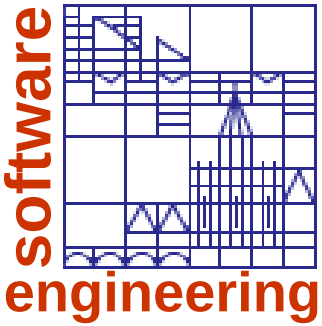
Entwurfsphase

Teil I: Grobentwurf (system design)

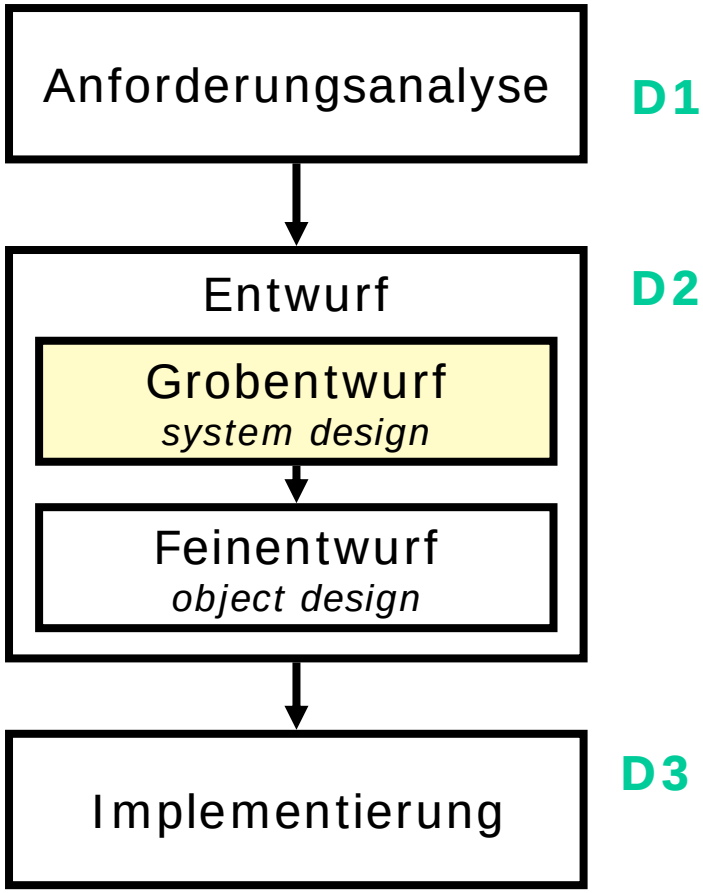
Johannes Leitner
leitner@inf.uni-konstanz.de

Universität Konstanz
Lehrstuhl für Software Engineering – Prof. Stefan Leue

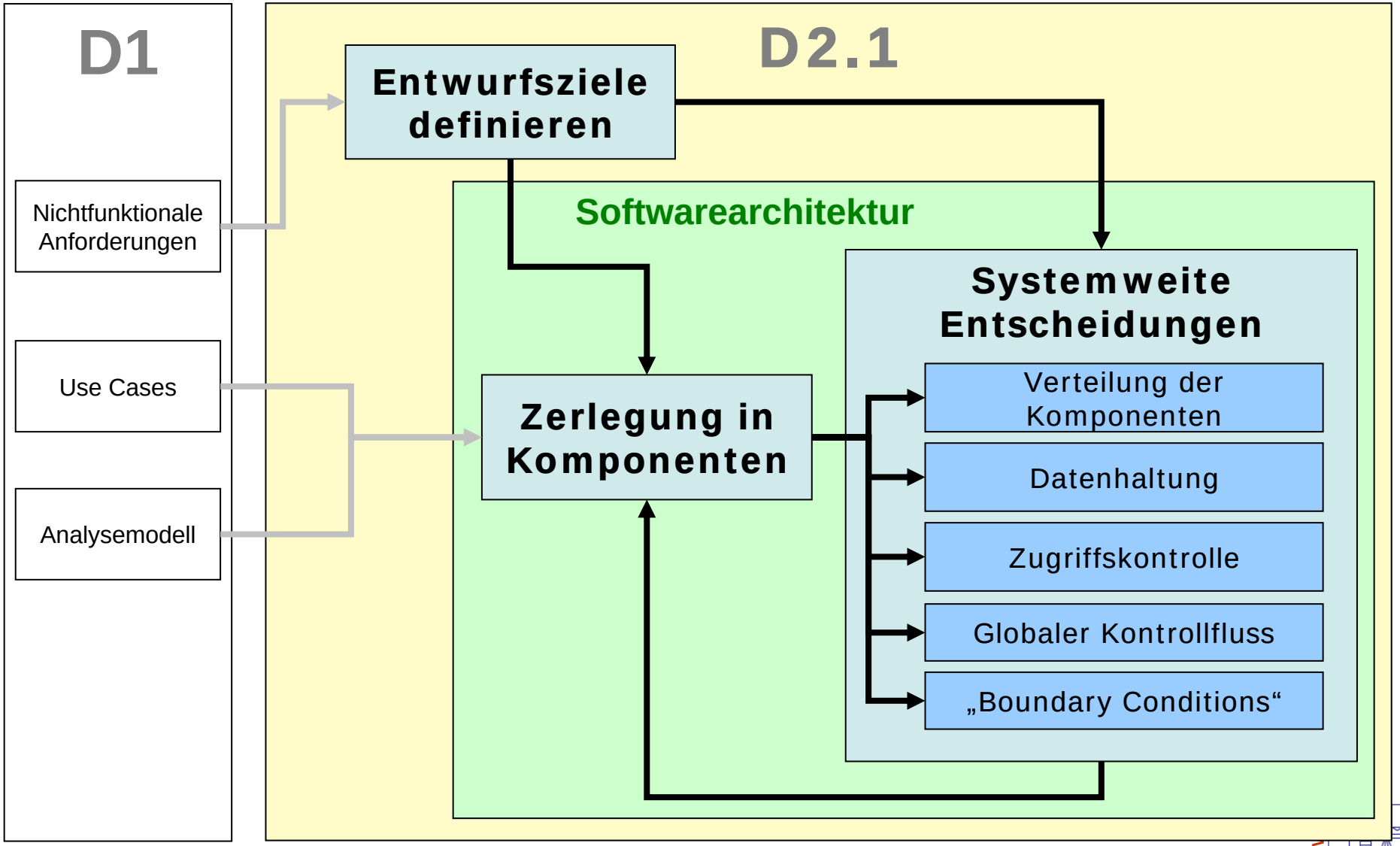


System Design

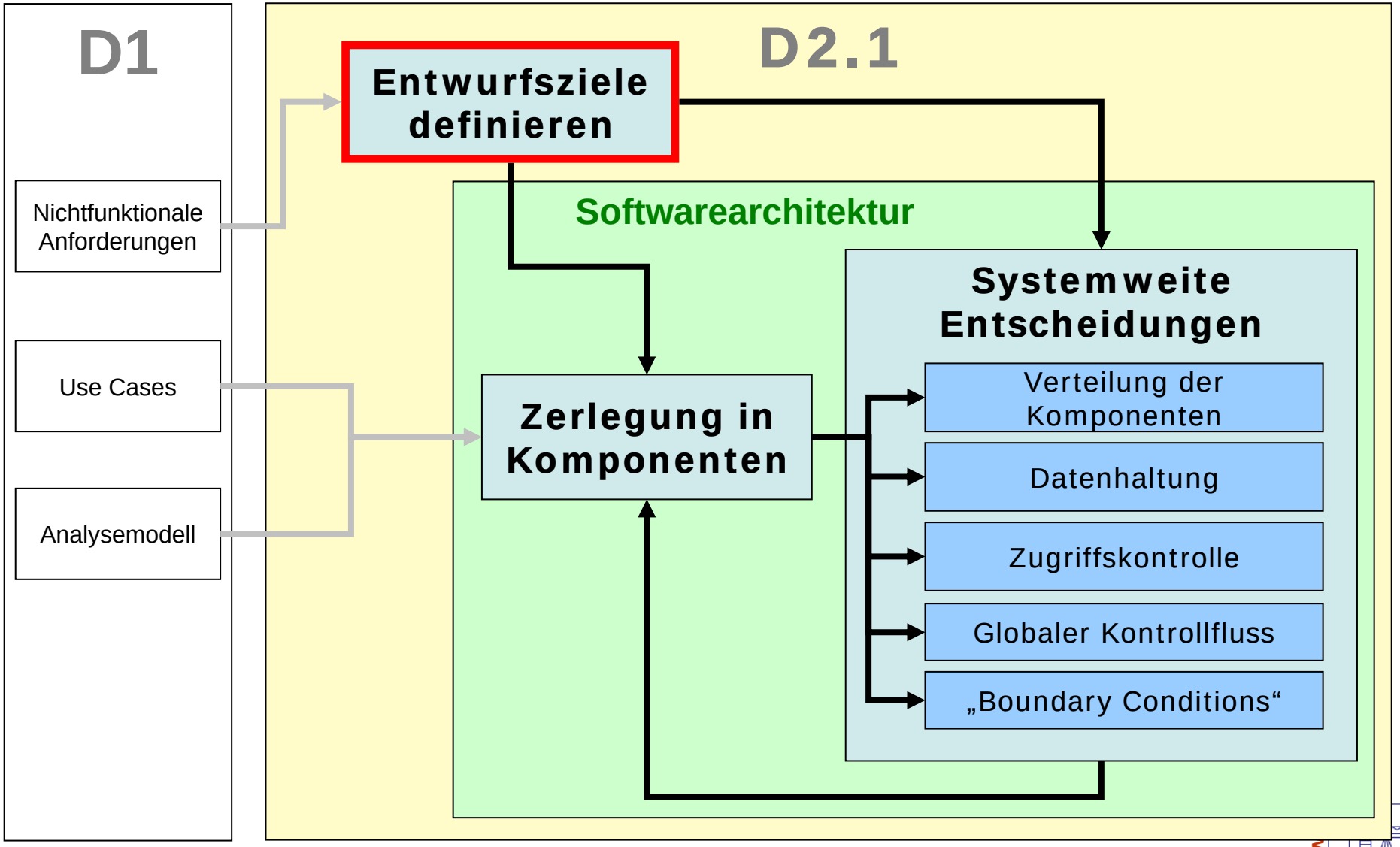
- Ergebnisse des Grobentwurfs:
  - **Entwurfsziele**
    - Beeinflussen alle weiteren Entwurfsentscheidungen
  - und **Systemarchitektur**
    - Zerlegung des Systems in handhabbare Teile



Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Systemweite Entscheidungen
Zerlegung in Komponenten
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

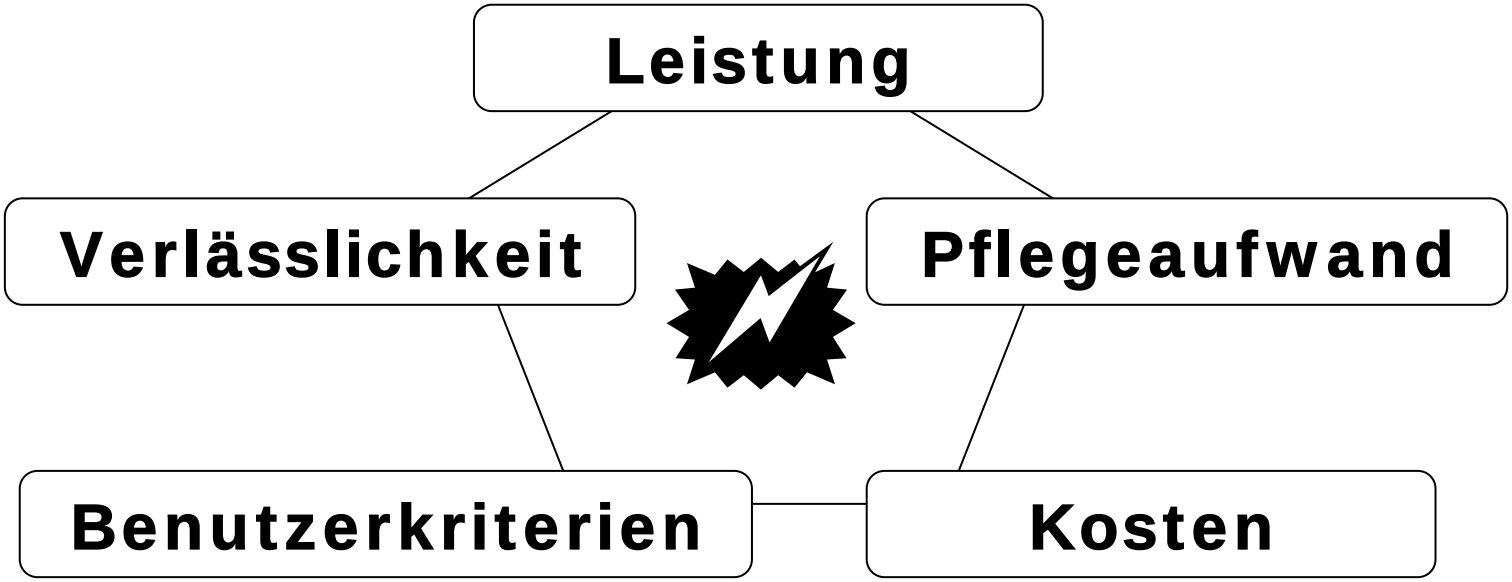
Entwurfsziele

- Erster Schritt im Systementwurf
- Sinn:
  Jede Entwurfsentscheidung kann anhand dergleichen Kriterien gefällt (und gerechtfertigt) werden
- Beeinflusst durch
  - Nichtfunktionale Anforderungen
  - Z.T. Analysemodell
  - Zusätzliche Kundenwünsche
- Vorgehensweise: Aus Liste allgemein erstrebenswerter Eigenschaften **auswählen**

Entwurfsziele

Gruppen

- Entwurfsziele lassen sich grob in Gruppen unterteilen:



Entwurfsziele

Leistung

- **Antwortzeit**
  - Wie schnell wird auf Benutzeranfragen geantwortet?
- **Durchsatz** (Throughput)
  - Wie oft kann das System seine Aufgaben in einer gegebenen Zeit erfüllen?
- **Speicherbedarf**

Entwurfsziele

Verlässlichkeit

- **Robustheit**
  - Wie gut kann das System mit ungültigen Benutzeingaben umgehen?
- **Zuverlässigkeit** (reliablity)
  - Wie groß ist die Abweichung von spezifiziertem und tatsächlichem Verhalten?
- **Verfügbarkeit**
  - Zeitanteil, in dem das System normal verwendet werden kann
- **Fehlertoleranz**
  - Fähigkeit, im Fehlerzustand weiter funktionsfähig zu bleiben
- **Security**
  - Widerstandsfähigkeit gegen bösartige Angriffe
- **Safety**
  - Sicherheit von Personen, etc. auch unter fehlerhaften Bedingungen

Entwurfsziele

Kosten

- **Entwicklungskosten**
- **Deployment-Kosten**
  - Z.B. Vor-Ort-Installation und Schulungen
- **Upgradekosten**
  - Kosten für das Ersetzen des alten Systems (→ Backwards Compatibility)
- **Wartungskosten**
  - Bug Fixes, Entwicklung von Erweiterungen
- **Verwaltungskosten**
  - Administration

Entwurfsziele

Wartbarkeit

- **Erweiterbarkeit**
  - Wie schwer ist es, neue Funktionalität zum fertigen Produkt hinzuzufügen?
- **Veränderbarkeit**
  - Wie schwer ist es, vorhandene Funktionalität im fertigen Produkt anzupassen?
- **Anpassbarkeit**
  - Kann das System leicht auf eine andere Anwendungsdomäne übertragen werden?
- **Portierbarkeit**
  - Kann das System leicht auf eine andere Plattform übertragen werden?
- **Lesbarkeit**
  - Kann man das System leicht durch Lesen des Codes verstehen?
- **Requirement Traceability**
  - Wie schwer ist es, Teile des Systems zu den Anforderungen zurückzuverfolgen?

