Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften

Sommersemester 2018

Nils Löwe / nils@loewe.io / @NilsLoewe

Ziele dieser Vorlesung

- Was ist Softwarearchitektur und wozu braucht man sie?
- Was muss ein Softwarearchitekt / eine Softwarearchitektin können?
- Wie konzipiert man eine große Anwendung?
- Wie werden Architekturen entworfen, dokumentiert und bewertet?

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Definitionen für Softwarearchitektur

Die Architektur eines Softwaresystems ist die Menge der Haupt-Designentscheidungen über das System. *(Taylor)*

(“A software system’s architecture is the set of principal design decisions about the system.”)

Die Software-Architektur ist die grundlegende Organisation eines Systems, dargestellt durch dessen Komponenten, deren Beziehungen zueinander und zur Umgebung sowie die Prinzipien, die den Entwurf und die Evolution des Systems bestimmen. *(Reussner)*

Software Architecture = { what, how, why } *(Perry and Wolf)*

The software architecture of deployed software is determined by those aspects that are the hardest to change. *(Chris Verhoef)*

Architektur besteht aus Strukturen

- die Komponenten (Bausteine), aus denen ein System besteht
- die wesentlichen (extern sichtbaren) Eigenschaften dieser Komponenten
- die Beziehungen der Komponenten untereinander

Architektur beschreibt eine Lösung im Sinne eines Bauplans

(die Architektur eines Gebäudes besteht aus einer Sammlung von Plänen - nicht aus Steinen und Zement)

Erst die *Implementierung* macht aus den Komponenten und Schnittstellen der Architektur ein reales System.

Architektur basiert auf Entwurfsentscheidungen

- Entscheidungen zum Entwurf der Komponenten
- Entscheidung für eine bestimmte Technologie
  Die Konsequenz vieler Entscheidungen können Architekten erst sehr viel später beurteilen!

Architektur schafft Verständlichkeit

- komplexe Anforderungen → geordnete Strukturen
- angemessene und problembezogene Dokumentation

Management: Anforderungen erfüllbar / erfüllt?

neue Mitarbeiter: Systemstruktur kennen lernen

Wartungsteams: betroffene Bestandteile leichter finden und Folgen von Änderungen abschätzen

Systembetreiber: Welche Software-Komponenten laufen auf welchen physischen Systemen ab?

Architektur ist der Rahmen für flexible Systeme

- stellt Flexibilität und Erweiterbarkeit sicher → *"framework for change"* (Tom DeMarco)

Architektur ist Abstraktion

- Essenzielle Aufgabe von Architekten: Weglassen von nicht benötigten Informationen
- Informationen werden bewusst gefiltert um die Darstellung lesbar und verständlich zu halten

Architektur schafft Qualität

Die Qualität eines Systems bezeichnet die Summe seiner nicht-funktionale Eigenschaften:

- Performance
- Verständlichkeit
- Flexibilität
- ...

Das sind meistens die schwierigen Anforderungen!

Architektur vs. Entwurf/Design?

- die Grenze ist fließend
- Design (oder Entwurf) bezeichnet den Prozess der Erstellung der Architektur

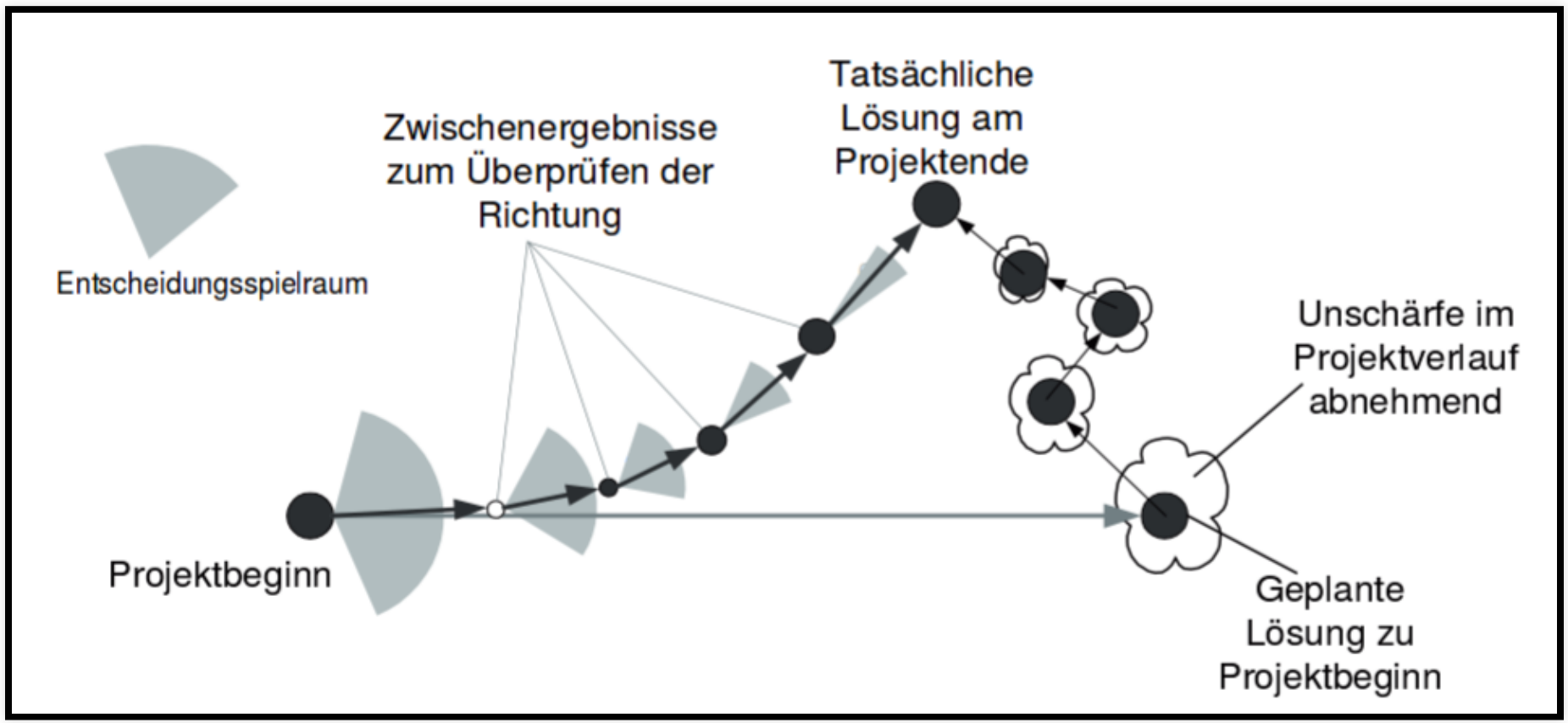
→ Gehen Sie mit diesen Begriffen pragmatisch um und suchen Sie nicht nach einer "formalen" Definition.

Die Aufgaben von Softwarearchitekten

"Das Leben von Software-Architekten besteht aus einer langen und schnellen Abfolge suboptimaler Entwurfsentscheidungen, die meist im Dunkel getroffen werden."

(Phillipe Kruchten)

Architekturen entstehen in Zyklen und Iterationen



Bildquelle: Starke / "Effektive Softwarearchitekturen" (5. Auflage)

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Seit wann gibt es den Begriff der Softwarearchitektur?

