Softwarearchitektur

4

Wolfgang Albrecht

Die Architektur eines Softwaresystems hat entscheidenden Einfluss auf ihre Güte und Qualität. Die Architektur einer Software beschreibt den inneren und äußeren Rahmen eines Softwaresystems. Durch die Architektur werden die Bausteine einer Software sowie die Schnittstellen, die Hierarchie der einzelnen Bausteine sowie die Beziehungen untereinander festgelegt und in strukturierter Form beschrieben. Die Softwarearchitektur ist somit die Zerlegung des Softwaresystems in seine einzelnen Bausteine.

Eine einmal festgelegte Softwarearchitektur lässt sich zu einem späteren Zeitpunkt im Lebenszyklus einer Software nur mit sehr viel Aufwand und Kosten verändern. Aus diesem Grund ist die Entscheidung über die zu verwendende Architektur für eine Software eine der wichtigsten Entscheidungen innerhalb des Entwicklungsprozesses der betreffenden Software. Sie hat auch maßgeblichen Einfluss auf

- Laufzeitverhalten,
- Betrieb,
- Betriebssicherheit,
- Verfügbarkeit,
- Flexibilität,
- Erweiterbarkeit,
- Wartbarkeit,
- Effizienz in der Entwicklung sowie
- Wiederverwendbarkeit

einer Software. Aufgrund dessen sollte die Architektur einer Software auf diese Anforderungen abgestimmt sein. Ebenso sollte bei der Auswahl eines einzuführenden Softwareproduktes die dieser Software zugrundeliegende Architektur betrachtet werden, da die Architektur Auskunft über maßgebliche Eigenschaften der Software gibt.

K.-H. Wehking, *Technisches Handbuch Logistik 2,*
https://doi.org/10.1007/978-3-662-60869-2_4

Auf Basis der vorliegenden Softwarearchitektur lassen sich bereits über Simulationsbetrachtungen erste Aussagen über die Einhaltung genereller Anforderungen (z. B. Laufzeitverhalten, Verfügbarkeit, etc.) an die Software treffen. Ebenso sind auf Basis der Softwarearchitektur bereits einige Risiken (z. B. zu hoher Abstimmungsaufwand zwischen Gewerken und Entwicklerteams bedingt durch zu hohem Grad an inneren Abhängigkeiten, Kombination unterschiedlicher Softwaretechnologien, etc.) für das Entwicklungsprojekt identifizierbar.

Zudem ist die Architektur einer Software ein wirkungsvolles Element für die Kommunikation innerhalb des Entwicklungsprojektes. Sie erleichtert das Verständnis der Entwickler im Team für die Struktur der Software und verkürzt so die Einarbeitungszeit neuer Mitarbeiter beziehungsweise neuer Mitglieder im Entwicklungsteam. Ebenso hilft die Softwarearchitektur den Entwicklern bei der Einordnung der von ihnen zu erstellenden Komponenten in Bezug auf das Gesamtsystem. Ebenso dem Anwender hilft die Darstellung der Softwarearchitektur beim Verständnis für das zu erstellende Softwaresystem. Entsprechend dem zuvor Gesagten ist auch die Dokumentation der Softwarearchitektur ein entscheidendes Qualitätskriterium innerhalb eines Softwareprojektes sowie bei der Beurteilung eines Softwareproduktes. Die Architektur einer Software unterstützt durch ihre Zerlegung der Software in einzelne Bausteine auch bei Aufbau einer Projektstruktur für die Entwicklung der Software indem Bausteine Entwicklerteams oder auch externen Lieferanten zugewiesen werden können. Im Umkehrschluss bedeutet dieses jedoch auch, dass die für die Realisierung der Software geplante Struktur durchaus auch Einfluss auf die Architektur einer Software haben kann, da bei parallel arbeitenden Entwicklerteams die technologischen Abhängigkeiten der von ihnen zu erstellenden Softwaremodule möglichst gering und klar definiert sein sollten. Eine direkte Entkopplung von Architektur und Projektstruktur kann beispielsweise zu ungeplanten und wiederkehrenden Abstimmungsaufwänden sowie Problemen bei Tests der einzelnen Module führen.

Eine Softwarearchitektur enthält normalerweise unterschiedliche Sichten, da die Personen, abhängig von ihren Rollen im Entwicklungsprozess, unterschiedlich auf die Architektur einer Software blicken werden. Zum besseren Verständnis sei als Analogie die CAD Zeichnung einer intralogistischen Förder- und Lageranlage genannt. Ebenso wie der Bauingenieur anders als der Konstrukteur der Förderer auf diese CAD Zeichnung blickt, haben die Softwareentwickler einen unterschiedlichen Blick auf die Architektur einer Software als der Projektmanager oder die Anwender. Entsprechend ist beim Aufbau und Darstellung der Softwarearchitektur zu bedenken, dass die Bedürfnisse der verschiedenen Zielgruppen in Form von Informationsgehalt, Detaillierungsgrad und Darstellungsform (z. B. Beschreibungssprache) berücksichtigt werden. In keinem Fall ist das Ergebnis des Architekturdesigns einer Software ein Dokument, welches ausschließlich von Entwicklern für Entwickler erstellt wurde.

