Softwarearchitekten und Softwarearchitekturen

Technik <-> Mensch <-> Organisation

Agenda

- Definition von Softwarearchitektur
- Softwarearchitekt
- Architekturentwurf
- Dokumentation
- Architekturpattern

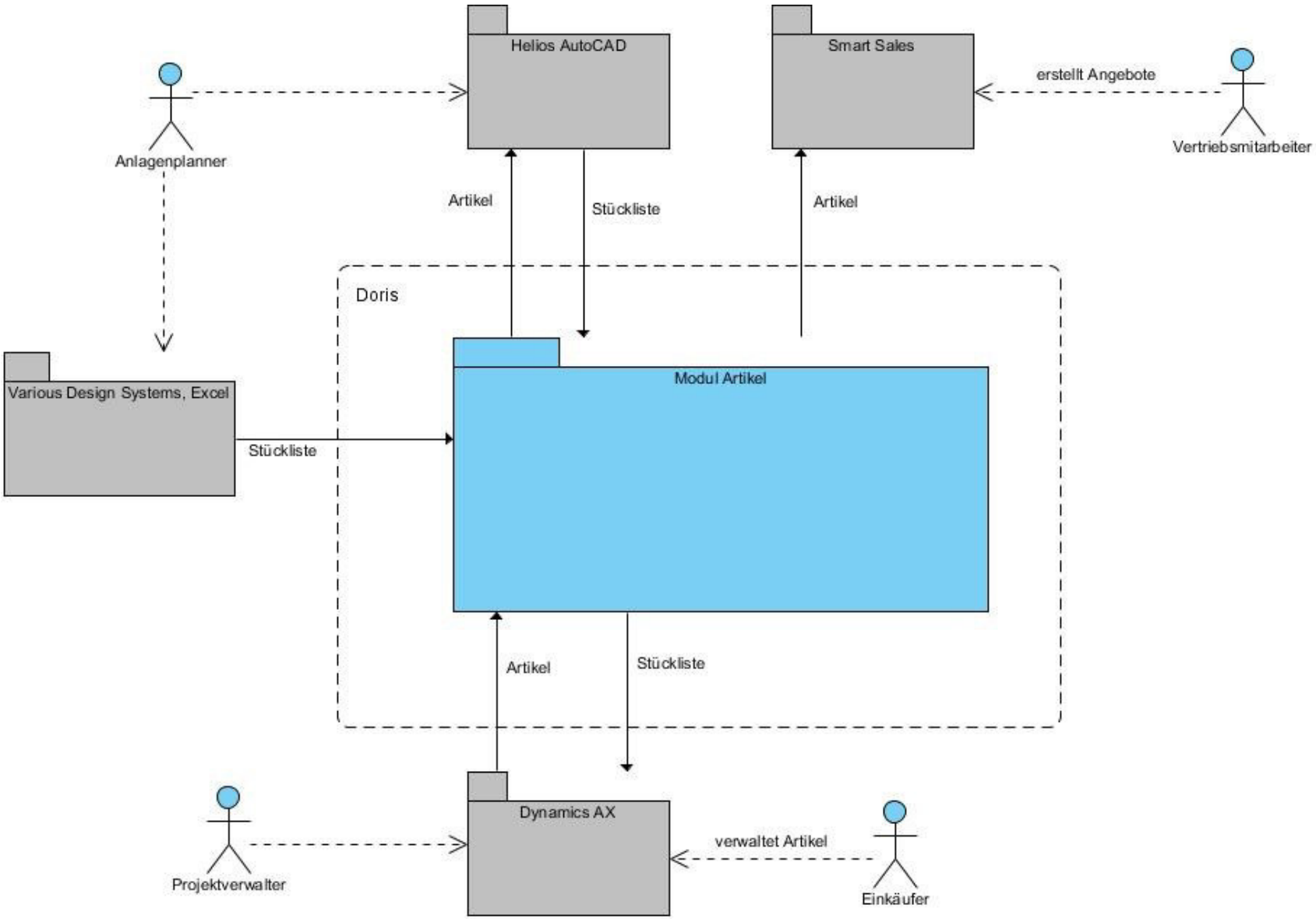
Definition Softwarearchitektur

Eine Softwarearchitektur besteht aus:

- Zerlegung des Problems in kleinere, verständliche und beherrschbare Komponenten
- Komponenten, deren Beziehungen zueinander und zur Umgebung
- Grundsätze, die Design und Entwicklung des Systems bestimmen

Definition Komponente

- Baustein eines Systems
- Kapselt ihr inneres Verhalten
- Kommuniziert mit anderen Komponenten über definierte Schnittstellen
- Ist wieder verwendbar und austauschbar


Helios AutoCAD
Smart Sales
erstellt Angebote
Anlagenplanner
Vertriebsmitarbeiter
Artikel
Stückliste
Artikel
Doris
Various Design Systems, Excel
Modul Artikel
Stückliste
Artikel
Stückliste
Dynamics AX
verwaltet Artikel
Projektverwalter
Einkäufer

Softwarearchitekt - Ausbildung

- SEI (Software Engineering Institute)
  - SOA, Architecture, ATAM
- The Open Group: TOGAF
- Zachman: Certified Enterprise Architect
- Oracle Certified Java EE 6
- iSAQB e.V.