Entwurfsziele

Benutzerkriterien

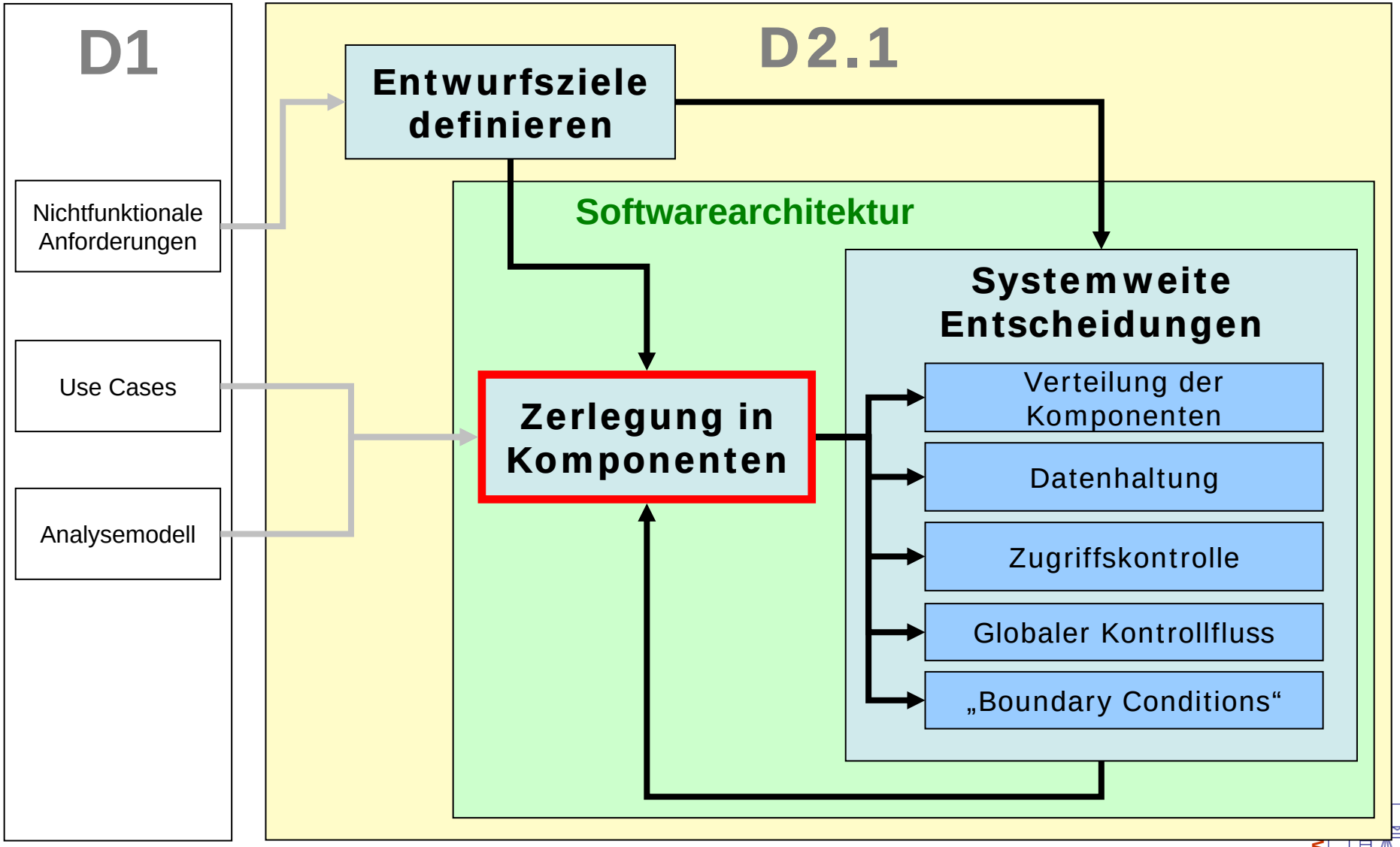
- **Nützlichkeit (utility)**
  - Wie gut unterstützt das System den Benutzer?
- **Benutzbarkeit (usability)**
  - Wie leicht fällt es dem Benutzer, das System zu verwenden?

Auswahl der Entwurfsziele

- Anforderungen widersprechen sich teilweise
  - Antwortzeit vs. Speicherverbrauch
  - Entwicklungskosten vs. Erweiterbarkeit
  - usw.

- Definition der Ziele
  - Auswählen anhand der **nichtfunktionalen Anforderungen**
  - **Trade-offs** abwägen
  - Widersprüchliche Ziele **priorisieren**
    - z.B. „Der Speicherverbrauch soll minimal sein, solange das die Antwortzeit nicht beeinträchtigt.“
  - Dokumentieren
    - In der Einleitung des Systementwurfsdokuments

Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

Zerlegung in Komponenten

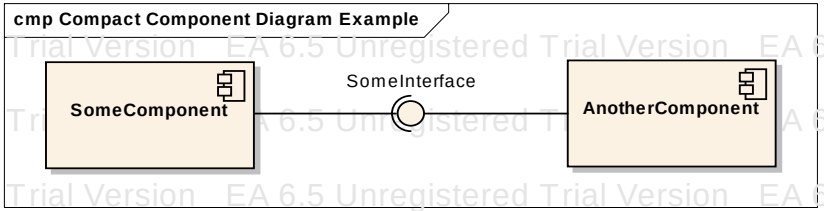
- **Gesamtsystem unüberschaubar komplex**
  - Aufspaltung in handhabbare Teile
  - Häufig: Teile, die von einzelnen Entwicklern oder Teilgruppen bearbeitet werden können
- Zerlegung in Komponenten = **Systemarchitektur**
- Wichtigste Aktivität des Systementwurfs

Was ist eine Komponente?

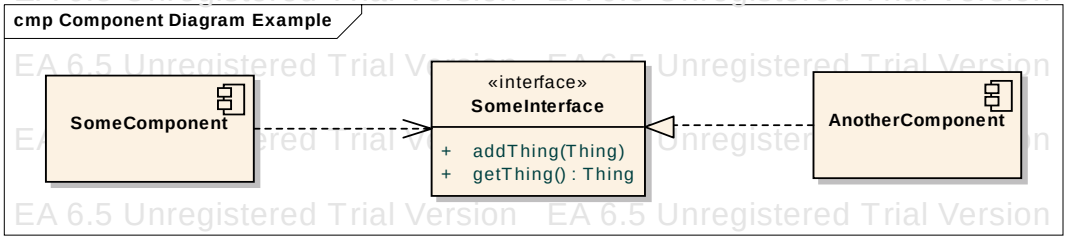
- **Teil des Systems, der nach aussen bestimmte Dienste anbietet**
  - Dienst (service) = Menge verwandter Operation mit gemeinsamem Zweck
- **Spezifikation dieser Operationen** = Schnittstelle der Komponente
  - Wichtiger Teil der Architektur

Komponenten
Darstellung

- Notation im Rahmen dieses Projektes orientiert am **UML 2.0 Komponentendiagramm**
- Verschiedene gleichwertige Darstellung möglich



Kurzdarstellung



Detaillierte Darstellung mit Schnittstellendefinition

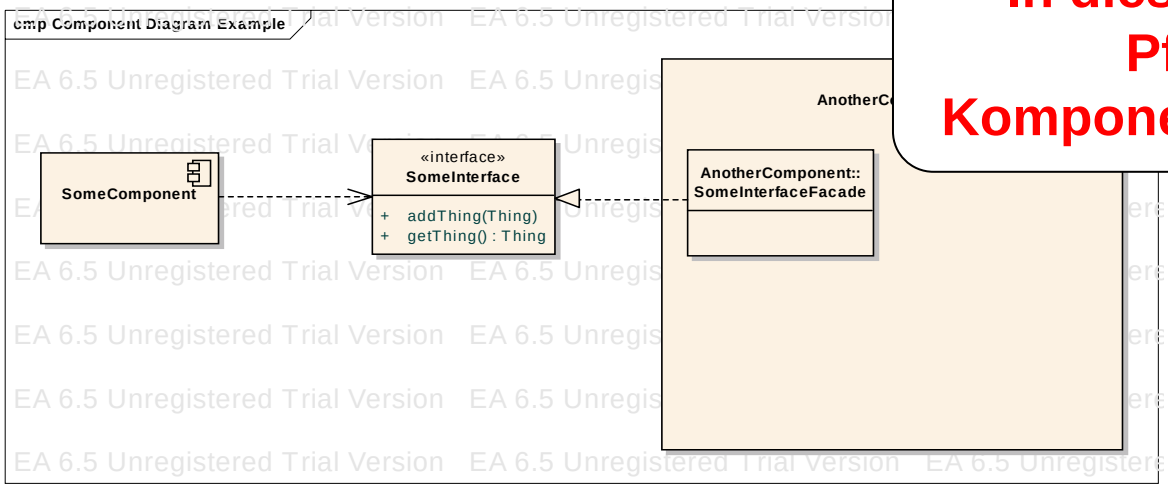
- Komponenten enthalten Klassen (oder auch andere Komponenten)

Komponenten

Bedeutung in Java / Implementierung

- Wir meinen hier nicht Komponenten im Sinne von EJBs
- Komponenten werden in der Implementierung **Packages**
- **Schnittstelle** = Menge aller öffentlichen Methoden von öffentlichen Klassen

- **Facade** Pattern:
  - „Klasse, die eine einheitliche, vereinfachte Schnittstelle für ein Teilsystem zur Verfügung stellt“
  - Häufig bei der Implementierung der Schnittstellen zwischen Komponenten verwendet

