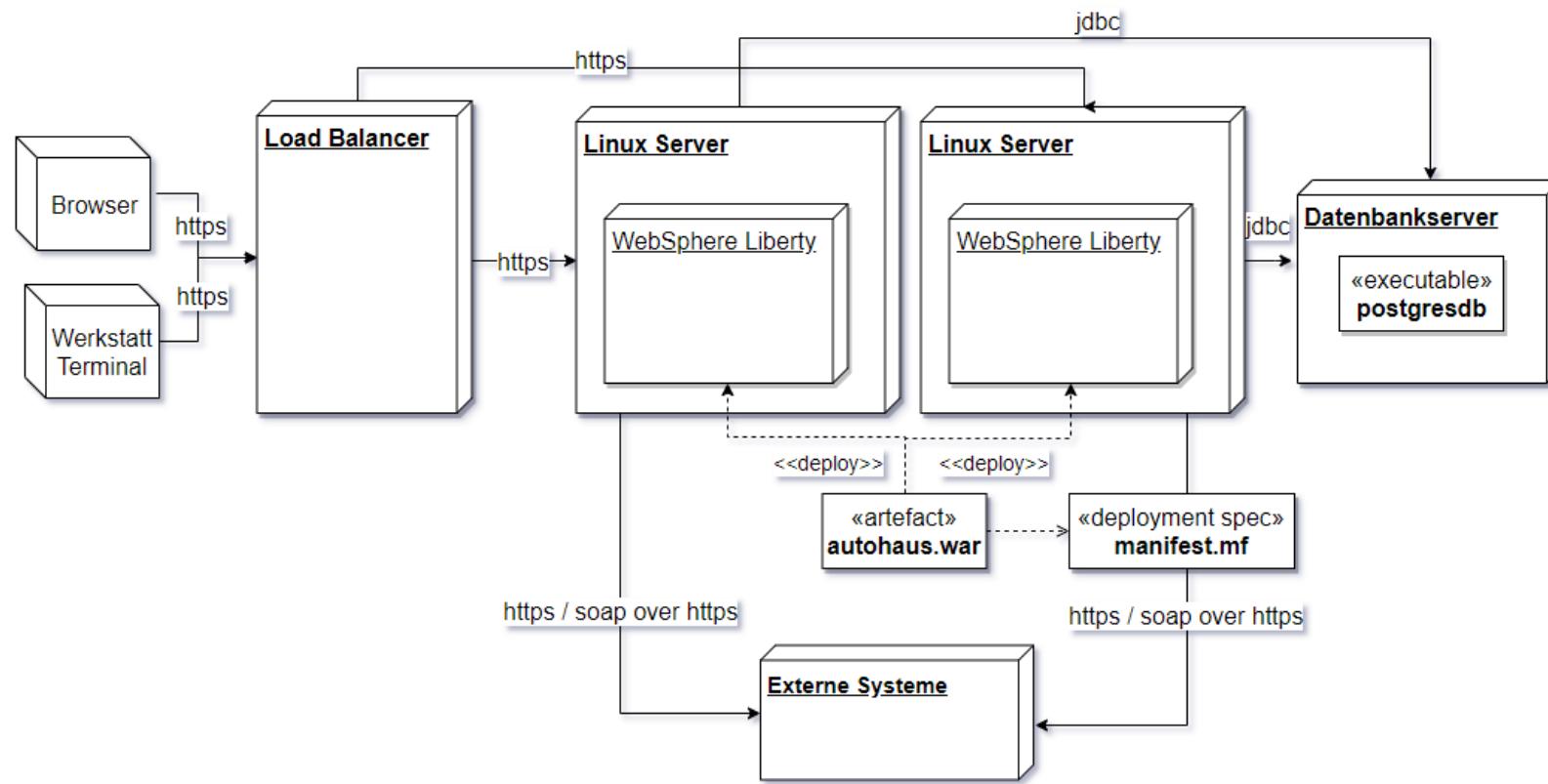
# Verteilungssicht

Beschreibt normalerweise in einer Kombination aus Diagrammen, Tabellen und Text ...

- die Verteilung des Systems auf mehrere Standorte, Umgebungen, Computer, Prozessoren usw. sowie die physischen Verbindungen zwischen ihnen
- Qualitäts- und/oder Leistungsmerkmale der Infrastruktur
- die Zuordnung von Softwareartefakten (Bausteinen) zu Elementen der Infrastruktur
- die Verfeinerung des technischen Systemkontexts.



# Verteilungssicht des Autohausverwaltungssystem



# **Stakeholder der Verteilungssicht**

---

- Betrieb
- Unternehmensarchitekt:innen
- Softwarearchitekt:innen
- Entwickler:innen
- Bewerter:innen im Rahmen einer Architekturbewertung

# Wechselwirkungen zwischen den Sichten

- Systemkontext als Basis
- Bausteine interagieren zur Laufzeit (Bausteinsicht und Laufzeitsicht)
- Bausteine sind bereitgestellt auf Hardware (Bausteinsicht und Verteilungssicht)
- Informationen, die bei der Erstellung einer Sicht gewonnen werden, verändern / verfeinern eine andere Sicht  
siehe Aktivitätsdiagramm, erstes Konzeptmodell, Bausteinsicht
- Änderungen und Entscheidungen für eine Sicht beeinflussen eine andere Sicht

Beispielweise zeigt die Laufzeitsicht, dass angedachte Abläufe mit den entworfenen Strukturen ineffizient sind. Die Strukturen, sprich die Bausteinsicht, muss angepasst werden.

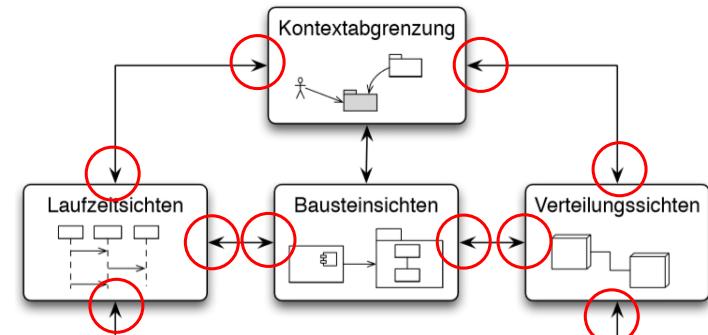


Abbildung aus  
[Starke 2015]

# Add-On: Dokumentation von Mustern

---

- Wann sollten Muster dokumentiert werden? Wenn
  - Das eingesetzte Muster ein oder mehrere **zentrale Qualitätsziele** adressiert
  - das Muster den Leser:innen **hilft**, die **Systemstrukturen** besser zu **verstehen**
  - Leser:innen (Entwickler:innen) sich an das Muster halten sollen
- Muster aus LZ 2, insbesondere Entwurfsmuster, kommunizieren ihre Kernidee durch ihren Namen  
Annahme: Entwurfsmuster der Gang of Four gehören zum Vokabular von Entwickler:innen und Architekten:innen (Wiedererkennungswert)
- Verwendung der Musternamen in Klassen- und Methodenbezeichnern
- Gleiches gilt für abstraktere Ebenen in Form von Architekturstilen und –mustern, wie z.B. Schichten, Broker, Microservices, REST, Messaging...  
Layer im Packagename / Event oder Ressource im Klassennamen / Model und View im Klassennamen

# Dokumentation von Mustern in arc42

**Architekturstile- und Muster mit grober Strukturierung im Zusammenspiel mit anderen Entscheidungen und Strategien.**

**Muster beschreiben vorwiegend Strukturen auf Baustein- und Klassenebene.  
Die Bausteinsicht ist der „natürliche“ Ort für Muster**

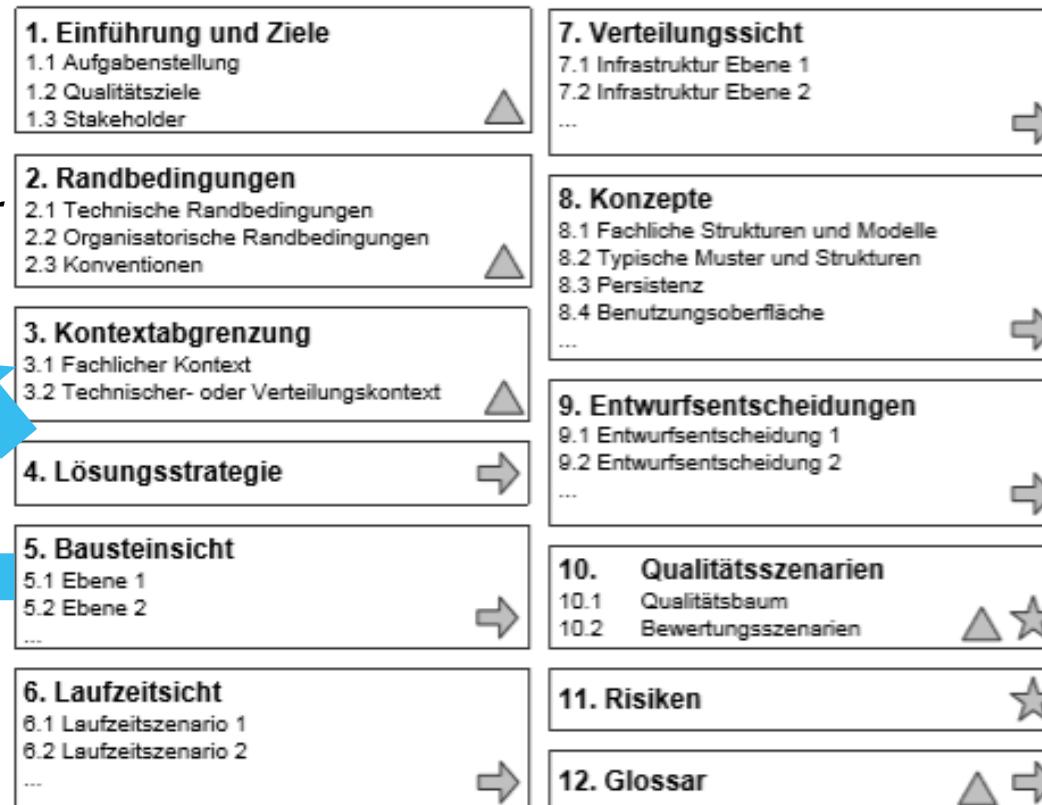


Abbildung aus [Zörner 2015]

# Übung

---

- Erstellen Sie eine Verteilungssicht der Anwendung VerSicher24mal7 unter Berücksichtigung ihrer gewählten Implementierungstechnologie und Laufzeitumgebung.