Aufgaben des Softwarearchitekten

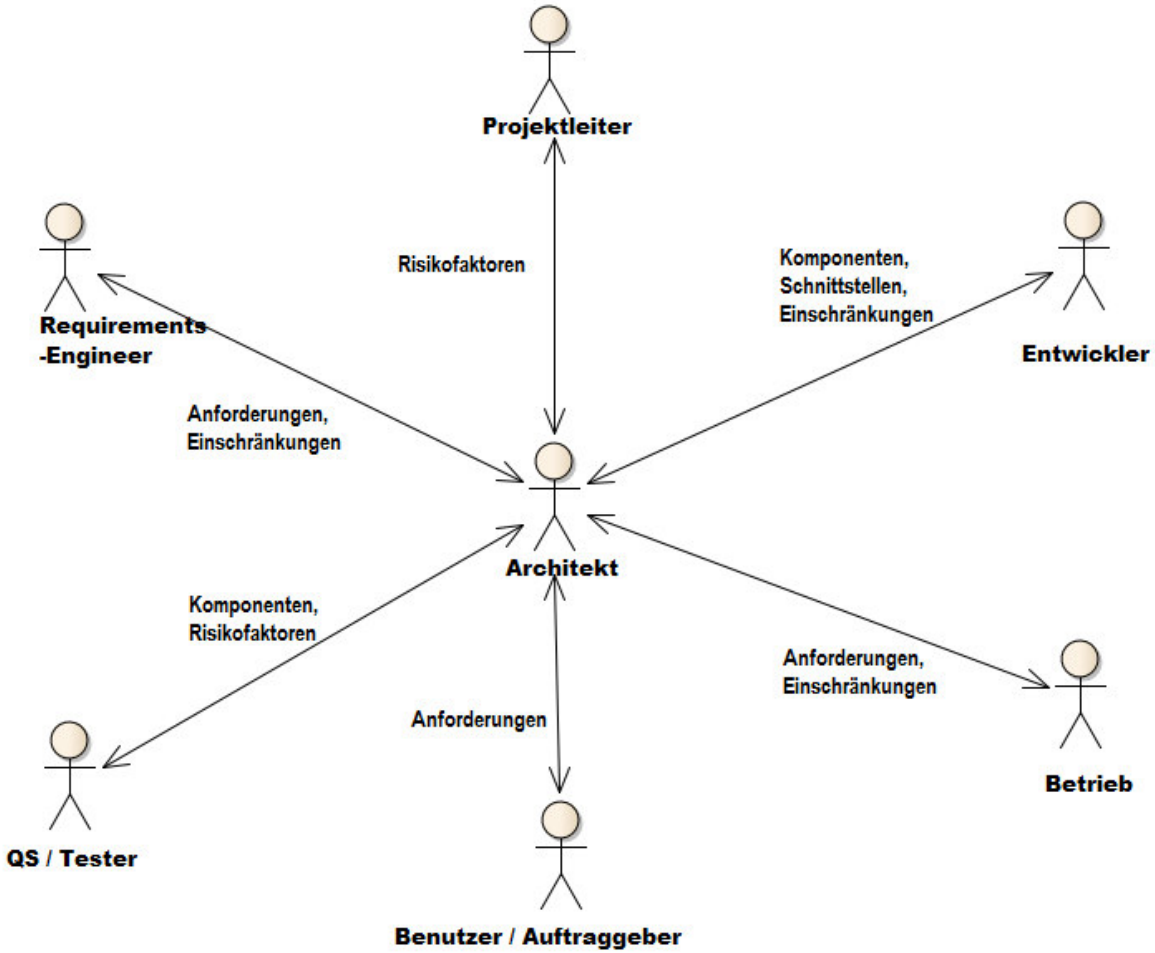
- Anforderungen und Randbedingungen klären
- System strukturieren
  - Systemzerlegung
  - Bausteinstruktur
  - Schnittstellen
- Übergreifende technische Konzepte erarbeiten / beschließen
- Dokumentieren und kommunizieren

Nutzen und Ziele

- Bindeglied zwischen Analyse und Umsetzung

- Bauplan des Systems
- Verständlichkeit und Sicherheit schaffen
- Flexibilität und Effizienz gewährleisten
- Qualität garantieren, Risiken erkennen

Stakeholder <-> Architekt



Beziehungen: Architekt - Projektleitung

- Risikoüberwachung und Eskalation
- Entscheidungskompetenz für Entwurfsentscheidungen
- Auswahl und Beurteilung der techn. Fähigkeiten von Bewerbern und Mitarbeitern
- Auswahl von Tools und Technologien
- Organisation des Entwicklerteams entsprechend des Architekturentwurfs

Beziehungen: Architekt – Req. Engineer

- Festlegung der Anforderungen und Rahmenbedingungen
- Zusammenführung unterschiedlicher oder widersprüchlicher Interessen
- Kompromissfindung: Funktionale, nichtfunktionale

Beziehungen: Architekt - Auftraggeber

- Verbreitung und Akzeptanz der Systemvision
- Einschränkungen und Vorgaben und Wünsche berücksichtigen
- Rückmeldungen einholen

Beziehung Architekt - Entwickler

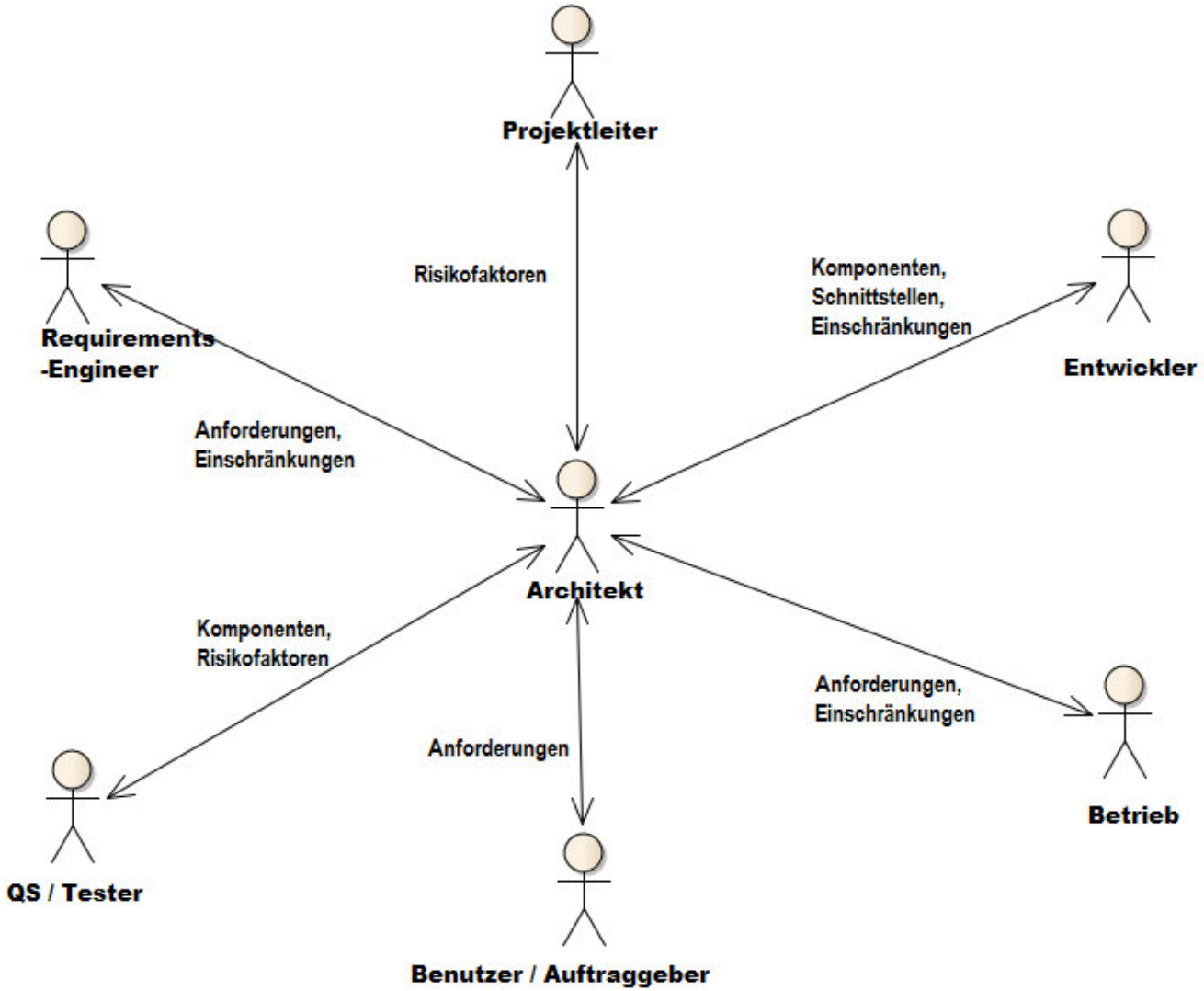
- Verbreitung der Akzeptanz der Architektur
- Training und Beratung des Entwicklerteams in den ausgewählten Werkzeugen und Technologien
- Moderation und Koordination an den Schnittstellen
- Durchsetzung der Architekturvorgaben
- Einholen von Rückmeldungen