Abhängig von der Komplexität kann die Architektur einer Software auch in mehrere Ebenen aufgebaut sein (siehe Abb. 4.1). Die unterste Ebene beinhaltet dabei in der Regel das Feindesign und geht direkt in die Implementierung der Software über.

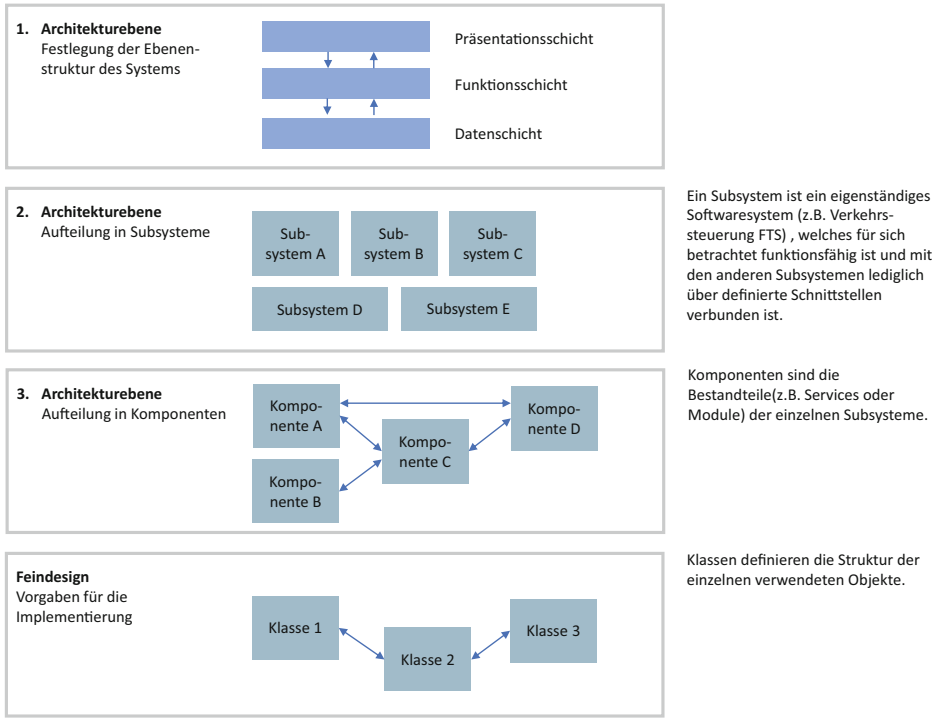


Abb. 4.1 Beispiele für die verschiedenen Ebenen der Entwicklung einer Softwarearchitektur. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

Generell gilt, dass die verschiedenen Architekturmuster nicht genormt oder standardisiert sind und auch gemischt eingesetzt werden können. Im Folgenden wird näher auf einige gängige Architekturmuster eingegangen.

Aufgrund der Bedeutung der Softwarearchitektur ist es entscheidend, den Entwurf dieser Architektur als bewusste Entscheidung im Rahmen eines Designprozesses und nicht Schritt für Schritt im Rahmen der fortschreitenden Entwicklung zu verstehen.

4.1 Darstellung gängiger Architekturmuster

Als Architekturmuster (engl. architectural pattern) bezeichnet man die verschiedenen Arten von Softwarearchitekturen. Im Folgenden sollen einige der gängigen Architekturmuster beschrieben werden.

Die Anwendung dieser Architekturmuster unterliegt wenig Vorgaben und in der Praxis kommen regelmäßig Mischformen von verschiedenen Architekturmustern sowie Teile von Architekturmustern vor. Dieses ist vollkommen zulässig, da es darum geht, eine

Softwarearchitektur zu definieren, welche die Anforderungen an ein Softwaresystem erfüllt. Im Umkehrschluss bedeutet dieses jedoch auch, dass nicht bei jeder Anwendung eines Architekturmusters alle generellen Eigenschaften dieser Architekturmuster automatisch auch für das zu entwickelnde oder zu betrachtende Softwaresystem gelten.

4.1.1 Mehrschichtarchitekturen

Mehrschichtarchitekturen sind eines der gängigsten Architekturmuster. Unterschieden wird zwischen sogenannten Zweischicht- (siehe Abb. 4.2) und Dreischichtarchitekturen (siehe Abb. 4.3). Bei diesem Architekturprinzip werden unterschiedliche Bereiche des Softwaresystems verschiedenen Schichten zugeordnet. Mehrschichtarchitekturen bieten den Vorteil einer Reduktion der in einem System vorhandenen Komplexität. Durch die Trennung in Schichten bietet sich auch die Möglichkeit einzelne Schichten unabhängig voneinander auszutauschen. Ein Beispiel hierfür könnte der Austausch der Präsentationsschicht eines Softwaresystems sein, um das Softwaresystem auch über eine Weboberfläche oder mobile Geräte nutzen zu können. In diesem Beispiel blieben die unterhalb der Präsentationsschicht vorhandenen Schichten weitgehend bis vollständig unberührt von dem Austausch der Präsentationsebene.