# LZ 3-6: Querschnittskonzepte dokumentieren und kommunizieren (R2)

---

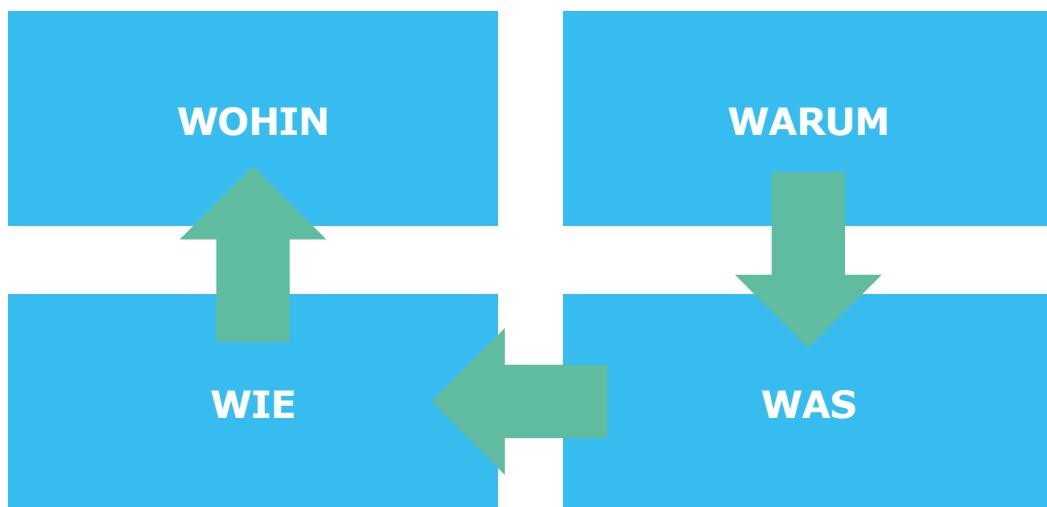
- Softwarearchitekt:innen können typische Querschnittskonzepte (synonym *Prinzipien*, *Aspekte*) adäquat dokumentieren und kommunizieren, z. B. Persistenz, Ablaufsteuerung, UI, Verteilung/Integration, Protokollierung.

# Dokumentation von querschnittlichen Architekturkonzepten

---

- **Identifikation** der relevanten technischen und fachlichen Querschnittsthemen
  - Welche Aspekte sollen innerhalb des Systems konsistent gelöst werden?
  - Welche technische Fragestellung sind darüber hinaus relevant?
- **Priorisierung** der Themen
  - Wo sind Unsicherheiten und Risiken?
  - Welches Thema ist für viele / alle fachlichen Bausteine relevant?
  - Welche Themen adressieren Qualitätsziele?
- **Konzeption und Dokumentation**  
Document Continuously - Dokumentation und Entwicklung verzahnen

# Dokumentation von Querschnittskonzepten



**Vier-Quadranten-Modell nach Vigenschow**

Aus [Zörner 2015]

## 1. Aufgabenstellung

1. Anforderungen, Architektur- und Qualitätsziele

2. Begriffe

**Warum**

## 2. Lösung

1. Kontext und Einflussfaktoren
2. Optionen im Lösungsraum
3. Lösung
4. Quellen

**Was**

## 3. Anwendung

1. Bibliotheken
2. Konfiguration
3. Beispiele

**Wie**

## 4. Ausblick

1. Offene Punkte
2. Nächste Schritte
3. Anregungen

**Wohin**

# Vier-Quadranten-Modell nach Vigenschow

---

## Warum (1)

- Relevanz für die Architektur aufzeigen
- Bezug auf Qualitätsziele, Anforderungen, Risiken und Architekturentscheidungen herstellen
- Wichtigkeit eines konsistenten Konzepts darstellen

## Was (2)

- Kontext und Einflussfaktoren beschreiben
- Alternativen aufzeigen (falls vorhanden)
- Detailbeschreibung des Konzepts inklusive Begriffe / Diagramm plus Text
- Verweise auf Quellen

## Wie (3)

- Konkrete Informationen für die Konzeptumsetzung
- Beispielcode, Tutorials, Screenshots

## Wohin (4)

- Beschreibung der Zukunftsvorstellung für das Lösungskonzept mit bereits eingeplanten Erweiterungspunkten
- Skizzieren der angedachten Weiterentwicklung

# Vier-Quadranten-Modell: Beispiel Logging

---

## Warum (1)

- Logging stellt ein querschnittliches Konzept dar, das von **allen Bausteinen einheitlich** umgesetzt werden muss
- Qualitätsziel: Bestimmung von Auslösern von Fehlern in weniger als 15 Minuten
- Qualitätsziel: Effektive Betriebsüberwachung (Aussagekräftige Fehlermeldung, Einheitliche Loglevel)

## Was (2)

- Zentrale Logging Komponente, Verwendung durch Entwickler und DevOps

## Was (2) (Forts.)

- Korrelation der Fehler über alle Ebenen der Anwendung hinweg
- Werkzeug zur Auswertung von Loginformationen
- Konkrete Umsetzung abhängig von Implementierungstechnologie
- Zipkin (verteiltes Tracing von Anwendungen), Umsetzung in Spring Cloud: Sleuth
- Alternativen:
  - Splunk
  - Elastic Search/Kibana

# Vier-Quadranten-Modell: Beispiel Logging

---

## Wie (3)

- Spring Boot Micoservices binden spring-cloud-starter-sleuth dependency ein, @NewSpan, ...
- Bei CloudFoundry als Runtime: Logging nach Standard-Out, loggregator sendet Loginformationen zu logstash, <https://medium.com/oneclicklabs-io/streaming-spring-boot-application-logs-to-elk-stack-part-1-a68bd7cccaeb>

## Wohin (4)

- Einbindung von UI-Logs
- Langzeitspeicherung von Security-relevanten Informationen
- Revisionssichere Speicherung

# LZ 3-8: Architekturentscheidungen erläutern und dokumentieren (R1 - R2)

---

- Softwarearchitekt:innen können:
  - Architekturentscheidungen systematisch herbeiführen, begründen, kommunizieren und dokumentieren
  - gegenseitige Abhängigkeiten solcher Entscheidungen erkennen, kommunizieren und dokumentieren.
- Softwarearchitekt:innen kennen Architecture-Decision-Records (ADR, siehe [Nygard 2011]) und können diese zur Dokumentation von Entscheidungen einsetzen (R2).