Beziehung Architekt – QS / Tester

- Festlegung der Testinfrastruktur
- Sicherstellung einer Mindestqualität
- Festlegung des Qualitätsmerkmale
- Übergabe der Komponenten an die QS
- Unterstützung und Zusammenarbeit bei White-Box Tests

Beziehung Architekt - Betrieb

- Sicherstellung der Betreibbarkeit
- Festlegung der Hardwareanforderungen
- Einschränkungen und Anforderungen festhalten und berücksichtigen
- Anforderung an SLA definieren


Projektleiter
Risikofaktoren
Requirements
-Engineer
Anforderungen,
Einschränkungen
Komponenten,
Schnittstellen,
Einschränkungen
Entwickler
Architekt
Komponenten,
Risikofaktoren
Anforderungen
Anforderungen,
Einschränkungen
QS / Tester
Benutzer / Auftraggeber
Betrieb

Architekten im modernen Umfeld - Risiken

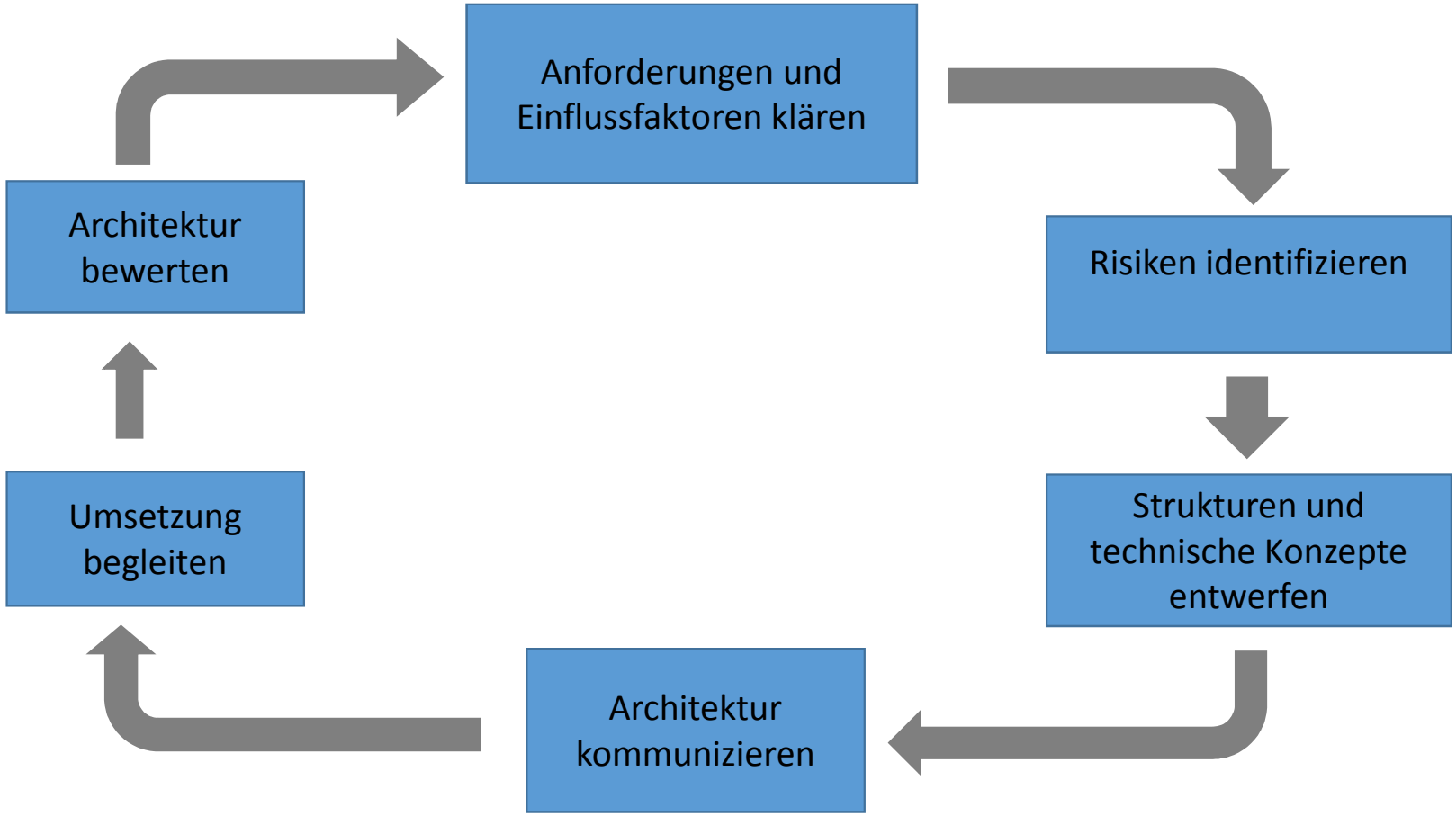
- Elfenbeinturm – Risiken :
  - Die Architektur kann nur aufwendig umgesetzt werden
  - Die Architektur hat mit der Realität nichts zu tun
  - Die Entwickler fühlen sich entmündigt

- Das Entwicklungsteam gestaltet die Architektur – Risiken :
  - Keiner fühlt sich verantwortlich
  - Das erste was dem Team einfällt wird umgesetzt
  - Architektur nach dem Prinzip : „das haben schon immer so gemacht“
  - Keine einheitliche Architektur

Architektur – Best Practise

- Architekturen im Team entwickeln
  - Alle Stakeholder einbinden und mitwirken lassen
- Architekturen iterativ entwickeln
- Komplexe Strukturen und große Risiken durch Szenarien und Prototypen absichern