Ein häufig bestehendes Problem bei Mehrschichtarchitekturen sind Laufzeitprobleme aufgrund der zwischen den Schichten auszutauschenden Daten. Dieses Problem zeigt sich häufig beim Aufbau beziehungsweise der Anzeige von Listen, welche aus einer größeren Anzahl von Datensätzen bestehen, da dann diese größeren Datenmengen gegebenenfalls über mehrere Schichten hinweg zu transportieren sind. Um dieses Problem zu lösen wird die ansonsten strikte Trennung zwischen den Schichten aufgeweicht, indem beispielsweise

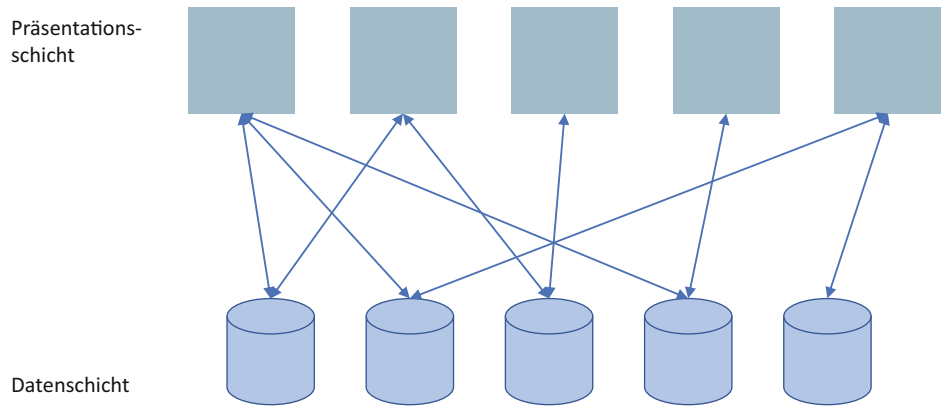


Abb. 4.2 Beispiel für eine Zweischichtarchitektur. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

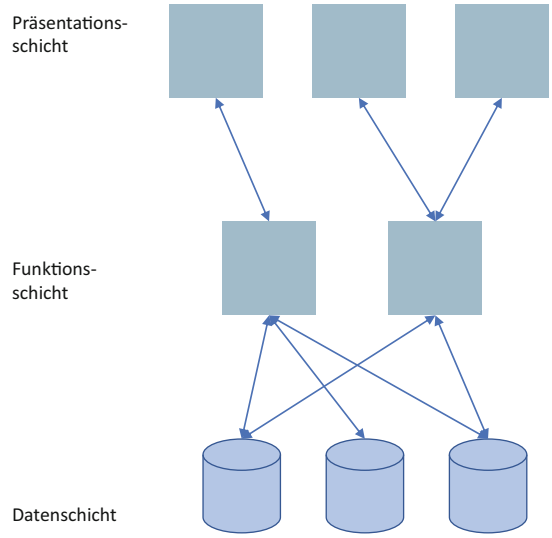


Abb. 4.3 Beispiel für eine Dreischichtarchitektur. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

aus der Präsentationsschicht ein direkter lesender Zugriff auf die Datenbank zugelassen wird, welcher bei strikter Auslegung der Mehrschichtarchitektur über eine der Zwischenebenen erfolgen müsste.

Grundsätzlich gilt für Mehrschichtarchitekturen, dass die einzelnen Schichten verschiedenen Rechnern (Vgl. Client-Server-Architekturen) zugeordnet sein können, aber nicht müssen. Softwaresysteme, welche eine Mehrschichtarchitektur besitzen, können auch auf einem einzelnen Rechner betrieben werden.

Eine Dreischichtarchitektur (siehe Abb. 4.3) weist eine zusätzliche Schicht auf, welche in der Praxis häufig für das zu entwickelnde Softwaresystem zentrale Funktionen enthält. In dieser Funktionsschicht könnte beispielsweise das Bestandsmanagement eines WMS als zentraler Funktionsbaustein angeordnet sein. Das Bestandsmanagement würde in diesem Beispiel Funktionen wie die Reservierung von Beständen als zentralen Service anderen Bausteinen der nächsthöheren Schicht zur Verfügung stellen. Ein Vorteil, welcher allen Mehrschichtarchitekturen gemeinsam ist, ist die Skalierung, da die Bausteine der verschiedenen Ebenen unterschiedlichen Rechnern zugewiesen werden können.

