

# Softwarearchitektur – Was ist das?

Aus der META-LEVEL Vortragsreihe  
„IT-Knowhow aus erster Hand“





- **Was ist Software-Architektur?**
- **Warum braucht man Software-Architektur?**
- **Was leistet (gute) Software-Architektur?**
- **Was tut ein Software-Architekt?**
- **Wie und wann werden Software-Architekturen erstellt?**
- **Beispiele für Architekturen**

## Auszug aus Wikipedia:

**„Der Begriff Architektur ist nicht eindeutig zu definieren... Jede nähere Definition ist nur im Kontext bestimmter Debatten um Inhalt, Aufgabe und Bedeutung der Architektur verständlich. Ähnlich wie in der Kunst ist es in der Architektur nicht möglich, eine eindeutige und kurze Begriffsbestimmung zu geben. Jede inhaltliche Bestimmung von Architektur ist kontrovers und im Kern ideologisch geprägt...“**

## Analoges Problem in der IT:

**Definitionsversuch des Software Engineering Institute (SEI) der Carnegie-Mellon Universität des Begriffs „Softwarearchitektur“ ergab über 50 verschiedene Definitionen.**

**Fachliteratur führt noch weitere „Definitionen“ ein.**

**ABER: Softwareentwickler mit gewisser Erfahrung haben intuitiv ein Gefühl dafür, was Softwarearchitektur ist.**

# Herkunft des Wortes



## Aus dem Altgriechischen:

**arché = „Anfang, Ursprung, das Erste, Prinzip“**

**techné = „Kunst, Handwerk“**

**→ Architektur = „erstes Handwerk“**

**→ architékto = „oberster Handwerker; Baukünstler; Baumeister“**

**Architektur ist Beschreibung eines Artefakts durch seine Teile und ihr Zusammenspiel.**

**Architektur ist eine Menge grundlegender Entscheidungen, die die Teile und ihr Zusammenspiel betreffen**

**Architektur legt fest:**

- **Struktur des Artefakts („Bauplan“): Welche Komponenten/Subsysteme gibt es?**
- **Zusammenspiel der Komponenten/Subsysteme („Ablaufplan“)**

**Das nach diesen Plänen konstruierte Artefakt ist der bestmögliche Kompromiss zwischen allen Anforderungen aller Personen, die ein Interesse am Artefakt haben und den herrschenden Randbedingungen (Umwelt, Budget, Qualifikation der am Konstruktionsprozess beteiligten, ...).**

## Beispiel aus dem Alltag: Architektur von Gebäuden



### Beobachtungen beim Bau von Gebäuden:

- **Die Art und Umfang der Planung hängt von der Art des zu erstellenden Gebäudes ab**
- **Die Planungen zu Gebäuden gleicher Art können sich durchaus unterscheiden.**
- **Je größer das Gebäude ist, um so wichtiger ist die Planung**
- **Das geplante Ergebnis muss mit allen beteiligten Parteien abgestimmt werden**
- **Schlecht geplante Gebäude führen zu höheren Kosten bei Bau, Nutzung und Instandhaltung und können ggf. sogar nicht wie vom Bauherrn beabsichtigt genutzt werden.**
- **Es gibt verschiedene Arten von Plänen für Gebäude (Grundriss, Raumplan,...)**

- **Architektur eines Artefakts wird von seiner Art und seiner Komplexität beeinflusst**
- **Ähnliche Artefakte können unterschiedliche Architekturen haben**
- **Je komplexer das Artefakt, desto wichtiger ist eine geeignete Architektur**
- **Die Architektur muss an den sich oft widersprechenden Anforderungen der Projektparteien ausgerichtet werden.**
- **Artefakte mit ungeeigneter Architektur können nicht wie vorgesehen genutzt werden und sind aufwändiger herzustellen und zu pflegen**
- **Jeder an der Herstellung und Nutzung des Artefakts Beteiligte hat / benötigt eine für ihn nützliche Beschreibung des Artefakts**  
→ **Architektursichten**