Architekturentwurf



Beispiel Modul Artikel

- Stelle die Ressource Artikel, die im ERP System gepflegt wird anderen Systemen insbesondere Smart Sales (Verkauf) und Helios (Planung) zur Verfügung
- Sorge für den Import von Stücklisten (verkaufte oder verplante Artikel) in das ERP System
- Das System muss in vier Monaten fertig sein

Stakeholder

- Abteilungsleiter
- Projektleiter
- Requirement Engineer
- Verschiedene Fachabteilungen
- Externer ERP Dienstleister
- Architekturboard
- Team

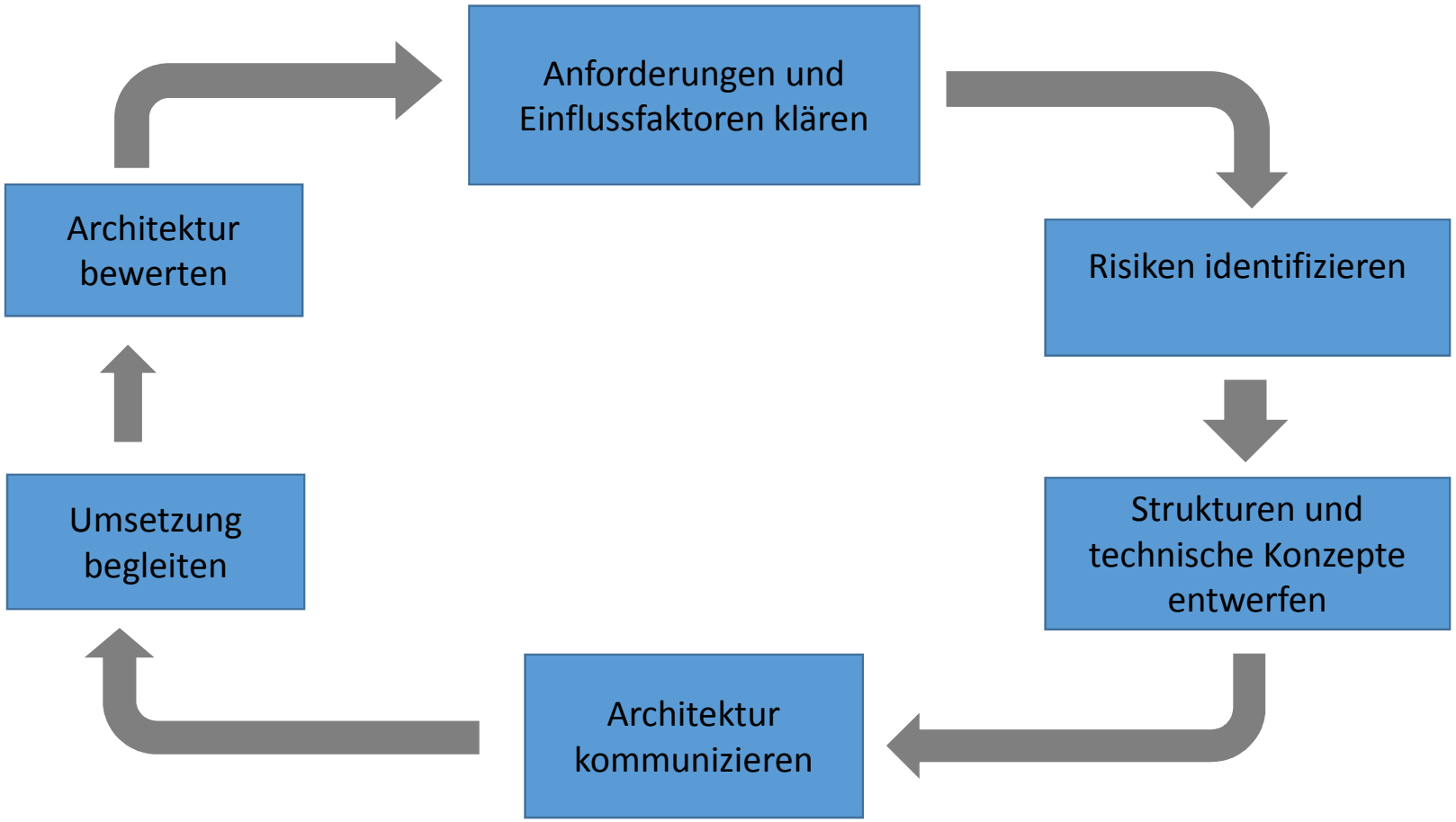
Einflussfaktoren

- Das System muss in vier Monaten fertig sein
- Es soll ein Modul der Integrationsplattform Doris werden
- Das Zielsystem Smart Sales wird parallel entwickelt, die Schnittstelle kann sich hier ändern
- Das Quellsystem ERP wird ebenso parallel entwickelt. Evtl. haben die Entwickler zu wenig Zeit um sich um die Anpassungen zu kümmern
- Die Zusammenarbeit mit dem externen Dienstleister muss noch etabliert werden

Risiken

- Anforderungen teilweise unklar
- Kleines Team
- Hoher Zeitdruck
- Zielsystem wird noch entwickelt
- Verfügbarkeit des externen Dienstleiters unklar
- Es fehlt Erfahrung mit neu eingesetzten Techniken