Der Entwurf und das Feindesign derartiger verteilter Systeme stellt eine besondere Herausforderung dar. So ist darauf zu achten, dass bei der Definition der Funktionen der unteren Schicht (Zweischichtarchitektur) beziehungsweise Schichten (Dreischichtarchitektur) die Möglichkeit von Deadlocks[1] ausgeschlossen wird. Aus diesem Grund gilt das Prinzip, dass Funktionen einer Schicht jeweils nur aus der nächsthöheren Schicht aufgerufen werden dürfen. In der Praxis fällt die Einhaltung dieses Prinzips jedoch nicht immer leicht, sodass

[1] Verklemmung oder gegenseitige Blockade von Prozessen, bei der jeder Prozess auf ein Ereignis wartet, welches nur von einem der anderen blockierten Prozesse ausgelöst werden kann.

auch fallweise hiervon abgewichen wird, wodurch wiederum die Gefahr eines möglichen Deadlocks entsteht.

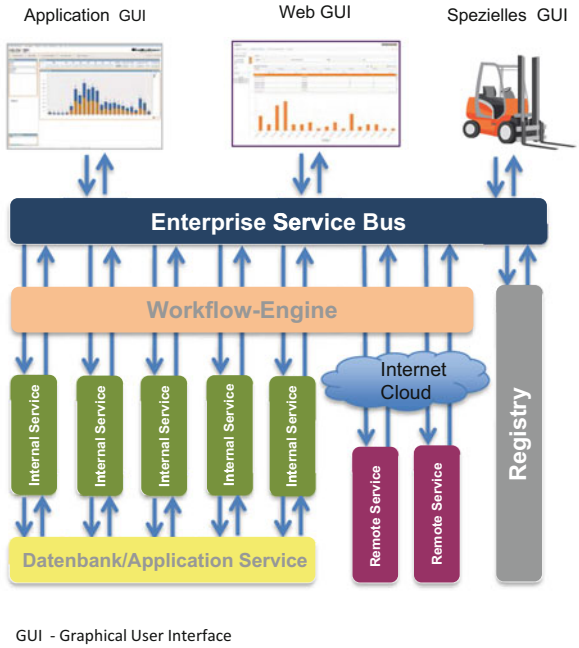
4.1.2 Serviceorientierte Architekturen

Serviceorientierte Architekturen (SOA) orientieren sich stark an Geschäftsprozessen und werden daher oft für die Entwicklung von Unternehmensanwendungen verwendet. Der Grundgedanke ist, dass einfachere Dienste (Services) zu komplexeren Geschäftsprozessen kombiniert werden können. Dabei ist das Wort „einfach“ nicht zwingend auf die Komplexität der Softwareimplementierung, sondern auf die Funktionalität des Service, bezogen auf den Geschäftsprozess, zu verstehen. Die Kombination einzelner Services zu komplexeren Prozessen wird im Kontext der serviceorientierten Architekturen als Orchestrierung bezeichnet. Diese Orchestrierung kann unterschiedlich, beispielsweise unter Nutzung eines sogenannten Workflow-Management, erfolgen. Ein wesentlicher Vorteil dieses Architekturmusters ist die Wiederverwendbarkeit der einzelnen Services sowie die Flexibilität, welche die Orchestrierung bietet. Serviceorientierte Architekturen werden zu den verteilten Systemen gezählt, da die einzelnen Services gegebenenfalls auch auf unterschiedlichen Rechnern ablaufen können. Ebenfalls cloud-basierte Services (Internet) können in die Architektur eingebunden werden. Die Services können in einem auf einer serviceorientierten Architektur basierenden Softwaresystem auch zur Laufzeit hinzugefügt, ausgetauscht oder entfernt werden. Die Verwaltung der im System verfügbaren Services erfolgt typischerweise über eine sogenannte Registry[2]. Dort meldet sich jeder Service an, wird registriert und steht anschließend systemweit zum Aufruf zur Verfügung. Eine andere Komponente (z. B. Workflow-Engine), welche einen Service nutzen will, fragt über die Registry nach, ob dieser Service verfügbar ist und über welchen Kommunikationsweg (z. B. Portnummer des Enterprise-Service-Bus) der Service erreichbar ist.

Die Kommunikation der einzelnen Komponenten (Dienstanbieter und Dienstempfänger) kann im einfachsten Fall über Punkt-zu-Punkt-Verbindungen über verschiedene Netzwerkprotokolle (z. B. CORBA, REST, JSON, o. ä.) erfolgen. Um die volle Flexibilität einer serviceorientierten Architektur nutzen zu können, kommen vielfach auch erweiterbare und protokollunabhängige Kommunikationsverfahren zum Einsatz. In diesem Fall wird häufig von einem Enterprise Service Bus gesprochen. Es haben sich verschiedene Standards für derartige Enterprise-Service-Bus-Konzepte am Markt etabliert, welche zum Teil frei verfügbar (z. B. OpenESB) oder anbieterspezifisch (z. B. SAP NetWeaver) sind.

