#9 TECHARTIKEL

GRUNDLAGEN DER SOFTWAREARCHITEKTUR

23.05.2024

Robert Jeutter

Inhalt

Grundlagen der Softwarearchitektur

Softwarearchitektur bildet das Rückgrat jedes Softwareprodukts und spielt eine entscheidende Rolle in dessen Entwicklung, Wartung und Skalierung. Dieser Artikel vermittelt ein tiefgehendes Verständnis darüber, was Softwarearchitektur ist, warum sie unerlässlich ist, und wie sie sich im Kontext moderner Technologieanforderungen stetig weiterentwickelt.

1. Die Essenz der Softwarearchitektur

Jedes Softwareprodukt besitzt eine Architektur – sei es geplant oder emergent. Eine gut durchdachte Softwarearchitektur ist entscheidend für die Qualität, Wartbarkeit und Erweiterbarkeit von Softwarelösungen. Ohne eine solide Architektur kann Software schnell zu einem unübersichtlichen und schwer wartbaren System werden, das die Entwickler vor große Herausforderungen stellt.

Definition und Kernaspekte

Softwarearchitektur bezieht sich auf die grundlegende Strukturierung eines Softwaresystems und umfasst eine Reihe von Prinzipien und Entscheidungen, die die Organisation des Systems, die Auswahl struktureller Elemente und deren Schnittstellen sowie deren Verhaltensweisen im Zusammenspiel definieren [1].

Der Kern jeder Architektur wird durch **folgende Elemente** gebildet:

- **Komponenten:** Dies sind die grundlegenden Bausteine der Architektur, wie Module, Klassen oder Pakete, die spezifische Funktionalitäten innerhalb des Systems implementieren. Eine Komponente kapselt ihr inneres Verhalten ab und kommuniziert nur über definierte Schnittstellen mit anderen Bausteinen. Dadurch ist jede Komponente zwangsläufig austauschbar und wiederverwendbar.

- **Beziehungen**: Die Art und Weise, wie diese Komponenten miteinander interagieren, kommunizieren und Daten austauschen, ist für die Funktionalität des gesamten Systems entscheidend. Beziehungen in einer Architektur bestehen zwischen einzelnen Elementen und gegenüber der Umgebung.

- **Entwurfsentscheidungen**: Sie beeinflussen den gesamten Lebenszyklus der Software, von der Entwicklung über den Betrieb bis hin zur Wartung. Entwurfsentscheidungen umfassen verschiedene Muster und Richtlinien, wie z.B. Schichtenarchitekturen, ereignisgesteuerte Architekturen oder Microservices, die je nach Projektanforderungen ausgewählt werden.

Die Notwendigkeit strategischer Planung

Investitionen in die Architektur zu Beginn eines Projekts können entscheidend sein, um spätere Kosten durch Fehlerbehebung und Anpassungen zu minimieren. Eine strategisch geplante Architektur ermöglicht es, frühzeitig potenzielle Schwachstellen zu identifizieren und effiziente, wartbare Lösungen zu entwickeln. Dadurch kann eine gut geplante Software für eine sehr lange Zeit betrieben und leicht an neue Anforderungen angepasst werden, statt eine große Legacy-Anwendung nach kurzer Zeit wieder durch eine Neuentwicklung ersetzen zu müssen.

2. Bedeutung der Softwarearchitektur

Die Rolle der Softwarearchitektur geht weit über die einfache Strukturierung von Code hinaus. Sie beeinflusst maßgeblich die Qualität und Lebensfähigkeit eines Softwareprodukts.

Gute Architektur trägt dazu bei, wichtige Qualitätsattribute wie Performance, Sicherheit, Skalierbarkeit und Robustheit zu gewährleisten. Sie ermöglicht es, dass Software auch unter veränderten Anforderungen effizient funktioniert und erweitert werden kann. Eine Auswahl von Qualitätsanforderungen kann z.B. der ISO 25010 [2] entnommen werden. Für jedes Projekt werden die Qualitätsattribute individuell ausgewählt, priorisiert und passend zugeschnitten gestaltet.

