
03 Applikationsarchitektur



Agenda

- Einführung
- Microsoft Application Architecture Guide
- Applikationsarchitektur
- Schlüsselkonzepte
- Architecture Design Principles
- Architecture Layers
- Architecture Tiers

Definition: Applikationsarchitektur

- [...] This, to me, is what application architecture is all about—it's about using today's tools and technologies to create as much **business value** as possible whilst keeping one eye on the **requirements and constraints** imposed by the business today, and one eye looking to tomorrow to **maximize ongoing value through scalability, flexibility and maintainability**. – David Hill, Microsoft.
- Freie Übersetzung:
Bei der Applikationsarchitektur geht es darum, aktuelle Werkzeuge und Technologien zu benutzen, um damit so viel Geschäftsnutzen wie möglich zu generieren. Dabei sollen einerseits die Anforderungen und Rahmenbedingungen des aktuellen Geschäfts beachtet und andererseits der zukünftige Mehrwert durch Skalierbarkeit, Flexibilität und Wartbarkeit maximiert werden.

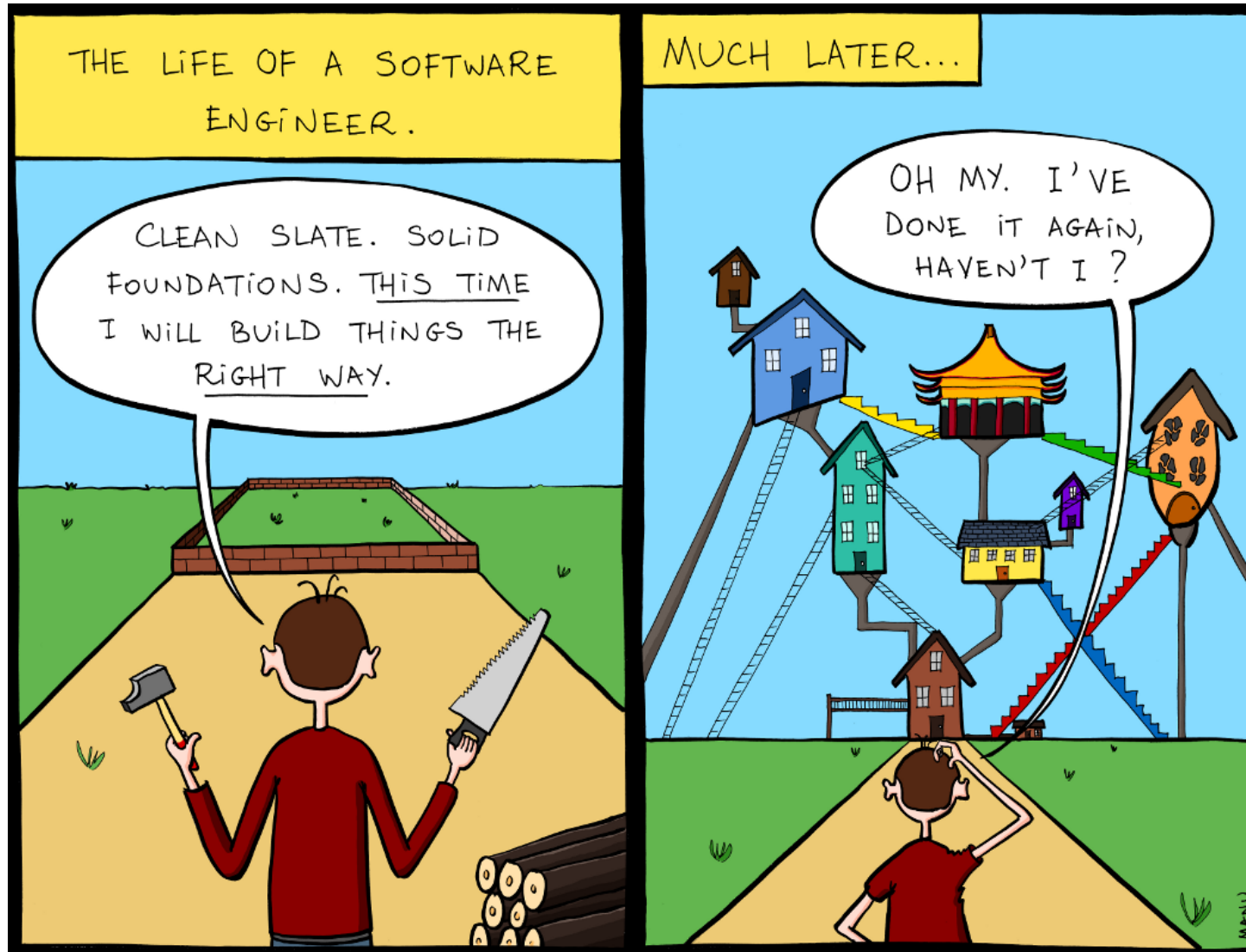
Definition: Applikationsarchitektur

- Software architecture refers to the **high level structures** of a software system, the discipline of creating such structures, and the documentation of these structures. These structures are **needed to reason about the software system**.

https://en.wikipedia.org/wiki/Software_architecture

- Freie Übersetzung:
Softwarearchitektur bezieht sich auf die groben Strukturen eines Softwaresystems, die Disziplin, die solche Strukturen erzeugt und die Dokumentation jener Strukturen. Dies Strukturen werden benötigt, damit man darüber sprechen und argumentieren kann.