**Gibt es überhaupt verschiedene Arten von Software? JA:**

- **Betriebssysteme**
- **„Embedded“ Systeme (Steuerungssoftware etc.)**
- **Compiler**
- **Datenbankmanagementsysteme**
- **Spiele**
- **Geschäftsanwendungen („Enterprise Applications“)**
- **Apps für Smartphones**
- **....**

# Typische Eigenschaften von Geschäftsanwendungen



- **persistente Daten**
- **(sehr) umfangreiche Daten**
- **Konkurrierender Zugriff auf die Daten**
- **verschiedene (benutzerabhängige) Sichten auf die Daten**
- **(viele) verschiedene Masken in der Benutzungsoberfläche**
- **Schnittstellen zu anderen (Geschäfts-)Anwendungen**
- **Komplexe (schwer zu vereinheitlichende) Abläufe**

## Typische Software-Art-spezifische Fragestellungen

**(Die Antworten auf die folgenden Fragen sind grundlegende Entwurfsentscheidungen!)**

- **Wie werden konkurrierende Zugriffe auf Daten verwaltet? Pessimistisches oder optimistisches Sperren?**
- **Bei OO-Anwendung mit GUI und RDBMS: Wie kommen die Objekte vom Hauptspeicher in die Datenbank (und zurück)? Wie gelangen die Objektattributwerte in die Steuerelemente und die manipulierten Werte wieder in das Objekt?**

**Wichtig: Diese Fragen sollten für ein Projekt geklärt werden und die Antworten von allen Entwicklern berücksichtigt werden.**

## Ziele einer Software-Architektur



- **Garantiert die Realisierung aller funktionalen Anforderungen in hoher Qualität (Performanz, Verfügbarkeit, Ergonomie, Skalierbarkeit, ...)**
- **Garantiert die innere Qualität der Software (Lesbarkeit, Wiederverwendbarkeit, Erweiterbarkeit)**
- **minimiert die Gesamtkosten des Systems über dessen gesamten(!) Lebenszyklus (-> Änderbarkeit!)**
- **maximiert die Zufriedenheit aller Stakeholder über den Lebenszyklus des Systems**

- **Architekturen bilden Übergang von der Analysephase zur Entwurfs- und Implementierungsphase, da grundlegende Entscheidung schon früh im SE-Prozess getroffen werden müssen.**
- **Architektur ist die gemeinsame Vision von der inneren Struktur der Software für alle Entwickler. Sie ist der Rahmen für Feinentwurf und Programmierung und muss deshalb dokumentiert und an die Designer / Programmierer kommuniziert werden. Zur Dokumentation hat sich die Erstellung verschiedener „Sichten“ etabliert (z.B. „4+1“)**
- **Architektur ist Basis für genaue Projektplanung (Aufgabenverteilung) und für realistische Aufwandsabschätzung**

## Aufgaben eines Softwarearchitekten

### Hauptaufgabe des Architekten:

**Entwurf einer Lösungsstruktur (Klasse von Lösungen) die alle Anforderungen enthält und den bestmöglichen Kompromiss zwischen den (oft widersprüchlichen) Interessen der Stakeholder zu finden (sog. "Transformation" der Anforderungen in die Lösungsdomäne).**

### Hierbei zu beachten laut [Fried]:

- **Komplexität und Änderbarkeit über den Lebenszyklus des bezogenen Systems**
- **Erfüllung der geforderten Qualitätsmerkmale (Ergonomie, Portierbarkeit, Performance, Skalierbarkeit etc.)**

**Teilaufgaben für den Architekten sind Strukturierung der Lösung und Ausrichtung der Lösung an der Aufgabenstellung.**

**„The life of a software architect is a long and rapid succession of suboptimal design decisions taken partly in the dark.“ *Philippe Kruchten***

**„Ein Architekt, der zu Beginn vollständige und konsistente Anforderungen benötigt, mag ein brillanter Entwickler sein – aber er ist kein Architekt“**

