

## Enterprise Architektur-Muster

**Modul “Software Engineering” (Prof. Dr. Andreas Both, Wi-Se 2024/2025)  
an der HTWK Leipzig**

## Anwendungsfall E-Commerce I

- Ziel: Backend für internationales E-Commerce-System
- MVP: Bestellungen, Bezahlung und Versand
- Zukünftig viele Nutzer und hoher Traffic erwartet
- Geringes Kapital für Infrastruktur
- Rechtliche Regularien teilweise unklar, weil international
- Hohe Sicherheitsanforderungen
- Agiles Team von acht fähigen Entwicklern
- Geldgeber wollen erste Auslieferung in zwei Wochen

## Anwendungsfall E-Commerce II

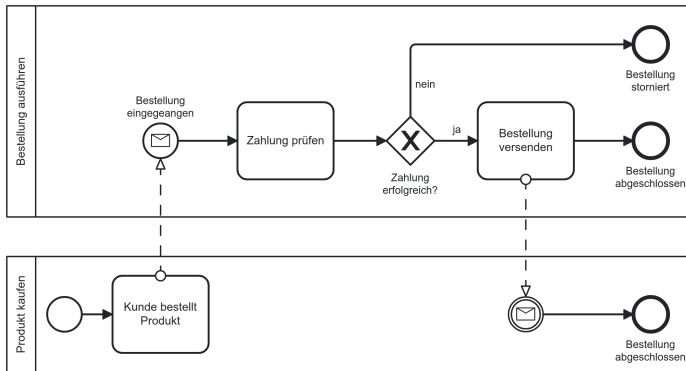


Abbildung 1: Geschäftsprozess zum Aufgeben einer Bestellung

# Grundlagen und Anforderungen

- Architectura lateinisch für: *Wissenschaft der Baukunst*
- Software-Architektur: Grundlegende Struktur und Beziehungen von Teilen einer Software [5]
- Enterprise-Architektur (EA): Grundlegende Struktur und Beziehungen von Komponenten eines Systems
- EA beschreibt Prozess und Ergebnis (vgl. Bedeutung latein. *architectura*)
- Ziel: Ausrichtung von Geschäft und IT
- Umsetzung durch EA-Muster: Spezifische Strategien zur Ausrichtung

# Monolithic Architecture

# Monolithic Architecture

- Monolith ist altgriechisch für *einheitlicher Stein*
- Einheitlich: Alles ist eins - alle Funktionalitäten in einer Komponente [2]
- Stein: Altes Material - früher gut. Heute schlecht?
- Problem: Enge Kopplung

## Monolithic Architecture: Beispiel E-Commerce



Abbildung 2: E-Commerce-Beispiel mit Monolithic Architecture

## Monolithic Architecture: Agilität

- Enge Kopplung ist fatal
  - Keine kleinen autonomen Teams
  - Erhöhte Komplexität und schwere Wartung  $\Rightarrow$  längere Iterationen  $\Rightarrow$  unflexibel
  - Funktionalitäten nicht wiederverwendbar  $\Rightarrow$  Mehraufwand & Duplikation  $\Rightarrow$  Änderungen teuer
  - Isolierte Funktionstests sind möglicherweise aufwendig
- Auslieferung nur im Ganzen & längere Iterationen  $\Rightarrow$  seltene Auslieferung
- Verpflichtung auf genau eine Technologie
- Horizontale Skalierung kaum möglich
- Aber: System ist nicht verteilt  $\Rightarrow$  Keine Intersystemkommunikation  $\Rightarrow$  Geringe Time-to-Market & möglicherweise einfache System-Tests



# Modular Monolithic Architecture

## Modular Monolithic Architecture

- Bisher: Eine eng gekoppelte Komponente
- Jetzt: Eine Komponente mit lose gekoppelten Teil-Funktionalitäten (Modulen)
- Evolution monolithischer Architektur, um Komplexität zu verringern [8]

## Modular Monolithic Architecture: Beispiel E-Commerce



Abbildung 3: E-Commerce-Beispiel mit Modular Monolithic Architecture

## Modular Monolithic Architecture: Agilität

- Im Vergleich zu Monolith: Verbesserungen bezüglich Entwicklung
  - Reduzierte Komplexität  $\Rightarrow$  bessere Wartbarkeit
  - Modularisierung ermöglicht kleine semi-autonome Teams
- Noch immer problematisch:
  - Deployment im Ganzen
  - Keine horizontale Skalierung
  - Keine Wiederverwendbarkeit von Funktionalitäten