In einer Welt, in der sich Geschäftsbedingungen und Technologien rasant ändern, muss Software flexibel bleiben. Die Architektur muss so gestaltet sein, dass sie leicht an neue Anforderungen angepasst werden kann, ohne dass das gesamte System neu entwickelt werden muss. Anforderungen können so schnell und konsistent verfolgt und weiterentwickelt werden, sparen dadurch wertvolle Zeit und können in kurzer Zeit auch durch die Entwickler umgesetzt werden.

Eine unzureichend durchdachte Architektur kann dazu führen, dass Systeme schnell veralten und schwer zu warten sind. Eine vorausschauende Architekturplanung schützt vor solchen Legacy-Problemen, indem sie klare Richtlinien für die Evolution der Software vorgibt.

3. Einflussfaktoren auf die Softwarearchitektur

Die Gestaltung der Softwarearchitektur wird von einer Vielzahl von Faktoren beeinflusst, die sowohl technischer als auch organisatorischer Natur sein können. Diese Faktoren zu verstehen und zu berücksichtigen, ist entscheidend für die Entwicklung effektiver und nachhaltiger Architekturen.

- **Technologische Trends:** Neue Technologien können die Möglichkeiten der Architekturgestaltung erweitern oder bestehende Ansätze veralten lassen. Die Auswahl der Technologie sollte daher sorgfältig erfolgen, um sicherzustellen, dass sie den langfristigen Anforderungen des Projekts gerecht wird. Es gilt jedoch zu verhindern, eine bestimmte Technologie zu wählen, weil diese gerade im Trend liegt. Die Wahl der Technologie sollte erst nach der Auswahl an Anforderungen und Qualitätsmerkmalen folgen.

- **Geschäftsstrategien und -ziele:** Die Architektur muss die Geschäftsziele unterstützen und flexibel genug sein, um sich an verändernde Marktbedingungen anpassen zu können. Dies erfordert eine enge Abstimmung mit den Stakeholdern, um die geschäftlichen Anforderungen präzise zu verstehen und umzusetzen. Nicht alle Stakeholder benötigen den gleichen Einfluss oder Rücksprachen - ein gutes Mittel zur Analyse ist eine Stakeholder-Priorisierungs-Matrix [3].

- **Gesetzliche Vorgaben:** In vielen Branchen, wie dem Finanzwesen oder dem Gesundheitswesen, müssen Softwareprodukte strenge regulatorische Anforderungen erfüllen. Die Architektur muss daher so gestaltet sein, dass Compliance sichergestellt ist. In solchen Fällen wird der Gesetzgeber meist als eigener Stakeholder betrachtet, der zufriedengestellt werden muss. Das Interesse dieses Stakeholders am Produkt selbst ist meistens jedoch sehr gering.

- **Standards und Best Practices:** Industriestandards und Best Practices können als Leitfaden für die Architekturgestaltung dienen und helfen, die Qualität und Interoperabilität der Software zu sichern. Beispiele sind die SOLID oder DRY Prinzipien.

- **Budget und Zeitrahmen:** Diese begrenzen oft, was architektonisch möglich ist. Ein effektives Architekturdesign findet Wege, um die besten Ergebnisse innerhalb dieser Grenzen zu erzielen. Es kann auch frühzeitig erkannt werden, welche Anforderungen oder Features nicht innerhalb dieser Rahmenbedingungen umsetzbar sind.

- **Fähigkeiten des Teams:** Die Fähigkeiten und Erfahrungen der beteiligten Entwickler und Architekten spielen eine große Rolle bei der Auswahl der Architekturstrategie. Eine komplexere Architektur kann möglicherweise bessere Ergebnisse liefern, erfordert jedoch entsprechend qualifizierte Mitarbeiter. Meist beginnt die Softwarearchitekturentwicklung mit dem Architekten und dem Product Owner oder Projektleiter bevor Entwickler für die Umsetzung hinzugezogen werden. Dies unterscheidet sich meist nach den organisatorischen Strukturen des Unternehmens.