Durch die Integration einer Workflow-Engine erhöht sich die Flexibilität eines auf einer serviceorientierten Architektur basierenden Softwaresystems (siehe Abb. 4.4). Eine Workflow-Engine ist ein Laufzeitsystem, welches zuvor definierte Arbeitsabfolgen (Workflows) interpretiert und ausführt. Diese Arbeitsabfolgen können Abfolgen einzelner Services sein, welche innerhalb eines auf einer serviceorientierten Architektur basierenden

[2] Registry – englisch für Registratur oder Registrierung.



Unterschiedliche Ausprägungen der Präsentationsschicht, welche verschiedene Wege darstellen, über die der Anwender das System nutzen kann. Beim speziellen GUI könnte es sich bspw. um eine Bedienoberfläche handeln, welche in das Cockpit eines Gabelstaplers integriert ist.

Abb. 4.4 Beispiel für eine serviceorientierte Architektur. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

Softwaresystems verfügbar sind. Für die Definition dieser Workflows stehen standardisierte Beschreibungssprachen (z. B. BPMN – business process model notation; siehe Abb. 4.5) zur Verfügung, welche zum Teil grafisch aufgebaut sind und allgemein verwendet werden.

Der Flexibilität dieser durch die Nutzung einer integrierten Workflow-Engine stehen Einschränkungen in der echtzeitnahen Verarbeitung von Ereignissen gegenüber. Dieses könnte zum Beispiel dazu führen, dass für die Entwicklung eines WMS dieses Architekturmuster verwendet wird, die direkte Materialflusssteuerung jedoch als separates Modul ausgelagert wird.

4.1.3 Microservicearchitekturen

Ein weiteres Architekturmuster, welches in letzter Zeit immer häufiger Verwendung findet, sind die sogenannten Microservices. Microservicearchitekturen folgen einem sehr stringenten Ansatz der Modularisierung. Der Gedanke der Modularisierung liegt fast allen Architekturmustern zugrunde. Das Besondere an Microservicearchitekturen ist die Unabhängigkeit der Module (Microservices) sowie eine lediglich lose Kopplung der Microservices untereinander. In Gegensatz hierzu gibt es beispielsweise in serviceorientierten Architekturen

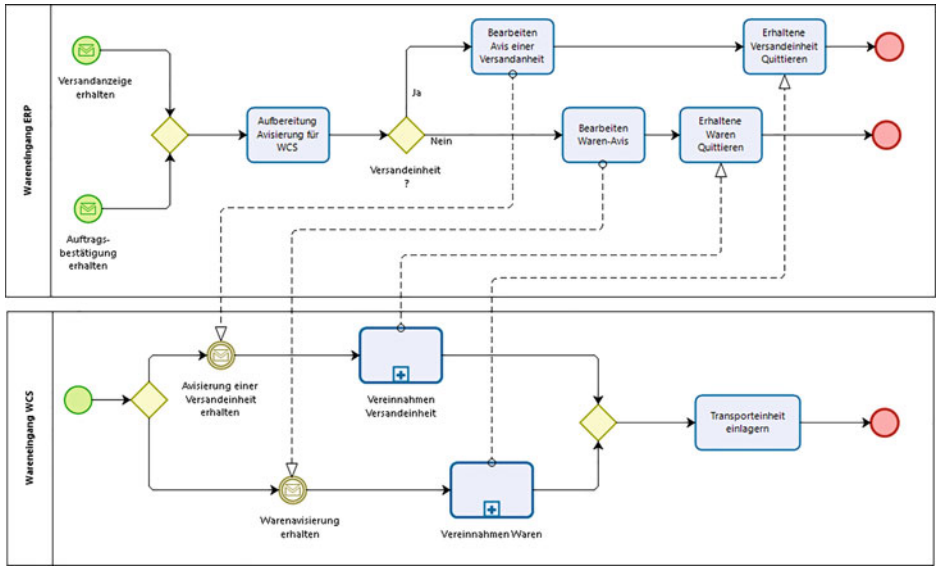


Abb. 4.5 Beispiel eines in BPMN beschriebenen Geschäftsprozess. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

einige zentrale Komponenten wie die Registry, die Orchestrierung oder gegebenenfalls auch den Enterprise Service Bus. Durch diese zentralen Komponenten entstehen technologische Abhängigkeiten sowie durch erforderliche Abstimmungen Abhängigkeiten im Entwicklungsprozess. Entsprechend ist das Gesamtsystem nicht funktionsfähig, wenn eine dieser zentralen Komponenten (z. B. der Enterprise-Service-Bus) nicht verfügbar ist. Bei Microservicearchitekturen werden technologische Abhängigkeiten durch zum Beispiel Verwendung entsprechender Kommunikationstechnologien vermieden. Ferner wird versucht durch entsprechende Designgrundsätze, welche im Folgenden erläutert werden, Abhängigkeiten im Entwicklungsprozess zu vermeiden.