Definition: Applikationsarchitektur



Definition: Applikationsarchitektur

- Gute Architektur
 - Minimiert das Geschäftsrisiko
 - Ist flexible genug, um zukünftige Anforderungen zu unterstützen

Microsoft Application Architecture Guide

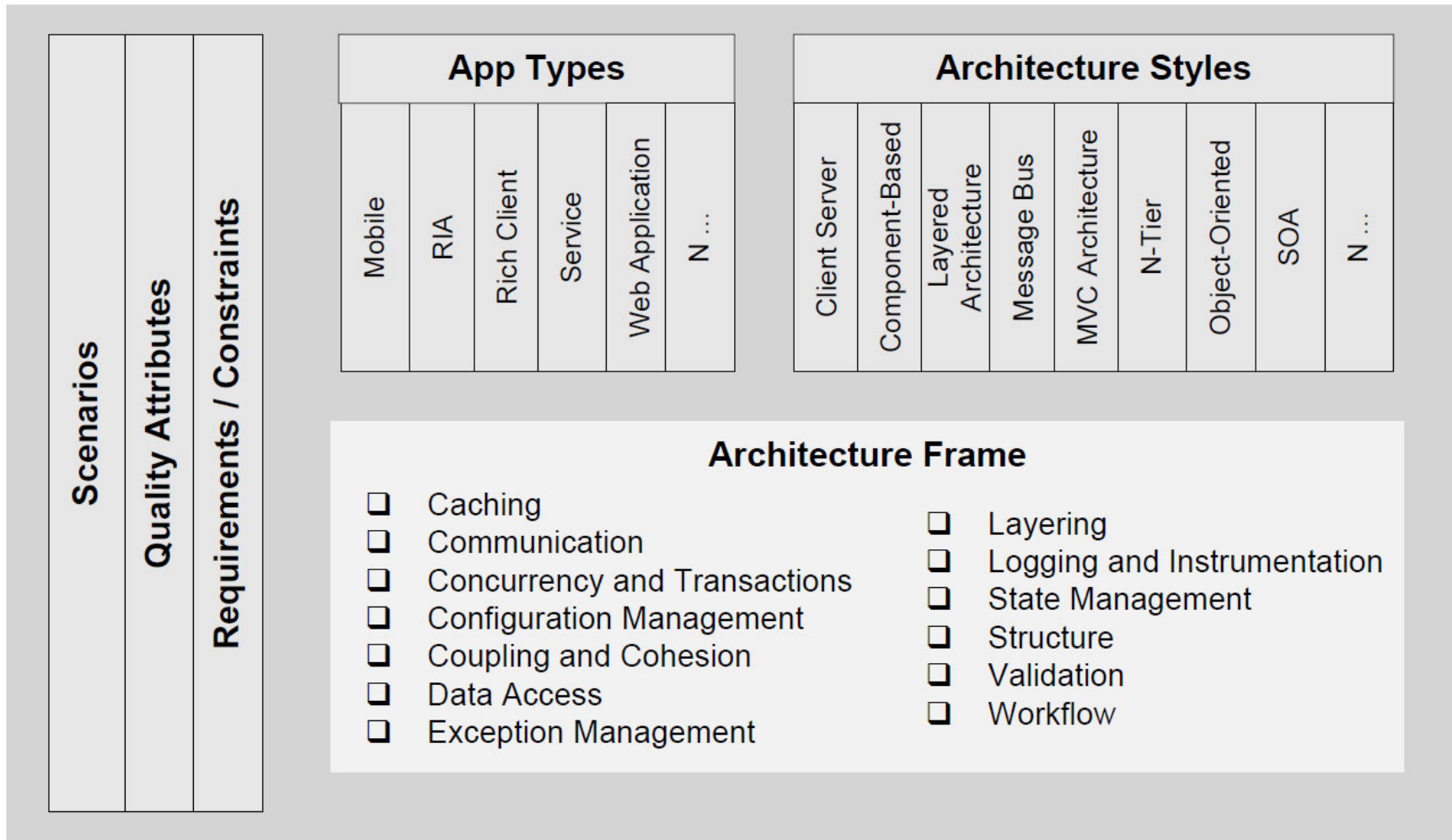
Microsoft Application Architecture Guide 2.0

- Umfassendes Handbuch zu
 - Application Types
 - Architecture Styles
 - Architecture Quality Attributes
 - Architecture Design
 - Deployment Topology
 - Strukturierung von Applikationen
 - Entscheidungsfindung zur Layer-Strategie
- Download
 - <https://msdn.microsoft.com/en-us/library/ff650706.aspx>



Microsofts Application Architecture Guide

Microsoft Architecture Meta-Frame



Schlüsselkonzepte (1/5)

■ Scenarios

- Verbindet Architektur mit realer Welt, z.B. Lösung soll Internet basierend sein => Web Applikation

■ Quality Attributes (ISO/IEC 9126)



- Security
- Performance
- Maintainability
- Availability
- Interoperability
- Scalability
- Testability
- usw.