# Service-oriented Architecture

## Service-oriented Architecture

- Bisher: Eng gekoppelte Funktionalitäten
- Jetzt: Lose Kopplung von Funktionalitäten durch Kapselung als Dienst
- Ziel Service-oriented Architecture (SOA): Wiederwendung von Funktionalitäten

## Service-oriented Architecture: Komponenten

- Service Provider: Stellt spezifischen Dienst bereit [1]
- Service Consumer: Nutzt bereitgestellten Dienst
- Broker: Vermittler, der Kommunikation zwischen Consumer und Provider regelt
- Service Registry: Sammlung von Metadaten zu Services und deren Provider

## Service-oriented Architecture: Struktur

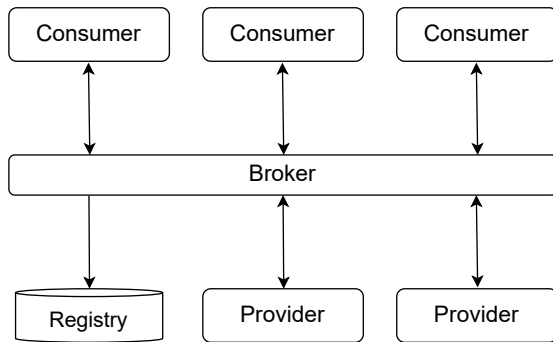


Abbildung 4: Aufbau der Service-oriented Architecture



## Service-oriented Architecture: Beispiel E-Commerce I

- OrderService: Dienst für Verwaltung von Bestellungen
- PaymentService: Dienst für die Abwicklung von Zahlungen (Service von PayPal, ...)
- ShipmentService: Dienst für den Versand (Service von DHL, ...)

## Service-oriented Architecture: Beispiel E-Commerce II

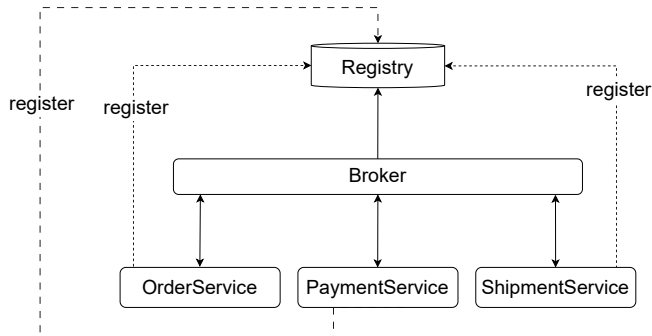


Abbildung 5: E-Commerce-Beispiel mit Service-oriented Architecture

## Service-oriented Architecture: Agilität

- Wie bei modularem Monolithen: Lose Kopplung  $\Rightarrow$  kleine autonome Teams
- Aber zusätzlich: Deployment in Teams  $\Rightarrow$  kürzere Iterationen & häufigere Auslieferung
- Dadurch: Flexibler gegenüber wechselnden Anforderungen
- Zeit- und Kosteneinsparungen durch Wiederverwendung von Diensten
- Eigenständige Dienste ermöglichen horizontale Skalierung
- Aber: Langfristig Abhängigkeiten zwischen Diensten, besonders für grob-granulare Dienste

# Microservice Architecture

## Microservice Architecture

- Bisher: SOA - Aufteilung in Services
- Jetzt: feiner granuliert und vollständig isolierte Services
- Jeder Service hat eigene private Persistenz
- Kommunikation untereinander über Netzwerkprotokolle
- API-Gateway vermittelt zwischen Client und Services

## Microservice Architecture: Struktur

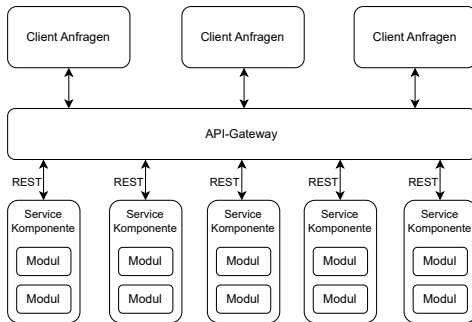


Abbildung 6: Aufbau einer Microservices Architecture

## Microservice Architecture: Beispiel E-Commerce I

- API-Gateway für Client Anfragen
- Auslagerung der Business Logik in isolierte Services
- Unabhängige Services für Bestellung, Bezahlung und Versand
- Aufteilung der zentralen Datenbank in private Persistenzen
- Erweiterung durch zusätzliche Services, horizontale Skalierung gut möglich
- Architekturmuster sehr passend für Anwendungsfall