***E. Rechtin***

- **Ist in alle Phasen des Softwareentwicklungsprozesses involviert**
- **Berät alle Projektbeteiligten (Projektleiter, Analytiker, Entwickler, Management, Auftraggeber, Qualitätssicherung)**
- **Ist Generalist (Fachdomänenwissen, Top-Level-Designer, beherrscht Technologien, kann implementieren), „Ein-Mann-Armee“**

## Strukturierungsprinzipien zur Architekturerstellung

- **Abstraktion**
- **„Divide et impera“: Aufteilung in beherrschbarere Teile**
- **Separation of Concerns (SoC, „Trennen von Belangen“)**
- **Modularisierung: Erstellung von Modulen mit hoher Kohäsion und schwacher Kopplung**
- **Information Hiding mittels Kapselung**
- **übliche Regeln des Softwareentwurfs wie S.O.L.I.D (Single Responsibility Principle, Open Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle)**

- **Denkprozess, der zur Reduktion des Informationsgehalts zu einem beobachtbaren Phänomen führt**
- **Nur die in der konkreten Situation relevanten Informationen werden erfasst, d.h. irrelevante Information wird vernachlässigt**
- **Ermöglicht die Beherrschung komplexer Phänomene durch Konzentration auf die wesentlichen Eigenschaften**
- **Beispiel: In OOA werden von realen Kunden nur die für die zu erstellende Anwendung relevanten Attribute im Fachobjekt „Kunde“ aufgenommen, d.h. das Fachobjekt ist eine Abstraktion des realen Kunden**



**Aufteilung eines Programms gemäß verschiedener Belange/Aspekte,  
z.B. Trennung von Fachlichkeit („Logik“) und der Technik**

**Beispiele solcher Belange:**

- **Präsentation (Benutzeroberfläche)**
- **Fachlichkeit (Geschäftslogik)**
- **Infrastruktur (Persistenz, Kommunikation und Verteilung, Integration Fremdsysteme, Sicherheit, Fremdbibliotheken, Hardware, ...)**

**Weiteres Beispiel: Trennung von Inhalt und Darstellung von Dokumenten (z.B. bei Webseiten und Content Management Systemen).**

**Modul = Programmteil mit (wohldefinierter) Schnittstelle**

**Kohäsion = Maß für Quellcode, wie gut er „zusammenpasst“ / „zusammengehört“**

- **Modul mit „hoher Kohäsion“ hat wenige, klare Zuständigkeiten.**
- **Modul mit „niedriger Kohäsion“ ist schwer zu verstehen, zu warten, wiederzuverwenden**

**Kopplung = Maß für den Grad der Abhängigkeit eines Moduls von anderen Modulen**

**Erstrebenswert: „schwache“, „lose“ Kopplung**

**Vorteil: einfachere Wiederverwendbarkeit / Austauschbarkeit**

**Nachteil: Ggf. ineffizient**

**Bedeutung: Verbergen von Entwurfsentscheidungen.**

**Vorteil: Abhängigkeiten zwischen verschiedenen Programmteilen werden verringert.**

**Gute Praxis: Verberge alle Entwurfsentscheidungen, die man nachträglich leicht ändern können möchte.**

**Information Hiding wird durch sog. Kapselung realisiert:**

- **Modul erhält eine (stabile) nach außen sichtbare Schnittstelle**
- **Alle Implementierungsdetails dieser Schnittstelle sind nur innerhalb des Moduls sichtbar, d.h. sie sind für andere Module unsichtbar**

## Ausrichtung von Architekturen

**Unter der Ausrichtung (engl. alignment) einer Architektur versteht man laut [Fried] das ständige Anpassen der Architektur an Einflussfaktoren und den beteiligten Parteien (sog. Stakeholder), so dass durch die Architektur eine gut zur Aufgabenstellung passende Lösung entsteht.**

**Während die Strukturierung implizit durch Entwicklerteam erfolgen kann, muss die Ausrichtung immer explizit erfolgen, d.h. aktiv von jemandem betrieben werden.**