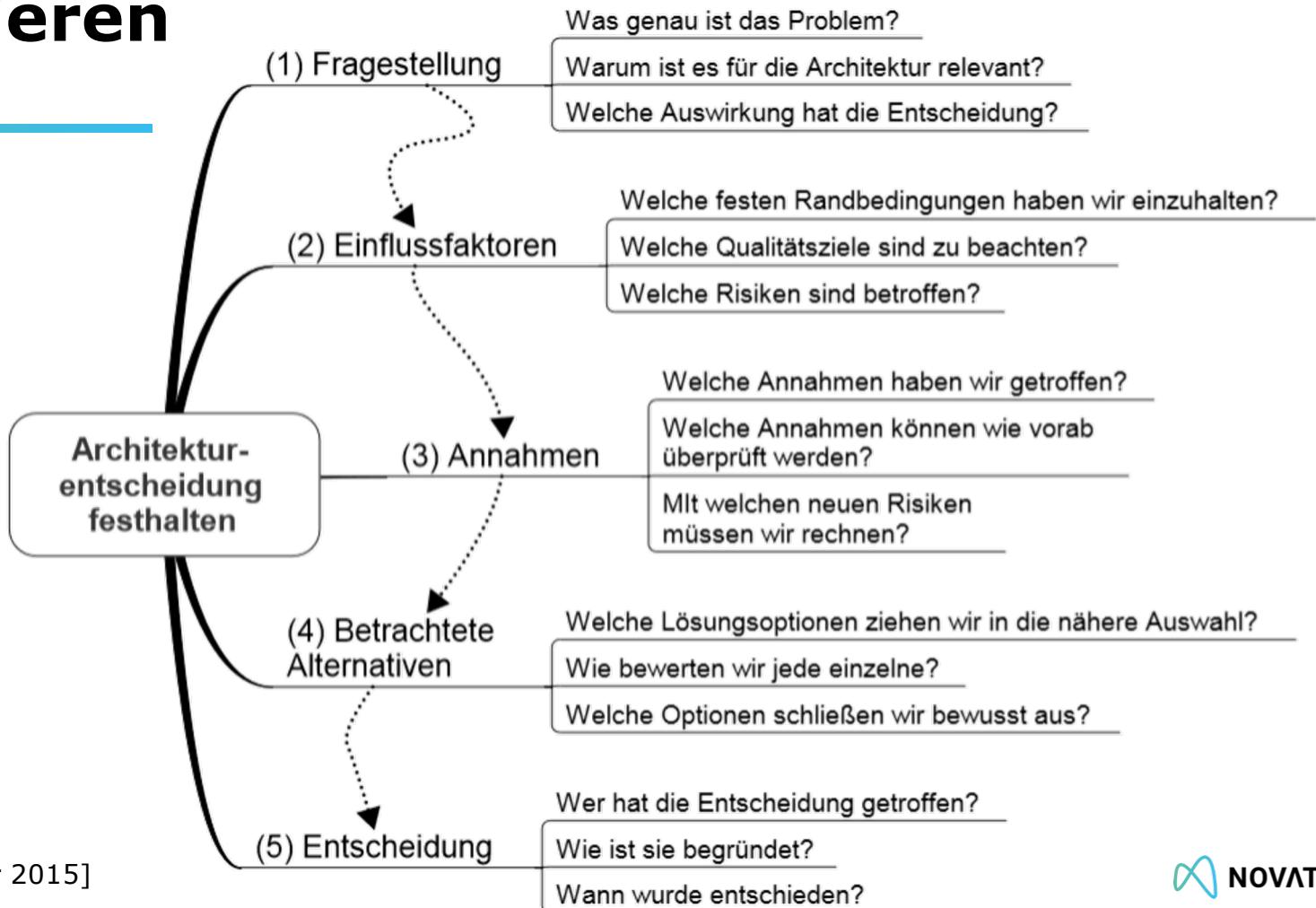
# Architekturentscheidungen erarbeiten und dokumentieren

---

## Typische Fragestellung

- Wie kommunizieren **Bausteine**?
- Wie integrieren wir **Umsysteme**?
- Welche **Technologie** für Problem XY?
- Make or Buy?
- Wie lösen wir **Querschnittsthemen**?  
(Persistenz, Logging, Caching, Sicherheit)
- Welche **Fachlichkeit** wird in welchem Baustein abgebildet?
- Welche **Geschäftsobjekte / Aggregates** existieren?
- Zuordnung von **Entitäten** zu Geschäftsobjekten und Aggregates

# Architekturentscheidungen erarbeiten und dokumentieren



# Architekturentscheidungen erarbeiten und dokumentieren

---



Zum Nachlesen

**Transparenz** bezüglich **Fragestellung** und **Lösung** herstellen

- Qualitätsziele definieren und priorisieren (siehe LZ 4)
- Einflussfaktoren und Rahmenbedingungen beschreiben
- Architektonische Fragestellung zu gewünschten Qualitätseigenschaften in einer Kreuztabelle in Beziehung setzen

# Architekturentscheidung: Ablaufsteuerung

---

## **Fragestellung:**

Auf welche Weise werden wiederkehrende Abläufe (Prozesse) explizit abgebildet, so dass diese bei Bedarf angepasst werden können.

## **Einflussfaktoren:**

- Es ist bereits ein kommerzielles BPM-Werkzeug im Einsatz. Allerdings existieren Andeutungen, dass die Lizenzen nicht verlängert werden sollen.
- Die Lösung sollte einfach einzusetzen sein und möglichst wenig Spezial Know-how voraussetzen.

## **Annahmen:**

Für den aktuell umzusetzenden Ablauf eines Outputmanagementsystems sind Anpassungen am Ablauf eher selten.

# Architekturentscheidung Ablaufsteuerung

| Kriterium               | Kommerzielles BPM      | Open Source BPM        | Eigenentwicklung auf Basis von Apache Camel |
|-------------------------|------------------------|------------------------|---|
| Fachliche Anforderungen | +                      | +                      | +   |
| Entwicklungsaufwand     | - (fehlendes Know-how) | - (fehlendes Know-how) | o   |
| Deployment-Aufwand      | -                      | o                      | +   |
| Pflegeaufwand           | - (Versionsupdates)    | - (Versionsupdates)    | o (Versionsupdates Software)                |
| Betrieb                 | -                      | +                      | +   |
| Zukunftsähigkeit        | - (Abbau geplant)      | o (erster Anwender)    | +   |
| Lizenzkosten            | -                      | +                      | +   |
| Performanz              | o                      | o                      | o   |
| Infrastrukturkosten     | -                      | +                      | +   |

# Architekturentscheidungen erarbeiten und dokumentieren

---

- Architekturentscheidung bzw. Lösungsstrategie kann wirkungsvoll in einer Tabelle dokumentiert werden
- Dadurch wird ein direkter Bezug zum „Warum“ hergestellt
- Dient als schnelle Übersicht für Entwickler:innen, Architekt:innen und Projektleiter:innen
- Details der ausgewählten Lösung sollten separat beschrieben werden  
(siehe Vier-Quadranten Modell und Abschnitt Lösungsstrategien in arc42)

# Architekturentscheidungen erarbeiten und dokumentieren



Zum Nachlesen

| Qualitätsziele | Lösungsansätze                                      |
|----------------|---|
| Ziel 1         | * Ansatz (a)<br>* Idee (b)                          |
| Ziel 2         | * Konzept (c)<br>* Ansatz (a)<br>* Entscheidung (d) |
| ...            | * ...   |

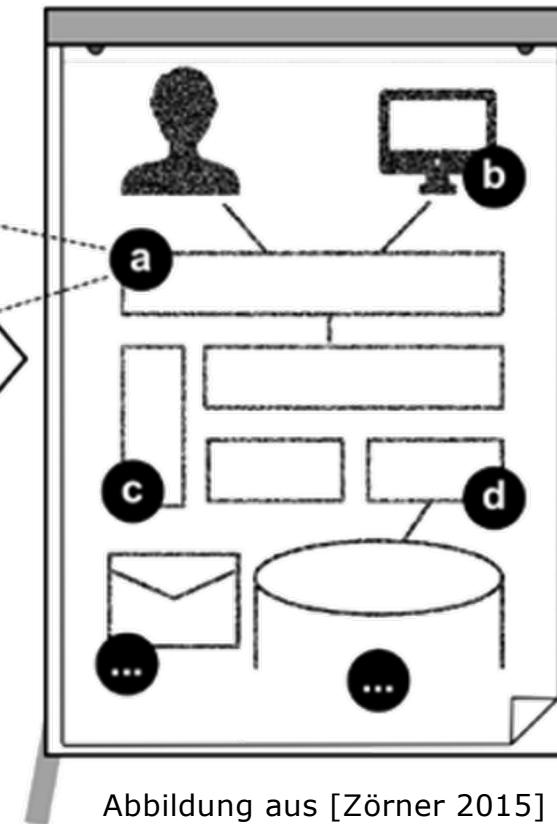


Abbildung aus [Zörner 2015]

# Dokumentation von Architekturentscheidungen in Form von Architecture Decision Records

- Das Konzept geht auf eine Idee von Michael Nygard zurück.  
<https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>
- Der Fokus liegt hier auf der Entscheidung, nicht auf dem Weg dorthin
- Die Struktur besitzt folgende Elemente:
  - **Titel** (title): ein kurzer Satz mit weniger als 50 Zeichen, der den Inhalt treffend beschreibt.
  - **Kontext** (context): Was ist das Problem, dass zu dieser Entscheidung geführt hat?
  - **Entscheidung** (decision): Was konkret ist es, dass wir vorschlagen oder machen?
  - **Status** (status): Beispiele wären *vorgeschlagen*, *angenommen*, *zurückgewiesen*, *ersetzt* oder *überholt*.
  - **Konsequenzen** (consequences): Was wird einfacher oder schwieriger auf Grund dieser Entscheidung?

# ADR: Fachliche Protokollierung

---

## **Titel**

Fachliche Protokollierung aller Kunden-Interaktionen

## **Kontext:**

Es sollen Auswertungen möglich sein wie viele Kunden noch aktiv sind sowie die Gründe für Neukunden und Abwanderung von Kunden auswertbar sein.

## **Entscheidung:**

Die Aktionen der Kunden werden mit einem fachlichen Grund versehen (Kauf / Verkauf / Leasing / Kündigung) und in einer NoSQL-Datenbank protokolliert, um zukünftig eine einfache Erweiterung der Daten zu ermöglichen.

