Architekturen betrieblicher Anwendungssysteme

Einführung in die Softwarearchitektur

Lehrstuhl für Wirtschaftsinformatik
Prozesse und Systeme
Universität Potsdam

Chair of Business Informatics
Processes and Systems
University of Potsdam

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

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

Lernziele

- Was unterscheidet eine Softwarearchitektur von einer Anwendungs- oder Systemarchitektur?
- Welche Aufgaben und Verantwortlichkeiten hat der Softwarearchitekt im Entwicklungsprozess?
- Warum sind klassische Architekturen (z. B. Monolith, 3-Schichten, SOA) heute oft unzureichend?
- Welche Rolle spielen Qualitätsattribute (nach ISO 25010, Bass et al.) für Architekturentscheidungen?
- Wie lässt sich die Ausrichtung der Architektur an Geschäftsprozessen (Business Process Alignment) bewerten und gestalten?

Was bedeutet Architektur?

Die Vitruv-Analogie

Kernprinzip (Vitruv, *De Architectura*, Buch I, Kap. 3)

„Architectura … constat ex … firmitate, utilitate, venustate."

(„Architektur ist die Kombination von Festigkeit, Nützlichkeit, Schönheit.") Nach Vitruvius Pollio (ca. 15 v. Chr.)

Firmitas: Das Gebäude ist stabil.

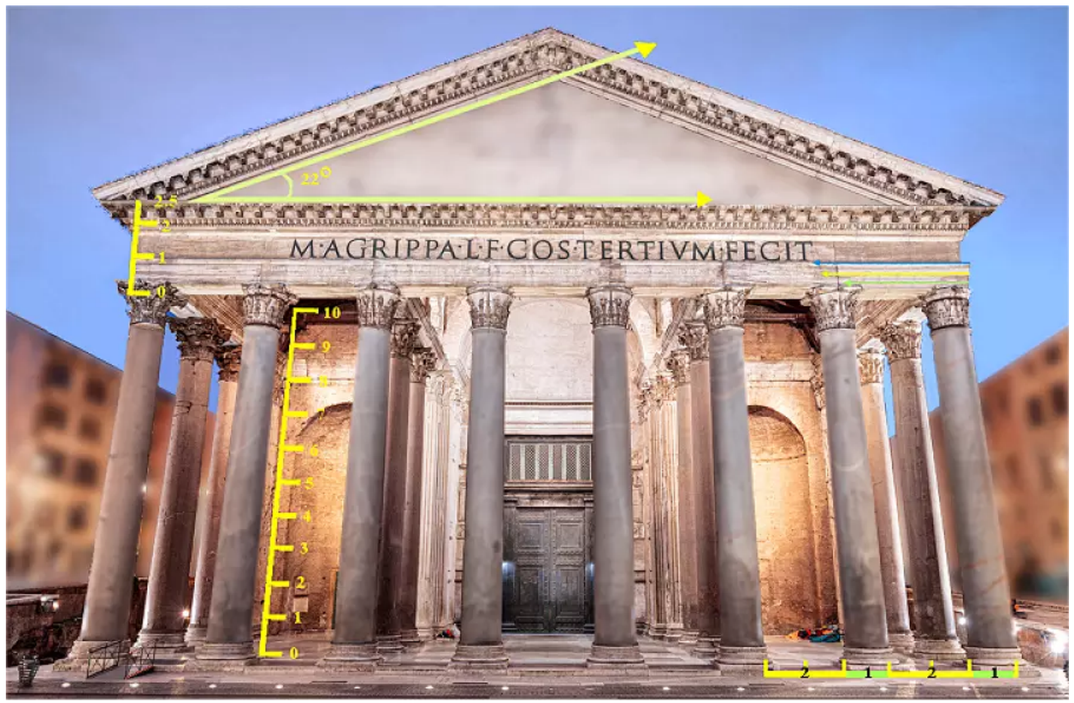
-> Das Softwaresystem ist langlebig und „stabil"/resilient gegenüber Änderungen

Utilitas: Das Gebäude erfüllt seine Funktion.

-> Das Softwaresystem erfüllt seine Anforderungen

Venustas: Das Gebäude ist ästhetisch gestaltet.

-> Das Softwaresystem weist klare, logische Strukturen auf.



Quelle: Vitruv, De architectura libri decem, ca. 15 v. Chr., Abbildung https://atouchofrome.com/roman-temple-architecture-explained-simply.html

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

Motivation und Einordnung

Warum Softwarearchitektur wichtig ist

- **Architektur bestimmt Qualitätseigenschaften** eines Systems (z. B. Performance, Sicherheit, Modifizierbarkeit) – nicht nur Funktionalität. *(Bass et al., 2021)*
- Qualität entsteht durch Architekturentscheidungen, nicht durch einzelne Codezeilen. *(Bass et al., 2021)*
- Architekturentscheidungen wirken langfristig auf Wartbarkeit, Integration und Kosten. *(Bass et al., 2021)*

Einordnung in betriebliche Anwendungssysteme

- **Softwarearchitektur:** Struktur eines einzelnen Systems (Komponenten, Schnittstellen, Beziehungen). *(Bass et al., 2021)*
- **(IT-)Unternehmensarchitektur:** Zusammenspiel mehrerer Systeme, Prozesse und Daten im Unternehmen (z. B. ERP, CRM, SCM).
- Abgrenzung: Softwarearchitektur ist die Bauweise eines Systems, eingebettet in eine übergeordnete Systemlandschaft. *(Keller, 2017)*

Quelle: Bass et al. 2021, Keller (2017)

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

Rolle des Softwarearchitekten

Zentrale Aufgabe

Der Softwarearchitekt ist die Brücke zwischen Anforderungen und Technik. Er gestaltet die **Struktur** eines Softwaresystems so, dass funktionale und nicht-funktionale Anforderungen erfüllt werden.