Konferenz über Softwaretechnik in Rom

Software Engineering Techniques. Report of a Conference Sponsored by the NATO Science Committee. Scientific Affairs Division, NATO, 1970, S. 12.

Warum?

Die Systeme wurden in den 1960ern so komplex, dass sie von mehreren Teams entwickelt werden mussten.

Beispiel: IBM OS/360

Planung

- Entwicklungskosten: 40 Mio. USD
- Lines of Code: 1 Mio.
- Fertigstellung: 1965

Beispiel: IBM OS/360

Realität

- Entwicklungskosten: 500 Mio. USD (Faktor 12,5)
- Lines of Code: 10 Mio. (Faktor 10)
- Fertigstellung: 1967 (2 Jahre zu spät)

Softwarearchitektur im Lauf der Zeit

Erste Beschreibung von ”Dekomposition, Zerlegung, Entwurf”

- 1970er: Eher im Kontext von Hardware genutzt
- 1972: *"On the criteria to be used in decomposing systems into modules"* von D. L. Parnas
- 1975: *"The Mythical Man Month"* von Frederick Brooks

Softwarearchitektur im Lauf der Zeit

Unabhängiges Teilgebiet der Softwaretechnik

Konzept der Schnittstellen und Konnektoren

- 1992: *"Foundations for the Study of Software Architecture"* von Dewayne Perry und Alexander Wolf
- 1995: *"Software Architecture Analysis Method"* des Software Engineering Institute

Softwarearchitektur im Lauf der Zeit

Allgemeine Verbreitung und ”Stand der Technik”

- 2000: *"IEEE 1471:2000 Norm Recommended Practice for Architectural Description of Software-Intensive Systems"*
- 2003: Zertifizierung als Softwarearchitekt durch die iSAQB (International Software Architect Qualification Board)
- 2003: UML 2.0 ist geeignet um Softwarearchitekturen zu beschreiben

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Warum überhaupt Sichten?

”Es ist eine offensichtliche Wahrheit, dass auch eine perfekte Architektur nutzlos bleibt, wenn sie nicht verstanden wird...”

Felix Bachmann und Len Bass in "Software Architecture Documentation in Practice"

1.

Eine einzelne Darstellung kann die Vielschichtigkeit und Komplexität einer Architektur nicht ausdrücken.

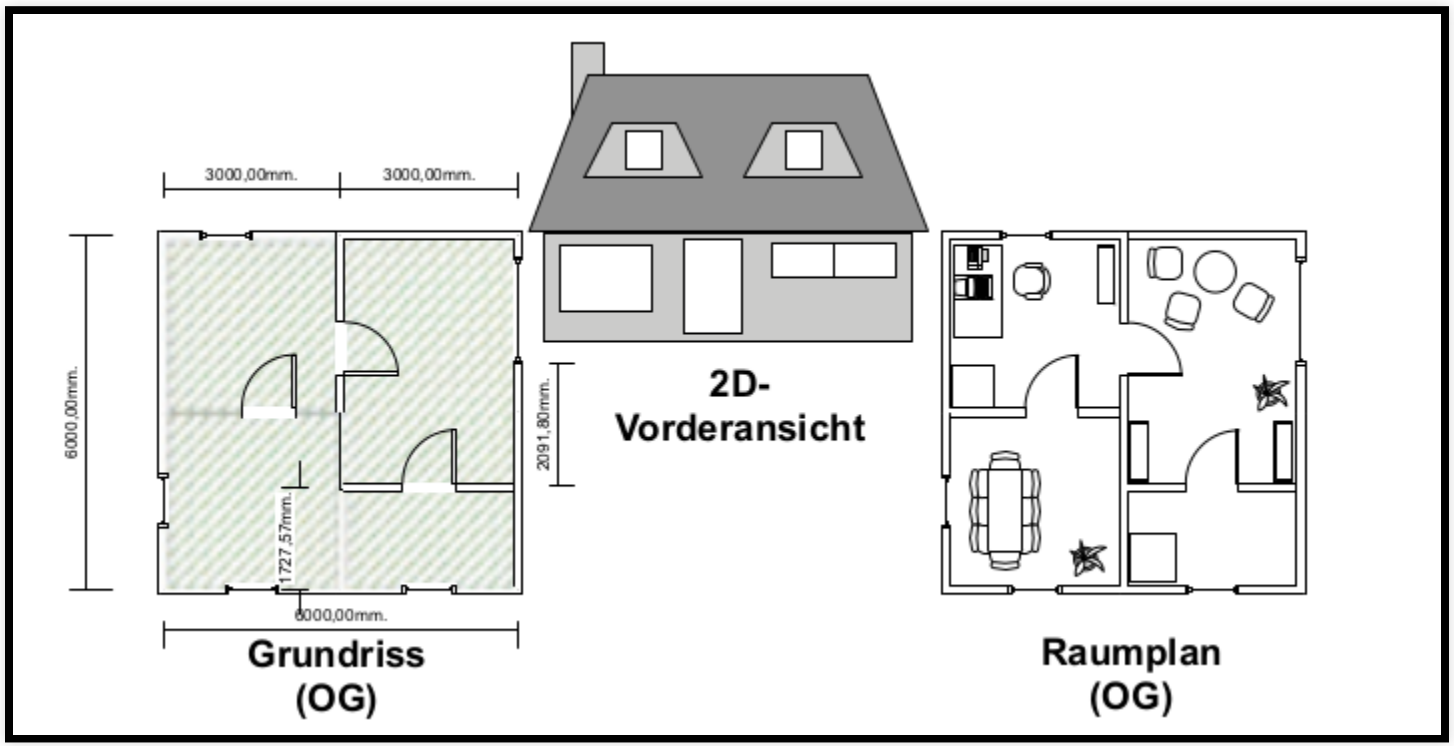
- Genauso wenig, wie man nur mit einem Grundriss ein Haus bauen kann.

2.

Sichten ermöglichen die Konzentration auf einzelne Aspekte des Gesamtsystems und reduzieren somit die Komplexität der Darstellung.

3.

Die Projektbeteiligten haben ganz unterschiedliche Informationsbedürfnisse.