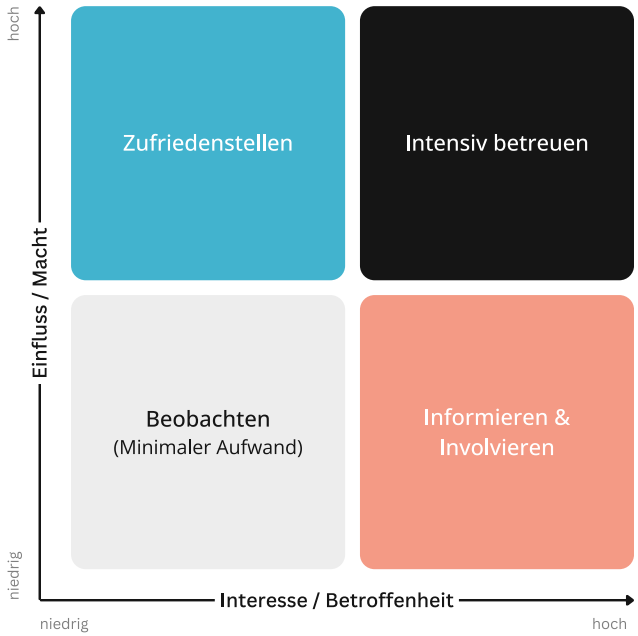

Abbildung 1: Beispiel einer Stakeholder-Priorisierungs-Matrix [4]

4. Herausforderungen in der Softwarearchitektur

Die Entwicklung einer robusten Softwarearchitektur ist kein trivialer Prozess und bringt zahlreiche Herausforderungen mit sich, die bewältigt werden müssen.

Balance zwischen Flexibilität und Kontrolle

Eine der größten Herausforderungen ist die Balance zwischen der Flexibilität der Architektur, um auf Veränderungen reagieren zu können, und der Notwendigkeit, eine konsistente und kontrollierbare Umgebung zu erhalten. Zu viel Flexibilität kann zu einem unkontrollierten System führen, während zu viel Kontrolle Innovationen behindern kann.

Integration und Schnittstellenmanagement

Die Fähigkeit, verschiedene Systemkomponenten nahtlos zu integrieren und effektiv zu verwalten, ist entscheidend. Dies umfasst die Gestaltung von Schnittstellen, die sowohl klar definiert als auch flexibel genug sind, um mit zukünftigen Änderungen umgehen zu können. Die Gestaltung von Schnittstellen erfordert ein tiefes Verständnis der beteiligten Systeme sowie der Daten und Funktionen, die über diese Schnittstellen ausgetauscht werden sollen. Schnittstellen sollten regelmäßig überprüft und getestet werden, um sicherzustellen, dass sie wie erwartet funktionieren und keine Sicherheitslücken aufweisen. Automatisierte Tests und kontinuierliche Integration können dazu beitragen, Probleme frühzeitig zu erkennen und zu beheben.

Sicherheit und Datenschutz

In der heutigen digitalen Ära sind Sicherheit und Datenschutz von größter Bedeutung. Die Architektur muss von Anfang an Sicherheitsmechanismen einbeziehen, die sowohl die Datenintegrität als auch die Privatsphäre der Nutzer schützen.

5. Agile Entwicklung und iterative Architekturgestaltung

In einem agilen Entwicklungsprozess wird die Softwarearchitektur nicht als statisches Element betrachtet, sondern als sich kontinuierlich weiterentwickelnde Struktur. Dieser Ansatz unterstützt die schnelle Anpassung an veränderte Anforderungen und ermöglicht es, innovative Lösungen effektiv zu implementieren.