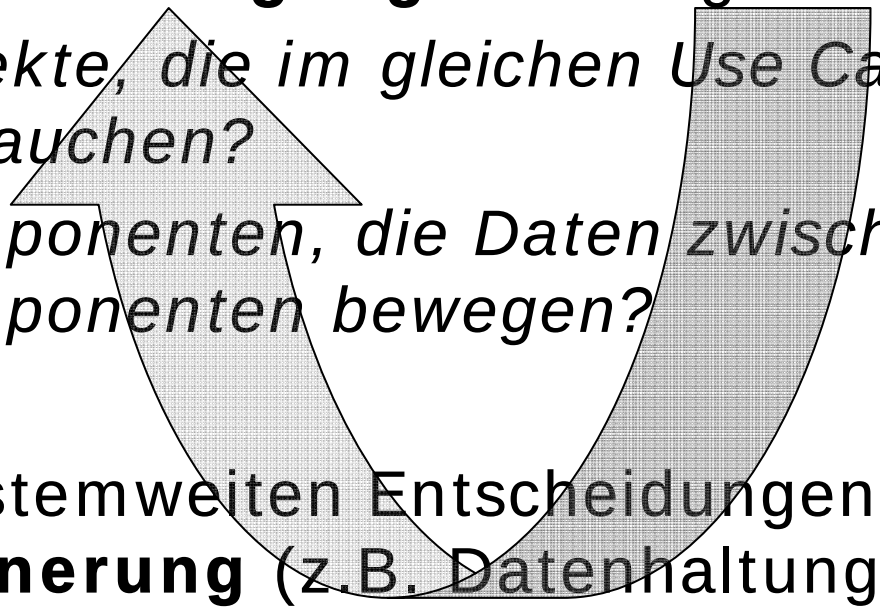


In dieser Veranstaltung Pflicht für alle Komponentenschnittstellen!

software engineering

Finden einer „guten“ Zerlegung?

- Kein Patentrezept
- Empfehlenswert: Iteratives Vorgehen



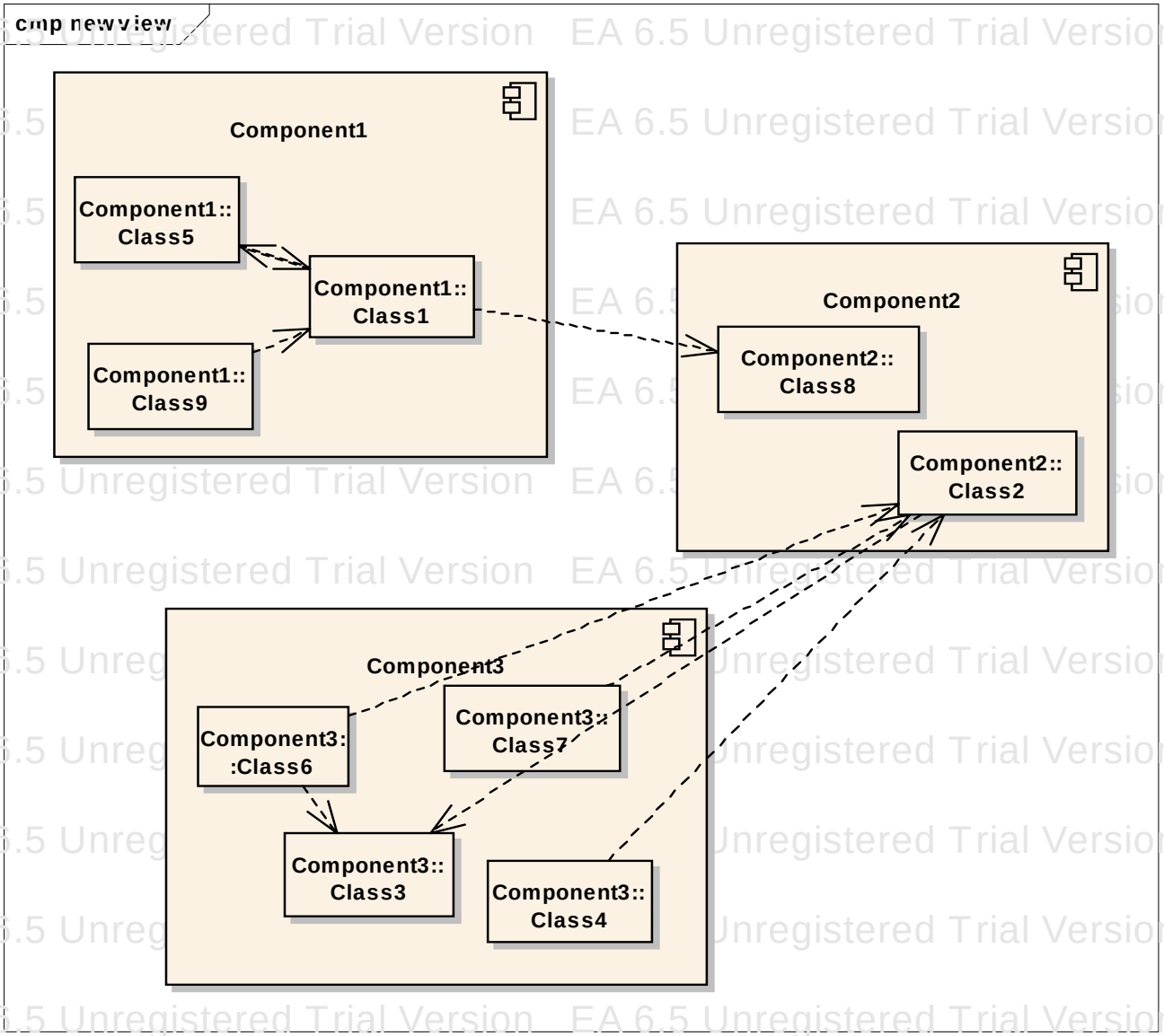
  - **Initiale Zerlegung** aus D1 gewinnen
    - *Objekte, die im gleichen Use Case auftauchen?*
    - *Komponenten, die Daten zwischen anderen Komponenten bewegen?*
  - Bei systemweiten Entscheidungen **Verfeinerung** (z.B. Datenhaltung, Sicherheit, etc.)

Bewertung einer Architektur

- **Coupling**
  - Anzahl von Dependencies zwischen zwei Komponenten
  - Informell spricht man auch von
    - **strongly coupled**
      - Viele Abhängigkeiten
      - Änderungen in einer Komponenten erfordern Änderungen der anderen
    - **loosely coupled**
      - nicht immer möglich
- **Cohesion**
  - Anzahl von Dependencies innerhalb einer Komponente
  - Geringe Cohesion → Komponente kann vielleicht zerschnitten werden
- Erstrebenswert: loosely coupled, highly cohesive

Coupling & Cohesion

Beispiel



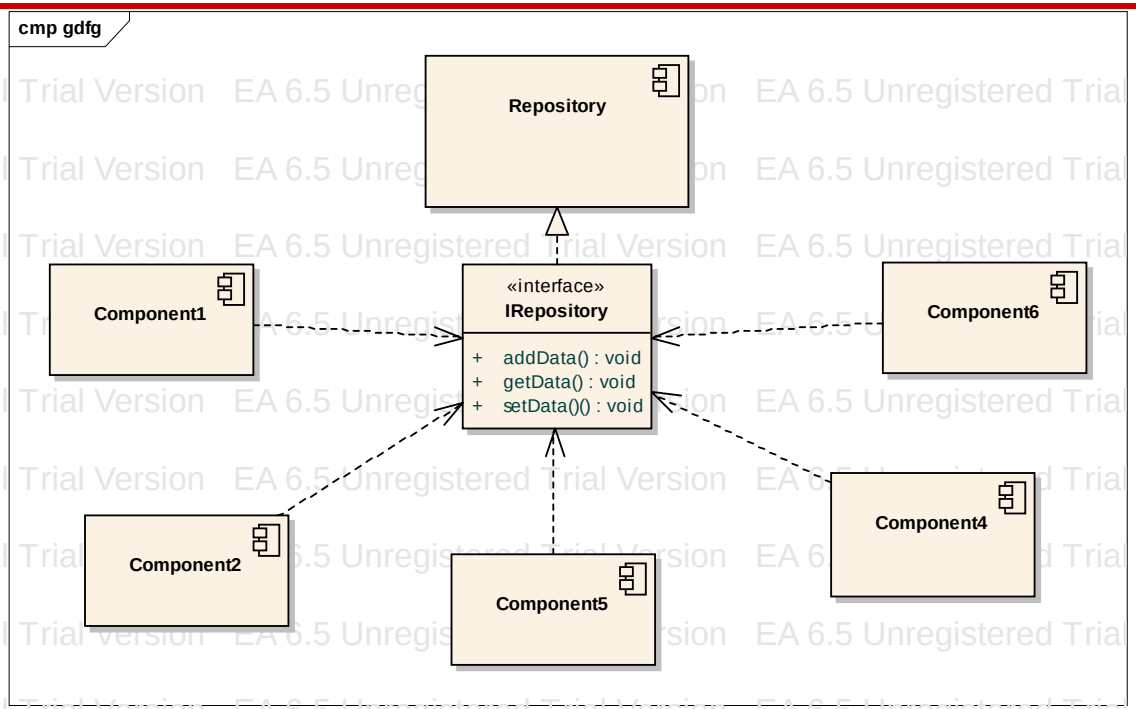
software engineering

Architekturstile

- Gänginge, erprobte Lösungen für Architekturprobleme
- Beschreiben grundlegendes Schema der Struktur eines Systems
- Können als Basis für spezielle Systemarchitektur verwendet werden
- Ähnlich *design patterns*, auf höherer Ebene
  - „architectural patterns“