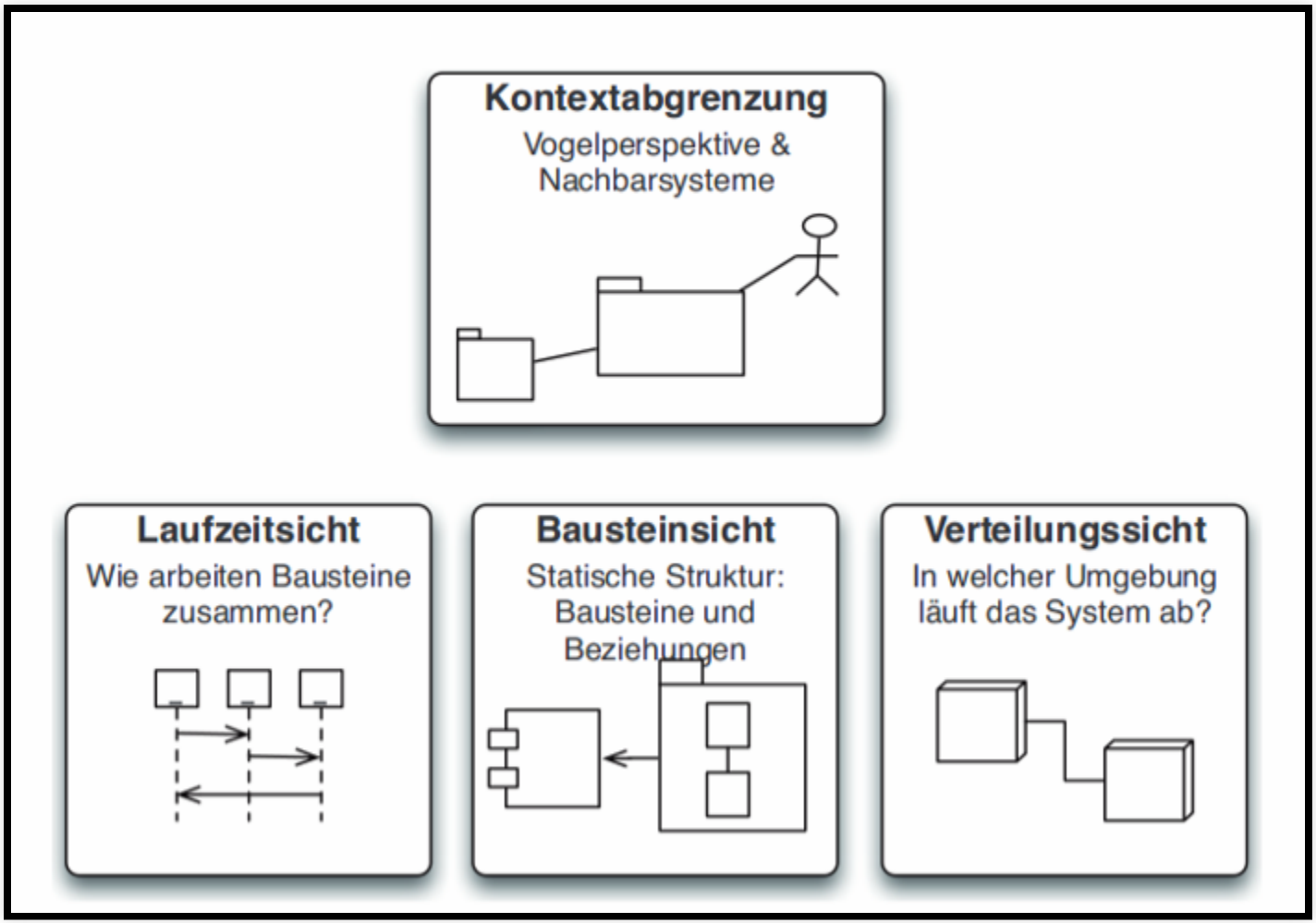
Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

Architekten müssen Projektbeteiligten die Architektur erklären bzw. sie verteidigen/vermarkten

- die entworfenen Strukturen
- die getroffenen Entscheidungen
- ihre Konzepte + Begründungen + Vor- und Nachteile

→ Mit Hilfe von unterschiedlichen Sichten lassen sich viele Aspekte von Architektur verständlich darstellen.

Überblick über die vier Sichten



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

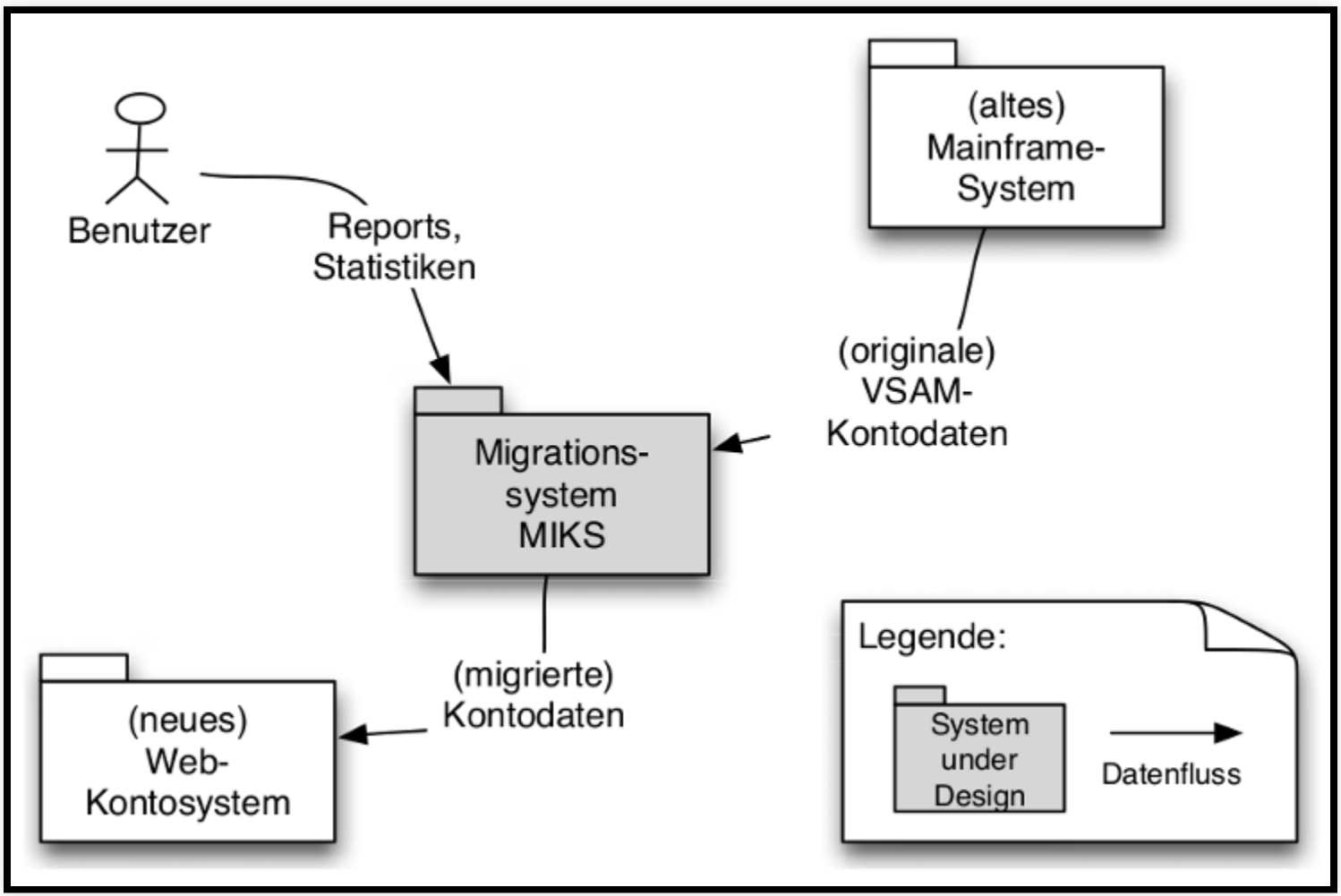
Kontextsicht

- Wie ist das System in seine Umgebung eingebettet?
- zeigt das System als Blackbox in seinem Kontext aus der Vogelperspektive

Kontextsicht - Enthaltene Informationen:

- Schnittstellen zu Nachbarsystemen
- Interaktion mit wichtigen Stakeholdern
- wesentliche Teile der umgebenden Infrastruktur

Kontextsicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

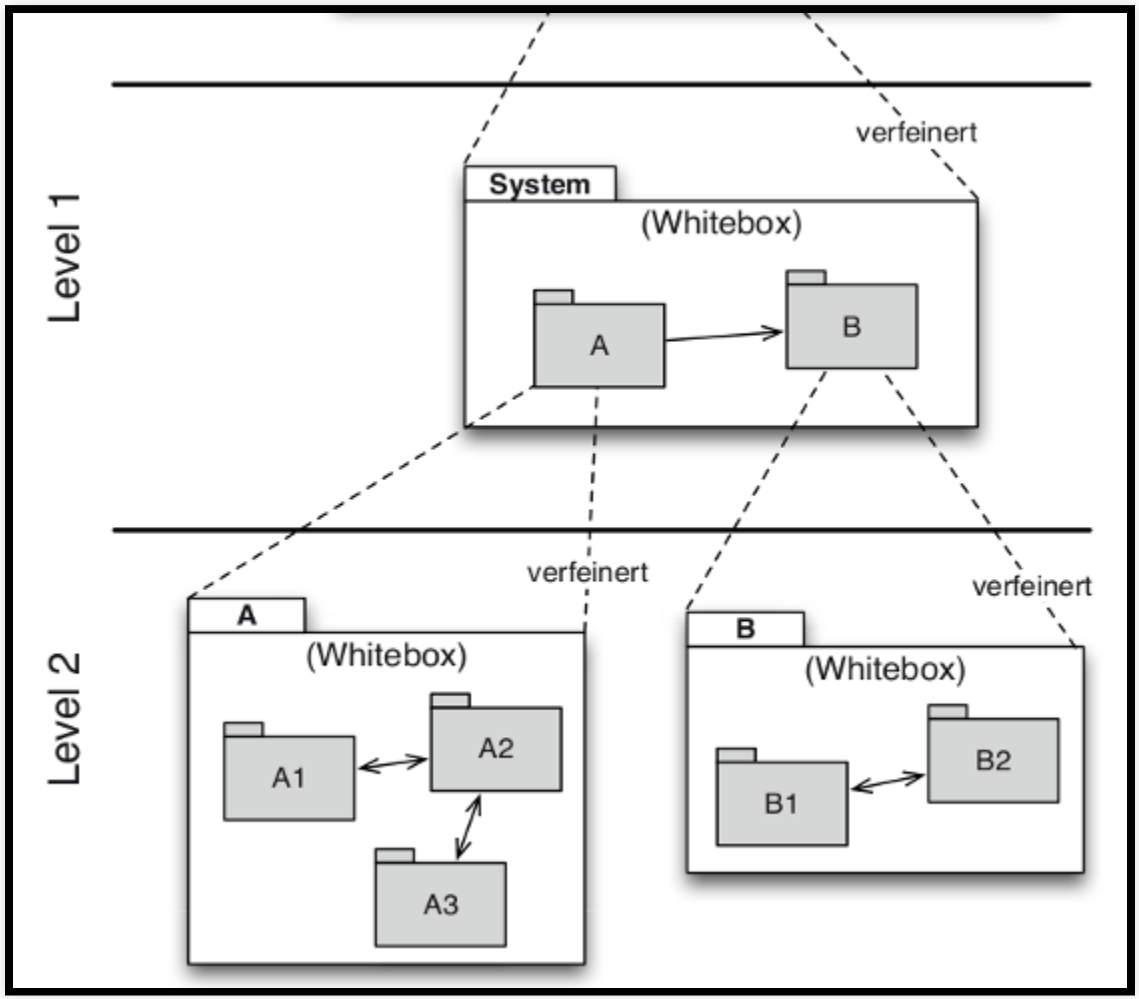
Bausteinsicht

- Wie ist das System intern aufgebaut?
- unterstützt Auftaggeber und Projektleiter bei der Projektüberwachung
- dienent der Zuteilung von Arbeitspaketen
- dient als Referenz für Software-Entwickler

Bausteinsicht - Enthaltene Informationen:

- statische Strukturen der Bausteine des Systems
- Subsysteme
- Komponenten und deren Schnittstellen

Bausteinsicht - Beispiel

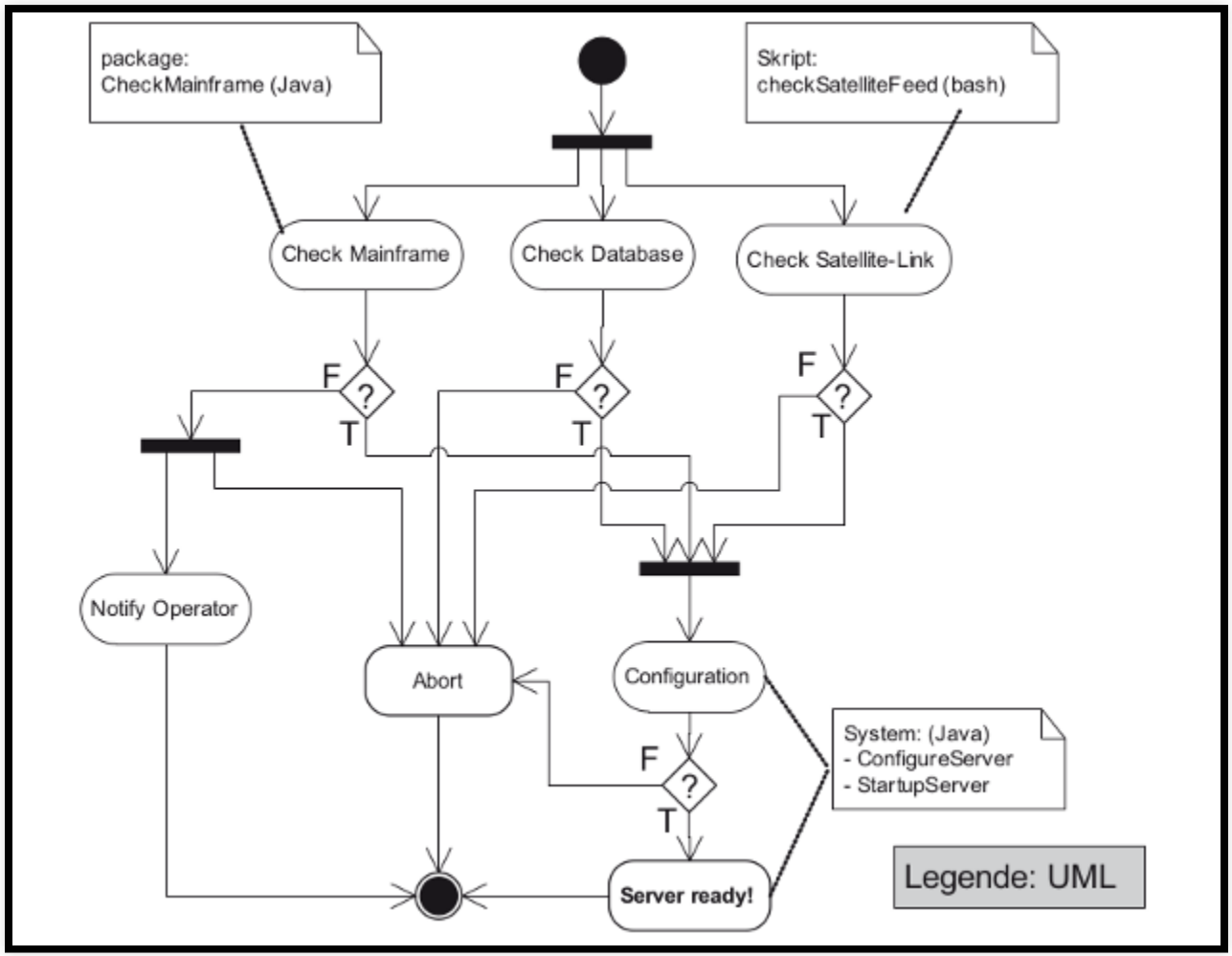


Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

Laufzeitsicht

- Wie läuft das System ab?
- Welche Bausteine des Systems existieren zur Laufzeit?
- Wie wirken die Bausteine zusammen?

Laufzeitsicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

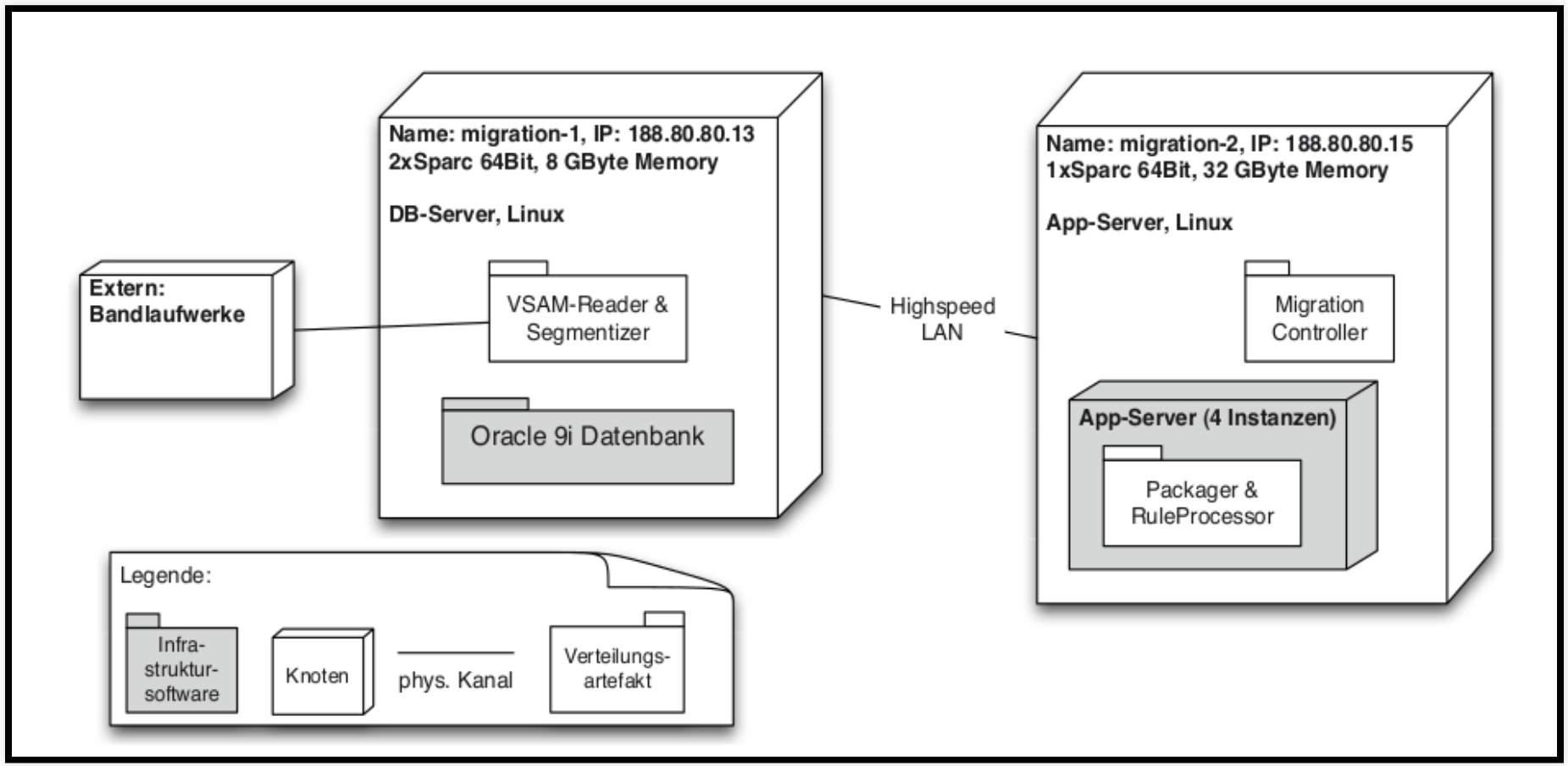
Verteilungssicht / Infrastruktursicht

- In welcher Umgebung läuft das System ab?
- zeigt das System aus Betreibersicht

Verteilungssicht - Enthaltene Informationen:

- Hardwarekomponenten: Rechner, Prozessoren
- Netztopologien
- Netzprotokolle
- sonstige Bestandteile der physischen Systemumgebung

Verteilungssicht - Beispiel



Bildquelle: "Effektive Softwarearchitekturen" von Gernot Starke

Gibt es noch weitere Sichten?

Empfehlung

- Verzichten Sie möglichst auf weitere Sichten
- Jede Sicht kostet Erstellungs- und Wartungsaufwand
- Die grundlegenden Aspekte der Architektur- und Systementwicklung decken die vier Sichten ab

Beispiel (nach Peter Hruschka):

```
Beim Häuserbau könnten Kakteen- und Orchideenzüchter nach der Sonnenein-
strahlung in einzelnen Räumen fragen und zum Wohle ihrer pflanzlichen Lieblinge
einen gesonderten Plan wünschen. Wie groß ist Ihrer Erfahrung nach die Zahl derer,
die beim Bau oder beim Kauf einer Immobilie diese „pflanzliche“ Sicht als Entschei
dungskriterium verwenden?
```

Wie viel Aufwand für welche Sicht?

Rechnen Sie damit, dass Sie 60 bis 80% der Zeit, die Sie für den Entwurf der Architektursichten insgesamt benötigen, alleine für die Ausgestaltung der Bausteinsicht aufwenden. Der ausschlaggebende Grund hierfür: Die Bausteinsicht wird oftmals wesentlich detaillierter ausgeführt als die übrigen Sichten.

Dennoch sind die übrigen Sichten für die Software-Architektur und das Gelingen des gesamten Projektes wichtig! Lassen Sie sich von diesem relativ hohen Aufwand für die Bausteinsicht in keinem Fall dazu verleiten, die anderen Sichten zu ignorieren.

Quelle: Starke/Effektive Softwarearchitekturen

Wechselwirkungen dokumentieren

Sie sollten in Ihren Architekturbeschreibungen die Entwurfsentscheidungen dokumentieren, die besonderen Einfluss auf andere Sichten haben. Beispielsweise bestimmt die Entscheidung für eine zentrale Datenhaltung in der Bausteinsicht maßgeblich den Aufbau der technischen Infrastruktur

Wechselwirkungen dokumentieren

- Bessere Nachvollziehbarkeit von Entscheidungen
- Auswirkungen von Änderungen werden vereinfacht
- Das Verständnis der Architekturbeschreibung wird einfacher, da die Zusammenhänge zwischen den Sichten klarer werden

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Was ist Qualität?