Aufgabenbereiche

- **Anforderungsanalyse und Übersetzung:** Übersetzt Geschäftsziele, Prozesse und QAs in technische Strukturen.
- **Strukturierung und Entwurf:** Definiert Komponenten, Schnittstellen und Interaktionen im System.
- **Qualitätssicherung:** Stellt sicher, dass Qualitätsattribute (z. B. Performance, Wartbarkeit, Sicherheit) erreicht werden.
- **Kommunikation und Moderation:** Vermittelt zwischen Entwicklern, Management und Fachexperten.
- **Entscheidung und Verantwortung:** Trifft und dokumentiert Architekturentscheidungen

Quelle: nach Bass, Clements & Kazman 2021; vgl. Broy 2010

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

Grundbegriff Softwarearchitektur

Definition (nach Bass et al. 2021, IEEE/SEI):

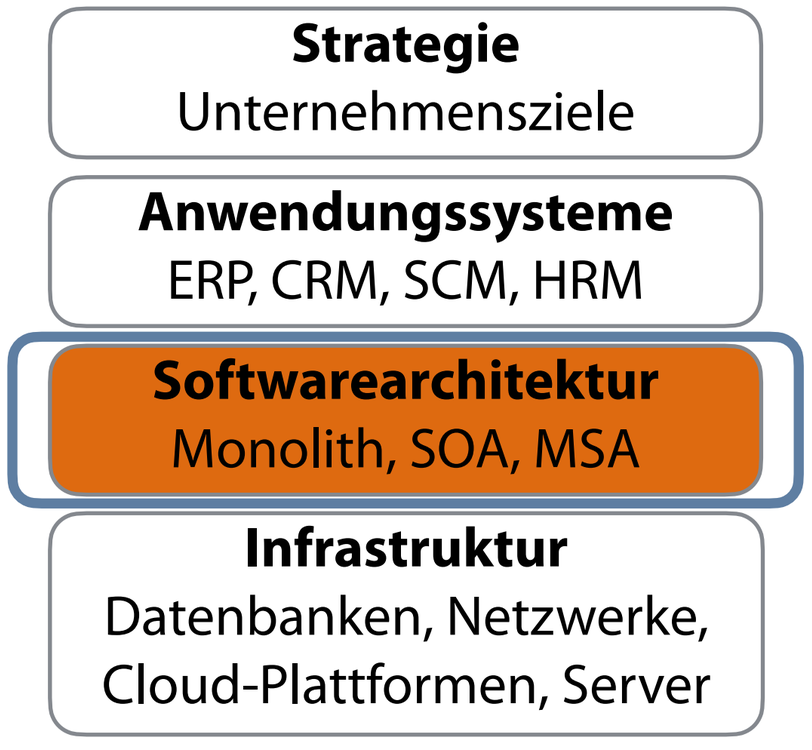
- „Softwarearchitektur umfasst die grundlegenden Strukturen eines Softwaresystems, die in seinen Komponenten, deren Beziehungen und den Prinzipien zur Gestaltung und Evolution bestehen."

Kernkonzepte:

- Komponenten: Bausteine der Software (z. B. Module, Services).
- Schnittstellen: Definieren, wie Komponenten interagieren.
- Beziehungen: Daten-/Kontrollflüsse zwischen Komponenten.

Qualitätsaspekte:

- Kopplung: lose -> Systeme bleiben änderungsfreundlich.
- Kohäsion: hohe innere Geschlossenheit von Komponenten.
- Wartbarkeit: einfache Anpassung an neue Anforderungen.
- Weitere Attribute: Performance, Sicherheit, Zuverlässigkeit.



Quelle: Bass et al. 2021

Architektur als Kommunikationsplattform

Kernidee

- Architektur dient nicht nur dem Entwurf von Systemen, sondern auch als **gemeinsame Sprache zwischen allen Beteiligten** – Entwicklern, Architekten, Management und Fachbereichen.

Warum Kommunikation zentral ist:

- Architektur ist ein gemeinsames Modell, das komplexe Systeme verständlich macht.
- Sie schafft ein gemeinsames Vokabular für Anforderungen, Entscheidungen und Kompromisse.
- Architekturentscheidungen müssen nachvollziehbar und kommunizierbar sein.
- Gute Architektur ist ein Kommunikationsartefakt: Diagramme, Dokumente, Modelle.
- Sie ermöglicht die Abstimmung zwischen Business-Zielen und technischer Umsetzung.

Architecture is the primary means of communication among stakeholders."
- Bass, Clements & Kazman (2021)