Iterative Entwicklung und Feedbackschleifen

Durch die iterative Entwicklung können Architekten und Entwickler Architekturkonzepte schrittweise einführen und verfeinern. Regelmäßiges Feedback von Stakeholdern und Endnutzern ist entscheidend, um sicherzustellen, dass die Architektur den tatsächlichen Bedürfnissen entspricht.

Zusammenarbeit im Team

Agile Architektur fördert die Zusammenarbeit zwischen verschiedenen Rollen, einschließlich Entwicklern, Architekten, Produktmanagern und Kunden. Diese Zusammenarbeit hilft, ein tiefes Verständnis der Ziele und Herausforderungen des Projekts zu entwickeln und gemeinsam effektive Lösungen zu gestalten.

6. Vertiefung der Qualitätssicherung in der Softwarearchitektur

Ein wesentlicher Aspekt der Softwarearchitektur, der oft übersehen wird, ist die ausdrückliche Definition des Problems, das die Software lösen soll. In vielen Architekturdokumentationen finden sich zwar detaillierte Beschreibungen der Struktur eines Systems, jedoch fehlen häufig klare Angaben darüber, welches spezifische Problem adressiert wird. Diese Lücke erschwert das Verständnis und die Bewertung der getroffenen Architekturentscheidungen erheblich.

Der Qualitätsbaum nach ISO 25010

Um die Qualität einer Softwarearchitektur vollumfänglich zu erfassen und zu verbessern, wird der Qualitätsbaum nach ISO 25010 eingesetzt. Dieser bietet eine strukturierte Methode zur Bewertung verschiedener Qualitätsmerkmale und deren Teilmerkmale.

Zuverlässigkeit, ein zentrales Qualitätsmerkmal, wird beispielsweise weiter in Fehlertoleranz und Wiederherstellbarkeit unterteilt. Fehlertoleranz bedeutet dabei, dass das System auch bei Ausfall einiger Komponenten funktionsfähig bleibt. Wiederherstellbarkeit hingegen fokussiert auf die schnelle Wiederherstellung des Systems nach einem Ausfall.