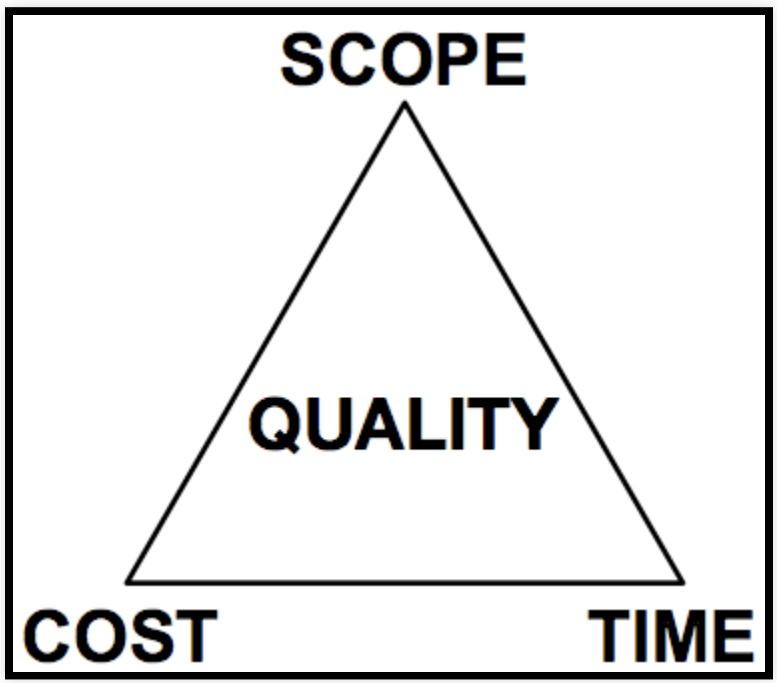
Was ist Qualität?

Duden: Qualität=„Beschaffenheit, Güte, Wert“

Was ist Qualität?

Die Qualität stimmt, wenn der Kunde wiederkommt und nicht das Produkt

Was ist Qualität?



Quelle: http://pm-blog.com/

Qualität ist ein wichtiges Ziel für Software-Architekten

Probleme von Qualität

- Qualität ist nur indirekt messbar
- Qualität ist relativ (jeweils anders für: Anwender, Projektleiter, Betreiber, ...)
- Die Qualität der Architektur korreliert nicht notwendigerweise mit der Codequalität
- Erfüllung aller funktionalen Anforderungen lässt keinerlei Aussage über die Erreichung der Qualitätsanforderungen zu

Beispiel funktionale Anforderung „Sortierung von Daten"

- Kann funktional erfüllt sein, aber nichtfunktional?
- Sortierung großer Datenmengen (Terabyte), die nicht mehr zeitgleich im Hauptspeicher gehalten werden können
- Sortierung robust gegenüber unterschiedlichen Sortierkriterien (Umlaute, akzentuierte Zeichen, Phoneme, Ähnlichkeitsmaße und anderes)
- Sortierung für viele parallele Benutzer
- Sortierung unterbrechbar für lang laufende Sortiervorgänge
- Erweiterbarkeit um weitere Algorithmen, beispielsweise für ressourcenintensive Vergleichsoperationen
- Entwickelbarkeit im räumlich verteilten Team

Qualitätsmerkmale nach DIN/ISO 9126

Funktionalität

Zuverlässigkeit

Benutzbarkeit

Effizienz

Änderbarkeit

Übertragbarkeit

Funktionalität

Existenz eines Satzes von Funktionen mit spezifizierten Eigenschaften

Zuverlässigkeit

Fähigkeit, Leistungsniveau über einen Zeitraum aufrecht zu erhalten

Benutzbarkeit

Aufwand zur Benutzung und individuelle Beurteilung der Benutzung

Effizienz

Verhältnis Leistungsniveau / eingesetzte Betriebsmittel

Änderbarkeit

Aufwand zur Durchführung von Änderungen

Übertragbarkeit

Eignung zur Übertragung in andere Umgebung

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Was sind Architekturmuster?

A pattern for software architexture describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, theis relationships, and the ways in which they collaborate.

(1996 / Pattern Oriented Software Architectute)

Ein Architekturmuster beschreibt eine bewährte Lösung für ein wiederholt auftretendes Entwurfsproblem

(Effektive Softwarearchitekturen)

Ein Architekturmuster definiert den Kontext für die Anwendbarkeit der Lösung

(Effektive Softwarearchitekturen)

Warum Architekturmuster?

Erfolg kommt von Weisheit.

Weisheit kommt von Erfahrung.

Erfahrung kommt von Fehlern.

Haben Sie jemals einen dummen Fehler zweimal begangen?

– Willkommen in der realen Welt.

Haben Sie diesen Fehler hundertmal hintereinander gemacht?

-Willkommen in der Software-Entwicklung.

Aus Fehlern kann man hervorragend lernen.

Leider akzeptiert kaum ein Kunde Fehler, nur weil Sie Ihre Erfahrung als Software-Architekt sammeln.

In dieser Situation helfen Heuristiken.

Heuristiken kodifizieren Erfahrungen anderer Architekten und Projekte, auch aus anderen Bereichen der Systemarchitektur.

Heuristiken sind nicht-analytische Abstraktionen von Erfahrung

Es sind Regeln zur Behandlung komplexer Probleme, für die es meist beliebig viele Lösungsalternativen gibt. Heuristiken können helfen, Komplexität zu reduzieren.

Andere Begriffe für Heuristiken sind auch „Regeln", „Muster" oder „Prinzipien".

Es geht immer um Verallgemeinerungen und Abstraktionen von konkreten Situationen.

Heuristiken bieten Orientierung im Sinne von Wegweisern, Straßenmarkierungen und Warnschildern.

Sie geben allerdings lediglich Hinweise und garantieren nichts. Es bleibt in Ihrer Verantwortung, die passen- den Heuristiken für eine bestimmte Situation auszuwählen:

Die Kunst der Architektur liegt nicht in der Weisheit der Heuristiken, sondern in der Weisheit, a priori die passenden Heuristiken für das aktuelle Projekt auszuwählen.

Architektur: Von der Idee zur Struktur

Ein klassischer und systematischer Ansatz der Beherrschung von Komplexität lautet „teile und herrsche“ (divide et impera). Das Problem wird in immer kleinere Teile zerlegt, bis diese Teilprobleme eine überschaubare Größe annehmen.

Anwendung auf Software-Architekturen:

klassische Architekturmuster

Horizontale Zerlegung: „In Scheiben schneiden"

Vertikale Zerlegung: „In Stücke schneiden"

weitere Architekturmuster

Alles ist möglich...

Überblick über Architekturmuster

Arten von Architekturmustern?