Architekturstile

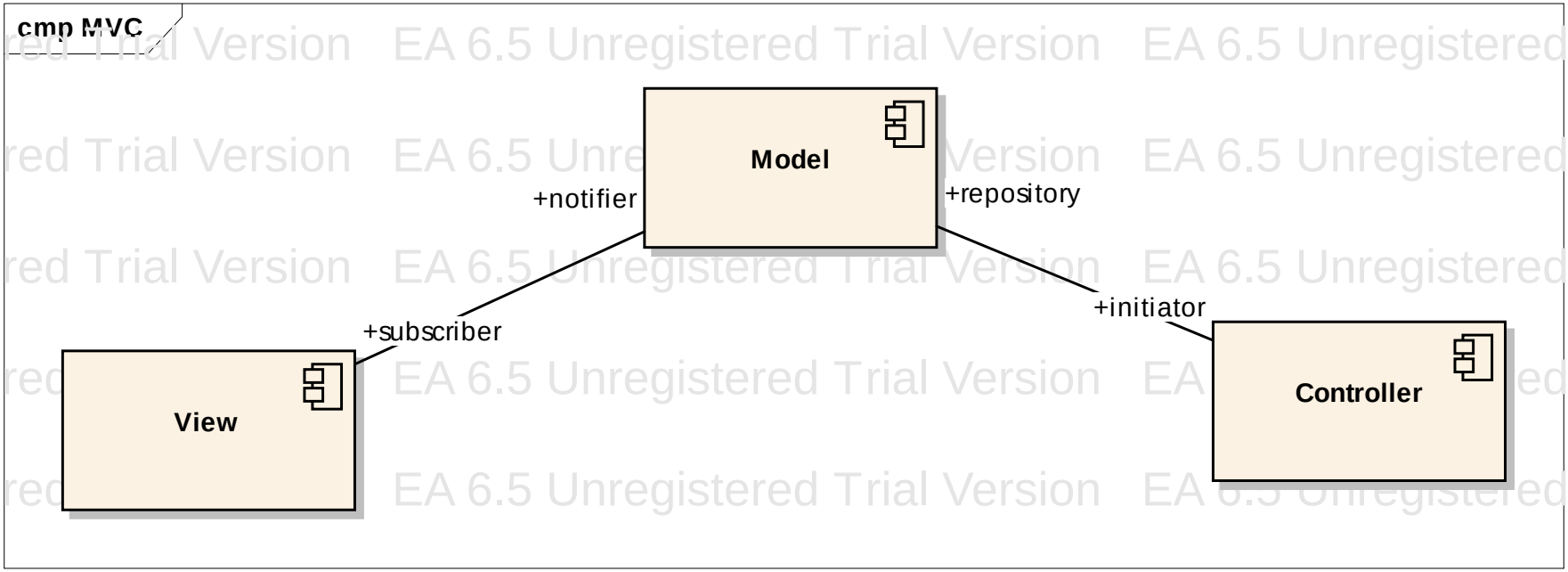
Repository



- Teilsysteme kommunizieren über gemeinsames Repository
  - Repository kann Kontrollfluss über **Observer-Muster** treiben
- Beispiele:
  - Sensor-Aktor-Systeme

Architekturstil

Model – View - Controller



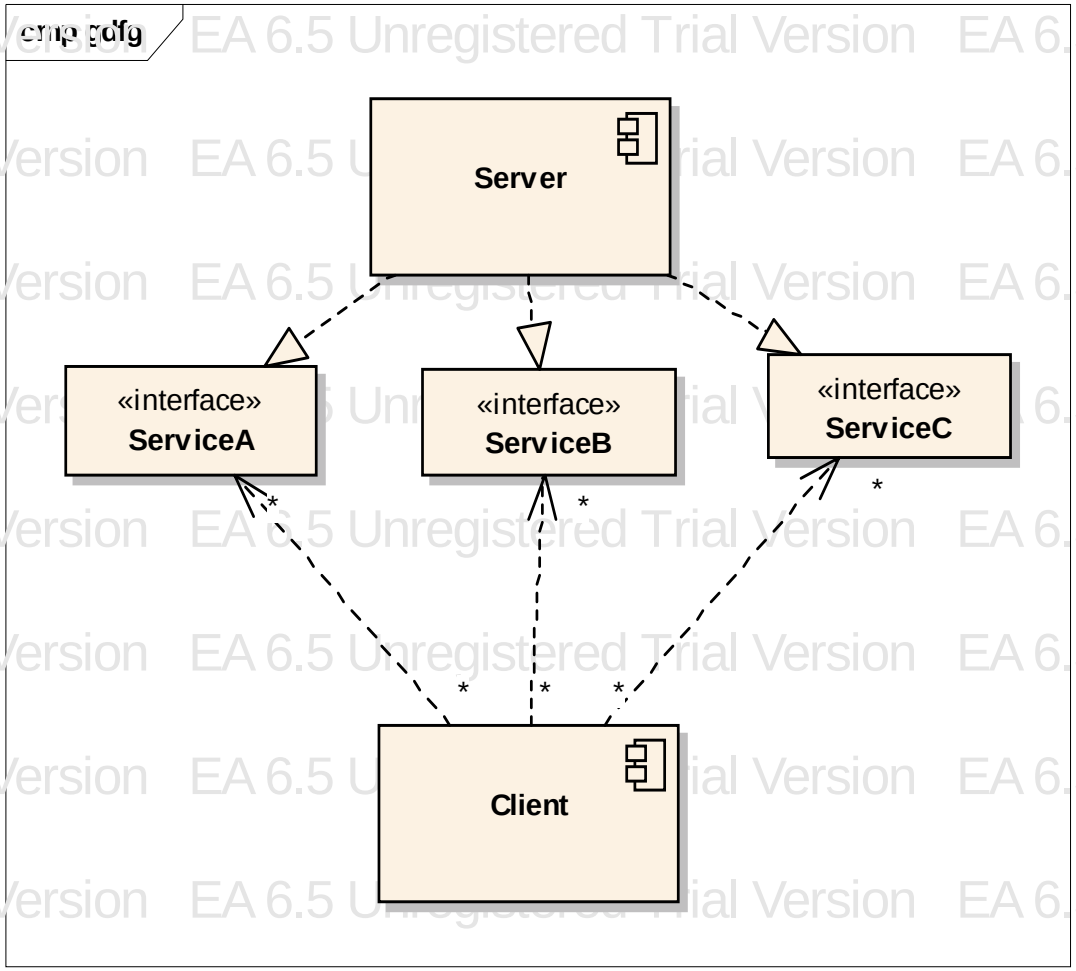
- View = Sicht des Benutzers auf Daten
- Model = Repräsentation der Daten
- Controller = Logik zum Ändern von Daten

- Ursprung: Smalltalk ~1980
  - Immer noch einer der wichtigsten Stile für Anwendersysteme
- Beispiele
  - GUI-Systeme (Swing, …), Web-Systeme (Struts, …)