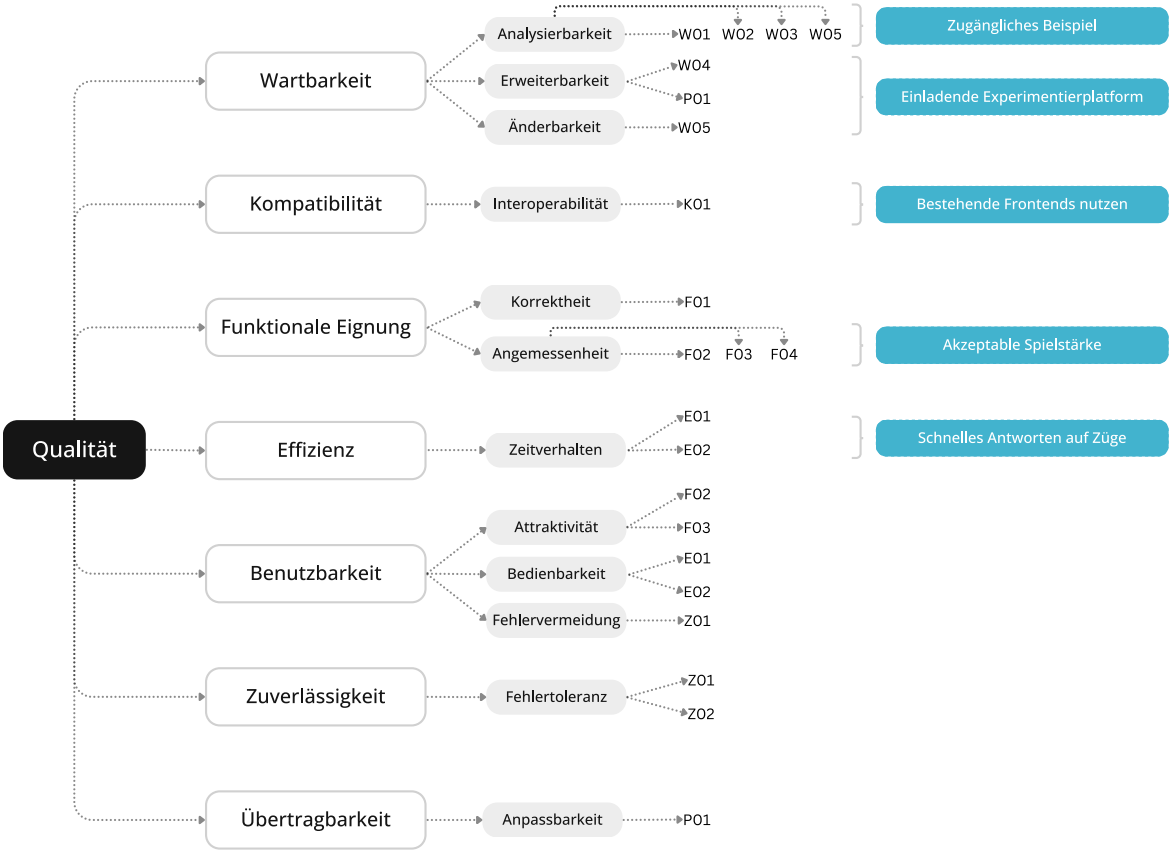

Abbildung 2: Qualitätsbaum aus DokChess [5]

Durch die Anwendung des Qualitätsbaums können Architekten sicherstellen, dass alle relevanten Qualitätsaspekte berücksichtigt werden. Dies fördert eine ganzheitliche Betrachtung der Softwarequalität, die über die übliche Konzentration auf Verfügbarkeit hinausgeht.

Qualitätsszenarien: Praktische Anwendung und Verifizierung

Trotz der Struktur, die der Qualitätsbaum bietet, reicht er allein nicht aus, um die Einhaltung der Qualitätsmerkmale effektiv zu überprüfen. Hier kommen Qualitätsszenarien ins Spiel. Diese beschreiben spezifische Ereignisse, die auf das System einwirken, und definieren konkrete Metriken, um das gewünschte Verhalten des Systems zu messen. Qualitätsszenarien machen Qualitätsmerkmale greifbar und leicht verifizierbar.

Arten von Qualitätsszenarien

Benutzungsszenarien
Sie adressieren die Anforderungen, die sich aus der Benutzung der Software ergeben.

Beispiel
Bei der Registrierung eines Benutzers sollte die Notwendigkeit, den Kundendienst zu kontaktieren, extrem niedrig sein, was durch entsprechende Benutzerfreundlichkeit erreicht wird.

Änderungsszenarien
Diese beschreiben, wie das System auf geplante Änderungen reagieren soll.

Beispiel
Die Unterstützung einer neuen Sprache ohne Änderungen am Code.

Ausfallszenarien
Sie betreffen die Reaktion des Systems auf Ausfälle.

Beispiel
Die schnelle Wiederherstellbarkeit ohne Datenverlust.

Diese Szenarien sind nicht nur für die Bewertung der aktuellen Architektur wertvoll, sondern auch richtungsweisend für erforderliche Anpassungen und Weiterentwicklungen. Indem Architekten Qualitätsszenarien formulieren und Lösungen zur Erfüllung dieser Szenarien entwickeln, können sie konkret aufzeigen, wie die Architektur die gestellten Anforderungen erfüllt.