Softwareentwurf

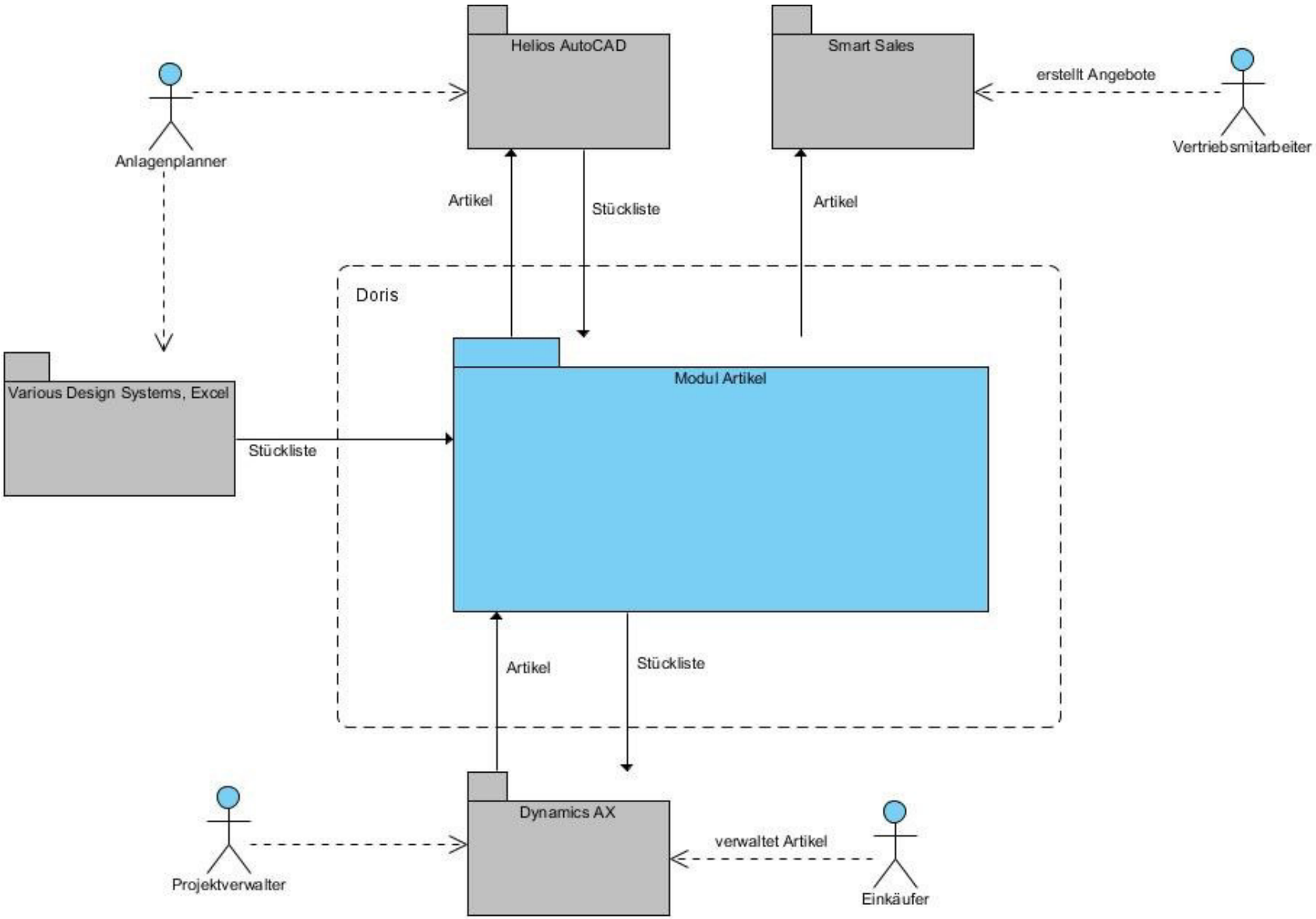


Vorgehen beim Entwurf

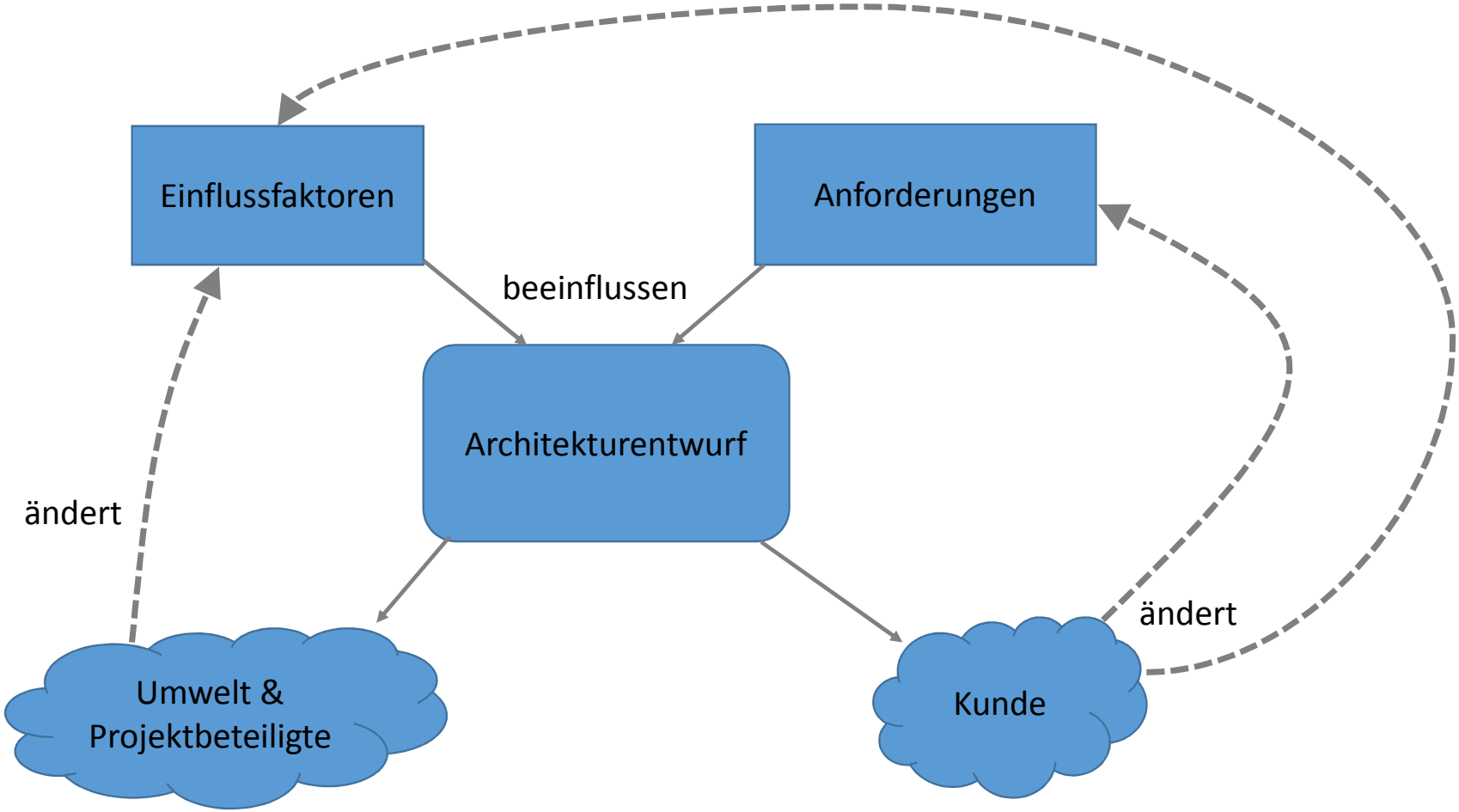
- Kernaufgabe kurz beschreiben und mit den Stakeholdern abstimmen
- Folgende Fragen klären:
  - Wer nutzt das System?
  - Wie wird das System genutzt?
  - Welche Schnittstellen gibt es?
  - Welche Daten werden verwaltet?
  - Wie wird das System gesteuert?
- Systemidee entwickeln und abstimmen