Quelle: Bass et al. 2021

QuizApp

Einwahldaten

- URL: https://quiz.lswi.de/login
- Lecture Code: aba19

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

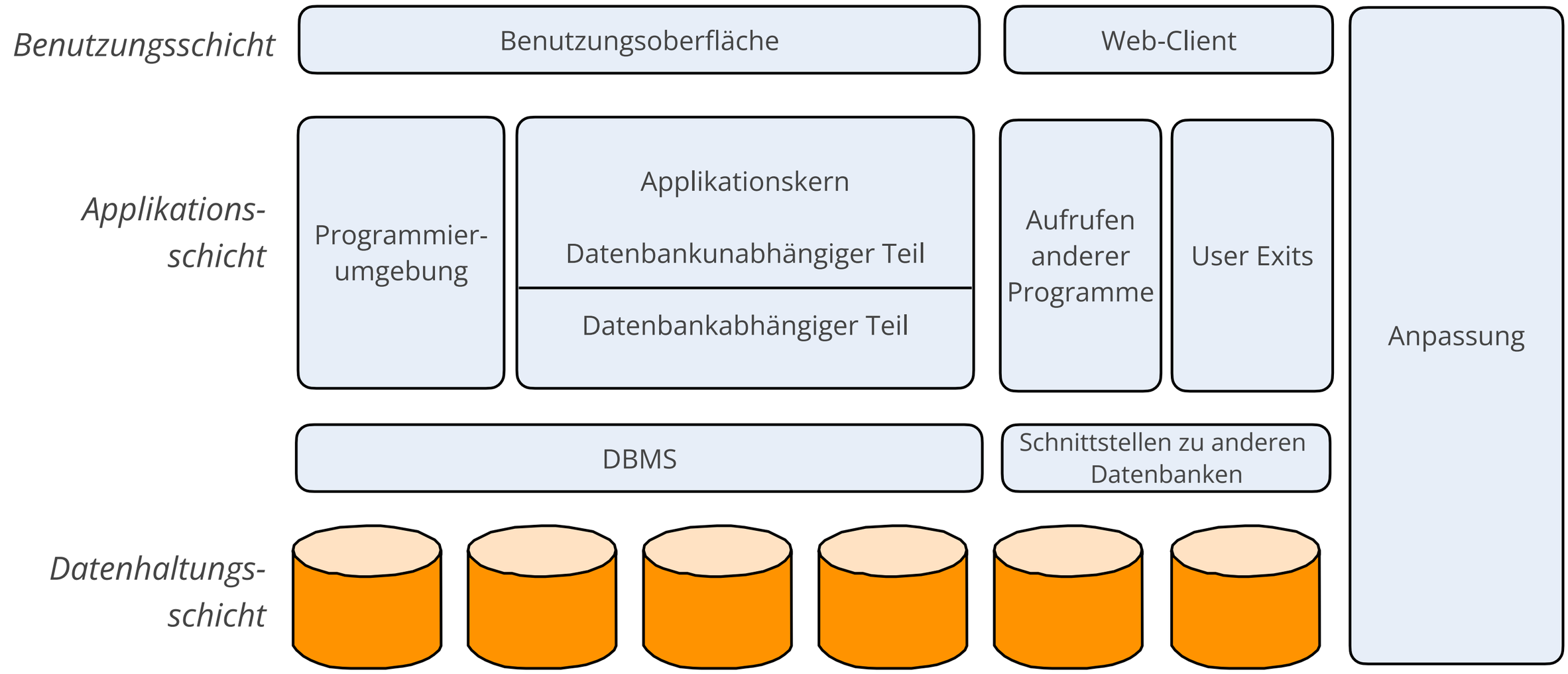
Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

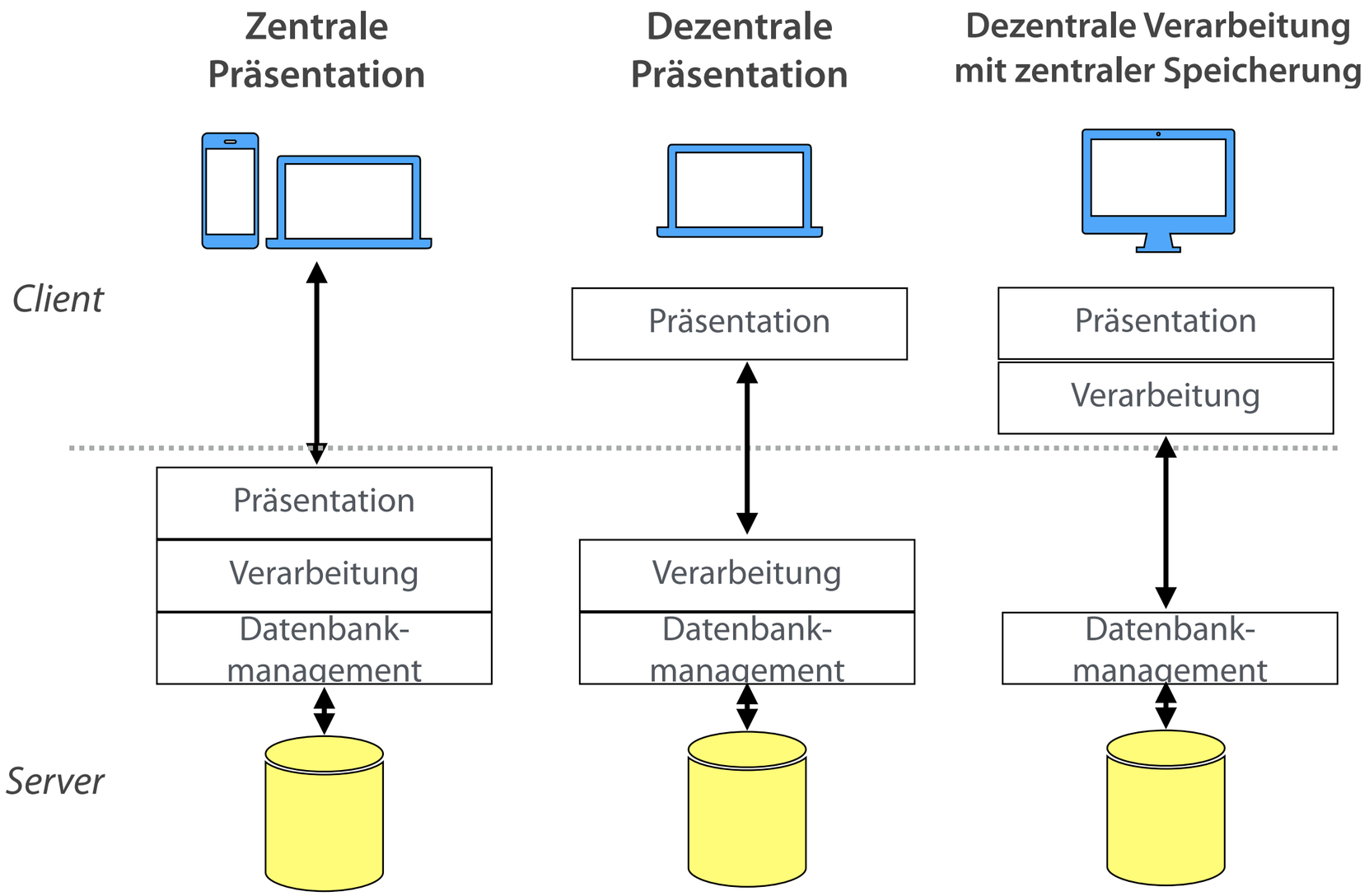
Klassische Architektur



Quelle: Gronau 2021

Verteilung der Systemfunktionen:

Client-Server Computing

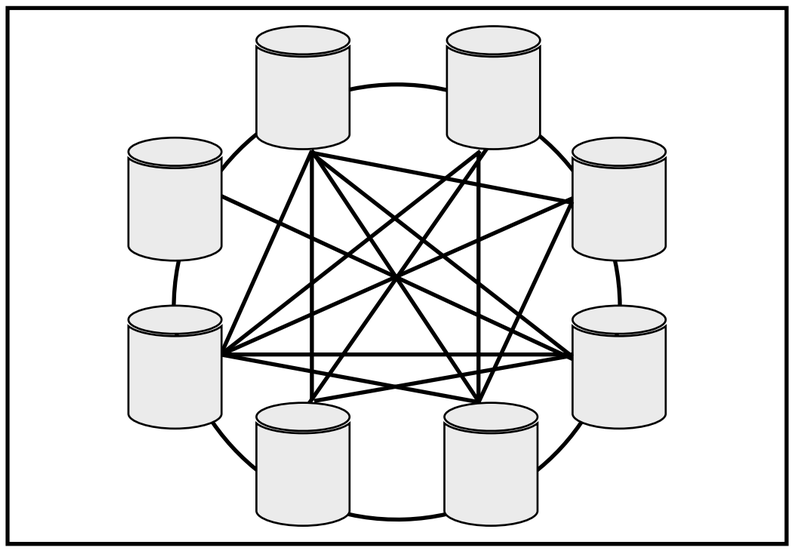


Client-Server-Computing erlaubt es, die Systemfunktionen auf verschiedene Weise auf mehrere Computer zu verteilen.

Quelle: Gronau 2021, S. 29

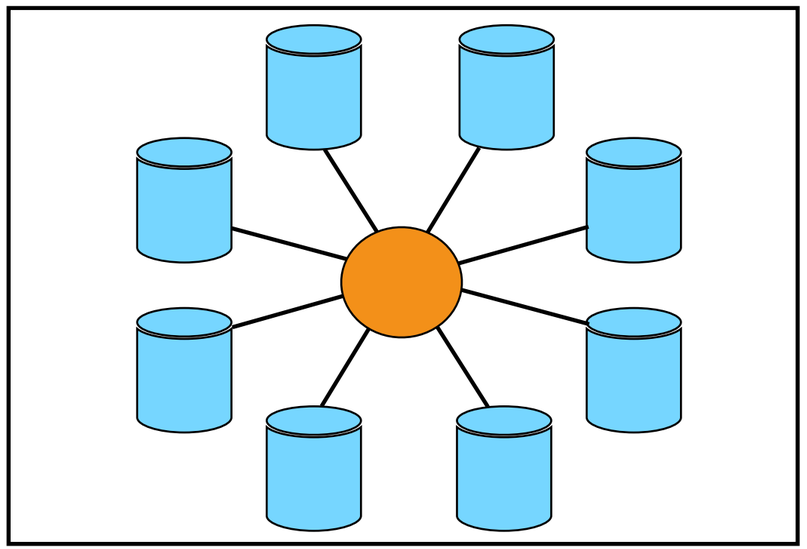
Prinzipien von Integrationsarchitekturen

Wie Systeme miteinander verbunden werden: Drei Integrationsarchitekturen im Vergleich



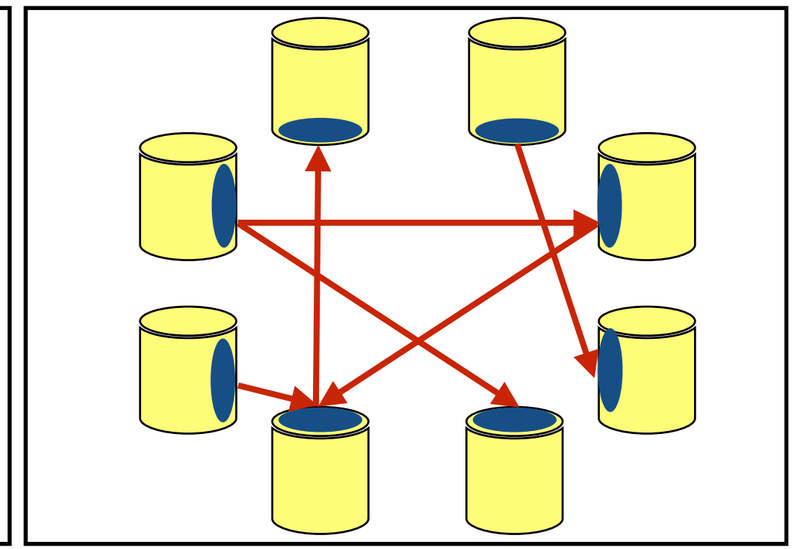
Punkt zu Punkt

- Direkte Verbindung zwischen jedem Systempaar
- Individuell entwickelte Schnittstellen
- Feste Kopplung, schwer wartbar
- Skalierungsproblem bei vielen Systemen: Anzahl der Verbindungen = $n(n-1)/2$



Hub and Spoke

- Zentrale Integrationsplattform (Hub) als Vermittler
- Systeme (Spokes) kommunizieren nur mit dem Hub
- Weniger Schnittstellen, aber:
- Single Point of Failure
- Transformation & Routing im Hub nötig



Service-orientierte Architektur (SOA)

- Lose Kopplung durch standardisierte Dienste (z. B. Web Services, REST APIs)
- Wiederverwendbare Services für verschiedene Prozesse
- Dezentrale Kommunikation zwischen Services
- Höherer Abstimmungsbedarf, aber große Flexibilität