## Microservice Architecture: Beispiel E-Commerce II

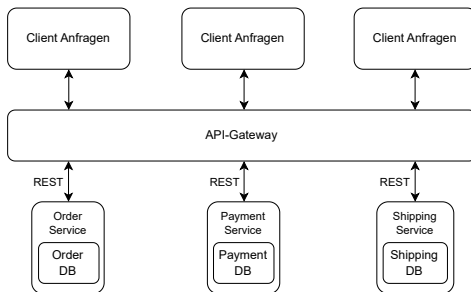


Abbildung 7: E-Commerce-Beispiel mit Microservices Architecture



## Microservice Architecture: Agilität

- Lose Kopplung der Services  $\Rightarrow$  einfaches Testen, Ausliefern und Entwickeln
- Kürzere Iterationen durch einfache Regressionstests
- Live-Deployments können ohne Downtime aktualisiert werden
- Services separat voneinander skalierbar
- Aber: Geringere Performance als andere Muster, da Netzwerklatenzen
- Aber: Aufwendige Integrationstests durch hohe Anzahl an Komponenten
- Aber: Initialer Aufwand höher, da Spezifizierung von Schnittstellen und Architektur nötig [7]
- Fazit: In vielen agilen Umgebungen sinnvoll, aber angemessene Granularität und korrekte Aufteilung wichtig

# Layered Architecture

## Layered Architecture

- Bisher: Isolierter Microservice pro Funktionalität
- Problem: Redundanz in Schnittstellen-Logik (Logging, Authentifizierung, ...)
- Jetzt: Aufteilung System in Schichten
- Ziel: Separation of Concerns & Layers of isolation [7]

## Layered Microservice Architecture: Beispiel E-Commerce

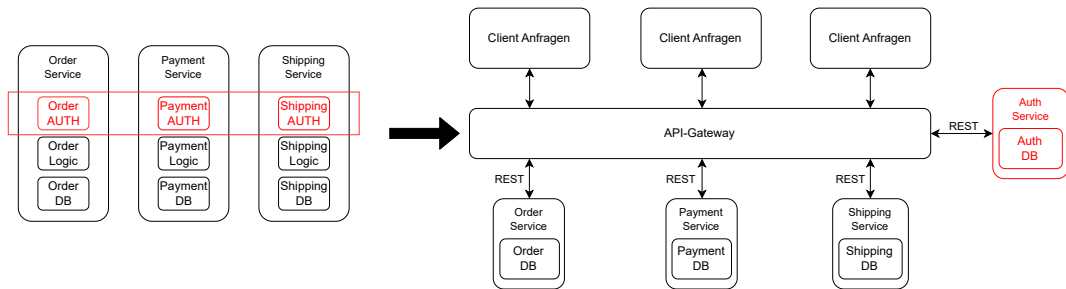


Abbildung 8: E-Commerce Beispiel mit geschichteter Microservice Architektur

## Layered Architecture: Agilität

- Separation of Concerns  $\Rightarrow$  kleine autonome Teams
- Layers of isolation  $\Rightarrow$  hohe Flexibilität
- Funktionalitäten wiederverwendbar  $\Rightarrow$  Auslieferung in kurzen Intervallen
- Kombination mit anderen EA-Mustern sehr sinnvoll

# Event-Driven Architecture

## Event-Driven Architecture

- Bisher: Expliziter Aufruf von Funktionalitäten
- Jetzt: Impliziter Aufruf durch Reaktion auf Ereignisse [4]
- System reagiert asynchron auf Zustandsänderung (Ereignis in System)
- Alte Idee: David Garlan und Mary Shaw, 1994, *An Introduction to Software Architecture*

## Event-Driven Architecture: Komponenten

- Event: Kapselt Information einer Zustandsänderung eines Systems [6]
- Produzent: Erzeugt Event
- Publisher: Publiziert erzeugtes Event
- Konsument: Reagiert auf Event
- Mediator: Vermittler zwischen Produzenten und Konsumenten
- Event-Broker: Infrastruktur für Gesamtheit der Vermittler