Beispiel – Modul Artikel

- Anwender: Anlagenplaner, Vertrieb, Einkäufer, Projektverwalter
- Schnittstellen zu Dynamic AX, Smart Sales, Design System
- Daten: Artikel, Stückliste
- Integration des Modul in die Integrationsplattform Doris
  - RESTFul
- Excel Import, Export
- Aufruf von den angeschlossenen Systemen, offline und manuell


Helios AutoCAD
Smart Sales
erstellt Angebote
Anlagenplanner
Vertriebsmitarbeiter
Artikel
Stückliste
Artikel
Doris
Various Design Systems, Excel
Modul Artikel
Stückliste
Artikel
Stückliste
Dynamics AX
verwaltet Artikel
Projektverwalter
Einkäufer

Softwarearchitektur iterativ entwickeln



Grundlegende Konzepte

- Einfachheit
  - So einfach wie möglich
  - So komplex wie nötig
- Entwurf nach Verantwortlichkeiten
  - Trennung von Technik und Fachlichkeit
  - Modularität
- Konzentration auf Schnittstellen
- Berücksichtigung von Fehlern

Qualität

- Manager und Auftraggeber
  - Kosteneffizienz
  - Flexibilität
  - Wartbarkeit
- Anwender
  - Hohe Performance
  - Einfache Benutzbarkeit
- Projektleiter
  - Parallelisierbarkeit
  - Gute Testbarkeit

Qualitätsmerkmale

- Funktionalität
  - Angemessenheit
  - Richtigkeit
- Zuverlässigkeit
  - Fehlertoleranz
  - Reife
- Benutzbarkeit
  - Verständlichkeit
  - Erlernbarkeit

Qualitätsmerkmale – Fortsetzung

- Effizienz
  - Zeitverhalten
  - Verbrauchsverhalten
- Änderbarkeit
  - Modifizierbarkeit
  - Analysierbarkeit
- Übertragbarkeit
  - Austauschbarkeit
  - Konformität

Qualitätsmerkmale Modul Artikel

- Interoperabilität
- Modifizierbarkeit
- Stabilität
- Zeitverhalten
- Ausfallsicherheit

Szenarien - Beispiele

- Ein neuer Artikel muss innerhalb eines Tages dem System Smarts Sales zur Verfügung stehen
- Ein neues System muss innerhalb von 10 PT Artikel verarbeiten können
- Bei einem Ausfall des Systems muss das System innerhalb von max. 12 Stunden wieder online sein

Szenarien – Vorlage

- Quelle des Auslösers
- Auslöser
- Umgebung
- Systembestandteil
- Antwort
- Antwortmetrik

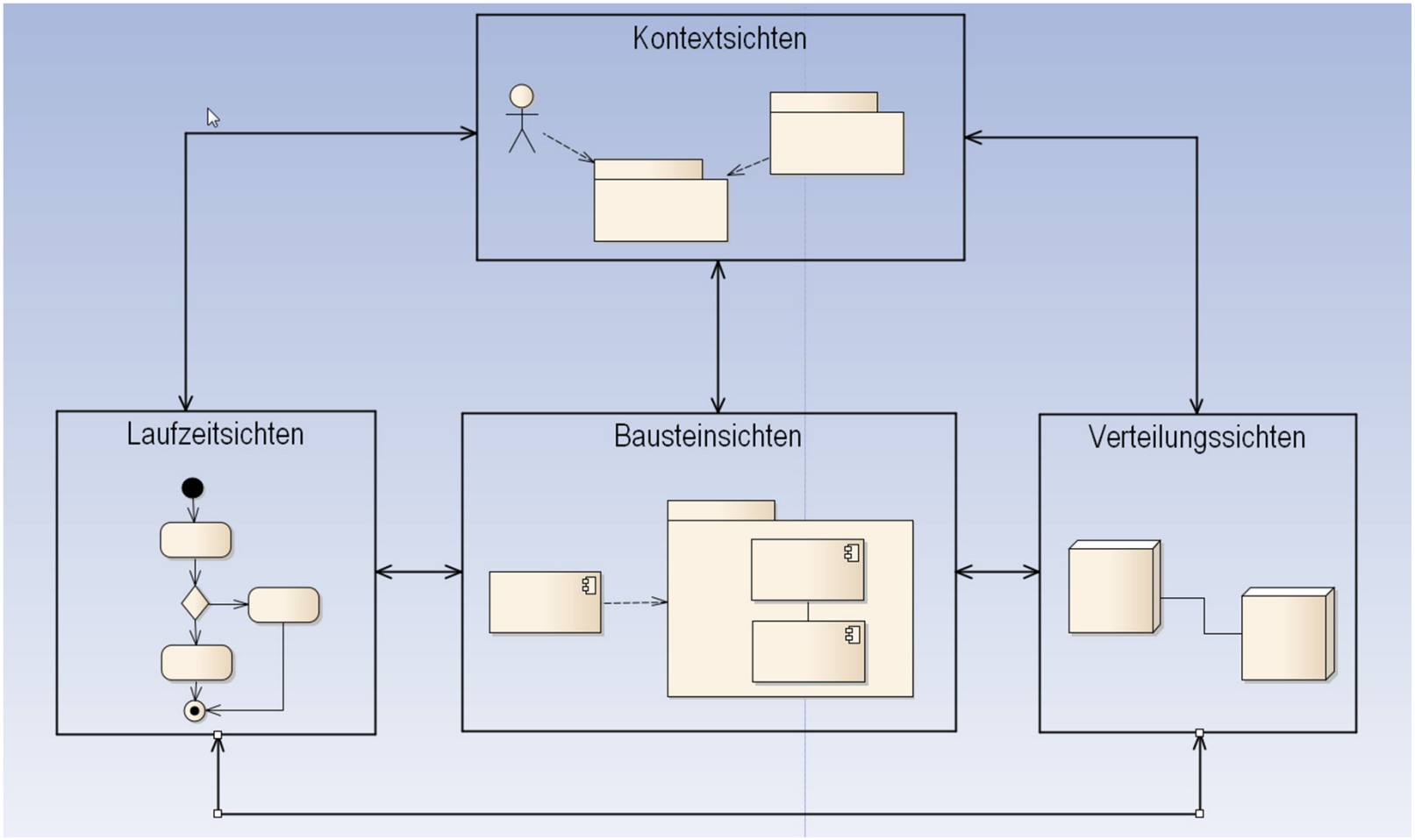
Weitere Konzepte - SOLID

- **S**ingle Responsible Principle
- **O**pen Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Dokumentation – Best Practise

- Template verwenden z.B. arc42
- Vier Arten von Sichten verwenden
- Einfach halten
- Standards verwenden z.B. UML
- Dokumentenverzeichnis
- Stakeholder
- Wichtige Entscheidungen und ihre Gründe festhalten