7. Architekturmuster und Strategien

Die Auswahl des richtigen Architekturmusters ist entscheidend für den Erfolg eines Softwareprojekts. Jedes Muster bietet spezifische Vorteile und ist für bestimmte Arten von Projekten geeignet [4]. Hier werden drei der gängigsten Architekturmuster kurz vorgestellt und deren Einsatzmöglichkeiten aufgezeigt:

Schichtenarchitektur (Layered Architecture)

Die Schichtenarchitektur ist eines der am häufigsten verwendeten Muster und teilt das System in logische Schichten von Funktionalität, die aufeinander aufbauen. Jede Schicht hat eine spezifische Aufgabe, wie Präsentation, Geschäftslogik oder Datenzugriff.

Vorteile

- Klare Trennung von Verantwortlichkeiten erleichtert die Wartung und Skalierung.
- Austausch oder Erweiterung einzelner Schichten ist ohne großen Einfluss auf andere Bereiche möglich.

Beispiel Bankensystem
Ein Bankensystem kann von einer Schichtenarchitektur profitieren, da es strenge Sicherheits- und Compliance-Anforderungen gibt. Die Trennung von Benutzeroberfläche, Geschäftslogik und Datenzugriffsebene ermöglicht es, Sicherheitskontrollen und Geschäftsregeln effizient zu implementieren und zu verwalten.

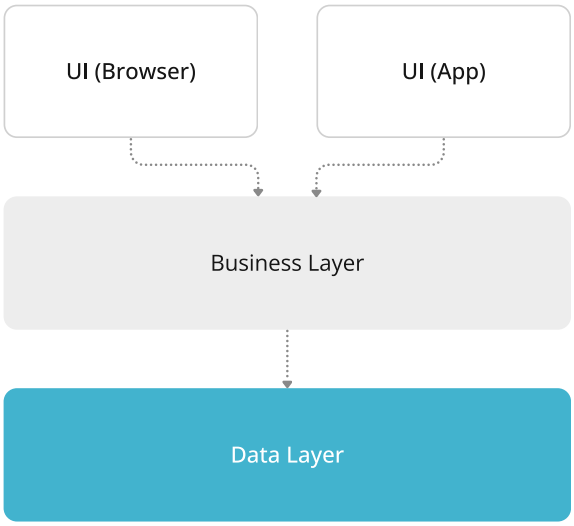

Abbildung 3: Aufbau einer Schichtenarchitektur

Ereignisgesteuerte Architektur (Event-Driven Architecture)

Diese Architektur eignet sich besonders für Anwendungen, die auf Ereignisse oder Nachrichten reagieren, wie beispielsweise komplexe Event-Processing-Systeme oder Echtzeitanalysewerkzeuge.

Vorteile

- Hohe Skalierbarkeit und Flexibilität durch asynchrone Verarbeitung.
- Erleichtert die Integration von neuen Komponenten, die auf Ereignisse reagieren.

Beispiel Real-Time Datenverarbeitung in der Logistik
In der Logistikbranche, wo Echtzeit-Datenverarbeitung kritisch ist, könnte eine ereignisgesteuerte Architektur verwendet werden, um die Bewegungen von Fahrzeugen und Paketen in Echtzeit zu verfolgen und darauf zu reagieren. Dieses Modell ermöglicht schnelle und flexible Reaktionen auf unvorhergesehene Ereignisse.