## **Status:**

Proposed

# **ADR: Fachliche Protokollierung**

---

## **Konsequenzen**

- Die Aktionen der Kunden können fachlich ausgewertet werden und auf dieser Basis können KPIs bestimmt werden (Neukunden-Rate, Prozent der Bestandskunden abhängig von der Laufzeit, Kündigungsrate, etc.).
- Es müssen Dashboards für die fachlichen Kennzahlen (KPIs) erstellt werden. Hier muss eine Entscheidung über die verwendete Technologie erfolgen.
- Eine NoSQL-Datenbank muss ausgewählt werden und in das bestehende System integriert werden.

# Übung

---

- Erstellen Sie eine Architekturentscheidung in VerSicher24mal7 für die von Ihnen gewählte Schnittstellentechnologie von mindestens einer Komponente oder eines der verwendeten Frameworks für die Implementierung.
- Setzen Sie dazu die Schablone für Architekturentscheidungen oder Architecture Decision Records ein.

## LZ 3-9: Dokumentation als schriftliche Kommunikation nutzen (R2)

---

- Softwarearchitekt:innen nutzen Dokumentation zur Unterstützung bei Entwurf, Implementierung und Weiterentwicklung (auch genannt *Wartung* oder *Evolution*) von Systemen.

# Zentrale Architekturbeschreibung

---

- **Kerndokument** einer Softwarearchitektur
- Enthält alle **architekturelevanten Informationen**
  - Aufgabenstellung, Ziele, Qualitätsanforderungen
  - Projektteilnehmer:innen und Stakeholder:innen
  - Randbedingungen
  - Risiken
  - Sichten, Entscheidungen, dokumentierter Einsatz von Muster
  - Quellcodeorganisation
  - Qualitätsmerkmale und -szenarien

# Zielgruppen-optimierte Inhalte

---

## Bereitstellung durch:

- Auf unterschiedliche Zielgruppen ausgerichtete **Dokumente**
- Zielgruppen sind Entwickler:innen, Betrieb, Tester:innen, Qualitätssicherer:innen, Enterprise Architektur, Projektleitung, Führungsebene und Management
- Wie kann man dies erreichen
  - **Schablonen**  
Diese schaffen allgemein Verständlichkeit und helfen beim „herauspicken“ von Zielgruppen relevante Informationen  
Gilt für den Leser und den Architekten beim Erstellen von Zielgruppen-optimierten Dokumentenartefakten
  - **Sichtenmodelle**

# Zielgruppen-optimierte Zusatzdokumente

| Dokument                     | Kurzbeschreibung   | Stakeholder  |
|------------------------------|--|--|
| <b>Architekturüberblick</b>  | Schnell lesbare Kurzfassung < 30 Seiten der zentralen Architekturbeschreibung<br>(Wichtigste Sichten, hoch priorisierte Qualitätsanforderungen, Kernentscheidungen)      | Projektleiter:innen / Product Owner / Agile Master<br>Unterste Führungsebene<br>Enterprise Architekt:innen<br>Neue Entwickler:innen / Architekten:innen<br>Qualitätsprüfer:innen |
| <b>Übersichtpräsentation</b> | Präsentation der (technischen) Architektur < 60 Minuten  | Projektleiter:innen / Product Owner / Agile Master<br>Enterprise Architekt:innen<br>Unterste Führungsebene bis Top Management  |
| <b>Architekturtapete</b>     | (Grafische) Darstellung wichtiger Architekturaspekte in einem Gesamtüberblick. Mittel zur Kommunikation der Architektur und für die interaktive Diskussion von Lösungen. | Projektleiter:innen / Product Owner / Agile Master<br>Entwickler:innen und Architekt:innen<br>Enterprise Architekt:innen   |

Aus [Gharbi 2015]. Dort gibt sind noch weitere Zusatzdokumente aufgeführt.

# **LZ 3-10: Weitere Hilfsmittel und Werkzeuge zur Dokumentation kennen (R3)**

---

- Softwarearchitekt:innen kennen:
  - Grundlagen mehrerer publizierter Frameworks zur Beschreibung von Softwarearchitekturen, beispielsweise:
    - ISO/IEEE-42010 (vormals 1471)
    - Arc42
    - C4
    - FMC
  - Ideen und Beispiele von Checklisten für die Erstellung, Dokumentation und Prüfung von Softwarearchitekturen
  - mögliche Werkzeuge zur Erstellung und Pflege von Architekturdokumentation

# Framework zur Beschreibung von Softwarearchitektur mit arc42

---



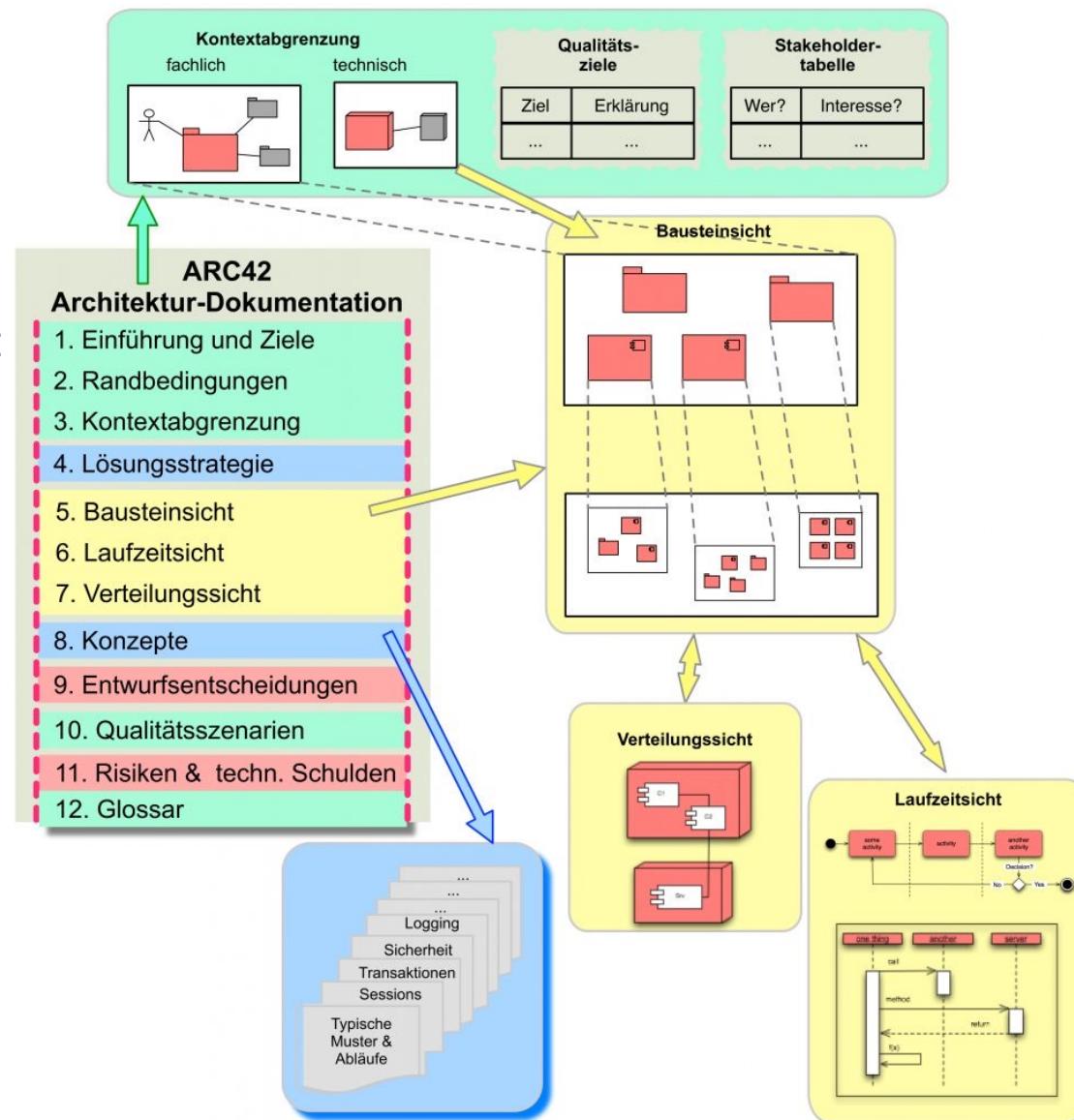
<https://arc42.de/>

- **Architekturprozess** (Wie wird Architektur erstellt?) und Vorlage zur **Architektdokumentation** (Wie wird Softwarearchitektur nachhaltig dokumentiert?)
- Best Practices auf Basis langjähriger Projekterfahrung
- Kämpft gegen das Vorurteil, dass das gesamte Dokument ausgefüllt werden muss
- Vielmehr hat dies jedoch den Charakter einer Checkliste und ist gleichzeitig die Basis zur Ableitung zielgruppengerechter Inhalte
- Ist frei und kostenlos in unterschiedlichen Formaten verfügbar

# arc42

<https://arc42.org/overview>

Die Farbgebung verdeutlicht die inhaltliche Einordnung der verschiedenen Entwurfsartefakte in die Dokumentation.