Die Vier Sichten



Architekturpattern

- SOA
- Hexagonal
- Micro Service
- Layer / Monolith

Architekturpattern – Conways Law

- Conways Law
  - Organisationen die Systeme modellieren sind auf Modelle festgelegt, welche die Kommunikationsstrukturen dieser Organisation abbilden
- Conways Corollary
  - Ein Software System dessen Struktur die Kommunikationsstruktur der Organisation widerspiegelt, funktioniert besser

SOA – Beispiel Deutsche Post AG

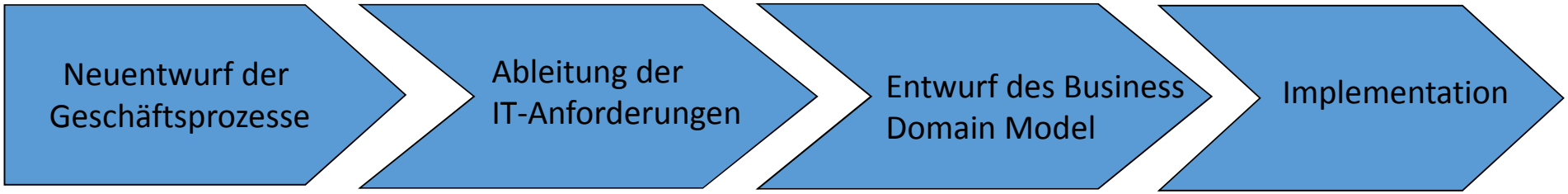
Motive:

- Investitionen sichern
- Neuentwicklung beschleunigen
- Zugriff auf Kerninformationen herstellen und vereinfachen

SOA - Lösungsidee

- Unterstützung von Geschäftsprozessen durch fachliche Services, die miteinander interagieren
- Services sind technikunabhängig
- Bildung einer Servicearchitektur (statt Anwendungsarchitektur)
- Möglichst wenig Redundanzen für Funktionen und Daten
- Integration von bestehenden Applikationen

Erneuerung der IT Landschaft



Service Backbone (SBB)

Standardapplikationen

- SBB kapselt Standardapplikationen wie SAP

- SBB stellt deren Funktionalität zur Verfügung

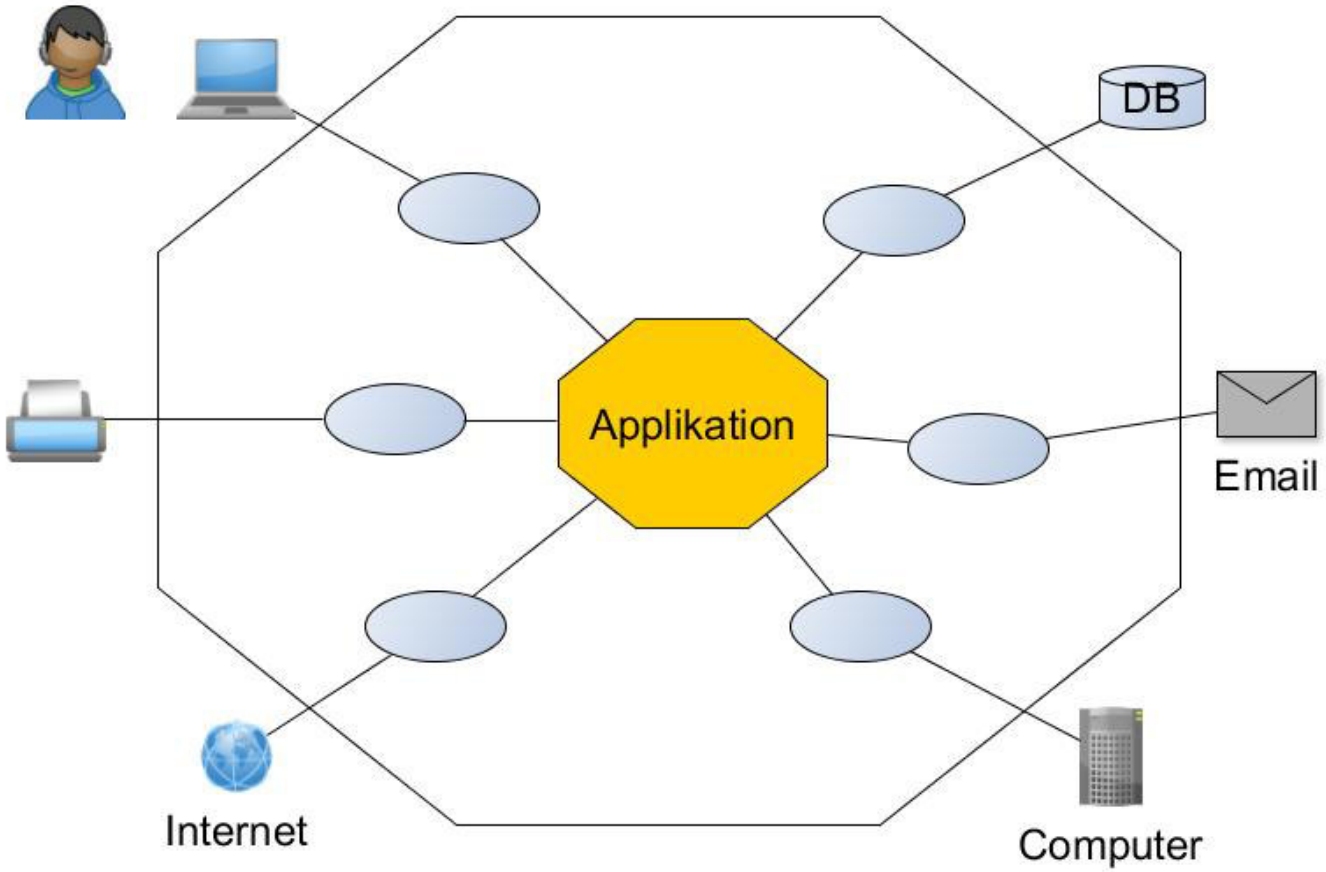
Service Entwurf

1. Identifizieren aller potenzieller Consumer
2. Spezifizieren der Geschäftsfunktionalität
3. Vergleichen mit Business Domain Model und existierenden Services
4. Definieren und Starten der Implementation
5. Service Beschreibung, SLA, XML Schema
6. Service Provider mit SBB verbinden
7. Registrieren

Motive der Hexagonalen Architektur

- Keine verteilte Business Logik
- Testbar in Isolation
- Verwendbar in Anwender Applikationen und Batch Systemen
- Wurde angewendet bei Flickr, Google Maps