Architekturstil

Client - Server

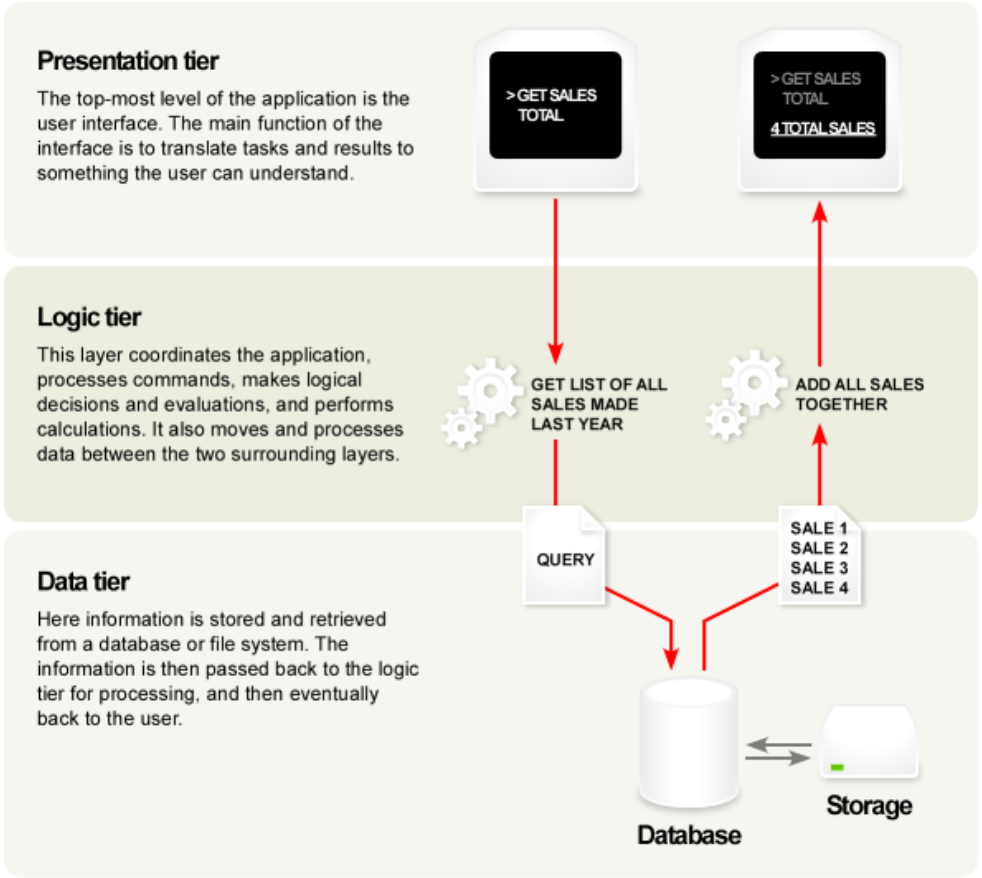
- Einige Server, viele Clients
- Unabhängiger Kontrollfluss, asynchrone Kommunikation
- Sinnvollerweise später auf unterschiedlichen Hardware-komponenten



Architekturstil

Schichten

- Zerlegung in Schichten, die nur paarweise von einander abhängen
- Klassisch: **3 Schichten**
  - Präsentation
  - Anwendungslogik
  - Datenhaltung



- Variation: Vier-Schichten
  - Presentation wird zu
    - Presentation Client
    - Presentation Server
  - Sinnvoll bei Webanwendungen usw.

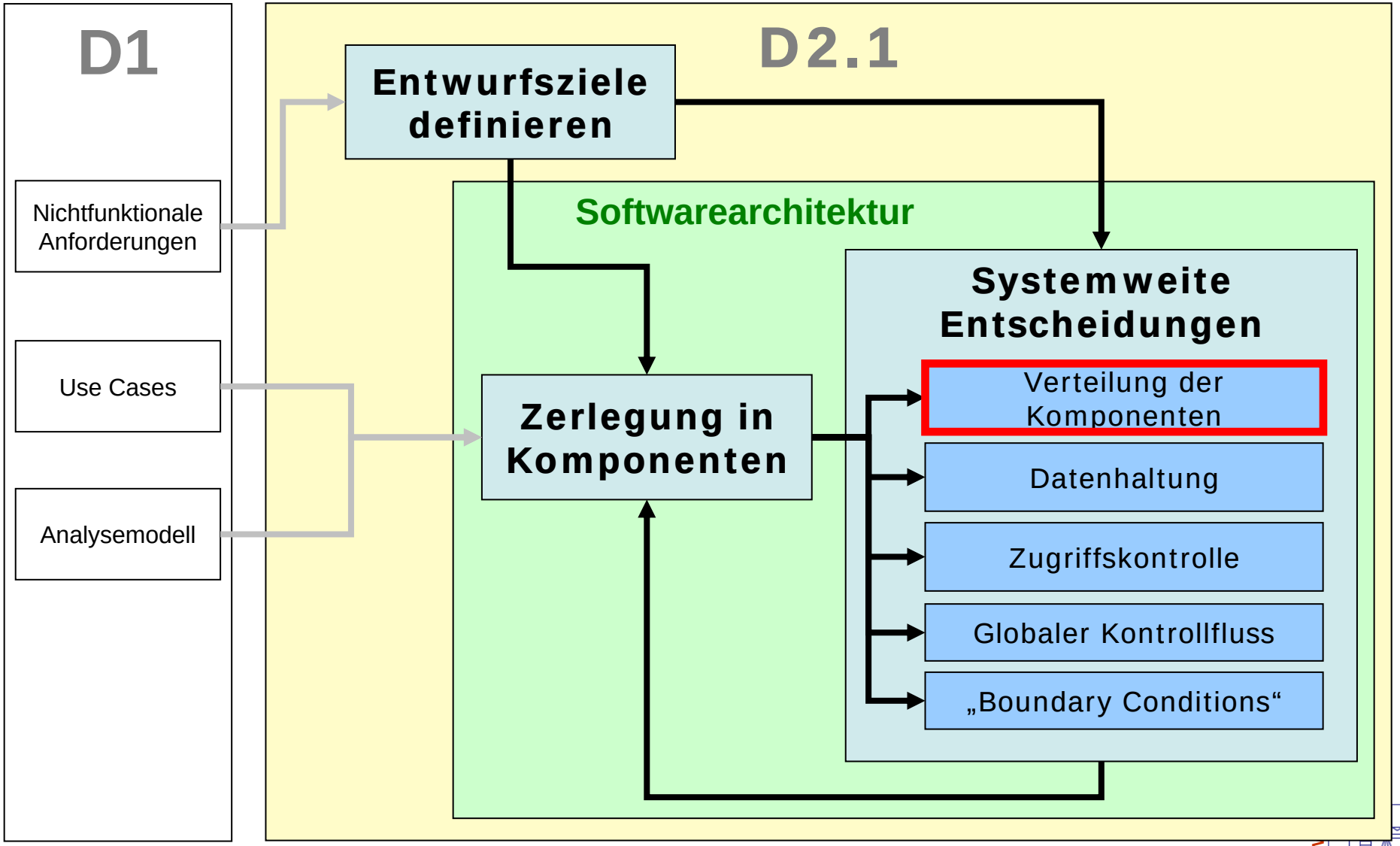
Unterschied zu MVC?

Weitere Architekturstile

- Pipeline (a.k.a. Pipes & Filters)
  - Unix, Compiler
- Peer-to-peer
- Usw.

- Bei Interesse siehe auch
  - Mary Shaw and David Garlan
    Software Architecture: Perspectives on an Emerging Discipline
    - David Garlan and Mary Shaw
      An Introduction to Software Architecture
      http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf
  - Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal
    Pattern-Oriented Software Architecture. A System of Patterns

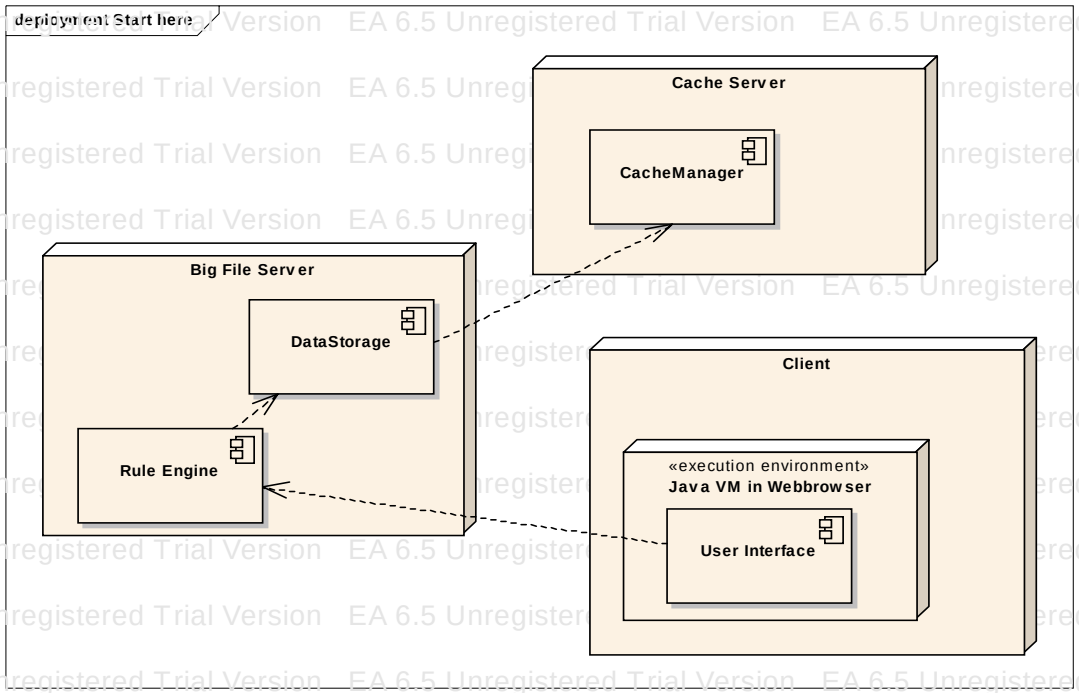
Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