**Zu klären:**

- **Wer sind zu berücksichtigende Stakeholder?**
- **Welche Einflussfaktoren gibt es?**

## Typische Stakeholder und ihre Interessen

- **Anwender: optimale Unterstützung des Tagesgeschäfts**
- **Fachabteilung: schnelle Umsetzbarkeit neue Geschäftsideen**
- **Projektleiter: stress- und risikofreies Projekt**
- **Systemverwalter: einfach betreibbares System**
- **Entwickler: Einsatz cooler Tools, Programmiersprachen und Technologien**
- **.....**

## Wie „macht“ der Architekt Ausrichtung?

### Architekt muss

- die Interessen und Anforderungen der Stakeholder abwägen und ausbalancieren
- vorhandene Widersprüche auflösen und Kompromisse finden.

Dazu benötigt der Architekt laut [Fried] folgende Fähigkeiten:

- Kommunikation, Moderation, Konfliktmanagement
- Führungsqualitäten
- Technologiewissen
- Praxiserfahrung mit den verschiedenen Rollen im Softwareentwicklungsprozess

**Wichtig: Glaubwürdigkeit des Architekten bei den Stakeholdern**

## Einflussfaktoren auf Softwarearchitekturen laut Starke



**„Information ist architekturelevant, wenn sie für Zwecke oder Ziele des Kunden notwendig ist.“ [Rechtin]**

- **Organisatorische / politische Faktoren**
- **Technische Faktoren: beeinflussen technisches Umfeld des Systems und dessen Entwicklung**
- **System- / Produktfaktoren**

- **Organisation und Struktur bei Auftraggeber und beim Projektteam: Ansprechpartner, Entscheidungsträger, Kooperationen, ...**
- **Ressourcen (Budget, Zeit, Personal): Festpreis, Zeitplan vs. Funktionsumfang, Releaseplan, ...**
- **Standards: Vorgehensmodell, Qualitätsstandards, Entwicklungswerkzeuge, Abnahme- und Freigabeprozesse ...**
- **Juristische Faktoren: Haftungsfragen, Datenschutz, Nachweispflichten, internationale Rechtsfragen ...**



- **Hard- und Softwareinfrastruktur: Prozessoren, Speicher, Netzwerk, Firewall, Betriebssysteme, Webserver, Middleware ...**
- **Betriebsart: Online / Batch, Verfügbarkeit, Wartungs- und Backupintervalle**
- **Programmiersprachen, Programmiervorgaben**
- **Bibliotheken, Frameworks, Komponenten**
- **Analyse- und Entwurfsmethoden**

- **Benutzerschnittstelle: Art und Stil der Interaktion, Konfigurierbarkeit durch Benutzer, ...**
- **Abhängigkeiten von bestehenden oder entstehenden Systemen**
- **Grad an Flexibilität: Wartbarkeit, Erweiterbarkeit, Anpassbarkeit / Konfigurierbarkeit zur Laufzeit**
- **Betrieb: Installation, Versionwechsel (laufender Betrieb oder offline), Protokollierung, „Service Level Agreements“**
- **Fehlertoleranz, Ausfallsicherheit**
- **Performance: Antwortzeitverhalten, Ressourcenverbrauch**
- **Kosten: Hardware, Softwareerstellung, Lizenzen**
- **Datenumfang**

# Vorgehensweise beim Erstellen von Architekturen



- Klären um welche Art von Software es sich handelt
- Gibt es Referenzarchitekturen?
- Architekturen verwandter Anwendungen anschauen
- Architekturmuster anschauen
- Frameworks anschauen (z.B. Spring, Cocoon, Hibernate, ...)
- Technologien anschauen (.NET, AJAX, ....)