# arc42 Struktur



Zum Nachlesen

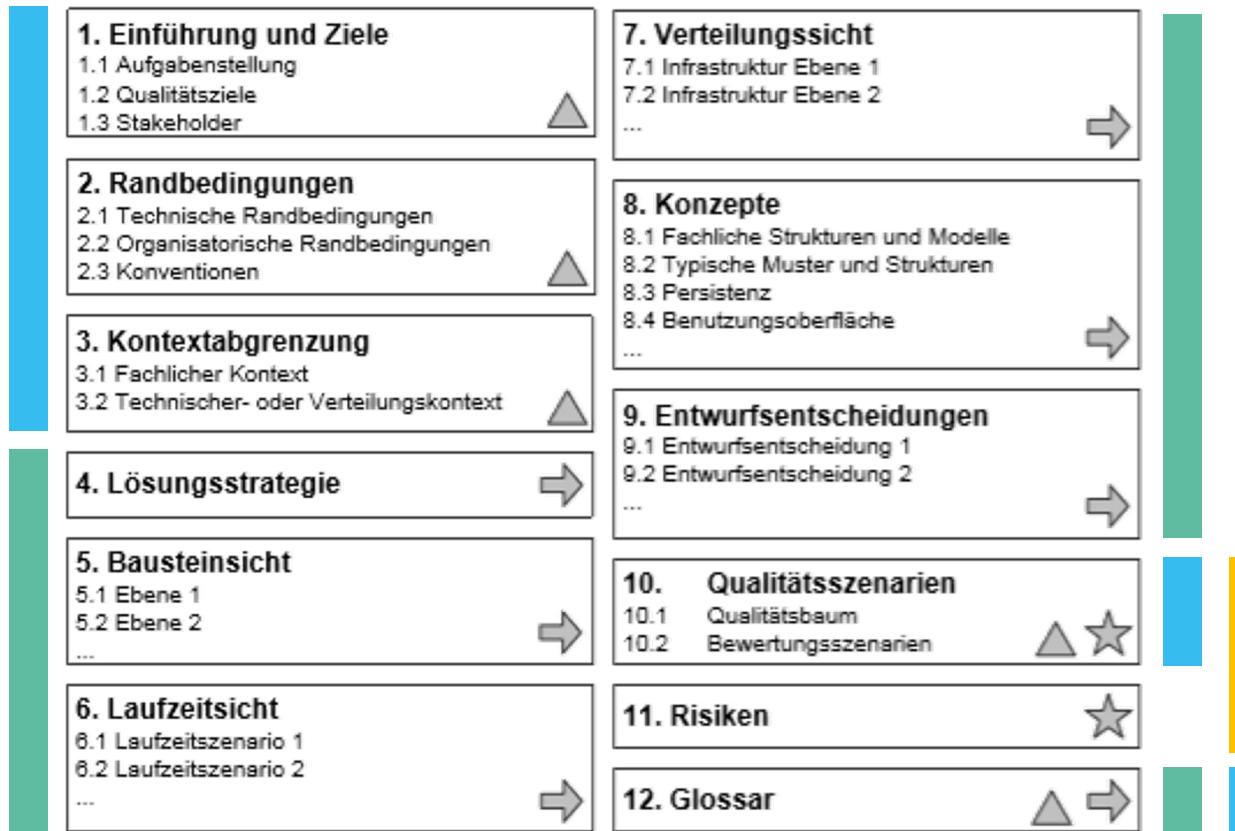


Abbildung aus [Zörner 2015]

# Weitere Frameworks zur Beschreibung von Softwarearchitektur

---

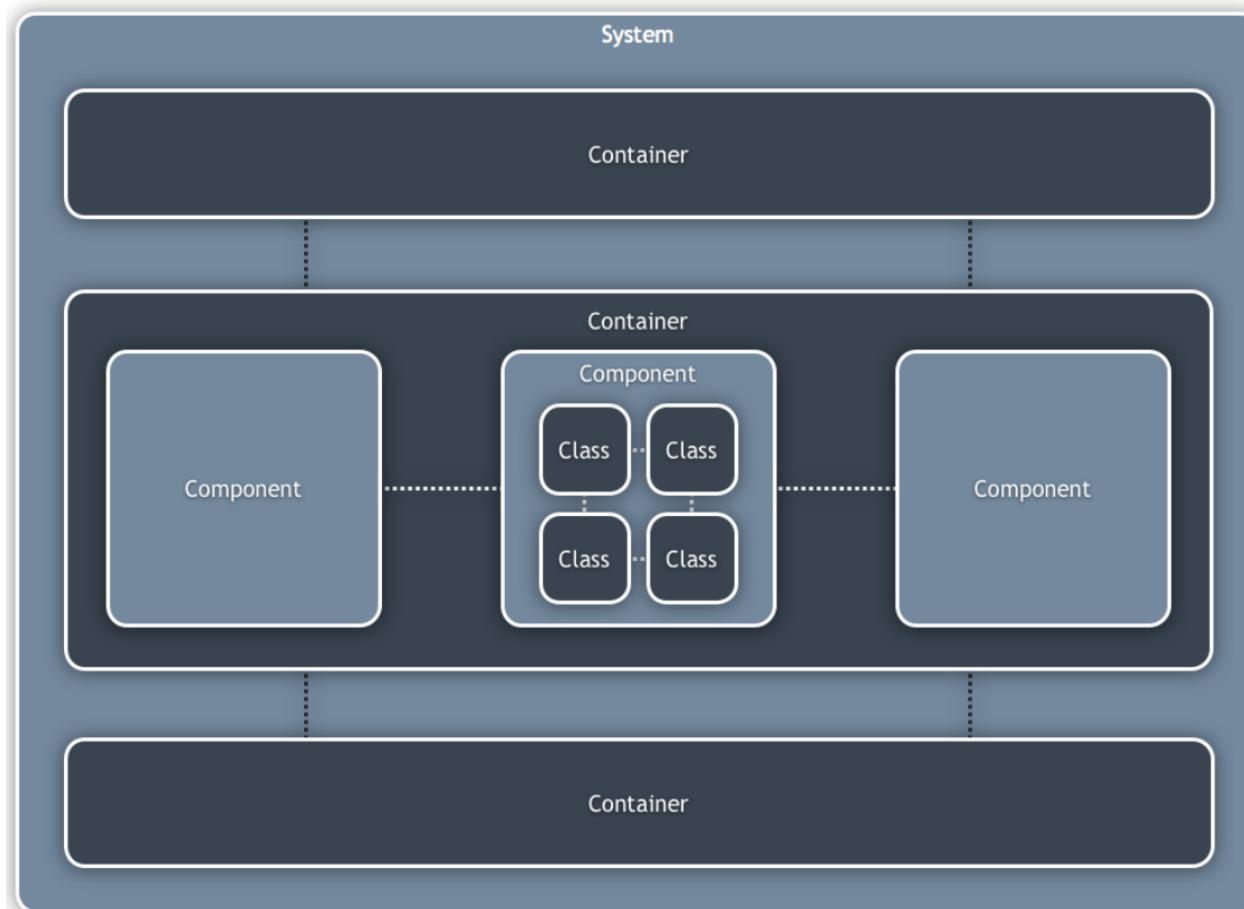
- **Recommended Practice for Architectural Description of Software-Intensive Systems**  
ISO/IEEE-42010 (ehemals IEEE 1471) ist ein Standard des IEEE
  - Definiert Begriffe für die Beschreibung von Softwarearchitekturen
  - Enthält jedoch keine konkreten Sichten oder Gliederungsvorgaben
- **Context, Container, Component and Class**  
C4 Model nach Simon Brown
- **Fundamental Modeling Concepts (FMC)**

# **Context, Container, Component and Classes (C4)**

---

- Fokussiert eine „leichtgewichtige“ Beschreibung der statischen Strukturen
- Hält UML und UML Tools für uneffektiv
- Hat eine „leichtgewichtige“ Modellierungssprache und ein eigenes Modellierungstool (<https://structurizr.com/>)
- Auf Klassenebene kann auch ein UML Klassendiagramm eingesetzt werden
- Bietet bei Bedarf auch Diagramme für dynamische Strukturen an

# Context, Container, Component and Classes (C4)



# **Context, Container, Component and Classes (C4)**

---



Zum Nachlesen

## **Context**

- Umsysteme und Akteure, die mit dem System interagieren

## **Container**

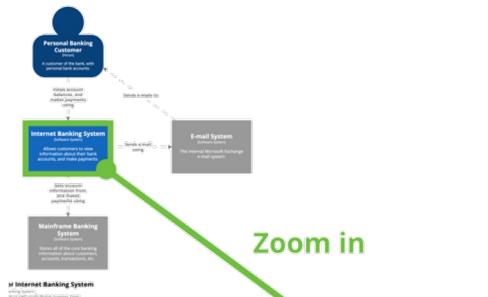
- Ausführungsumgebung für Komponenten
- Ein Container ist nur durch eine Schnittstelle ansprechbar
- war, jar, node module, http-server etc.