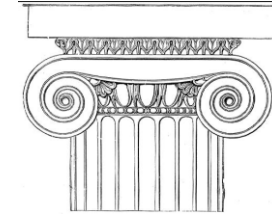
■ Requirements und Constraints

- Anforderungen, welche die Architekturauswahl einschränken

Schlüsselkonzepte (2/5) – Application Types

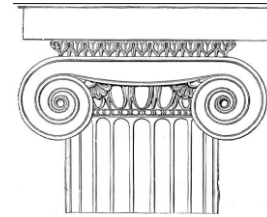
- Mobile Applications (iPhone, Android, Windows Mobile) 
 - Web Applications oder Rich Client Applikationen
- Rich Client Applications (WPF, WinForms)
 - Business-Applikationen mit hohen GUI-Ansprüchen, oft Stand-alone, mit lokalen Ressourcen
- Rich Internet Applications (Silverlight)
 - Business-Apps mit mittleren GUI-Ansprüchen, Multi-Browser- und Multi-OS-Support
 - Werden im Browser in einer Sandbox ausgeführt
- Services (SOAP => WCF, REST => WebAPI) 
 - Ermöglichen lose Kopplung zwischen Client und Server
 - Bieten Geschäftslogik für Clients an. Lokale oder verteilte Lösung.
- Web Applications (ASP.NET, ASP MVC, SPA)
 - Multi-Browser- und -OS-Support, niedrige GUI-Anspr., Ressourcen auf Server

Schlüsselkonzepte (3/5) – Architecture Styles



- **Client – Server**
 - Die Applikation wird aufgeteilt in Client und Server; der Client konsumiert Services, welche auf dem Server gehostet werden
- **Component-Based Architecture**
 - Dekomposition der Architektur in Komponenten mit klar definierten Kommunikationsschnittstellen
- **Layered Architecture**
 - Die Applikation wird in Layers aufgeteilt: Presentation, Business und Data
- **Message Bus**
 - Der Message Bus kann Meldungen (basierend auf einer bekannten Menge von Meldungsformaten) empfangen und versenden
 - Kann mit anderen Systemen kommunizieren, ohne den Empfänger zu kennen, da er aus der Meldung extrahiert wird

Schlüsselkonzepte (4/5) – Architecture Styles



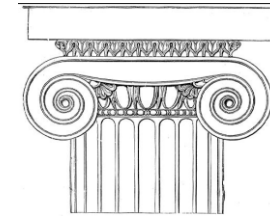
- N-tier / 3-tier
 - Segmentiert die Funktionalität der Applikation in sogenannten Tiers (Schichten).
 - Ähnlich wie die Layer-Architektur, aber jede Schicht ist physikalisch getrennt
- Object-Oriented
 - Architekturstil basiert auf der Aufteilung in Task-Objekte
 - Jedes Objekt enthält Daten und hat Methoden
- Separated Presentation
 - Separieren der Geschäftslogik und Daten vom GUI (MVC-Stil)
- Service-Oriented Architecture (SOA)
 - Applikationen, welche Funktionalität als Services anbieten oder selbst konsumieren. Sie benutzen Contracts und Messages.

Schlüsselkonzepte (5/5) – Architecture Frame

■ Weitere Architektur-«Hot Spots»

- Authentication and Authorization
- Caching and State
- Communication
- Composition
- Concurrency and Transactions
- Configuration Management
- Coupling and Cohesion
- Data Access
- Exception Management
- Logging and Instrumentation
- User Experience
- Validation
- Workflow

■ Details: Siehe Kapitel 3 Architecture and Design Guides



Architecture Design Principles

Key Design Principles/Regeln (1/2)

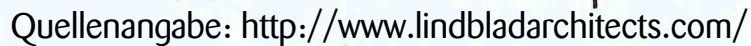
- Separation of concerns
 - Separieren der Funktionalität/Verantwortlichkeit in Einheiten (Packages, Klassen) mit möglichst keiner Überlappung
 - Frage: Ist diese Klasse verantwortlich für ...?
- Single Responsibility Principle (SRP)
 - Jedes Modul oder Komponente sollte nur für eine Funktionalität zuständig sein
- Principle of least knowledge
 - Eine Komponente sollte keine Details einer anderen Komponente oder Objekt wissen (Law of Demeter, LoD)
- Don't Repeat Yourself (DRY)
 - Keine Duplizierung einer spezifischen Funktionalität in anderen Komponenten

Architecture Design Principles

Key Design Principles/Regeln (2/2)

- Avoid doing a big design upfront
 - Falls die Anforderungen an die Applikation noch unklar sind, sollte der Architektur-Design-Effort klein gehalten werden
 - Die Architektur nicht von Beginn an bereits in allen Details designen. → Grosser Aufwand, um das Design konsistent zu halten bei Änderungen während des Projektes
- Prefer composition over inheritance
 - Wann immer möglich, sollte Komposition statt Vererbung angewandt werden, da Vererbung in eine stärkere Kopplung zwischen den Parent- und Child-Klassen resultiert. Dies kann zur Limitierung der Wiederverwendbarkeit der Klassen führen.