Quelle: Gronau 2021, S. 35

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

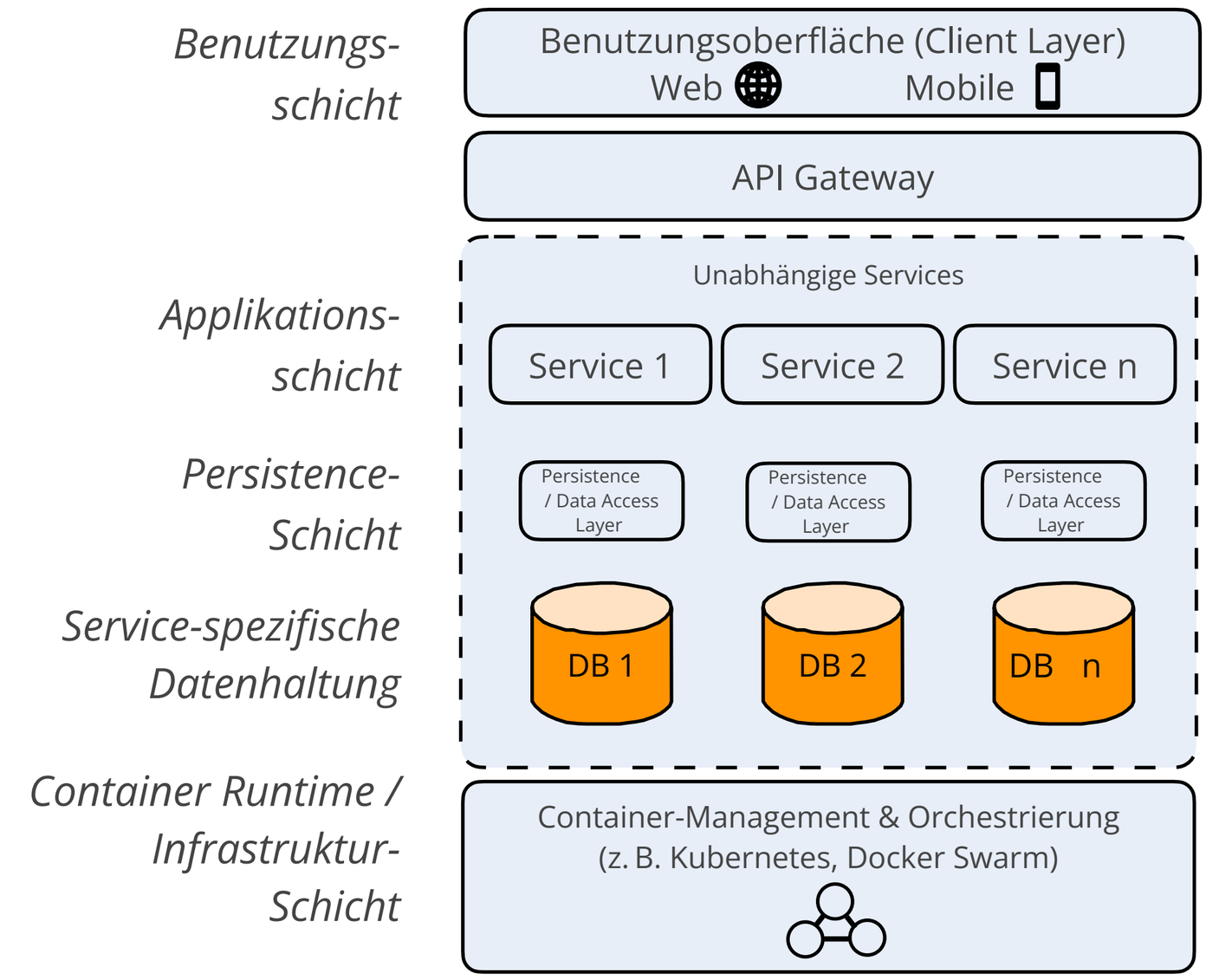
Microservices als moderner Architekturstil

Definition (nach Newman 2021):

- Kleine, autonome Services, die unabhängig deployed werden können.
- Jeder Service hat eigene Logik und eigene Datenhaltung.
- Kommunikation über leichte Schnittstellen (REST, gRPC, Messaging).

Prinzipien

- Lose Kopplung, starke Kohäsion.
- Um Geschäftsdomänen modelliert (*Domain-Driven Design*).
- Automatisiertes Deployment (DevOps, CI/CD).



Quelle: Newman (2021)

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

25010 – Softwarequalität

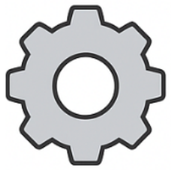
- Architekturentscheidungen zielen nicht nur auf Funktionalität, sondern auf Qualitätseigenschaften.
- ISO/IEC 25010 definiert acht zentrale Kategorien von Softwarequalität.
- Diese Kategorien bilden den Rahmen für die Bewertung von Softwarearchitekturen.
- *Beispiel:* Caching erhöht Performance, kann aber Wartbarkeit verringern

  -> Qualitätseigenschaften stehen oft in Zielkonflikt.

Quelle: Bass et al. 2021; ISO/IEC 25010

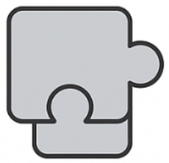
Vom Attribut zur Metrik
Wie Qualität messbar wird

- Qualitätsattribute beschreiben **was** wichtig ist (z. B. Performance, Wartbarkeit, Sicherheit).
- Metriken zeigen **wie gut** diese Attribute erfüllt werden.



Performance

- Response Time
- Throughput
- Resource Utili.
- Latency



Maintainability

- Modularity
- Change Effort
- Testability
- Code Complexity

Security

- Vulnerability Density
- Mean Time to Detect
- Mean Time to Recover
- Access Control Effect.

Reliability

- Availability
- Mean Time Between Failures
- Failure Rate

Qualität wird beobachtbar und bewertbar – Grundlage für Architekturentscheidungen.