## **Component**

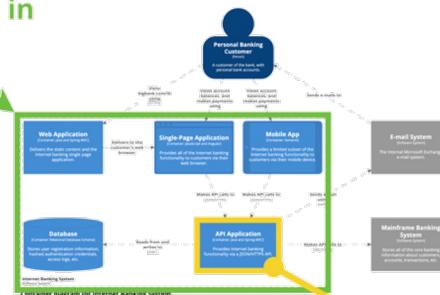
- Logische Gruppierung von eins bis n Klassen

## **Classes**

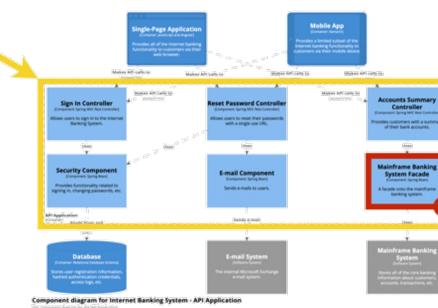
- Kleinster Baustein eines Softwaresystems



**Zoom in**



Zoom in



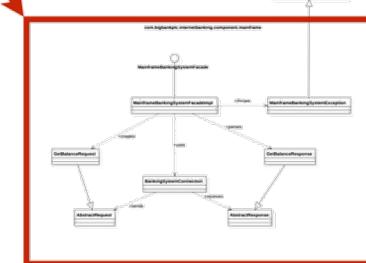
**Zoom in**

# Context, Container, Component and Classes (C4)

<https://leanpub.com/b/software-architecture>

und

<https://c4model.com/>



## Level 1

# Context

## Level 2

# Containers

## Level 3

# Components

# Level 4

# Code



# Fundamental Modeling Concepts (FMC)

---

## **Compositional Structures (Block diagram)**

- Statische Strukturen in Form von Komponenten und deren Beziehungen
- Ebene Bausteine (iSAQB) oder Component (Brown)

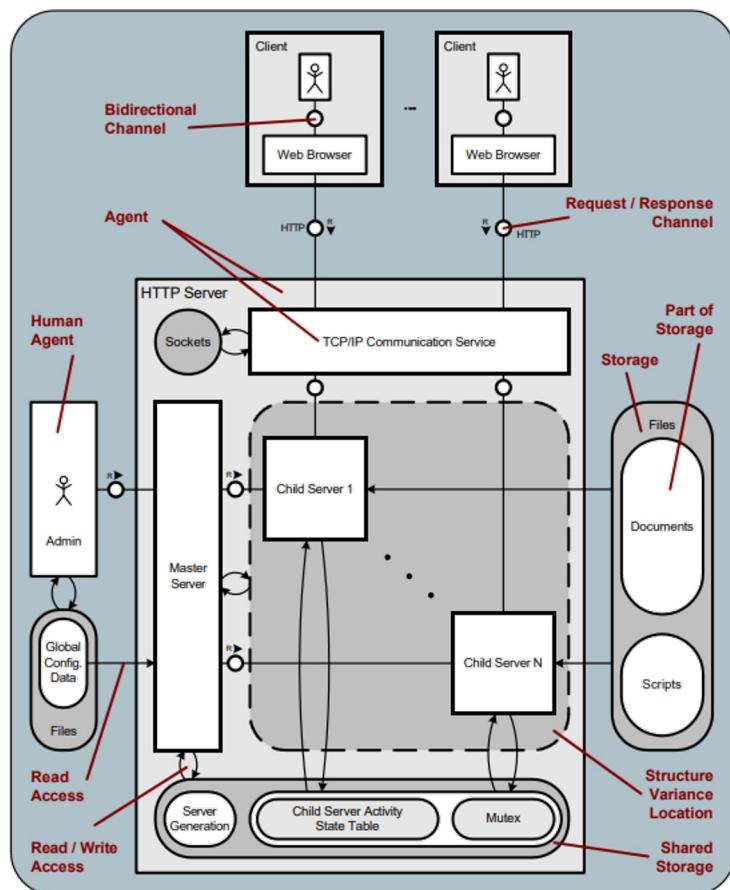
## **Dynamic Structures (Petri nets)**

- Dynamische Systemabläufe zur Laufzeit

## **Value Range Structures (Entity relationship diagram)**

- Wertebereichsstrukturen und mathematische Strukturen

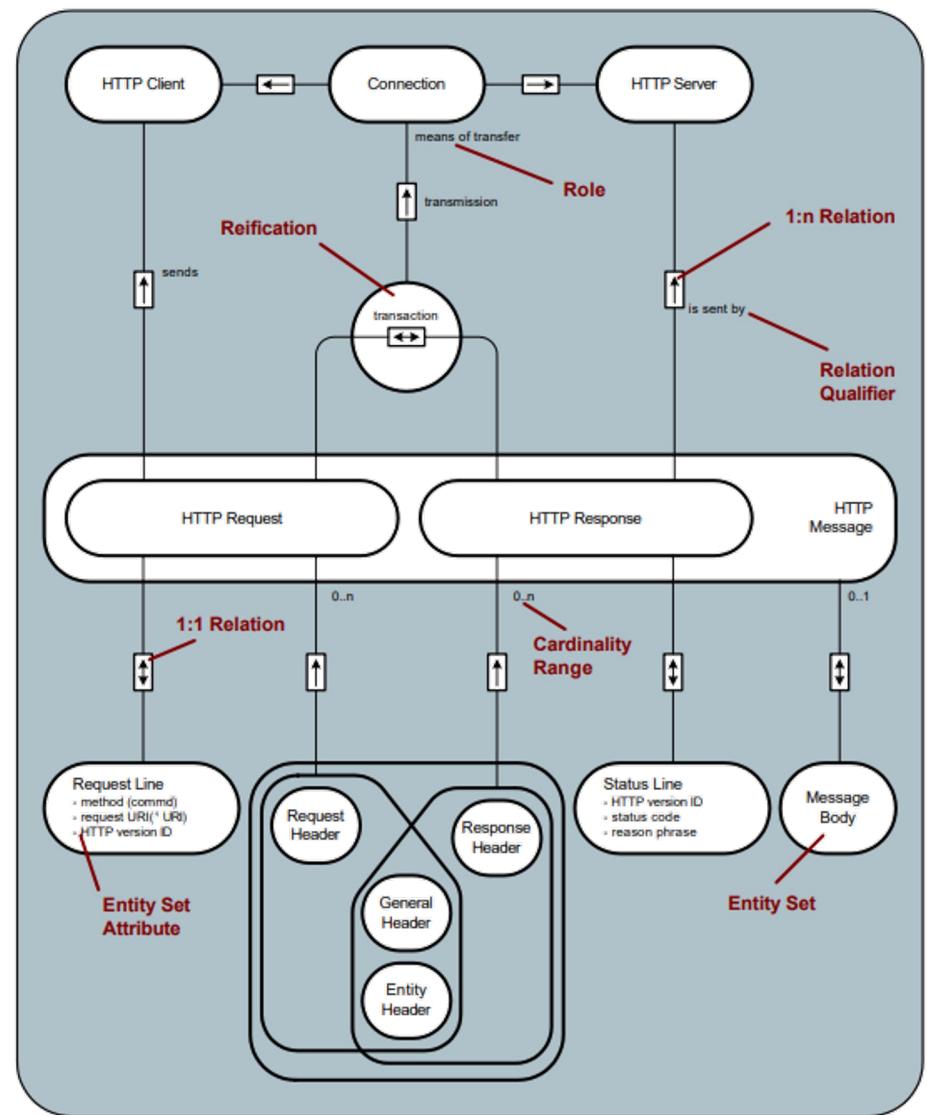
# FMC Diagramme



Block diagram

[http://www.fmc-modeling.org/notation\\_reference](http://www.fmc-modeling.org/notation_reference)

141



Entity relationship diagram

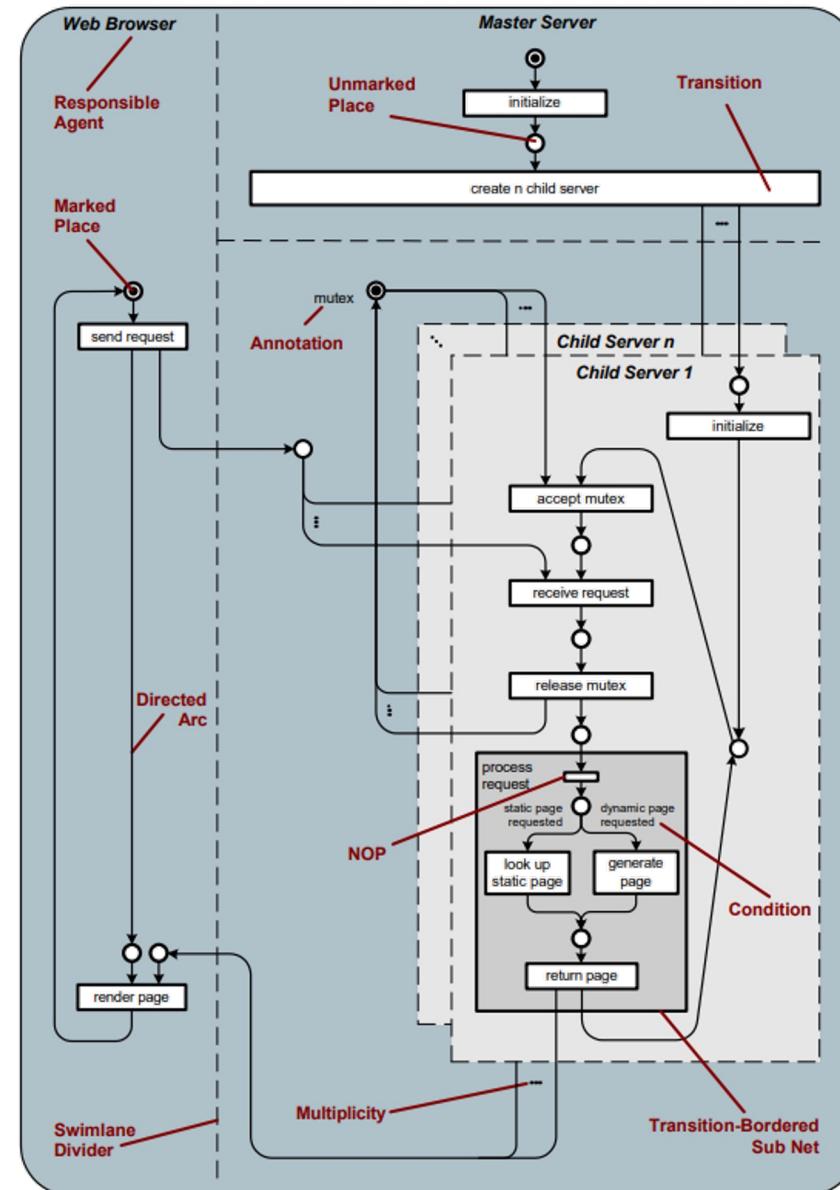
NOVATEC

# FMC Diagramme

142

[http://www.fmc-modeling.org/notation\\_reference](http://www.fmc-modeling.org/notation_reference)