## Event-Driven Architecture: Struktur

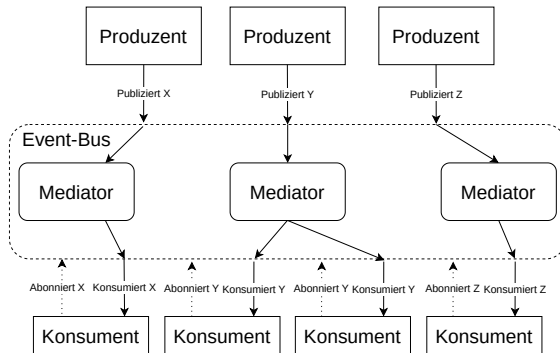


Abbildung 9: Vertrag zwischen Produzenten und Konsumenten am Event-Bus

## Event-Driven Architecture: Beispiel E-Commerce I

- OrderCreated: Genau dann, wenn Bestellung aufgegeben wird
- PaymentProcessed: Genau dann, wenn Bezahlvorgang abgeschlossen wird
- ShipmentInitiated: Genau dann, wenn Bestellung versandt wird
- Event-Kette: OrderCreated → PaymentProcessed → ShipmentInitiated
- Implementierung in Diensten: OrderService, PaymentService, ShipmentService

## Event-Driven Architecture: Beispiel E-Commerce II

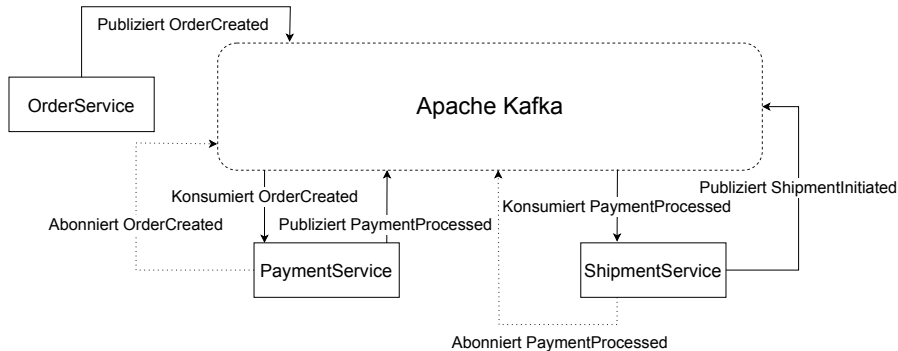


Abbildung 10: E-Commerce-Beispiel mit Event-Driven Architecture

## Event-Driven Architecture: Agilität

- Event ist Vertrag zwischen Produzent und Konsument am Event-Broker  
⇒ Hohe Kohäsion ⇒ Lose Kopplung
- Feature: Menge von Events, deren Produzenten und Konsumenten  
⇒ Klare Abgrenzung ⇒ einfach definierbare Iterationen
- Events sind sehr realitätsnah - domain-driven
- Sehr hohe Flexibilität & maximale Skalierung durch lose Kopplung
- Schnelle Auslieferung, kurze Intervalle
- Exzellente Kombination mit Microservices & Cloud-Integration
- Aber: Erhöhte Komplexität ⇒ Hohe Anforderungen an Entwickler

# Cloud-native Architecture

## Cloud-Native Architecture

- Bisher: Vorab-Allokation von Ressourcen
- Jetzt: Allokation genau dann, wenn notwendig
- Cloud-Native: Explizit für die Cloud entwickelte Applikationen [3]
- Annahme: Infrastruktur ist in ständigem Wandel
- Folgerung: Infrastruktur auslagern - an Cloud-Vendor
  - Globale Nutzung durch Geo-Redundanz: Starke Verteilung und hohe Verfügbarkeit
  - Auto-Scaling: Dynamische Skalierung basierend auf Nachfrage
  - Pay-as-you-go, Scale-to-zero: Nur verwendete Ressource wird bezahlt
  - Zero Downtime

## Cloud-Native Architecture: Technologie

- Containerisierung: Jede Komponente eines Systems ist Container
- Dynamische Orchestrierung: Aktives Container-Management zur Ressourcenoptimierung
- Microservice-Architektur ist Fundament - ergänzt durch Cloud-Dienste
- Fully Managed Cloud-Services: Business-Logik statt Infrastruktur