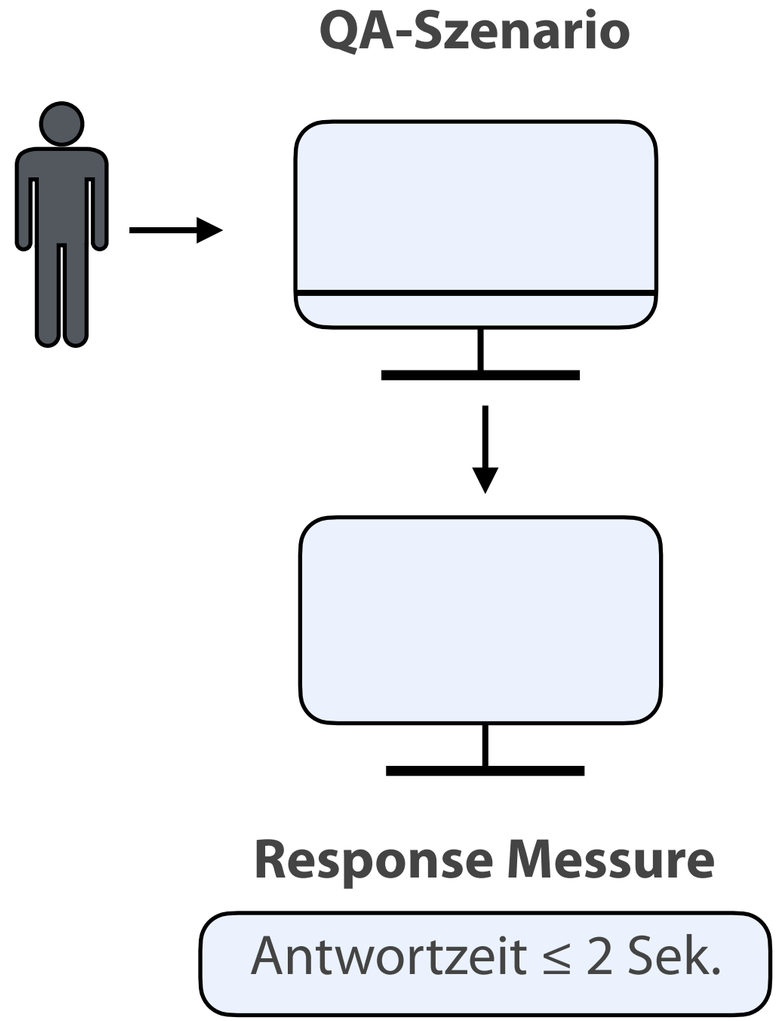
Quelle: Bass et al. 2021; ISO/IEC 25010

Präzise Qualitätsanforderungen

Beispiel eines Quality Attribute Scenarios (Performance)

Quality Attribute Scenarios (Bass et al. 2021)

- Beschreiben Qualitätsanforderungen präzise:
  - **Stimulus**: Ein Nutzer sendet eine Suchanfrage über die Weboberfläche.
  - **Environment**: Das System befindet sich unter hoher Last (mehrere gleichzeitige Benutzeranfragen).
  - **Artifact**: Webserver und Datenbank, die für die Verarbeitung der Anfrage zuständig sind.
  - **Response:** Das System verarbeitet die Anfrage und liefert eine Ergebnisliste zurück.
  - **Response Measure**: Antwortzeit ≤ **2 Sekunden** für 95 % aller Anfragen.



Quelle: Bass et al. 2021; ISO/IEC 25010

Präzise Qualitätsanforderungen

Beispiel: Maintainability-Metrik (Change Effort)

- **Stimulus:** Eine fachliche Regel im Bestellprozess wird geändert (z. B. neue Rabattlogik).
- **Environment:** Laufender Betrieb, reguläre Release-Zyklen.
- **Artifact:** Betroffene Services im Order- und Pricing-Bereich.
- **Response:** Anpassung des Codes und Tests.
- **Response Measure:** Anzahl geänderter Module / Services.

Berechnung Beispiel

- Zwei Services müssen angepasst werden
- Gesamtsystem: 10 Services

$$\text{Change Effort} = \frac{BetroffeneServices}{GesamtzahlServices}$$

$$\text{Change Effort} = \frac{2}{10} = 0{,}2$$

Interpretation

- Niedriger Wert → gute Änderbarkeit
- Hoher Wert → starke Kopplung, schlechte Wartbarkeit

Quelle: Bass et al. 2021; ISO/IEC 25010

Präzise Qualitätsanforderungen

Beispiel: Reliability-Metrik (Availability)

- **Stimulus:** Ein Service wird über einen Zeitraum von 30 Tagen betrieben.
- **Environment:** Produktivbetrieb.
- **Artifact:** Rechnungsservice.
- **Response:** Service ist verfügbar oder nicht verfügbar.
- **Response Measure:** Anteil der Zeit, in der der Service verfügbar ist.

Berechnung Beispiel

- Gesamtzeit: 720 Stunden
- Ausfallzeit: Drei Stunden

$$\text{Availability} = \frac{Uptime}{TotalTime}$$

$$\text{Availability} = \frac{720 - 3}{720} = 0{,}996$$

Interpretation

- Wert nahe 1 → hohe Zuverlässigkeit
- Kritisch für kundennahe Prozesse

Quelle: Bass et al. 2021; ISO/IEC 25010

Präzise Qualitätsanforderungen

Beispiel: Security-Metrik (Mean Time to Recover - MTTR)

- **Stimulus:** Sicherheitsvorfall (z. B. fehlerhafte Zugriffskonfiguration).
- **Environment:** Produktivbetrieb.
- **Artifact:** Authentifizierungsservice.
- **Response:** Vorfall wird erkannt und behoben.
- **Response Measure:** Zeit bis zur vollständigen Wiederherstellung.