**Laut [POSA1] dienen Architekturmuster als mentale Bausteine beim Architekturentwurf, d.h. ein einzelnes Muster deckt nicht immer den kompletten Architekturentwurf ab, sondern nur einen einzelnen Aspekt**

- **Interaktive Systeme: Model-View-Controller**
- **Vom Chaos zur Struktur: Schichten, Pipes-and-filters, Blackboard**
- **Verteilte Systeme: Broker**
- **Adaptierbare Systeme: Microkernel**

- **Schichten: Teile sind Abstraktionsschichten zuordnenbar**
- **Pipes-and-filters: Für Systeme, die Datenströme bearbeiten. Teile sind hintereinandergeschaltet und bearbeiten den Datenstrom. Beispiel: Compiler**
- **Blackboard: Für Systeme, die in noch nicht vollständig verstanden sind und für die es keine deterministische Lösung gibt. Die Teile organisieren ihre Zusammenarbeit nach definierten Regeln in Abhängigkeit der aktuell zu bearbeitenden Daten. Beispiel: Spracherkennungssoftware**



**Oftmals Anwendung von Separation of concerns zur Trennung von Fachlichkeit und Technik**

**Ideal: Geschäftslogik liegt in einer Form vor, die einfach geändert werden kann (s. Inferenzregelmaschine JBoss Drools)**

**Organisatorische(!) Aufteilung des Quellcodes in die Teile**

- **Präsentation (Benutzeroberfläche)**
- **Fachlichkeit (Geschäftslogik)**
- **Infrastruktur (Persistenz, Kommunikation und Verteilung, Integration Fremdsysteme, Sicherheit, Fremdbibliotheken, Hardware, ...)**

# Model-View-Controller MVC



## Standardarchitektur bzw. Architekturprinzip (Muster) für

- Dialoganwendungen
- Verteilte Anwendungen

## Teile

- Model (Daten)
- View (Darstellung der Daten)
- Controller (Steuerung, Reaktion auf Benutzereingaben)

**Berühmtes MVC-basiertes Framework ist Struts für Java-Webanwendungen.**

## **Kommunikation (bei nicht-verteilter Anwendung):**

- **View kennt Model (Lesen von Daten)**
- **Controller kennt View und Model (Weitergabe von Benutzereingaben über View ins Model)**
- **Model kennt nicht explizit View oder Controller: Verwendung des Beobachtermusters (Model ist Subjekt, View und ggf. Controller sind Beobachter)**

## **Verteilung von Zuständigkeiten:**

- **View ist Benutzeroberfläche**
- **Geschäftslogik kann auf Controller und Model verteilt sein**

- **Aufteilung eines Systems in Teile (sog. „Schichten“) mit fester Zuständigkeit. Jede Schicht bietet über eine Schnittstelle ihre Dienste an.**
- **Die Schichten sind sog. Abstraktionsebenen zugeordnet**
- **Schichten einer Abstraktionsebene benutzen zur Realisierung ihrer Dienste lediglich Schichten der unmittelbar darunterliegenden Abstraktionsebene.**
- **Beispiel: Client-Server-Architektur („2-Schicht-Architektur“), OSI-Schichtenmodell für Netzwerkprotokolle**

## Vor- und Nachteile von Schichtenarchitekturen

- + Nutzer einer Schicht muss nichts über die darunterliegenden Schichten wissen**
- + Minimierung von Abhängigkeiten zwischen den Teilen eines in Schichten aufgeteilten Systems → einfache Kommunikation**
- + Implementierung einer Schicht kann ausgetauscht werden, wenn angebotene Dienste verfügbar bleiben**
  
- Kapselung gewisser Änderungen nur ungenügend, z.B. muss ein neues Datenfeld bei Client-Server-Architektur im Client und in der Datenbank eingebaut werden, d.h. alle Schichten müssen geändert werden**
- Zusätzliche Schichten können Performance beeinträchtigen**
  - „Layer-Bridging“**