## Cloud-Native Architecture: Beispiel E-Commerce

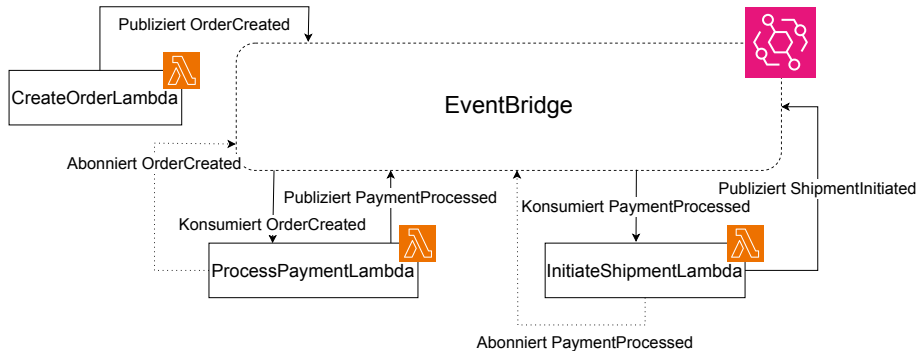


Abbildung 11: E-Commerce-Beispiel mit Cloud-Native Architecture in AWS



## Cloud-Native Architecture: Agilität

- Alle agilen Vorteile von Microservice- und Event-Driven Architecture
- Fokus auf Business-Logik & einfaches Deployment  $\Rightarrow$  Kurze Iterationen
- Maximale Flexibilität für Nachfrage durch Auto-Scaling
- Finanzielle Agilität: Pay-as-you-go
- Aber: Kostenrisiken durch Auto-Scaling und Pay-as-you-go
- Achtung: Vendor-Lock-In durch proprietäre Fully Managed Cloud-Services

# Microkernel Architecture

## Microkernel Architecture

- Aufteilung der Anwendung in zwei Arten von Komponenten [7]
- Kern:
  - Minimale Funktionalität
  - Bietet Schnittstelle für Erweiterungen/Plugins
  - Enthält Plugin-Registry
- Module:
  - Erweitern Kern um Funktionalitäten
  - Kommunikation über definierte Schnittstellen
  - Lose gekoppelt, unabhängig und isoliert voneinander
  - Verbindung über verschiedene Wege möglich (REST, Messaging, Objekt Instanziierung, ...)

## Microkernel Architecture: Struktur

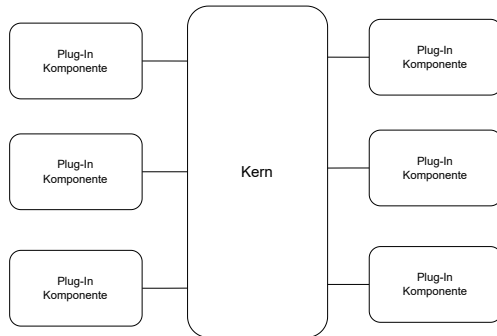


Abbildung 12: Aufbau einer Microkernel Architecture

## Microkernel Architecture: Beispiel E-Commerce I

- Kern: Großteil der Funktionalität
- Module: Auslagerung von Business-Logik zum Bezahlen und Versenden
- Einfache Erweiterbarkeit durch neue Dienstleister
- Komplexer Kern, hohe Kopplung der übrigen Funktionalitäten
- Als einziges Architekturmuster eher nicht geeignet, jedoch in Kombination mit anderen sinnvoll

## Microkernel Architecture: Beispiel E-Commerce II



Abbildung 13: E-Commerce-Beispiel mit Microkernel Architecture

## Microkernel Architecture: Agilität

- Lose Kopplung der Module  $\Rightarrow$  schnelle Reaktionsfähigkeit auf Änderungen
- Module können einfach ausgetauscht werden  $\Rightarrow$  geringe Downtime
- Einfach testbar, da Module unabhängig und isoliert voneinander
- Kurze Iterationen und Auslieferungszeiten bei Erweiterung der Anwendung
- Aber: Hoher initialer Aufwand durch teuren Kern, eher hohe Time-to-Market
- Aber: Hoher Aufwand, wenn Kern später Anpassungen benötigt
- Fazit: In ausgewählten Anwendungsfällen sinnvoll, aber nicht universell in agilen Umgebungen einsetzbar