Chaos zu Struktur / Mud-to-structure

Verteilte Systeme

Interaktive Systeme

Adaptive Systeme

Domain-spezifische Architektur

Chaos zu Struktur / Mud-to-structure

- Organisation der Komponenten und Objekte eines Softwaresystems
- Die Funktionalität des Gesamtsystems wird in kooperierende Subsysteme aufgeteilt
- Zu Beginn des Softwareentwurfs werden Anforderungen analysiert und spezifiziert
- Integrierbarkeit, Wartbarkeit, Änderbarkeit, Portierbarkeit und Skalierbarkeit sollen berücksichtigt werden

Chaos zu Struktur / Mud-to-structure

Layers

Pipes und Filter

Blackboard

Domain-driven Design

Verteilte Systeme

- Verteilung von Ressourcen und Dienste in Netzwerken
- Kein "zentrales System" mehr
- Basiert auf guter Infrastruktur lokaler Datennetze

Verteilte Systeme

Serviceorientierte Architektur (SOA)

Peer-to-Peer

Client-Server

Interaktive Systeme

- Strukturierung von Mensch-Computer-Interaktionen
- Möglichst gute Schnittstellen für die Benutzer schaffen
- Der eigentliche Systemkern bleibt von der Benutzerschnittstelle unangetastet.

Interaktive Systeme

Model View Controller (MVC)

Model View Presenter

Presentation-Abstraction-Control (PAC)

Adaptive Systeme

- Unterstützung der Erweiterungs- und Anpassungsfähigkeit von Softwaresystemen.
- Das System sollte von vornherein mögliche Erweiterungen unterstützen
- Die Kernfunktionalität sollte davon unberührt bleiben kann.

Adaptive Systeme

Mikrokernel

Reflexion

Dependency Injection

Anti-Patterns

Anti-Patterns?

Ein Anti-Pattern ist in der Softwareentwicklung ein häufig anzutreffender schlechter Lösungsansatz für ein bestimmtes Problem. Es bildet damit das Gegenstück zu den Mustern (Entwurfsmuster, Analysemuster, Architekturmuster, …), welche allgemein übliche und bewährte Problemlösungsansätze beschreiben.

Überblick über Antipatterns

Projektmanagement Anti-Patterns

Architektur Anti-Pattern

Code Smells

Organisations Anti-Pattern

Projektmanagement Anti-Patterns

Blendwerk

Aufgeblähte Software

Feature creep

Scope creep

Brooks’sches Gesetz

Death Sprint

Death March

Architektur Anti-Patterns

Big Ball of Mud

Gasfabrik

Gottobjekt

Innere-Plattform-Effekt

Spaghetticode

Sumo-Hochzeit

Code Smells

Zwiebel

Copy and Paste

Lavafluss

Magische Werte

Reservierte Wörter

Unbeabsichtigte Komplexität

Organisations Anti-Pattern

Wunderwaffe

Das Rad neu erfinden

Das quadratische Rad neu erfinden

Body ballooning

Empire building

Warme Leiche

Single head of knowledge

Organisations Anti-Pattern II

Management nach Zahlen

Vendor Lock-In

Design by Committee

Boat Anchor

Dead End

Swiss Army Knife

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Dokumentation von Architekturen

Nutzen von Templates

Beispiele:

- arc42
- Normen
- Software Guidebook

ARC42

(Dr. Gernot Starke / Dr. Peter Hruschka)

http://www.arc42.de/

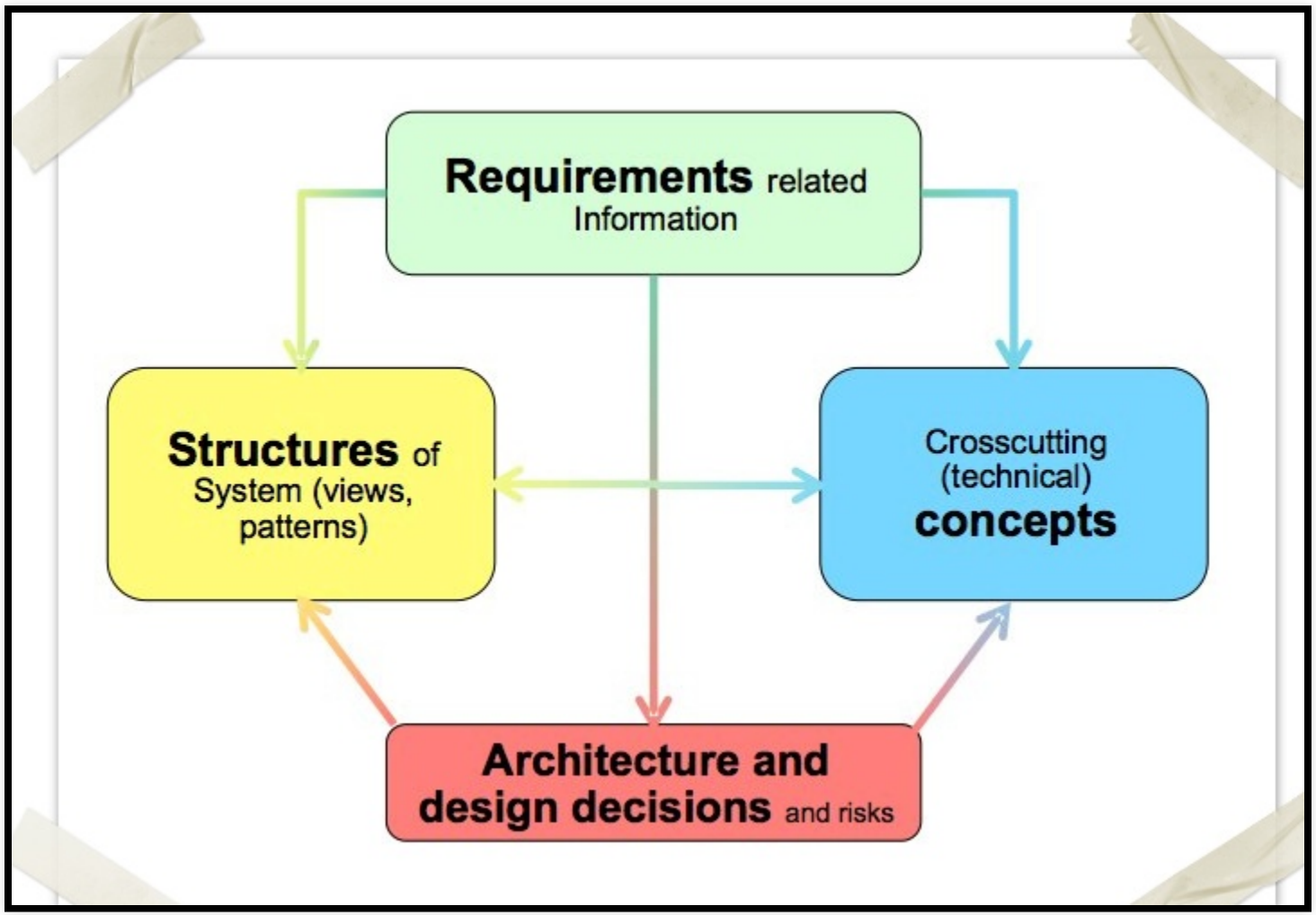
arc42 unterstützt Software- und Systemarchitekten. Es kommt aus der Praxis und basiert auf Erfahrungen internationaler Architekturprojekte und Rückmeldungen vieler Anwender.

Dokumentation von Architekturen

ARC42

1. Einführung und Ziele
2. Randbedingungen
3. Kontextabgrenzung
4. Lösungsstrategie
5. Bausteinsicht
6. Laufzeitsicht
7. Verteilungssicht
8. Querschnittliche Konzepte/Muster
9. Entwurfsentscheidungen
10. Qualitätsszenarien
11. Risiken
12. Glossar

ARC42


Requirements related Information
Structures of System (views, patterns)
Crosscutting (technical) concepts
Architecture and design decisions and risks

IEEE Standards

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards or implementations thereof.