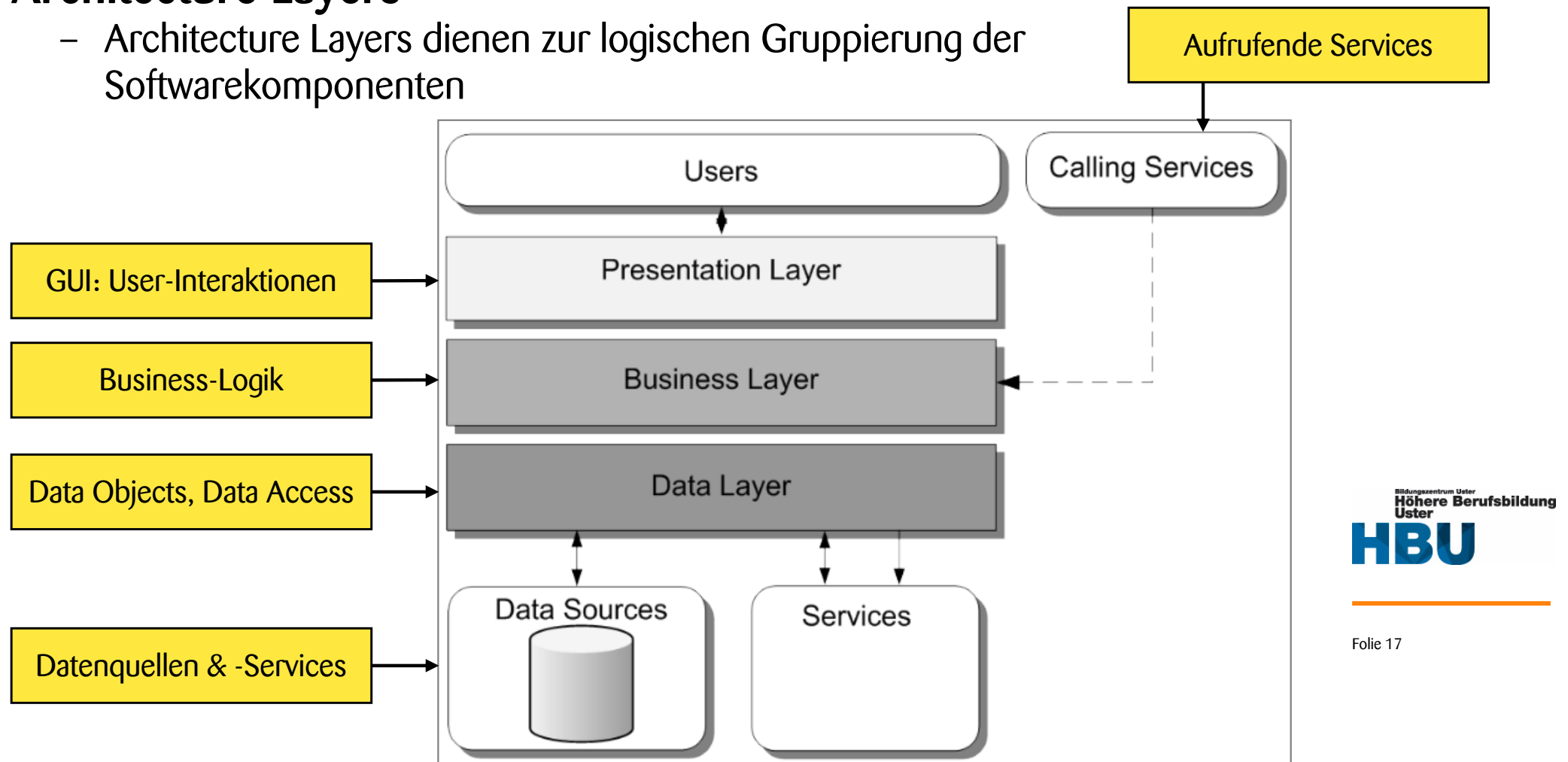
Layers am Beispiel einer Stadtarchitektur/-planung