## Zusammenfassung

- EA ist dynamische Strategie (Vision), um IT auf Business auszurichten
- Wahl EA beeinflusst Agilität in Entwicklung substantiell
- (Modularer) Monolith: Eine (aus Funktionalitäten bestehende) Komponente
- SOA: Wiederverwendbarkeit von Funktionalitäten durch (meist grobe) Dienste
- Micro-Kernel: Kernfunktionalität wird um Zusatzfunktionalitäten erweitert
- Microservices: Isolierung von Funktionalitäten durch feine Dienste
- EDA: Asynchrone Reaktion auf Ereignisse
- Cloud-Nativ: Auslagerung Verwaltung Infrastruktur  $\Rightarrow$  Fokus auf Businesslogik
- Es gibt keine beste Architektur - Wahl ist anwendungsspezifisch und meist Kombination mehrerer EA-Muster



# Übungsaufgaben

## Übungsaufgabe I

Das E-Commerce-Beispiel aus der Einleitung soll um Nutzer-Authentifizierung erweitert werden. Sie haben sich zuvor für eine Microservice-Architektur entschieden und die in der Einleitung genannten Anforderungen implementiert. Die Authentifizierung wird in verschiedenen Komponenten benötigt. Erläutern Sie, wie Sie die Authentifizierung in die Architektur integrieren.

## Übungsaufgabe II

Das E-Commerce-Beispiel aus der Einleitung soll um Logging erweitert werden. Sie haben sich zuvor für eine cloud-native Event-Driven-Architektur entschieden und die in der Einleitung genannten Anforderungen implementiert. Das Logging wird in verschiedenen Komponenten benötigt.

Erläutern Sie, wie Sie das Logging in die Architektur integrieren.

Bedenken Sie dabei, dass Logs aus verschiedenen Komponenten möglicherweise zur Auswertung zusammengeführt werden müssen und somit die Reihenfolge von Logs relevant ist.

## Übungsaufgabe III

Das E-Commerce-Beispiel aus der Einleitung soll um ein E-Mail-Notifikationssystem erweitert werden. Dieses soll Nutzern E-Mails bei jeder Statusänderung einer ihrer Bestellungen zustellen.

Untersuchen Sie für alle in diesem Papier betrachteten Architekturen, wie das E-Mail-Notifikationssystem in die bestehende Architektur integriert werden kann und welche Vor- und Nachteile diese Integration in die jeweilige Architektur mit sich bringt.

Für welche der Architekturen ist die Integration des E-Mail-Notifikationssystems am einfachsten?

## Literatur I

- [1] Phil Bianco, Rick Kotermanski und Paulo Merson. *Evaluating a service-oriented architecture*. Citeseer, 2007.
- [2] Grzegorz Blinowski, Anna Ojdowska und Adam Przybyłek. “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”. In: *IEEE Access* 10 (2022), S. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803.
- [3] Dennis Gannon, Roger Barga und Neel Sundaresan. “Cloud-Native Applications”. In: *IEEE Cloud Computing* 4.5 (2017), S. 16–21. DOI: 10.1109/MCC.2017.4250939.
- [4] David Garlan und Mary Shaw. *An Introduction to Software Architecture*. Techn. Ber. CMU/SEI-94-TR-021. Accessed: 2025-Jan-2. Jan. 1994. URL: [https://insights.sei.cmu.edu/library/an-introduction-to-](https://insights.sei.cmu.edu/library/an-introduction-to-software-architecture/)

## Literatur II

- [5] Danny Greefhorst und Erik Proper. “The Role of Enterprise Architecture”. In: *Architecture Principles: The Cornerstones of Enterprise Architecture*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 7–29. ISBN: 978-3-642-20279-7. DOI: [10.1007/978-3-642-20279-7\\_2](https://doi.org/10.1007/978-3-642-20279-7_2). URL: [https://doi.org/10.1007/978-3-642-20279-7\\_2](https://doi.org/10.1007/978-3-642-20279-7_2).
- [6] Ramakrishna Manchana. “Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries”. In: *International Journal of Science and Research (IJSR)* 10 (Jan. 2021), S. 1706–1716. DOI: [10.21275/SR24820051042](https://doi.org/10.21275/SR24820051042).
- [7] Mark Richards. *Software Architecture Patterns*. O'Reilly Media, Inc., 2015, value. ISBN: 9781491925409.

## Literatur III

- [8] Ruoyu Su und Xiaozhou Li. “Modular Monolith: Is This the Trend in Software Architecture?” In: *Proceedings of the 1st International Workshop on New Trends in Software Architecture*. New York, NY, USA: Association for Computing Machinery, 2024, S. 10–13. ISBN: 9798400705601. DOI: 10.1145/3643657.3643911. URL: <https://doi.org/10.1145/3643657.3643911>.