## Beispiel Schichtenarchitektur mit MVC (1)



**Gesucht: Architektur für interaktive Geschäftsanwendung**

**Lösung: Verfeinerte MVC-basierte Schichtenarchitektur**

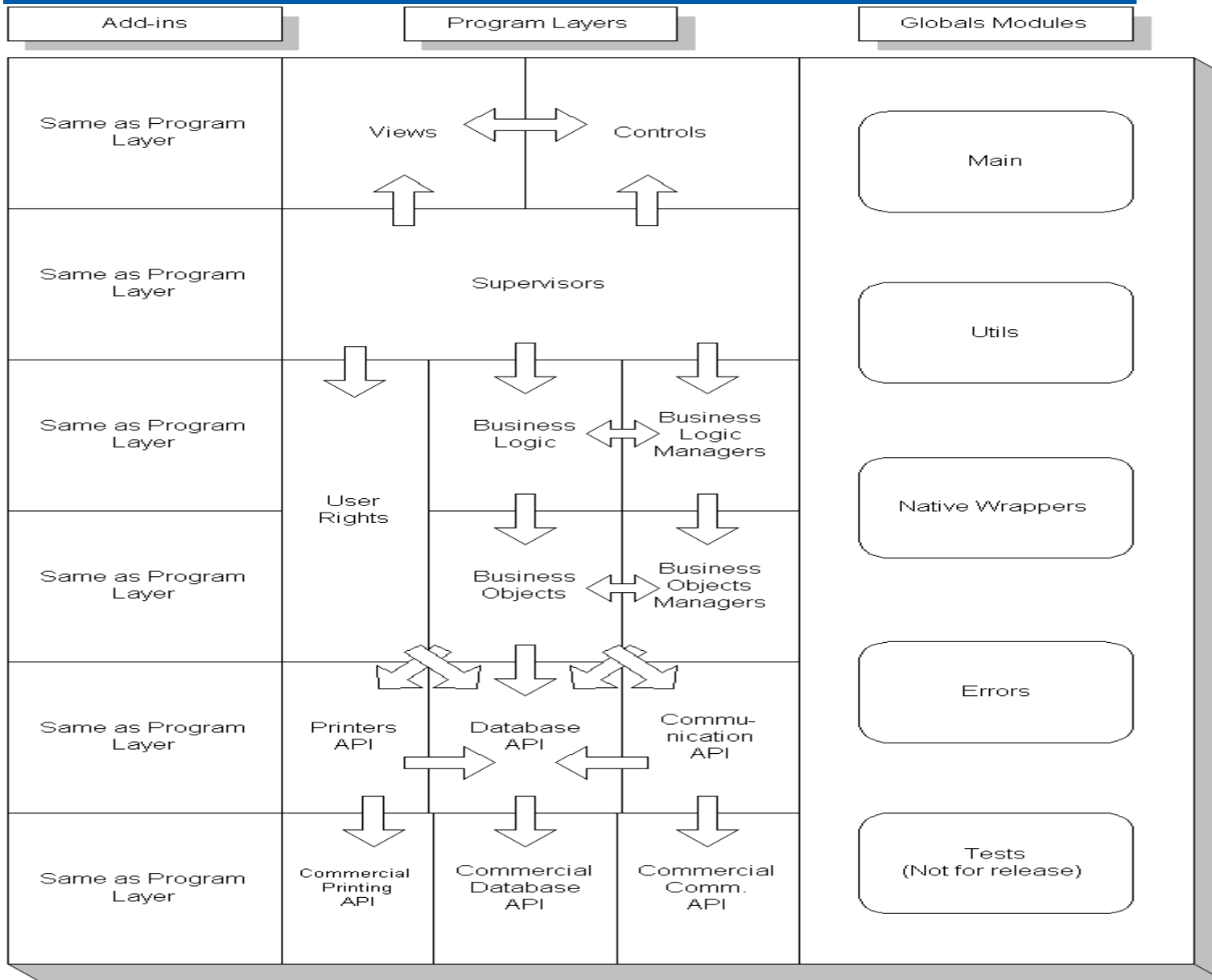
- **Präsentation (View)**
- **Steuerung (Controller, Reaktion auf Benutzereingaben)**
- **Fachlogik**
- **Datenbankzugriff**

**Bemerkung: Zur Entstehung dieser Architektur s. folgende Folie „Projektszenario 1“.**

## Beispiel Schichtenarchitektur mit MVC (2)

### **Aufgaben DB-Schicht:**

- **Unabhängigkeit vom Persistenzmechanismus (Bibliotheken, Datenbankservern) garantieren**
- **Austauschbarkeit der Zugriffstrategie auf Speichermedium (z.B. Verwendung von Stored Procedures statt SELECT,INSERT,...) gewährleisten**
- **Bei RDBMS: OR-Mapping**
- ....