Architecture Layers

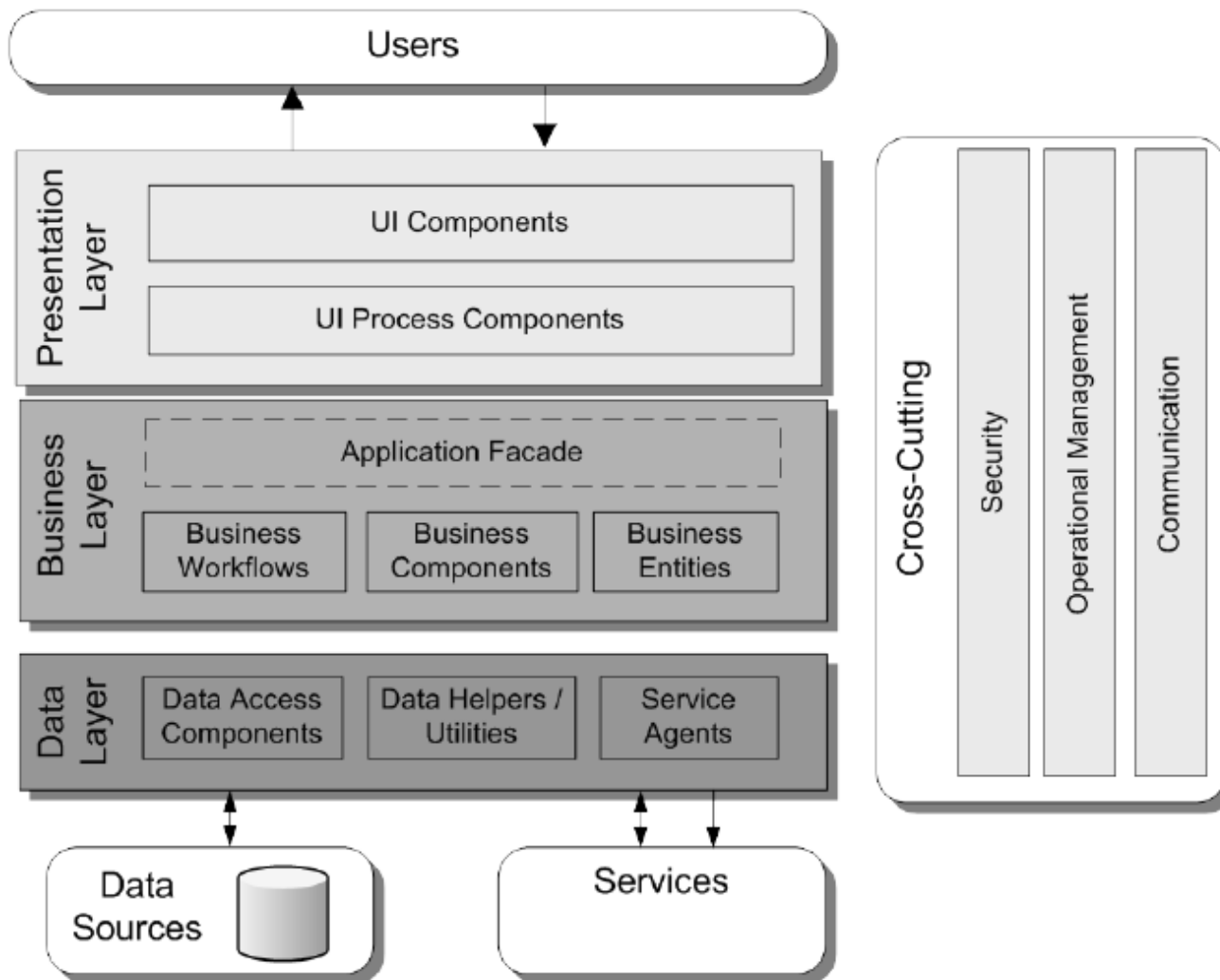
Architecture Layers

- Architecture Layers dienen zur logischen Gruppierung der Softwarekomponenten



Architecture Layers

Referenzarchitektur für .NET-Applikationen



Architecture Layers

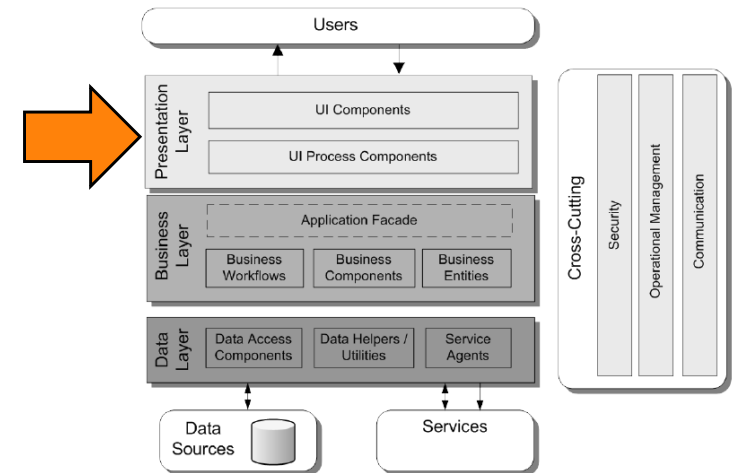
Presentation Layer

■ UI Components

- Komponenten für die Benutzerinteraktionen
- Z.B. Console, WinForms, WPF, GUI-Controls
- Formatierte Ausgabe der Daten und Display Rendering
- Validierung der Benutzereingaben

■ UI Process Components

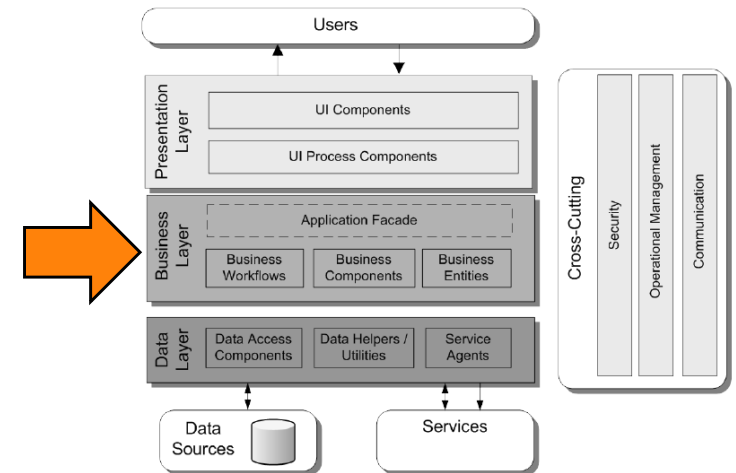
- Manchmal ist es sinnvoll die Benutzerinteraktionen mit einem Prozess zu kontrollieren/orchestrieren
- UI-Prozessklassen von den GUI-Klassen separieren
- State Management separieren
- Erlauben das Benutzen der Interaktionen von anderen User Interfaces