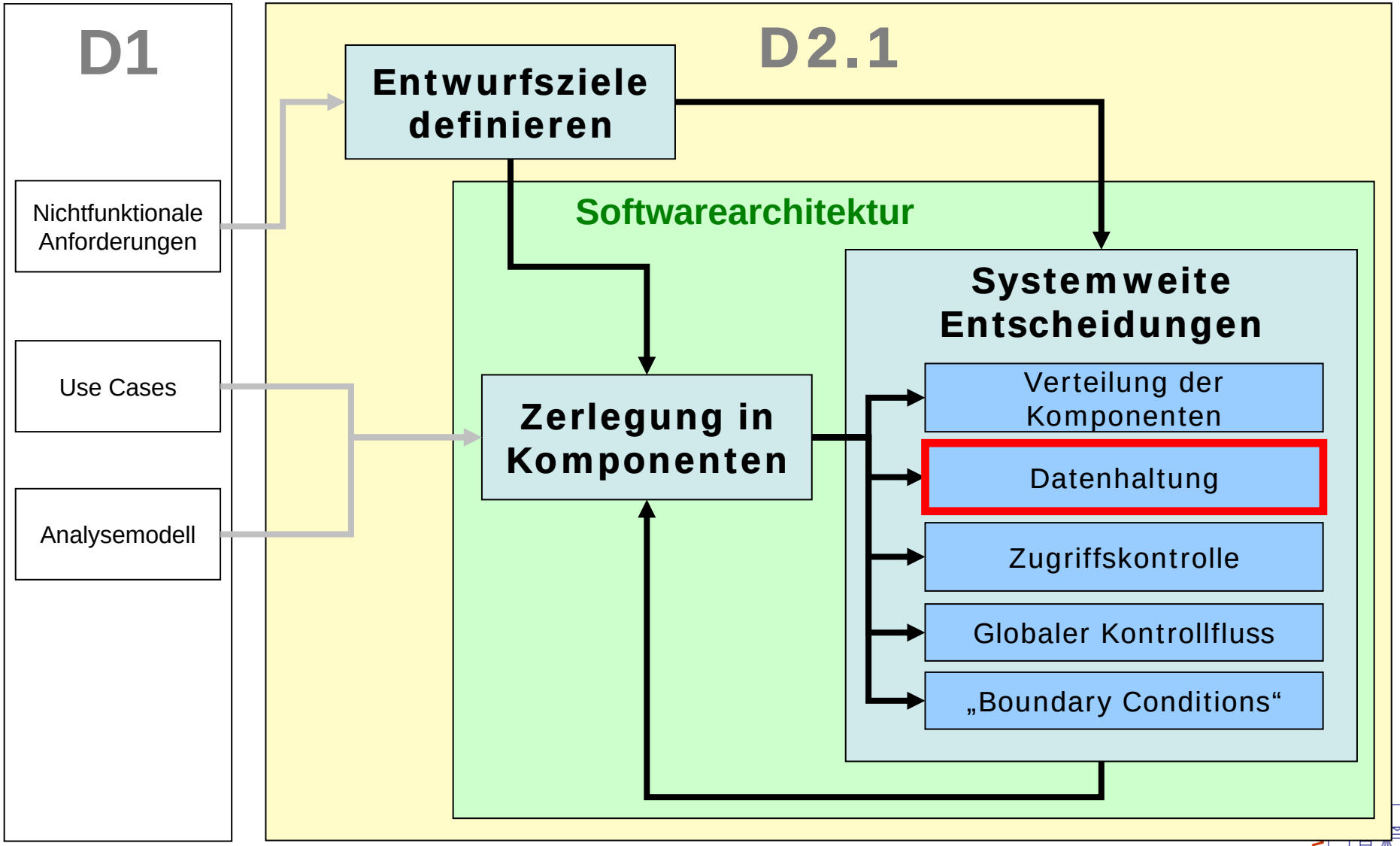
Verteilung der Subsysteme

- Aufteilung der Komponenten auf Rechner, etc.
  - „Was läuft auf dem Client, was auf dem Server?“