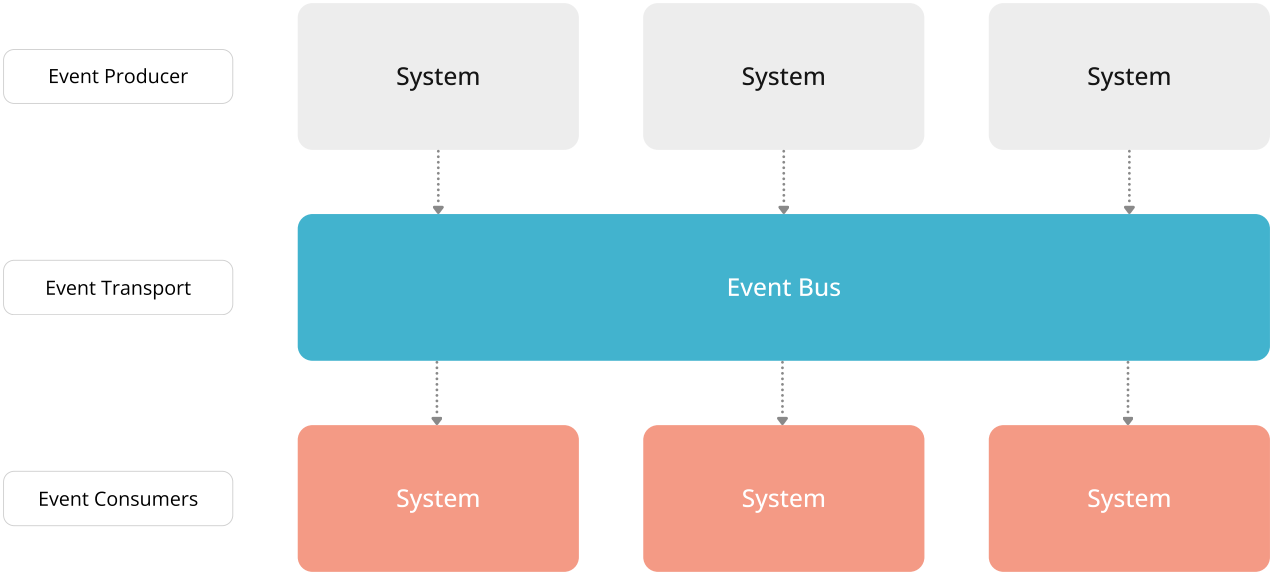

Abbildung 4: Aufbau einer Event-Driven Architektur

Microservice-Architektur

Ein grundlegendes Merkmal der Mikroservices ist ihre Unabhängigkeit: sie kommunizieren untereinander ausschließlich über wohldefinierte Schnittstellen. Diese Modularität wird durch Designprinzipien wie hohe Kohäsion und geringe Kopplung unterstützt, die sicherstellen, dass jeder Service eine klar abgegrenzte Funktionalität bietet und minimale Abhängigkeiten zu anderen Services aufweist. Interna, wie Datenbankschemata, bleiben innerhalb der Microservices gekapselt und sind von außen nicht zugänglich, was die Sicherheit und Wartbarkeit des Gesamtsystems erhöht.

Vorteile

- Fördert die Skalierbarkeit und erleichtert die kontinuierliche Integration und Bereitstellung.
- Jeder Microservice kann unabhängig entwickelt und skaliert werden, was die Agilität fördert.

Beispiel E-Commerce-Plattform
Ein Online-Einkaufsportal könnte von einer Microservice-Architektur profitieren, da einzelne Services wie Bestellabwicklung, Zahlungsverkehr und Kundenservice unabhängig voneinander entwickelt und skaliert werden können. Dies ermöglicht schnelle Updates und Verbesserungen einzelner Teile des Portals ohne Ausfallzeiten oder Störungen des Gesamtsystems.