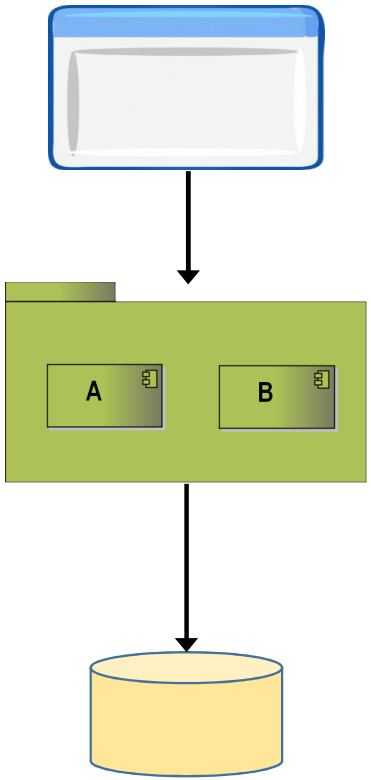
Hexagonale Architektur – Ports & Adapter

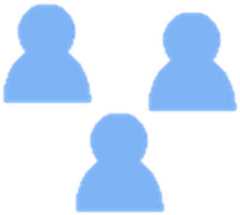


Motive - Schichtenarchitektur

- Schichten können unabhängig voneinander entwickelt werden
- Schichtenbildung minimiert Abhängigkeiten zwischen Komponenten
- Schichtenbildung ist ein leicht verständliches Strukturprinzip

Schichtenarchitektur





GUI Schicht

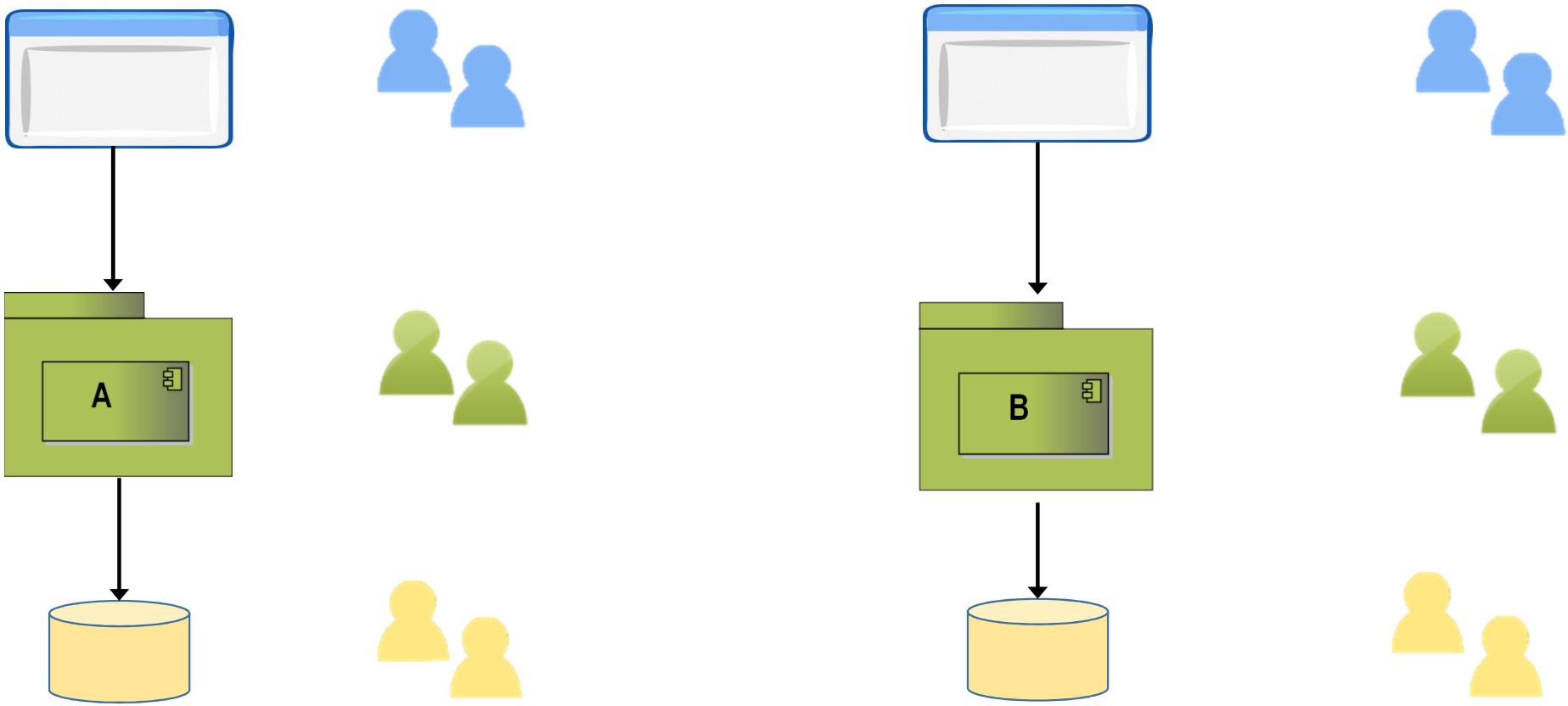
Mittelschicht

Datenbankschicht

Motive Microservices

- Klare Modularisierung
- Gute Skalierbarkeit
- Einfaches Deployment
- Teams können unabhängig voneinander arbeiten
- Aufteilung der Teams anhand von Fachlichkeit

Microservices



Best Practise

- Die Architektur sollte der Organisationsstruktur entsprechen
- Beides sollte der Problemstellung angepasst sein
- Auch für die Organisation gelten ähnliche Empfehlungen:
  - Single Responsible Prinzip
  - Konzentration auf die Schnittstellen

Zusammenfassung

- Softwarearchitektur ist wichtig
- Die Arbeit kann von der Rolle Softwarearchitekt übernommen werden
- Teamarbeit ist wichtig
- Dokumentation pragmatisch angehen
- Man kann verschiedene Architekturpattern anwenden
- Organisation und Architektur hängen eng miteinander zusammen

Quellenverzeichnis

- Effektive Softwarearchitekturen – Gernot Starke
- Arc42 – www.arc42.de
- Aim42 – www.aim42.org
- Knigge für Softwarearchitekten – Peter Hruschka, Gernot Starke
- Software Architectur in Praxis – Len Bass, Paul Clements, Rick Kazman
- Pattern Oriented Software Architecture (Volume 4) – Frank Buschmann, Kevlin Henney, Douglas C. Schmidt
- Pattern of Enterprise Application Architecture – Martin Fowler

Quellenverzeichnis

- SOA in der Praxis – Nicolai Josuttis
- Enterprise SOA – Dirk Krafzig, Karl Blanke, Dirk Slama