Petri net



NOVATEC

# **Werkzeuge zur Erstellung und Pflege von Dokumentationen**

---

- Word / Power Point / Visio / PDF / HTML Dokumente
- Wiki / Confluence
- Documentation as Code (ASCIIDOC, Markdown)
- Diagram as Code (PlantUML)
- IDE und Modellierungswerkzeuge

# Modellierungswerkzeuge



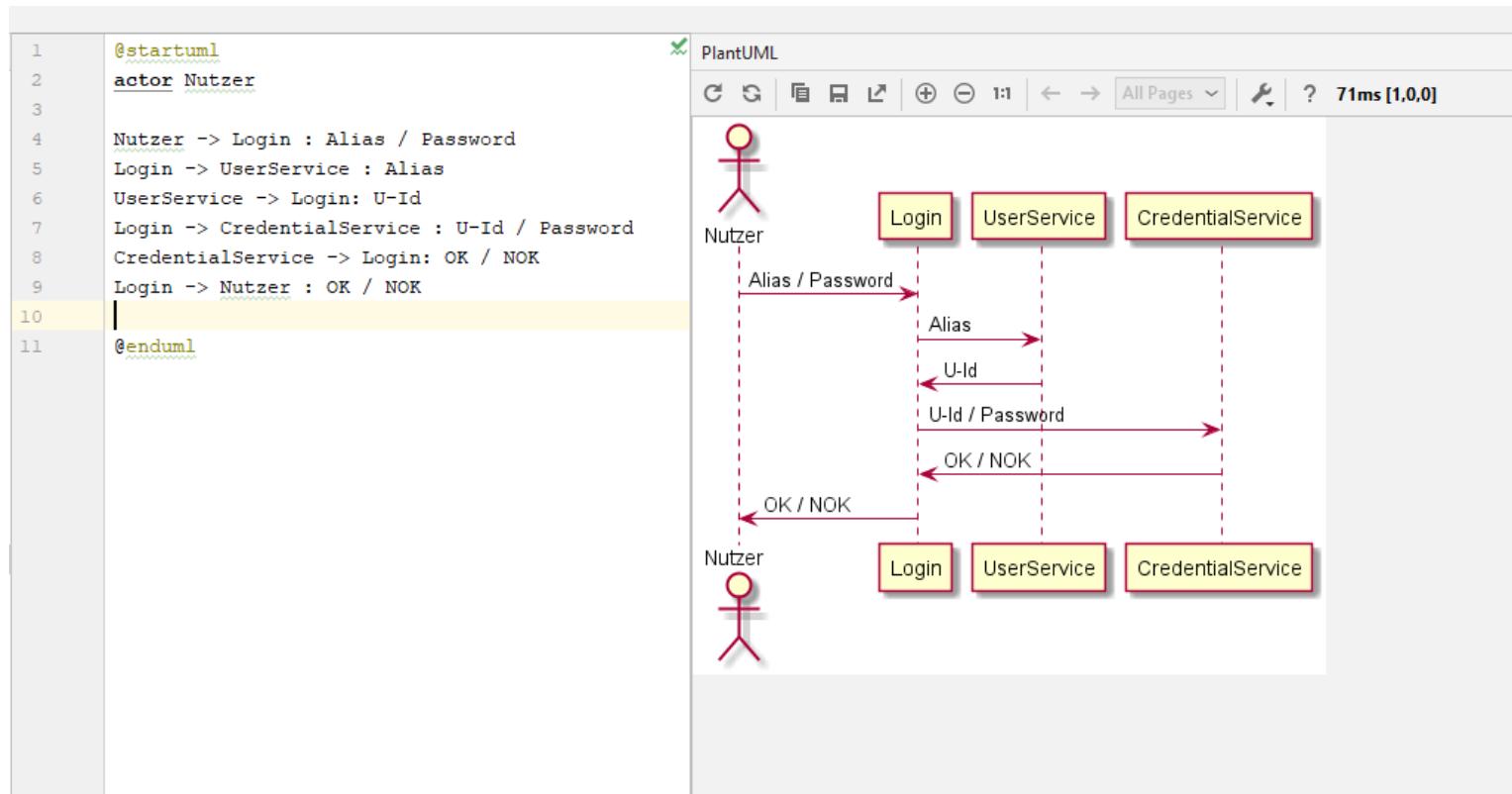
Zum Nachlesen

Modellierungswerkzeuge dienen dazu fachliche und technische Modelle von Software darzustellen. Dabei werden meist grafische Abstraktionen der Realität erstellt.

| Anforderungen  | Herausforderungen  |
|--|--|
| <ul style="list-style-type: none"><li>Unterstützung diverser Modellierungs-methoden (UML, ER, BPMN, freie Notation)</li><li>Unterstützung von Sichten / UML-Modellen</li><li>Unterstützung statischer und dynamischer Modelle</li><li>Validierung / Plausibilisierung von Modellen</li><li>Abhängigkeiten zwischen Modellen</li><li>Integration der Versionsverwaltung</li><li>Mehrbenutzerfähigkeit</li><li>Reverse Engineering von Quellcode</li></ul> | <ul style="list-style-type: none"><li>Grafische Darstellungen (UML-Modelle, Mindmaps, freie Diagramme) können meist nicht parallel bearbeiten werden. Zusammenführen von Versionen meist aufwändig und fehleranfällig.</li><li>Technischen Notationen für viele Stakeholder ungeeignet</li><li>Meist keine IDE-Integration. Deshalb fehlende Akzeptanz bei Entwicklern</li></ul> |

sparx Enterprise Architect, draw.io, Magic Draw, RSA, *Tipp: PlantUML*

# PlantUML – Diagram as Code



# Diskussion

---

- Wie haben Sie Ihre Architekturen in der Vergangenheit dokumentiert und welche Werkzeuge kamen zum Einsatz?
- Würden Sie jetzt anders vorgehen?

# Quellen

---

[Gharbi 2018] Gharbi, Mahbouba [u.a]: Basiswissen für Softwarearchitekten. Aus- und Weiterbildung nach iSAQB-Standard zum Certified Professional for Software Architecture – Foundation Level. 3. überarb. Aufl. Heidelberg, 2018

[Starke 2015] Starke, Gernot: Effektive Software-Architekturen: Ein praktischer Leitfaden. 7. überarb. Aufl. München, 2015

[Starke und Hruschka 2015] Starke Gernot, Peter Hruschka: arc42. Pragmatische Hilfe für Softwarearchitekten. München

[Toth 2015] Stefan Toth: Vorgehensmuster für Software-Architektur: Kombinierbare Praktiken in Zeiten von Agile und Lean. 2. Aufl., Hanser Verlag, München, 2015.

[Zörner 2015] Zörner, Stefan: Softwarearchitekturen dokumentieren und kommunizieren. Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten. 2. Aufl. München, 2015

[Rupp und Queins 2012] Rupp Chris, Stefan Queins: UML glasklar: Praxiswissen für die UML-Modellierung. 4. Aufl., München, 2012

Icons made by <https://www.freepik.com/> & <https://www.flaticon.com/>

<https://adr.github.io/>



# iSAQB Foundation Level

Beschreibung und Kommunikation von  
Softwarearchitekturen

Beispielfragen



# Beispielfragen

---

## Frage 1

A-Frage: Wählen Sie eine Option aus 1 Punkt

Welche der vier Sichten sollte eine gute Architekturbeschreibung umfassen?

- (a) Kontextsicht, Fachliche Sicht, Technische Sicht, Infrastruktursicht
- (b) Kontextsicht, Bausteinsicht, Laufzeitsicht, Verteilungssicht
- (c) Kontextsicht, Bausteinsicht, Schnittstellensicht, Verteilungssicht
- (d) Bausteinsicht, Schnittstellensicht, Verteilungssicht, Infrastruktursicht
- (e) Kontextsicht, Technische Sicht, Schnittstellensicht, Verteilungssicht

# Beispielfragen

---

## Frage 1

A-Frage: Wählen Sie eine Option aus

1 Punkt

Welche der vier Sichten sollte eine gute Architekturbeschreibung umfassen?