Die Ziele, welche mit dem Einsatz von Microservicearchitekturen verfolgt werden sind:

- Kurzzyklische/kontinuierliche Software-Deployments[3],
- Hohe Systemverfügbarkeit,
- Skalierbarkeit
- Hohe Effizienz in der Softwareentwicklung sowie
- Risikovermeidung in der Entwicklung, Weiterentwicklung und Wartung.

[3]Mit Deployment wird der Prozess der Zurverfügungstellung, Installation und Konfiguration neuer Versionen eines Softwaresystems bezeichnet.

Microservices sind eigenständige Module eines Softwaresystems, welche klar abgegrenzte Aufgaben übernehmen, untereinander über Schnittstellen Daten austauschen und zusammenarbeiten, um die Gesamtfunktionalität des Softwaresystems bereitzustellen. Anders als die Services innerhalb einer serviceorientierten Architektur, welche für sich keine durch den Endanwender abrufbare Funktionalität bereitstellen, stellen Microservices einen durch den Endanwender bereits nutzbaren Funktionsumfang zur Verfügung. In einer serviceorientierten Architektur entsteht diese nutzbare Funktionalität erst über die Orchestrierung mehrerer Services.

In der Designphase sind die Funktionen des Gesamtsystems so den Microservices zuzuweisen, dass die dem Architekturmuster zugrundeliegende funktionale Eigenständigkeit der einzelnen Services sich ergibt. Auf der anderen Seite ist darauf zu achten, dass die einzelnen Services nicht zu groß werden, da anderenfalls der Quellcode der einzelnen Services zu unübersichtlich wird und somit die Forderung nach einer hohen Effizienz in der Entwicklung sowie einer einfachen Weiterentwicklung nicht erfüllt werden kann.

Da die Microservices jeweils einen in sich geschlossenen Funktionsumfang zur Verfügung stellen, ist bei Ausfall eines Microservices das Gesamtsystem weiterhin verfügbar. Lediglich der durch den betroffenen Service bereitgestellte Funktionsumfang steht während der Störungsdauer nicht zur Verfügung.

Jeder Microservice ist eigenständig und kann nach einer eigenen internen Struktur aufgebaut (siehe Abb. 4.6) sein. Gleiche Klassen, welche in mehreren Microservices verwendet werden, werden beispielsweise aufgrund der geforderten Unabhängigkeit in jedem Microservice lokal definiert, um Abhängigkeiten zu vermeiden. Eine Verbindung zwischen den einzelnen Services besteht nur zeitweise über die Schnittstelle. Als Kommunikationsprotokolle kommen in der Regel Webservices wie HTTP[4] und REST[5] zum Einsatz. Wobei HTTP ein Kommunikationsprotokoll und REST ein Paradigma zur Implementierung von Kommunikationsschnittstellen in verteilten Systemen darstellt. Bei REST-basierenden Schnittstellen wird eine Abhängigkeit der Schnittstelle von Zuständen vermieden, um die Unabhängigkeit der einzelnen Komponenten sicherzustellen.

Mit Blick auf die Eigenständigkeit erhält jeder Microservice entweder eine eigene Bedienoberfläche oder bringt seine eigenen Funktion in ein gemeinsames Framework (GUI Client) zur Laufzeit ein, sodass dem Bediener eine gemeinsame Dialogebene für die Bedienung zur Verfügung steht. Durch diese gemeinsame Bedienoberfläche wird die Forderung nach einer hohen Systemverfügbarkeit nicht in Frage gestellt, da diese als eigene separate Instanz auf jedem Arbeitsplatz-PC verfügbar ist.

Häufig werden die einzelnen Microservices auch separat auf sogenannten virtuellen Maschinen installiert oder in einem Container zusammengefasst. Durch diese Maßnahme wird u. a. der Deployment-Prozess vereinfacht, da bei der Auslieferung der Service inklusive der gesamten Laufzeitumgebung zur Verfügung gestellt und installiert werden kann.

[4] HTTP – Hypertext Transfer Protokoll.

[5] REST – Representional State Transfer. Einsatz hauptsächlich für die Maschine-Maschine Kommunikation.