IEEE Standards

- IEEE 802: LAN
- IEEE 802.3: Carrier sense multiple access with collision detection (CSMA/CD)
- IEEE 802.11: Wireless LAN
- IEEE 830: Recommended Practice for Software Requirements Specifications
- IEEE 1394: FireWire/i.Link Bussysteme
- IEEE 1471: Recommended Practice for Architectural Description of Software-Intensive Systems
- IEEE 9945: Portable Operating System Interface (POSIX®)

IEEE Standards - Kosten

- IEEE 830: 171$
- Journals: 26.500$ / Jahr
- Standards Library: *custom quote*

Software Guidebook

- Template von Simon Brown aus dem Buch *"Software Architecture for Developers"*
- Buch: https://leanpub.com/software-architecture-for-developers
- Beispiel: https://leanpub.com/techtribesje (kostenlos)

Software Guidebook

Welche Informationen wünsche ich mir, wenn ich in ein neues Projekt komme?

- Karten
- Sichten
- Geschichte
- Praktische Informationen!

Software Guidebook

Product vs project documentation

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Frameworks

Was ist ein Framework?

Frameworks

Ein Framework ist kein fertiges Programm, es stellt einen Rahmen zur Verfügung.

Frameworks

- Ein Framework ist eine semi-vollständige Applikation.
- Es stellt für Applikationen eine wiederverwendbare, gemeinsame Struktur zur Verfügung.
- Entwickler integrieren dass Framework in ihre eigene Applikation ein, und erweitern es um die Applikationslogik.
- Frameworks stellen eine kohärente Struktur zur Verfügung, anstatt eine einfache Menge von Hilfsklassen anzubieten.

Frameworks

- Ein Framework gibt in der Regel die Anwendungsarchitektur vor.
- Ein Framework definiert den Kontrollfluss der Anwendung
- Ein Framework definierte die Schnittstellen für die Applikation.

Frameworks

Eine allgemeingültige Definition von Frameworks gibt es aufgrund der hohen Anzahl von Diversitäten nicht.

Frameworks

Vorteile

- Wiederverwendung von Code
- Grundfunktionalität muss nicht immer wieder implementiert werden
- Es existieren genormte Schnittstellen z.B. zu Datenbanken
- Frameworks erleichtern die Programmierarbeit und sparen Entwicklungszeit
- Frameworks können den Stil entscheidend verbessern

Frameworks

Nachteile

- Frameworks erhöhen die Komplexität der Anwendung
- Frameworks stecken voller Know-How und eine effiziente Anwendung erfordert Profiwissen
- Frameworks nehmen nicht das Verständnis der Grundlagen ab, auch wenn oft so gearbeitet wird
- Dokumentationen sind größtenteils unzureichend

Frameworks

Wie wähle ich ein Framework aus?

Popularität und Community

Wie wahrscheinlich finde ich Hilfe und Entwickler?

Philosophie

A tool developed by professionals for their own needs will obviously meet the demands of other professionals.

Sustainability / Nachhaltigkeit

Kann das Framework "mitwachsen"?

Support

Gibt es professionelle Hilfe neben der Community?

Technik

Wie gut ist das Framework implementiert?

Security

Wie schnell werden Sicherheitslücken reportet und geschlossen?

Dokumentation

Wie gut, ausführlich und verständlich ist das Framework dokumentiert?
Wie aktuell ist die Doku?

Lizenz

Ein Framework unter GPL Lizenz verlangt z.B., dass die Anwendung auch unter der GPL steht. MIT dagegen nicht.

Entwickler-Kapazität

Wie wahrscheinlich werde ich Entwickler finden?

Hosting Requirements

Wie einfach kann ich die Anwendung deployen?

Einfache Installation?

Wie schnell ist ein neues Projekt eingerichtet?

Lernkurve

Wie komplex ist das Framework?

Inhalte / Funktionen?

- AJAX
- Authentication
- Authorization
- Caching
- Data Validation
- Templating engine
- URL mapping / rewriting
- ...?

DB Abstraktion / ORM

Wie einfach/mächtig ist das Object Relational Mapping?

JS Library

Welche JS Bibliothek ist per default dabei?

Unit Testing

Wie sehr ist TDD Teil der Philosophie, wie ist der Tool-Support?

Skalierbarkeit?

Wie einfach lässt sich die Anwendung bei Bedarf skalieren?

Ausprobieren!

Reviews lesen reicht nicht, Erfahrungen und das look&feel zählen!

Wann brauche ich ein Framework?

- Die Anwendung basiert im Wesentlichen auf CRUD Operationen
- Die Anwendung wird relativ groß
- UI und Anwendungslogik sollen getrennt werden
- Authentication und andere Grundfunkionen werden intensiv genutzt
- Zeitdruck + Das Framework ist bereits bekannt

Wann brauche ich KEIN Framework?

- Ich brauche nur einen kleinen Teil des Frameworks (z.B. ORM)
- Zeitdruck + Das Framework ist nicht bekannt
- "Frameworks lösen jedes Problem"

Vorbereitung auf Klausuraufgaben

Klausur

- 5.7.2018
- 90 Minuten
- keine Hilfsmittel

Vorbereitung auf Klausuraufgaben

Grundlagen & Geschichte

- Was waren die Gründe für Softwarearchitektur?
- Nennen Sie drei typische Aufgaben von Softwarearchitekten
- Welche nicht-funktionalen Anforderungen sollten Sie beim Architekturentwurf ggf. berücksichtigen?
- Was besagt das Brook'sche Gesetz?
- Was besagt 'Conways Law'?
- In welcher Beziehung stehen Architektur und Design?

Vorbereitung auf Klausuraufgaben

Sichten

- Welche Arten von Sichten gibt es?
- Was sollte eine Kontext-Sicht enthalten?
- Was enthält eine Verteilungssicht?
- Sind mehr als die vier klassischen Sichten sinnvoll?

Vorbereitung auf Klausuraufgaben

Architekturmuster

- Welche Aufgabe haben Architekturmuster?
- Nennen und erläutern Sie drei Arten von Architekturmustern
- Nennen und erläutern Sie drei Anti-Patterns
- Für welche Systeme würden Sie eine Blackboard-Architektur verwenden?
- Für welche Systeme wird das MVC Muster typischerweise verwendet?
- Für welche Systeme wird das MVP Architekturmuster typischerweise verwendet?
- Für welche Systeme wird das Microkernel Architekturmuster typischerweise verwendet?

Vorbereitung auf Klausuraufgaben

Dokumentation

- Was ist der Grundgedanke hinter dem "Software Guidebook"?
- Wie ist das ARC42 Template entstanden?

Vorbereitung auf Klausuraufgaben

Frameworks

- Was ist ein Framework?
- Wann ist die Verwendung eines Frameworks sinnvoll?
- Wann ist die Verwendung eines Frameworks nicht sinnvoll?
- Nennen sie drei Vorteile für die Verwendung von Frameworks!
- Nennen sie drei Nachteile für die Verwendung von Frameworks!
- Nennen sie drei Auswahlkriterien für Frameworks!
- Was ist Dependency Injection?
- Nennen Sie drei Beispiele für Crosscutting Concerns

Fragen?

Unterlagen: ai2018.nils-loewe.de