- (a) Kontextsicht, Fachliche Sicht, Technische Sicht, Infrastruktursicht
- (b) Kontextsicht, Bausteinsicht, Laufzeitsicht, Verteilungssicht
- (c) Kontextsicht, Bausteinsicht, Schnittstellensicht, Verteilungssicht
- (d) Bausteinsicht, Schnittstellensicht, Verteilungssicht, Infrastruktursicht
- (e) Kontextsicht, Technische Sicht, Schnittstellensicht, Verteilungssicht

# Beispielfragen

---

## Frage 2

P-Frage: Wählen Sie die **drei besten** Aspekte aus 3 Punkt

Welche Informationen sind Teil der Schablone für die Blackboxsicht eines Bausteins?

- (a) Kurzbeschreibung
- (b) Diagramm
- (c) Elementkatalog
- (d) Schnittstellen
- (e) Variabilitäten

# Beispielfragen

---

## Frage 2

P-Frage: Wählen Sie die **drei besten** Aspekte aus 3 Punkt

Welche Informationen sind Teil der Schablone für die Blackboxsicht eines Bausteins?

- (a) Kurzbeschreibung
- (b) Diagramm
- (c) Elementkatalog
- (d) Schnittstellen
- (e) Variabilitäten

# Beispielfragen

---

## Frage 3

K-Frage: Wählen Sie für jede Zeile „Richtig“ oder „Falsch“ aus 2 Punkte

Welchen Aussagen zur Modellierung und Dokumentation stimmen Sie zu?

Richtig      Falsch

- |                          |                          |   |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | <input type="checkbox"/> | (a) Als Architekt:in bin ich alleine für die Modellierung und Dokumentation zuständig               |
| <input type="checkbox"/> | <input type="checkbox"/> | (b) Dokumentation und Modellierung sind das gleiche, werden nur zu einem anderen Zeitpunkt erledigt |
| <input type="checkbox"/> | <input type="checkbox"/> | (c) Ein Diagramm benötigt in der Regel keinen Text  |
| <input type="checkbox"/> | <input type="checkbox"/> | (d) Im arc42 Template muss nicht alles ausgefüllt werden  |

# Beispielfragen

---

## Frage 3

K-Frage: Wählen Sie für jede Zeile „Richtig“ oder „Falsch“ aus 2 Punkte

Welchen Aussagen zur Modellierung und Dokumentation stimmen Sie zu?

Richtig      Falsch

- |                                     |                                     |   |
|-------------------------------------|-------------------------------------|---|
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | (a) Als Architekt:in bin ich alleine für die Modellierung und Dokumentation zuständig               |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | (b) Dokumentation und Modellierung sind das gleiche, werden nur zu einem anderen Zeitpunkt erledigt |
| <input type="checkbox"/>            | <input checked="" type="checkbox"/> | (c) Ein Diagramm benötigt in der Regel keinen Text  |
| <input checked="" type="checkbox"/> | <input type="checkbox"/>            | (d) Im arc42 Template muss nicht alles ausgefüllt werden  |



# iSAQB Foundation Level

Beschreibung und Kommunikation von  
Softwarearchitekturen

Selbstlernkontrolle



# Beispielfrage

---

Wer ist kein Stakeholder der Architekturdokumentation:

- Entwickler:innen
- Auftraggeber:innen
- Endbenutzer:innen
- Betrieb

# Beispielfrage

---

Wer ist kein Stakeholder der Architekturdokumentation:

- Entwickler:innen
- Auftraggeber:innen
- **Endbenutzer:innen**
- Betrieb

# Beispielfrage

---

Welches ist kein gültiges UML-Diagramm:

- Paketdiagramm
- Verteilungsdiagramm
- Prozessdiagramm
- Anwendungsfalldiagramm

# Beispielfrage

---

Welches ist kein gültiges UML-Diagramm:

- Paketdiagramm
- Verteilungsdiagramm
- **Prozessdiagramm**
- Anwendungsfalldiagramm

# Beispielfrage

---

Wer ist kein Stakeholder der Kontextsicht:

- Unternehmensarchitekt:innen
- Auftraggeber:innen
- UI-Expert:innen
- Betrieb
- Keine der genannten Rollen

# Beispielfrage

---

Wer ist kein Stakeholder der Kontextsicht:

- Unternehmensarchitekt:innen
- Auftraggeber:innen
- **UI-Expert:innen**
- Betrieb
- Keine der genannten Rollen

# Beispielfrage

---

Wer ist kein Stakeholder der Bausteinsicht:

- Unternehmensarchitekt:innen
- Projektleiter:innen
- Entwickler:innen
- Keine der genannten Rollen

# Beispielfrage

---

Wer ist kein Stakeholder der Bausteinsicht:

- **Unternehmensarchitekt:innen**
- Projektleiter:innen
- Entwickler:innen
- Keine der genannten Rollen

# Beispielfrage

---

Welche Informationen sind nicht Teil der Bausteinsicht:

- Interner Aufbau der Bausteine
- Schnittstellen
- Entwurfsentscheidungen
- Platzierung der Bausteine innerhalb der Infrastruktur

# Beispielfrage

---

Welche Informationen sind nicht Teil der Bausteinsicht:

- Interner Aufbau der Bausteine
- Schnittstellen
- Entwurfsentscheidungen
- **Platzierung der Bausteine innerhalb der Infrastruktur**

# Beispielfrage

---

Wer ist kein Stakeholder der Laufzeitsicht:

- Tester:innen
- Projektleiter:innen
- Entwickler:innen
- Keine der genannten Rollen

# Beispielfrage

---

Wer ist kein Stakeholder der Laufzeitsicht:

- Tester:innen
- **Projektleiter:innen**
- Entwickler:innen
- Keine der genannten Rollen

# Beispielfrage

---

Welche der folgenden Elemente sollte eine Schablone zur Schnittstellenbeschreibung enthalten:

- Interaktionen
- Qualitätsmerkmale
- Entwurfsentscheidungen
- Fehlerbehandlung
- Datentypen

# Beispielfrage

---

Welche der folgenden Elemente sollte eine Schablone zur Schnittstellenbeschreibung enthalten:

- **Interaktionen**
- **Qualitätsmerkmale**
- **Entwurfsentscheidungen**
- **Fehlerbehandlung**
- **Datentypen**

# Beispielfrage

---

Welche Aspekte umfassen den Bereich Security einer Schnittstelle?

# Beispielfrage

---

Welche Aspekte umfassen den Bereich Security einer Schnittstelle?

z.B.

- **Authentifizierung**  
Username und Passwort, OAuth2, OpenID, SAML
- **Autorisierung**  
Administratoren, Rollen und Rechte, Role based access control pattern, Attribute based access control,

# Beispielfrage

---

Wer ist kein Stakeholder der Verteilungssicht:

- Tester:innen
- Unternehmensarchitekt:innen
- Betrieb
- Entwickler:innen
- Keine der genannten Rollen

# Beispielfrage

---

Wer ist kein Stakeholder der Verteilungssicht:

- **Tester:innen**
- Unternehmensarchitekt:innen
- Betrieb
- Entwickler:innen
- Keine der genannten Rollen

# **Beispielfrage**

---

Welche Wechselwirkungen zwischen Systemkontext und Verteilungssicht existieren?

# Beispielfrage

---

Welche Wechselwirkungen zwischen Systemkontext und Verteilungssicht existieren?

Systemkontext ist Basis für alle Sichten

Zusammenhang: Wo laufe ich? Auf welcher Umgebung sind Nachbarsysteme (z.B. Relevant wegen Verfügbarkeit)? Wie sieht der technische Kommunikationskanal aus?

# **Beispielfrage**

---

Nennen Sie mindestens drei querschnittliche Konzepte, die bei einer Architektur zu berücksichtigen sind und die technischen Fragestellungen hierzu.

# Beispielfrage

---

Nennen Sie mindestens drei querschnittliche Konzepte, die bei einer Architektur zu berücksichtigen sind und die technischen Fragestellungen hierzu.

1. **Logging** => Wohin Loggen wir unsere Informationen und welche Logging-Library wollen wir verwenden
2. **Persistenz** => CAP vs. ACID / Verfügbarkeit vs. Konsistenz
3. **Security** => Role based access vs. attribute based access

# Beispielfrage

---

Welches sind keine relevanten Fragestellungen im Hinblick auf eine Architekturentscheidung:

- Auswahl des UI-Frameworks
- Auswahl der Methodik für den Softwareentwicklungsprozess (agil versus iterativ / inkrementell)
- Auswahl der Programmiersprache
- Auswahl der Betriebsform (on-premise versus Cloud-basiert)

# Beispielfrage

---

Welches sind keine relevanten Fragestellungen im Hinblick auf eine Architekturentscheidung:

- Auswahl des UI-Frameworks
- **Auswahl der Methodik für den Softwareentwicklungsprozess (agil versus iterativ / inkrementell)**
- Auswahl der Programmiersprache
- Auswahl der Betriebsform (on-premise versus Cloud-basiert)



## **Novatec Consulting GmbH**

Dieselstraße 18/1

Berta-Benz-Platz 1

T. +49 711 22040-700

[info@novatec-gmbh.de](mailto:info@novatec-gmbh.de)

[www.novatec-gmbh.de](http://www.novatec-gmbh.de)