Architecture Layers

Business Layer – Core Functionality

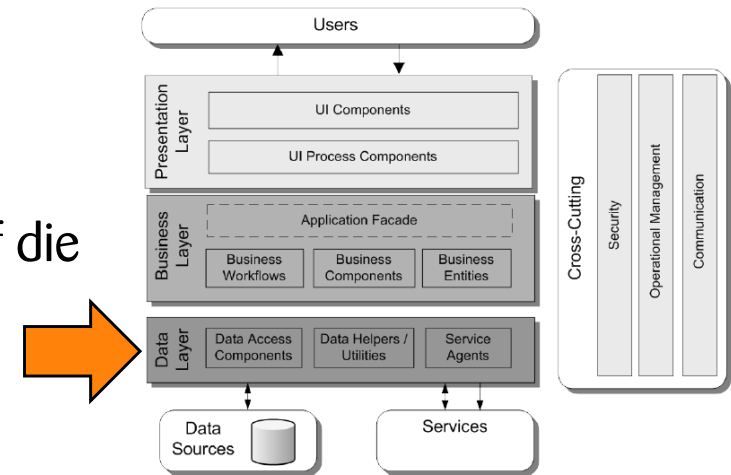
- **Application Façade (Optional)**
 - Kombiniert mehrere Business Operations in eine Single Operation mit Messages
 - Sinnvoll bei verteilter n-tier Architektur (Optimierung der Kommunikation durch eine Methode)
- **Business Components**
 - Representiert die Business-Logik/-Rules/-Tasks
- **Business Workflow**
 - Representiert einzelne Arbeitsschritte eines Business Workflow
- **Business Entity Components**
 - Business Entities repräsentieren Datenobjekte in der realen Welt, welche von den Components verwendet werden. Z.B. Datenstrukturen, Objektbäume, Listen, DataSets usw.



Architecture Layers

Data (Access) Layer

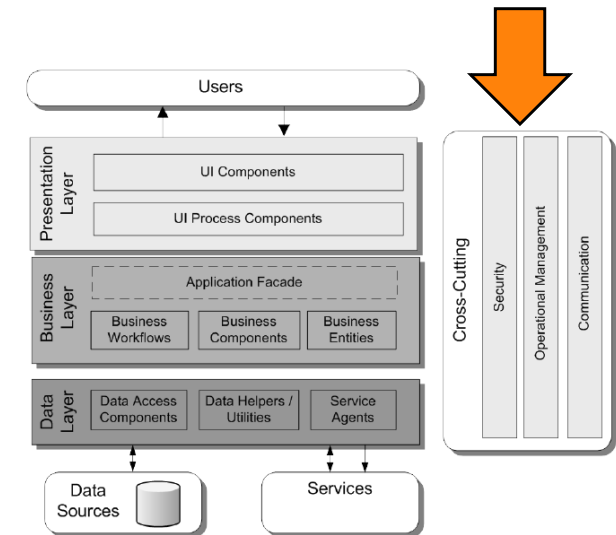
- Data Access Components
 - Diese Komponenten abstrahieren die Zugriffslogik, um auf die Datenquellen zugreifen zu können
- Data Helper & Utility Components
 - Gemeinsam benutzte Helper- und Utility-Komponenten (Klassen), welche oft auch in anderen Applikationen wieder verwendet werden können
- Service Agents
 - Falls eine Business-Komponente einen externen Service konsumiert
 - Implementiert oft Mapping-Klassen, um Daten vom Service-Format ins interne Format zu konvertieren und umgekehrt