Berechnung Beispiel

- Drei Vorfälle mit Behebungszeiten: 1h, 2h, 3h

$$\text{MTTR} = \frac{\Sigma \text{Wiederherstellungszeiten}}{\text{Anzahl Vorfälle}}$$

$$\text{MTTR} = \frac{1 + 2 + 3}{3} = 2 \text{ Stunden}$$

Interpretation

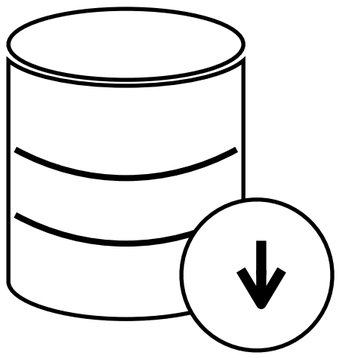
- Niedriger MTTR → hohe Sicherheits- und Reaktionsfähigkeit
- Relevant für kritische Infrastrukturen

Quelle: Bass et al. 2021; ISO/IEC 25010

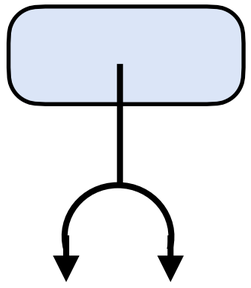
Architekturlösungen

Tactics and Patterns

- Beispiele:
  - *Performance:* Caching, Load Balancing
  - *Modifiability:* Schichtenarchitektur, Abstraktionen
  - *Availability:* Heartbeat, Failover
  - *Security:* Authentifizierung, Verschlüsselung



Caching



Load Balancing

Schichtenmodell

Quelle: Bass et al. 2021; ISO/IEC 25010

Quick Check 2
Vorlesung 10: Fragerunde 2

Auditorium Quiz App

STUDENT

Veranstaltungs-
schlüssel:

aba19

https://quiz.lswi.de/login

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

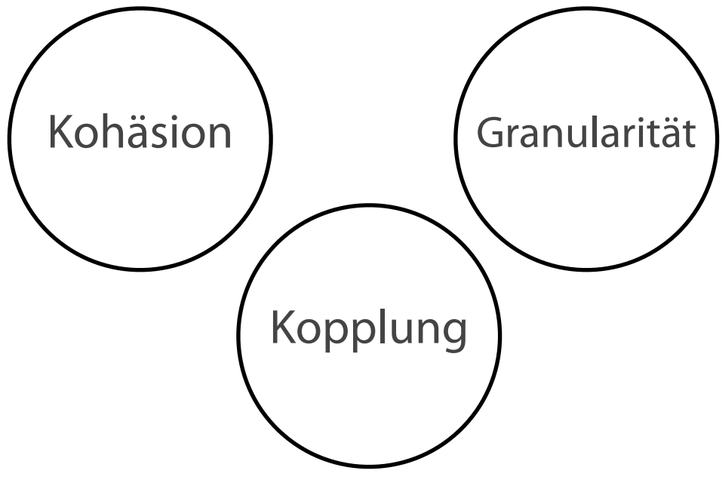
Decomposition

Zerlegung von Systemen

Warum Zerlegung?

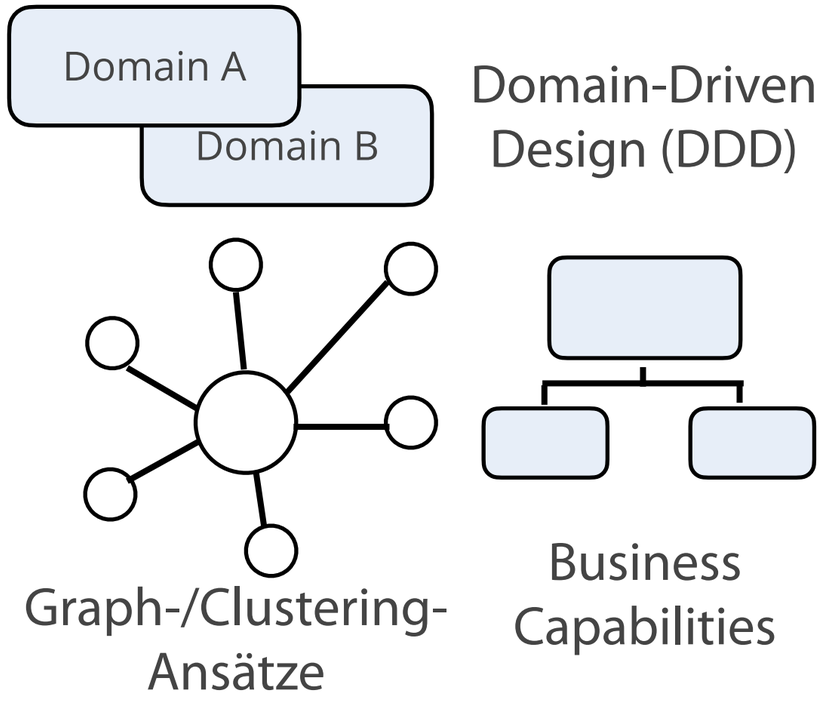
- Systeme werden zu groß und unübersichtlich -> brauchen Struktur.
- Ziel: Kohäsion hoch, Kopplung niedrig, Granularität angemessen.
- Grundlage für Wartbarkeit, Skalierbarkeit, Wiederverwendbarkeit.

Klassische Metriken



Von klassischen Maßen zu modernen Methoden

Moderne Methoden



Quelle: Bass et al. (2021); Evans (2003); Newman (2021)

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

Business Process Alignment (BPA)

Motivation:

- Klassische Metriken (Kohäsion, Kopplung, Granularität) bewerten nur technische Aspekte.
- Morderne, service-orientierte Anwendungssystemen unterstützen Geschäftsprozesse.
- Ziel: Architektur = technische Struktur + Prozessunterstützung.
- *Odoo Sales Prozess*: Quotation → Order → Delivery → Invoice → Payment
- Zerlegung in Services wird anhand von BPA-Metriken bewertet.