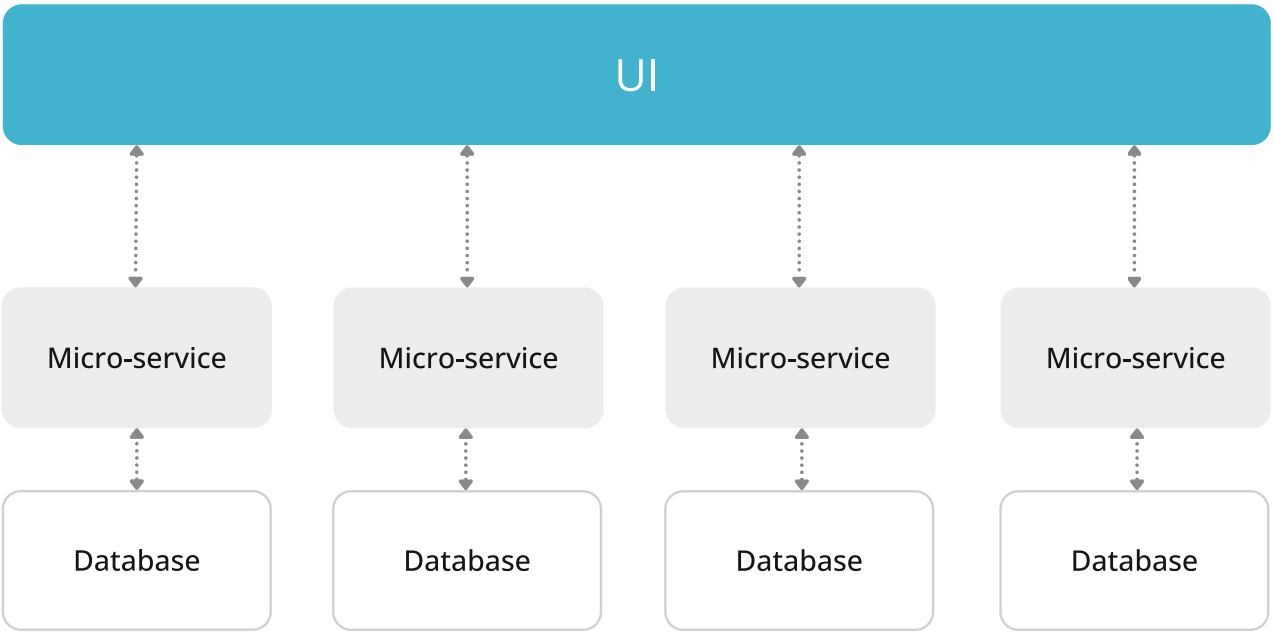

Abbildung 5: Aufbau einer Microservice Architektur

Weitere Architekturmuster

Es gibt noch eine große Anzahl weiterer Architekturmuster, deren Umfang diesen Artikel jedoch sprengen würden und trotzdem nennenswert sind:

- **Client Server:** Beispielsweise oft in Webanwendungen.
- **Peer to Peer:** Keine zentrale Koordination der Knoten.
- **Broker Modell:** Broker koordiniert Kommunikation zwischen Client und Services.
- **Blackboard Modell:** Enthält "Gedächtnis" für Services.
- **Main/Worker (früher Master/Slave):** Verteilen der Arbeit auf mehrere Knoten.
- **Plug-In:** Kern-Anwendung, die durch Plug-Ins um neue Funktionen erweitert werden kann.

8. Fazit

Die herkömmliche Auffassung von "guter" Architektur, die oft mit sauberem Code oder Skalierbarkeit gleichgesetzt wird, reicht nicht aus, um Qualitätsszenarien effektiv zu adressieren. Auch die Wahl spezifischer Frameworks oder Programmiersprachen ist oft nicht ausschlaggebend für die Erfüllung der Anforderungen.

Die Wahl der richtigen Architektur ist ein kritischer Schritt, der weitreichende Konsequenzen für die Leistung, Wartbarkeit und Skalierbarkeit eines Softwareprodukts hat. Durch die Verwendung von etablierten Methoden, Architekturmustern und Strategien können Entwickler robuste Systeme erstellen, die den Anforderungen moderner Geschäftsprozesse und Technologien gerecht werden.

Die Beispiele zeigen, wie unterschiedlich die Anforderungen und Lösungen sein können, abhängig von der spezifischen Situation und den Zielen des Projekts. Eine gründliche Analyse der Anforderungen und eine sorgfältige Planung der Architektur sind unerlässlich, um die langfristigen Ziele der Softwareentwicklung erfolgreich zu unterstützen.

Quellen

1. **"Software-Architektur: Worauf es ankommt" von Eberhard Wolff**
2. **ISO 25010**
3. **projekte-leicht-gemacht.de**
4. **iSAQB**
5. **dokchess.de**