Architecture Layers

Cross Cutting

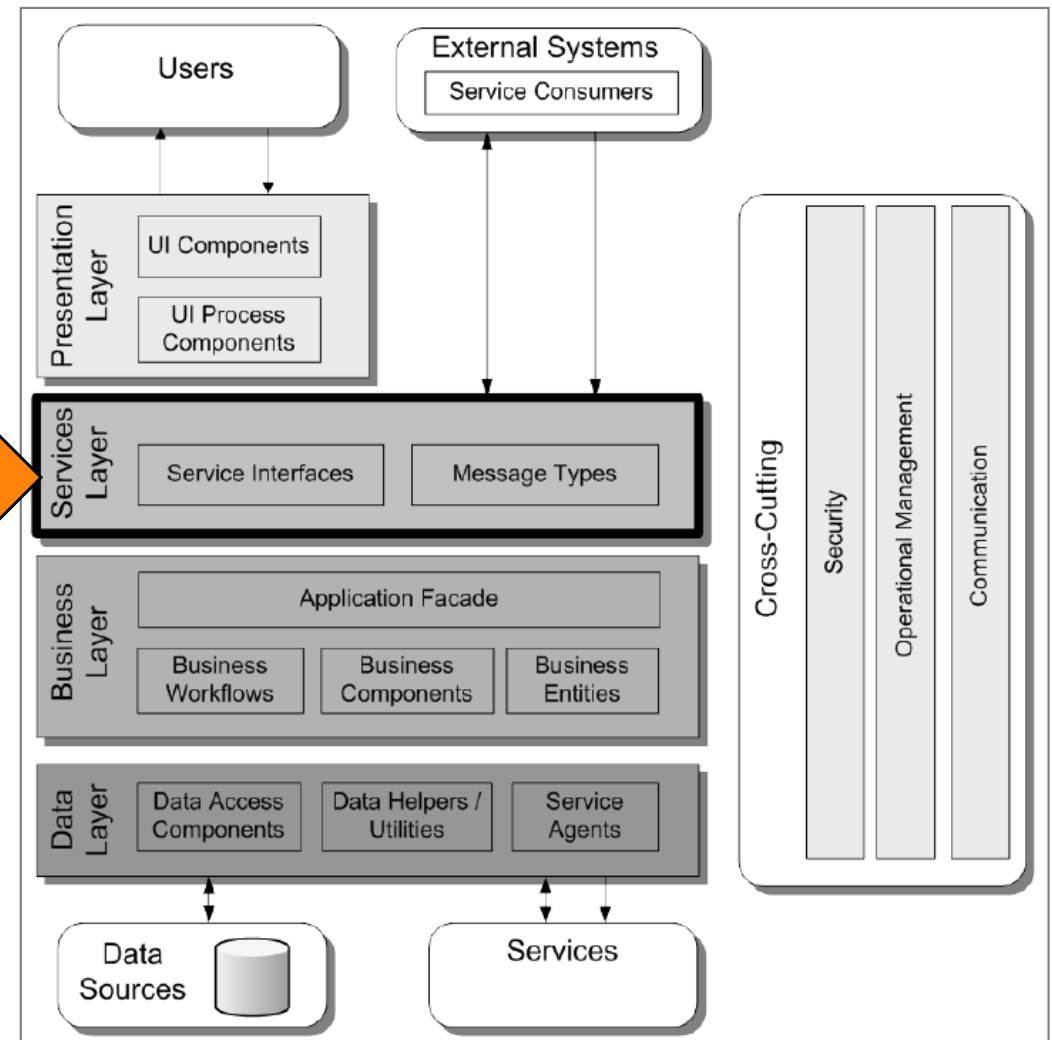
- Enthält Funktionalität, welche von diversen Layern verwendet wird
- **Security**
 - Beinhaltet i.d.R. Komponenten zur Authentifizierung, Authorisierung und Validierung
- **Operational Management**
 - Beinhaltet i.d.R. Komponenten zur Exception Handling, Logging, Performance Counters, Konfiguration und Tracing
- **Communication**
 - Beinhaltet i.d.R. Komponenten zur Kommunikation mit anderen Services und Applikationen



Architecture Layers

Referenzarchitektur mit Service Layer

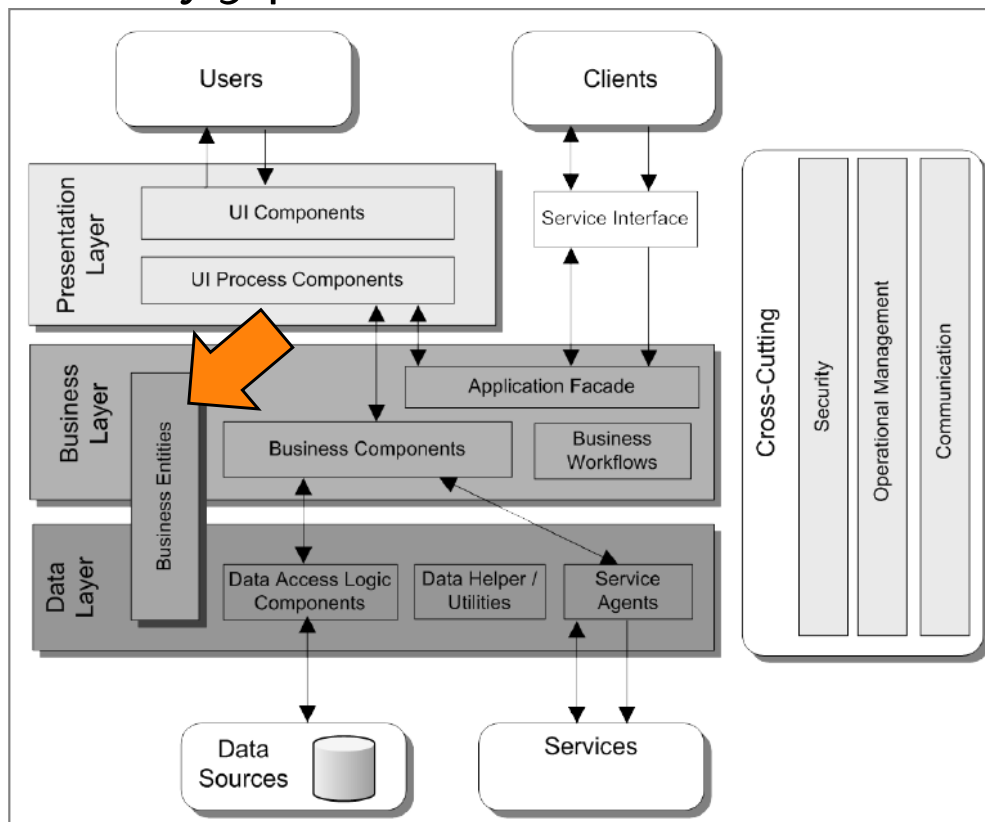
- Durch die Einführung eines Service Layer können Business Operations anderen Applikationen oder externen Systemen zur Verfügung gestellt werden
- **Service Interfaces**
 - Bietet Interfaces mit Business Operationen an (oft als Web Services implementiert)
Input: Request-Meldungen
Output: Response-Meldungen
- **Message Types**
 - Der Datenaustausch wird mit Request- und Response-Meldungen realisiert. D.h. die Objekte und Datenstrukturen werden zu Meldungen transformiert.