## Beispiel Schichtenarchitektur mit MVC (3)

### Gelöste Probleme:

- **Objektsensitive Controls (statt datenbanksensitive Controls)**
- **Vermeidung der Explosion der Anzahl von Fachobjekten bei Anzeige große Datenmengen in sogenannten Grids (s. Screenshot auf nächster Folie)**
- **Einfluss der Technologie: Mit Delphi VCL waren eigene Unterklassen von Controls nötig. In .NET sind die Controls von Haus aus objektsensitiv, es musste lediglich der Infrastrukturcode (immer wiederkehrende Aufgaben beim Binden von Fachobjekten mit Controls) erstellt werden.**

# Screenshot Maske mit Grid

Artikelverwaltung

Erfassen    Ändern    Löschen

Nummer	Bezeichnung	Mengeneinheit
4711	Ordner A4	Stück
4712	Ordner A5	Stück
0815	Büroklammern	Schachtel

Nummer:

Bezeichnung:

Mengeneinheit:

Speichern    Abbruch

## Beispiel verteilte Schichtenarchitektur (1)

### **Aufgabe: Physische Verteilung logischer Schichten (Layer to Tier)**

#### **Pro**

- **Skalierbarkeit (Lastverteilung)**
- **Ausfallsicherheit (Backupserver)**
- **Sicherheitsaspekte**

#### **Contra**

- **Laufzeiteinbußen**
- **Erhöhter Implementierungsaufwand**
- **Höhere Programmkomplexität**
- **Erschwerte Testbarkeit**

## Beispiel verteilte Schichtenarchitektur (2)



- **Client-Schicht in Web-Browser**
- **Präsentationsschicht auf Webserver (JSP, Servlets)**
- **Geschäftslogikschicht auf Application Server (EJBs)**
- **Datenhaltungsschicht auf Datenbankserver (Oracle)**

## Einfluss physischer Verteilung auf Architektur

### Neue Schichten zur Kapselung von Infrastrukturaspekten (Kommunikation zwischen Tiers) nötig:

- **Schicht über Fachobjekte, damit diese „remotefähig“ sind. Grobkörnige einheitliche Schnittstelle von „Geschäftsdiensten“ anbieten, mit der alle möglichen Clients effizient kommunizieren können (Entwurfsmuster „(Session) Facade“)**
- **Schicht im Client, die Kommunikation mit entfernten Objekten bündelt und den Client vom „Remote“-Aspekt der Fachlogik entkoppelt (Muster „Business Delegate“, „Proxy“)**
- **Generell Einsatz von Lösungen, die Performanceverlust durch entfernte Kommunikation ausgleichen (Muster „Value Object“ und „Value List Handler“)**

## Brauche ich in meinem Projekt einen Architekten?

**Genauer: Muss es eine Rolle „Architekt“ in meinem Projekt geben?**

**Ob diese Rolle von einer oder mehreren Personen wahrgenommen in Voll-/Teilzeit übernommen wird, ist eine sekundäre Fragestellung.**

**Diese Frage ist pauschal nicht beantwortbar.**

**Beim Finden einer Antwort zu berücksichtigende Fakten:**

- **In jedem Projekt müssen architektonische Überlegungen stattfinden**
- **die Architektur muss mindestens die geeignete Strukturierung der Lösung gewährleisten**
- **die Risikofaktoren des Projekts (Zeithorizont, Komplexität, Personenanzahl, Erfahrung aller Beteiligten, Zeit-/Kostendruck, „Politik“)**