BPA-Metriken:

Metrik	**Basierend auf …**	**Bezug zur Literatur**	Herleitung
PSC	BPMN-basierte Prozessmodellierung	Daoud et al. (2020); Delgado et al. (2018)	analog zu Code Coverage in der Softwarequalität (abgedeckte Schritte ÷ alle Schritte)
BCF	Capability-Mapping und Domain-Driven Design	Drieschner et al. (2023); Özkan et al. (2025)	analog zu Jaccard Similarity Index (Überschneidung zwischen Service und Capability)
CIS	Change-Impact-Analysen in MSA	Li et al. (2025); Ortiz et al. (2023)	abgeleitet aus Change Propagation Metric (betroffene Services ÷ alle Services)
PCT	Distributed Tracing / End-to-End-Monitoring	Hui et al. (2025); Taibi & Systä (2019)	analog zu Traceability Coverage (vollständig nachverfolgte Instanzen ÷ alle Instanzen)
PGC	Ziel-Prozess-Abgleich (EA)	Zimmermann et al. (2018); Delgado et al. (2018)	abgeleitet aus Goal Alignment Metrics (gewichteter Mittelwert aus Alignment-Scores)

Präzise Qualitätsanforderungen

BPA-Metrik (Process Step Coverage – PSC)

- **Stimulus:** Analyse eines Geschäftsprozesses (z. B. Odoo Sales).
- **Environment:** Modellbasierte Analyse (BPMN).
- **Artifact:** Services, die Prozessschritte implementieren.
- **Response:** Prozessschritte sind einem Service zugeordnet.
- **Response Measure:** Anteil abgedeckter Prozessschritte.

Berechnung Beispiel

- Prozess hat 5 Schritte
- 4 Schritte sind Services zugeordnet

$$\text{PSC} = \frac{4}{5} = 0{,}8$$

$$\text{PSC} = \frac{durchServicesabgedeckteProzessschritte}{alleProzessschritte} = 0{,}8$$

Interpretation

- PSC = 1 → vollständige Prozessabdeckung
- PSC < 1 → Prozesslücken oder implizite Logik

Quelle: Bass et al. 2021; ISO/IEC 25010

Übung: Microservices & Business Process Alignment

Beispiel:

- *Odoo Sales Prozess*: Quotation → Order → Delivery → Invoice → Payment
- Zerlegung in Services wird anhand von BPA-Metriken bewertet.
- Architekturen werden nicht nur **technisch**, sondern auch **prozessual** bewertet.
- Klassische Metriken ≠ Unterstützung von Geschäftsprozessen.
- In der Übung:
  - Zerlegung eines Systems in Services
  - Bewertung mit **technischen** und **BPA-Metriken**
  - Ziel: Architekturentscheidungen begründet vergleichen

→ Bonuspunkte für saubere Modellierung und nachvollziehbare Argumentation

Motivation und Einordnung

Rolle des Softwarearchitekten

Grundbegriffe Softwarearchitektur

Klassische Architekturen

Microservices

Qualitätsaspekte und Architekturlösungen

Decomposition

Business Process Alignment

Ausblick

Ausblick

- Softwarearchitektur ist **keine einmalige Entscheidung**, sondern ein kontinuierlicher Gestaltungsprozess.
- Der Nutzen von Microservices hängt stark von Prozess- und Domänenbezug ab.
- Metriken helfen, Architekturentscheidungen **transparent, vergleichbar und begründbar** zu machen.
- Business Process Alignment gewinnt besonders bei **ERP-, Plattform- und Legacy-Migrationen** an Bedeutung.
- In Forschung und Praxis: Bedarf an systematischen, prozessorientierten Bewertungsansätzen.

Quelle: Ortiz et al. (2023)

Literatur

Ahlemann, F., Stettiner, E., Messerschmidt, M., Legner, C. (2012). Strategic Enterprise Architecture Management. Berlin, Heidelberg, New York: Springer.

Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice*. Addison-Wesley Professional.

Daoud M, El Mezouari A, Faci N, Benslimane D, Maamar Z, El Fazziki A. Towards an automatic identification of microservices from business processes. In: Proceedings of the 29th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). 2020, 42–47

Delgado, A., Ruiz, F., & García-Rodríguez de Guzmán, I. (2018). A reference model-driven architecture linking business processes and services

Dern, G. (2009). Management von IT-Archtekturen, Vieweg+Teubner.

Drieschner, C., Sensoy, M., Weking, J., & Krcmar, H. (2023). Business Capability Mining: Opportunities and Challenges.

Gronau, N. (2006). Wandlungsfähige Informationssystemarchitekturen: Nachhaltigkeit bei organisatorischem Wandel (2. Aufl). GITO-Verlag.

Gronau, N. (2023). Handbuch der ERP-Auswahl. 3. Aufl. Berlin 2023

Hanschke, I. (2023). Strategisches Management der IT-Landschaft: Ein praktischer Leitfaden für das Enterprise Architecture Management. Carl Hanser Verlag GmbH Co KG.

Hui, M., Wang, L., Li, H., Yang, R., Song, Y., Zhuang, H., ... & Li, Q. (2025). Unveiling the microservices testing methods, challenges, solutions, and solutions gaps: A systematic mapping study. *Journal of Systems and Software*, *220*, 112232.

Keller, W. (2017). IT-Unternehmensarchitektur, 3., überarb. u. erw. *Aufl. dpunkt, Heidelberg*.

Li, N., Liu, Y., Lei, M., Ma, X., & Guo, Q. (2025). Research on Evaluation Model of Microservice Transformation Based on Business Consistency. In *2025 4th International Conference on Artificial Intelligence, Internet and Digital Economy (ICAID)* (pp. 31-37). IEEE.

Ortiz, J., Torres, V., & Valderas, P. (2023). Microservice compositions based on the choreography of BPMN fragments: facing evolution issues. *Computing*, *105*(2), 375-416.

Özkan, O., Babur, Ö., & van den Brand, M. (2025). Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness. Journal of Systems and Software, 112537. https://doi.org/10.1016/j.jss.2025.112537

Taibi, D., & Systä, K. (2019, May). A decomposition and metric-based evaluation framework for microservices. In *International Conference on Cloud Computing and Services Science* (pp. 133–149). Cham: Springer.

Zimmermann, A., Schmidt, R., Sandkuhl, K. (2018). Enterprise Composition Architecture for Micro-Granular Digital Services and Products. In B. Andersson, B. Johansson, S. Carlsson, C. Barry, M. Lang, H. Linger, & C. Schneider (Eds.), *Designing Digitalization (ISD2018 Proceedings)*. Lund, Sweden: Lund University. ISBN: 978-91-7753-876-9. http://aisel.aisnet.org/isd2014/proceedings2018/ISDevelopment/4.