Architecture Layers

Referenzarchitektur mit Shared Business Entities

- In vielen Architekturen werden Business-Entitäten im Business Layer und Data Layer gemeinsam benutzt
- Dazu werden die Business-Entitäten in einer separaten Assembly gepackt

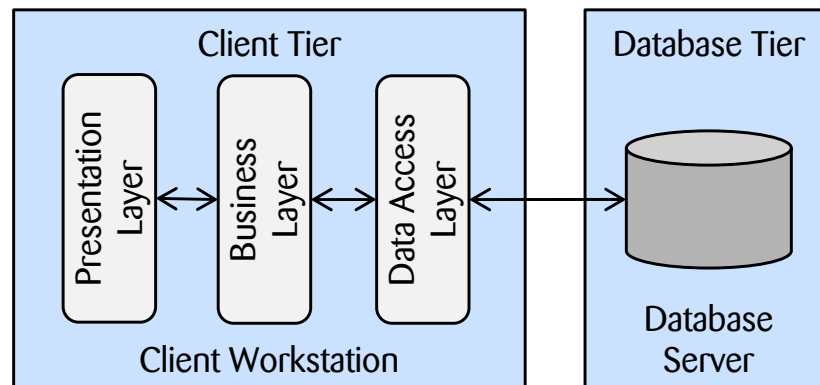


Was sind Tiers?

- «Tiers» repräsentieren eine physikalische Trennung (Schichten) von Presentation, Business, Services und Daten auf verschiedene Computer und Systeme
- Übliche Architecture Designs sind:
 - 2-Tier
 - 3-Tier
 - N-Tier
- Die Kommunikation zwischen den Tiers kann mit einer Firewall oder Reverse Proxy geschützt werden

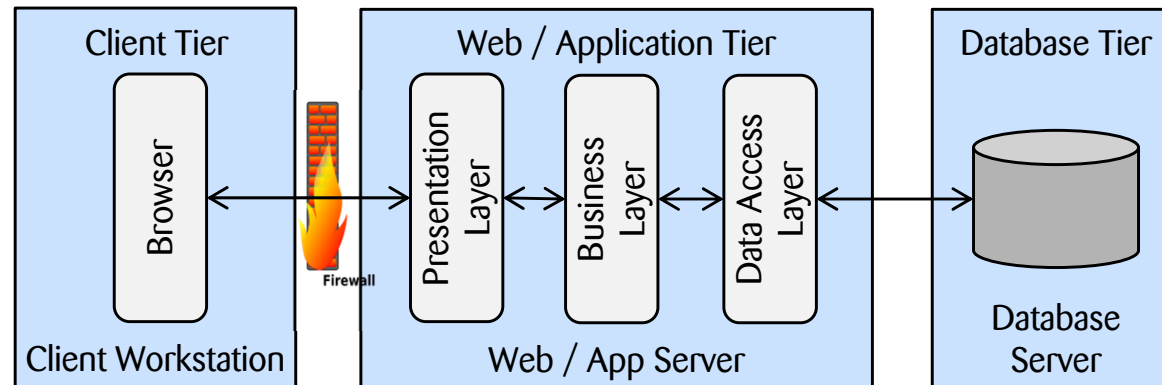
2-Tier-Architekturen

- Eine 2-Tier-Architektur entspricht einer physikalischen Struktur mit den zwei Hauptkomponenten: Client und Server
- Client und Server können auf der gleichen oder auf zwei verschiedenen Servern laufen.
- Beispiel: Fat-Client-Applikation mit separatem, zentralen Datenbankserver



3-Tier-Architekturen

- Bei einer 3-Tier-Architektur ist die Applikation auf einem separaten Server installiert; der Application Server kommuniziert mit dem Datenbankserver
- Dieses Architecture Pattern ist üblich für eine Web-Applikation mit Datenbank



Architecture Tiers

N-Tier-Architekturen

- Die N-Tier-Architektur trennt physikalisch den Presentation Layer von Business und Data Access Layer. Oft wird dies aus Security- oder Skalierbarkeit-Gründen gemacht.