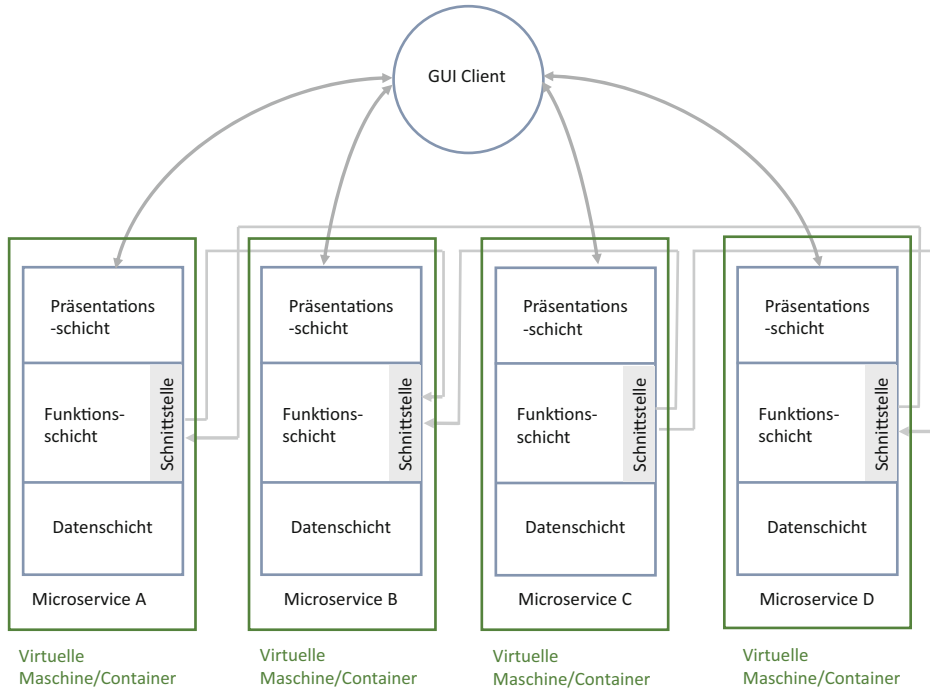


Abb. 4.6 Beispiel für eine Microservice-Architektur. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

Die einzelnen Microservices innerhalb eines Systems können aufgrund der Eigenständigkeit der Microservices gegebenenfalls unter Nutzung unterschiedlicherer Technologie (z. B. Programmiersprache) aufgebaut sein. Hierdurch können zum Beispiel spezielle auf die Aufgabe optimierte Technologien (z. B. Programmbibliotheken) verwendet werden.

Der interne Aufbau eines Microservices wird durch die Schnittstelle von der Außenwelt abgeschirmt. Aufgrund des klaren funktionalen Fokus, welchen jeder Microservice besitzt, besteht die Möglichkeit der Spezialisierung der einzelnen Microservices und die Nutzung von auf die Aufgabe des Service hin optimierten Technologien.

Microservices können zusammen auf einem Rechner oder auf jeweils auch separaten Rechnern betrieben werden, wodurch Microservice-basierte Softwaresysteme hochgradig skalierbar sind. Je nach Auslegung der einzelnen Microservices können von kritischen Services auch mehre Instanzen nebeneinander betrieben werden, wodurch beispielsweise auch Multiprozessor/-kernarchitekturen besser ausgenutzt werden können. Käme es zum Beispiel bei einem WMS zu Verzögerungen bei der Reservierung von Beständen und Erzeugung von Kommissionieraufträgen, könnte gegebenenfalls der für die Bestandsverwaltung zuständige Microservice mehrfach parallel betrieben werden, sodass mehrere Anfragen parallel bearbeitet werden könnten.

Die Skalierbarkeit wird auch durch eine Verwendung von virtuellen Maschinen vereinfacht. Ob nun jeder Microservice einer separaten virtuellen Maschine oder mehrere Microservices zusammen auf einer virtuellen Maschine betrieben werden, hängt von beispielsweise der Implementierung oder auch einer logischen Zusammengehörigkeit ab. Mit Blick auf die Skalierbarkeit können dann, abhängig von der Lastsituation, die virtuellen Maschinen dann den realen Servern innerhalb eines Netzwerkes zugewiesen werden.

Im Zusammenhang mit Microservicearchitekturen wird immer wieder das Gesetz von Conway[6] genannt. Nach Conway gibt es einen Zusammenhang zwischen der Organisation eines Projektes zur Entwicklung eines Softwaresystems und der Architektur des im Rahmen des Projektes zu entwickelnden Softwaresystems. Im Rahmen von streng hierarchisch organisierten Projekten werden demnach auch hierarchisch monolithisch strukturierte Softwaresysteme entstehen. Im Umkehrschluss lässt sich die Frage stellen, wie ein Projekt strukturiert sein muss, um möglichst effizient Software entwickeln zu können? Eine erstrebenswerte Struktur bestünde aus mehreren parallel agierenden Entwicklerteams, zwischen welchen nur ein geringer Abstimmungs- und Kommunikationsbedarf besteht. Notwendige Abstimmungen bedeuten zusätzlichen Aufwand sowie auch Wartezeiten. Eine solche Projektstruktur wird durch eine Microservicearchitektur unterstützt. Jedem Microservice kann ein Entwicklerteam zugeordnet werden, welches für die Entwicklung und Weiterentwicklung dieses Moduls verantwortlich ist und sich dadurch spezialisiert. Auch in diesem Punkt ist wichtig, dass im Feindesign die Funktionen des Softwaresystems so den zu entwickelnden Microservices zugeordnet werden, dass sich eine funktionale Eigenständigkeit der einzelnen Microservices ergibt, wodurch wiederum auch die Unabhängigkeit der Entwicklerteams ermöglicht wird. Dabei ist auch auf die Größe des Microservice zu achten, da es einem Entwicklerteam schwer fallen wird einen unübersichtlich großen Quellcode mit hoher Qualität und Effizienz weiterzuentwickeln. Wird ein solcher Service im Laufe der Entwicklung zu groß, empfiehlt es sich dieses Modul in mehrere Microservices aufzugliedern.