**Orientierung bieten evtl. die folgenden Projektszenarien.**

## Projektszenario 1: kleines Entwicklerteam



**Aufgabe: Re-Engineering eines Produkts, d.h. überwiegend Strukturierung**

- **2 erfahrene Entwickler erstellen iterativ die Architektur**
- **sich aus der Architektur ergebender anwendungsunabhängiger Code (Infrastruktur- und Technologiecode) wurde von diesen Entwicklern als Framework realisiert**
- **Framework wurde mit Unit-Tests und einer kleinen Testanwendung getestet**
- **Zwei weitere junge Entwickler ergänzten das Team zur Erstellung des eigentlichen Produkts mit der neuen Architektur. Aufteilung in „Architekturentwickler“ und „Anwendungsentwickler“.**
- **Anwendungsentwickler geben bei Entwicklung des Produkt wichtige Impulse zur Verbesserung des Architekturframeworks durch die Architekturentwickler.**
- **Lerneffekt für junge Entwickler, die nach und mehr Architekturentwicklertätigkeiten übernehmen können**

## Projektszenario 2: „Moses-Architektur“

**Der Architekt erstellt die Architektur ohne sich mit den Stakeholdern abzustimmen. Die Stakeholder (zunächst die Entwickler) kommen mit der Architektur erst nach deren „Fertigstellung“ in Kontakt.**

**Ergebnis: Architektur passt nicht zur Aufgabenstellung, weil keine Ausrichtung stattfand.**

**Konsequenzen:**

- **Architektur hat die Entwickler bei der Erstellung der Software behindert**
- **Unproduktive Wartezeiten der Entwickler bei dringend nötigen Architekturanpassungen**

## Architekturen und Agilität

**Agile Methoden (Scrum, XP) kennen keine Rolle „Architekt“, weil es sich um „Vorabplanung“ handelt, die Agilität hemmt („Big up front design“).**

**Trotzdem ist es auf Grund von vorhandenen Risikofaktoren explizit eine Rolle „Architekt“ zu besetzen.**

**Problem: Viele architekturelevante Anforderungen (Wartbarkeit, Verfügbarkeit, Änderbarkeit, Testbarkeit, ....) lassen sich wegen ihres Querschnittscharakters schlecht auf Iterationen und „Sprints“ abbilden.**

**Zur Zeit werden in der „agilen Welt“ nach Lösungen gesucht bzw. schon in der Praxis auf ihre Tauglichkeit geprüft.**

## Wie wird man Architekt?

- **In möglichst vielen Rollen des Softwareentwicklungsprozesses mitarbeiten oder zumindest mit diesen Rollen zusammenarbeiten**
- **besonders schwierig sind die Fähigkeiten zur Ausrichtung der Architektur zu erlernen (falls überhaupt erlernbar).**
- **Architekt wird man im wesentlichen, in dem man Architekturen erstellt**

**Es bedarf Zeit und Erfahrung, um Softwarearchitekt zu werden.**

## Populäre Missverständnisse und Irrtümer

- **„Gute Architektur muss nie mehr geändert werden, egal welche zukünftigen Anforderungen entstehen.“**
- **„Ich hätte die Architektur aber besser hingekriegt“**
- **„Architekten sind im wesentlichen Top-Level-Designer. Jeder erfahrene Entwickler kann das.“**
- **Frage: „Wie sieht die Architektur ihrer Software aus?“**  
**Antwort: „Wir machen <beliebige Technologie einsetzen>!“**



**[Fried] U. Friedrichsen - „Wer braucht einen Architekten“ in Objektspektrum  
Mai/Juni 2010**

**[Rechtin] E. Rechtin, M. Maier: The Art of Systems Architecture, CRC Press, 2000**

**[Starke] G. Starke, Effektive Software-Architekturen, Hanser Verlag, 2002**

**[POSA1] F. Buschmann, R. Meunier, H. Rohnert, P. Somerlad, M. Stal: Pattern-  
oriented Software Architecture, Addison-Wesley, 1996**