Die Eigenständigkeit und Entkopplung der Entwicklerteams wird ebenso durch die technologische Unabhängigkeit der verschiedenen Microservices unterstützt. Da jeder Service eine eigene technologische Struktur aufweist, welche vom jeweiligen Entwicklerteam festgelegt wird, besteht nur ein geringer Bedarf an technologischen Abstimmungen zwischen den verschiedenen Teams.

In Mehrschicht- und serviceorientierten Architekturen werden den Entwicklerteams in der Regel Komponenten eines Softwaresystems zugewiesen, welche keine funktionale Eigenständigkeit aus Sicht des Anwenders besitzen (siehe Abb. 4.7). Demgegenüber haben die Entwicklerteams in einem Entwicklungsprojekt, welches auf einer Microservicearchitektur basiert, einen Fokus, welcher in sich geschlossenen und vom Anwender abrufbaren Funktionen entspricht. Entsprechend ist auch eine direkte Rückkopplung hinsichtlich Funktionsfähigkeit und auch Nutzwert für den Endanwender gegeben.

[6] Melvin Conway ist ein amerikanischer Informatiker.

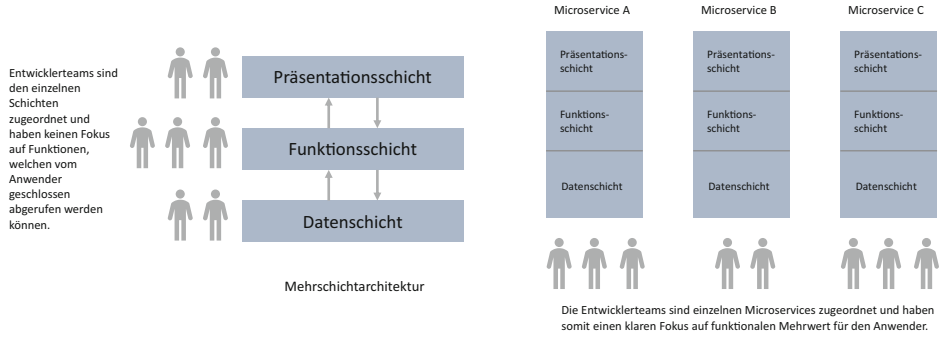


Abb. 4.7 Gegenüberstellung der Zuordnung von Entwicklerteams bei verschiedenen Architekturmustern. (Quelle: Dipl.-Ing. Tech. Informatik Wolfgang Albrecht)

Microservicearchitekturen folgen einem sehr pragmatischen Ansatz. Anders als bei anderen Architekturmustern, welche eine klare technologische Gliederung und somit Wiederverwendbarkeit einzelner Module verfolgen, steht bei einer Microservicearchitektur die Struktur des Entwicklungsprojektes, der Deploymentprozess und der Betrieb im Zentrum. Diesem Ansatz liegt ebenfalls die Erkenntnis zugrunde, dass die anfangs klare Struktur einer Mehrschichtarchitektur oder auch einer serviceorientierten Architektur über den gesamten Lebenszyklus des betreffenden Softwaresystems schrittweise verloren geht. Anders bei einer auf Microservice basierenden Architektur. Eventuelle Nachteile durch zum Beispiel mehrfaches Definieren von Klassen oder anderen Komponenten wird durch eine höhere Effizienz in der Entwicklung ausgeglichen.


BEUMERGROUP
MANCHE DENKEN, SORTIERTECHNIK SEI KOMPLIZIERT.
WIR DENKEN ANDERS.
Wir kennen die besonderen Herausforderungen des täglichen Warenverkehrs. Daher haben wir mit der neuen Familie der BG Sorter innovative und hocheffiziente Sortieranlagen entwickelt, die sich modular an den jeweiligen Aufstellungsort und Bedarf anpassen lassen, intuitiv zu bedienen und für die Wartung leicht zugänglich sind. Sie sortieren Stückgüter unterschiedlichster Größe zuverlässig, effizient und besonders geräuscharm – auch bei hohem Durchsatz.
Mehr Informationen unter beumer.com.
MADE DIFFERENT