- UML Deployment Diagram

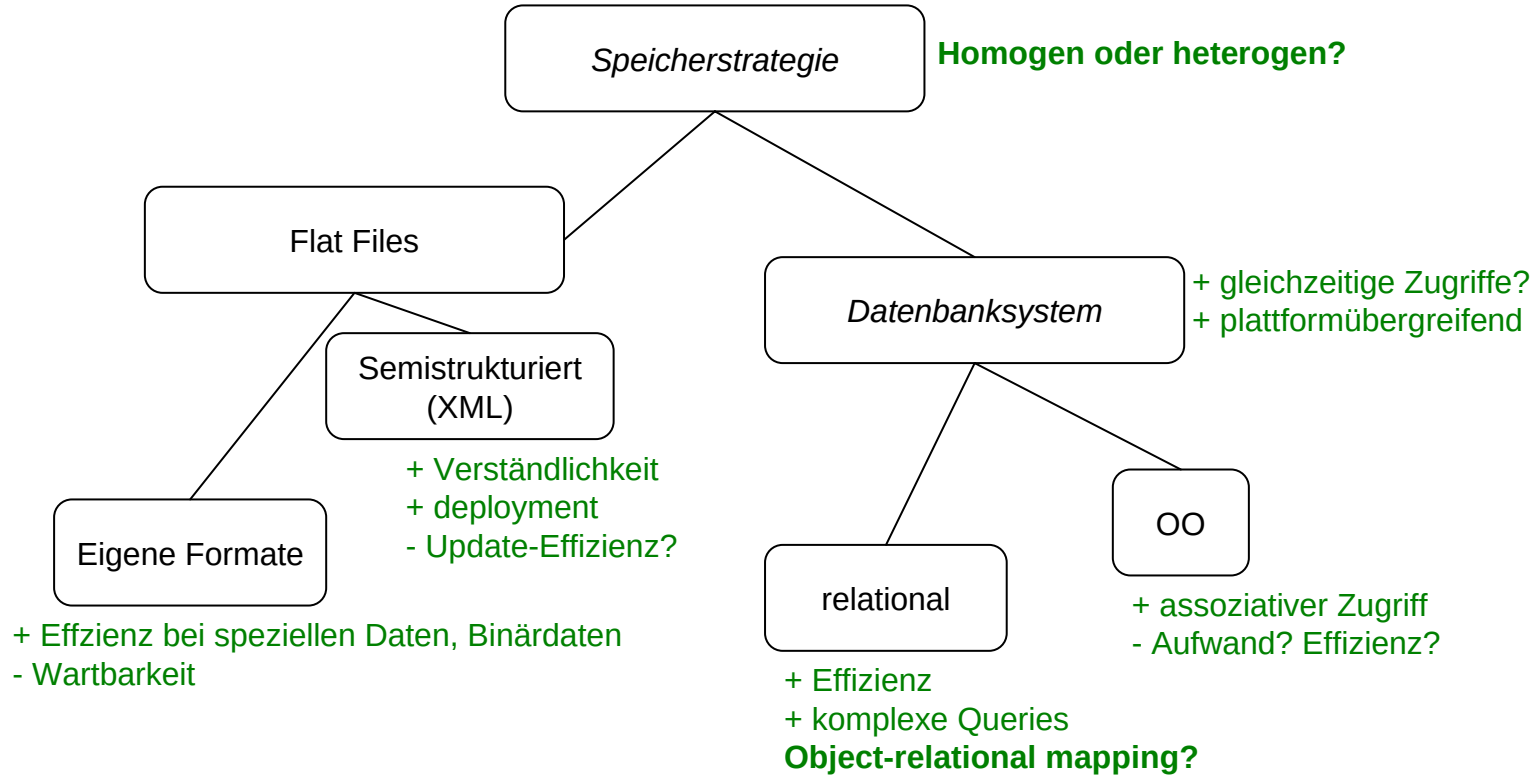


Überblick

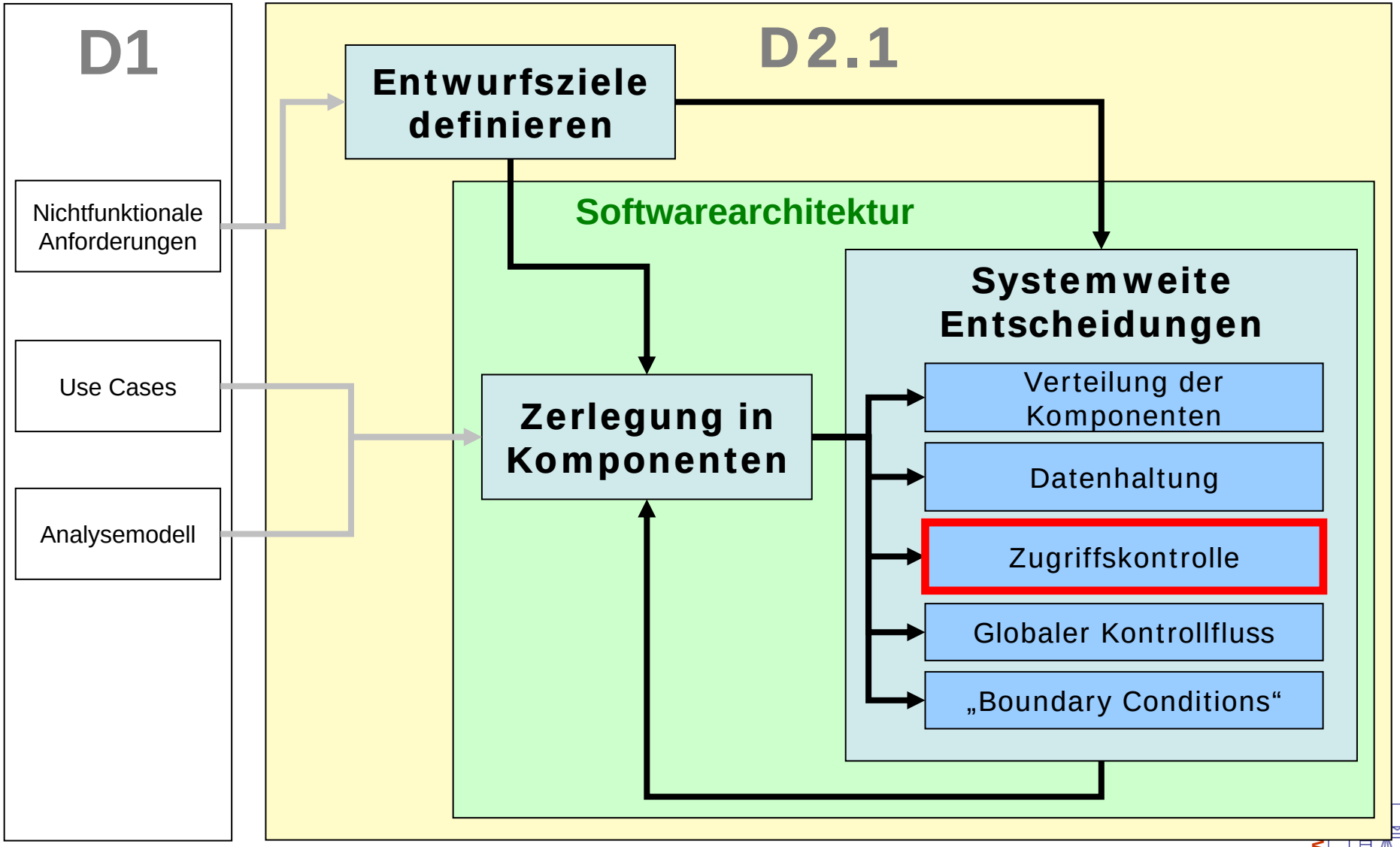

D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

Datenhaltung

- Welche Daten sind persistent?
  - „Was muss einen Systemneustart überleben?“
- Strategie für das Speichern dieser Daten wählen
  - Beeinflusst von Entwurfszielen
  - Kann Architektur beeinflussen



Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

Zugriffsrechte

- Benutzer haben verschiedene Rollen
  - Aktoren in den Use Cases aus D1
- Vorgehensweise
  - Identifiziere Schnittstellen, auf die Aktoren zugreifen
  - Beschreibe Zugriffsrechte über Tabelle
  - **Access Matrix**

	Kursverwaltung	Studienordnung	...
Student	showCourse()	view()	...
Verwaltung	addCourse() delCourse() showCourse()	add() remove()	...
...	...	...	...

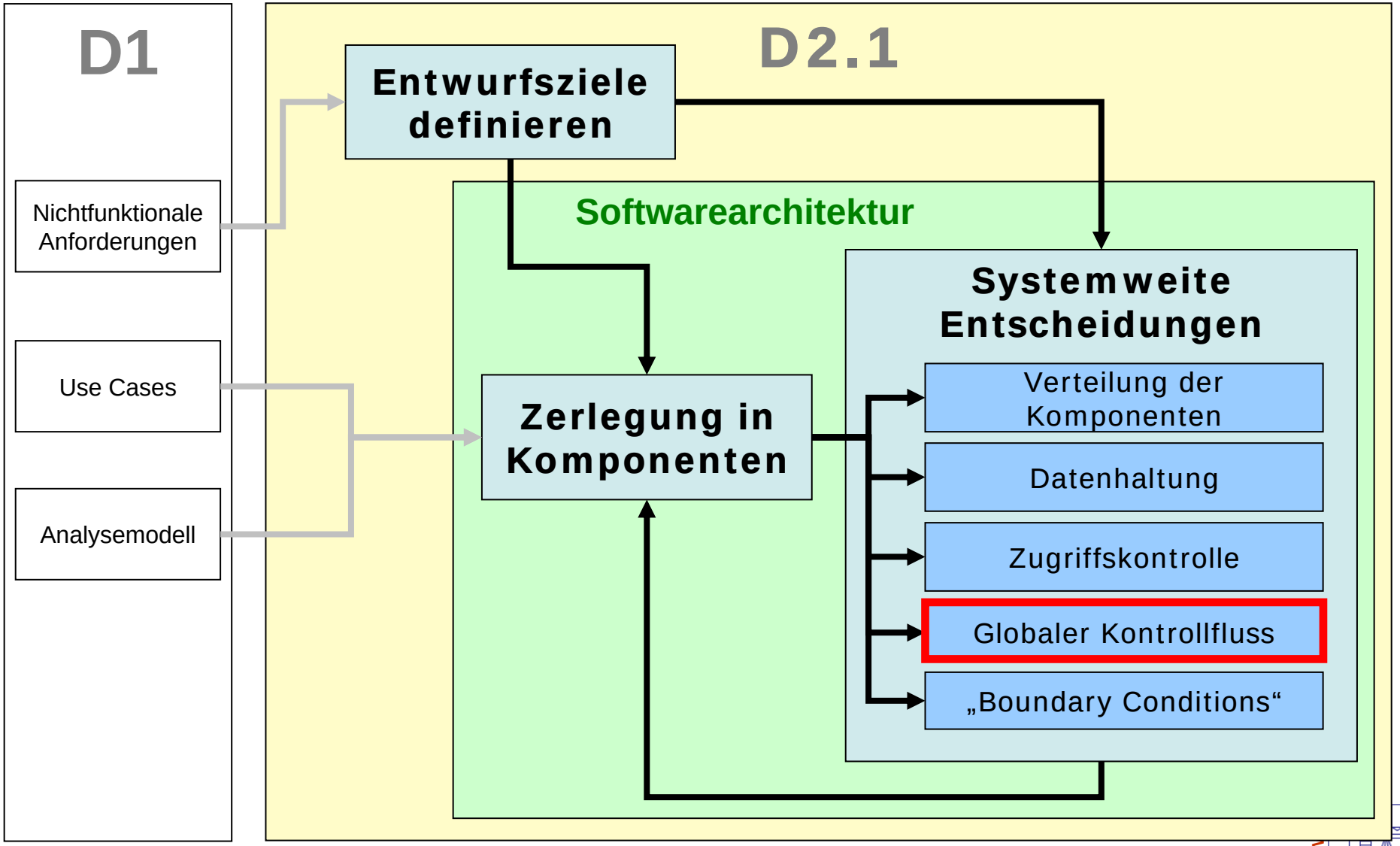
Zugriffskontrolle

- Umsetzung dieser Matrix festlegen
- **Alternativen**
  - Global Access Table
  - Access Control List
    - Assoziiere (Aktor, Operation) mit Klasse
  - Capabilities
    - Assoziiere (Klasse, Operation) mit Aktor

- Weiteres:
  - **Verschlüsselung** der Kommunikation zwischen verteilten Komponenten?
  - **Authentifzierung** der Aktoren

- Normalerweise: Off-the-shelf Lösungen gegenüber Eigenimplemtierungen bevorzugen

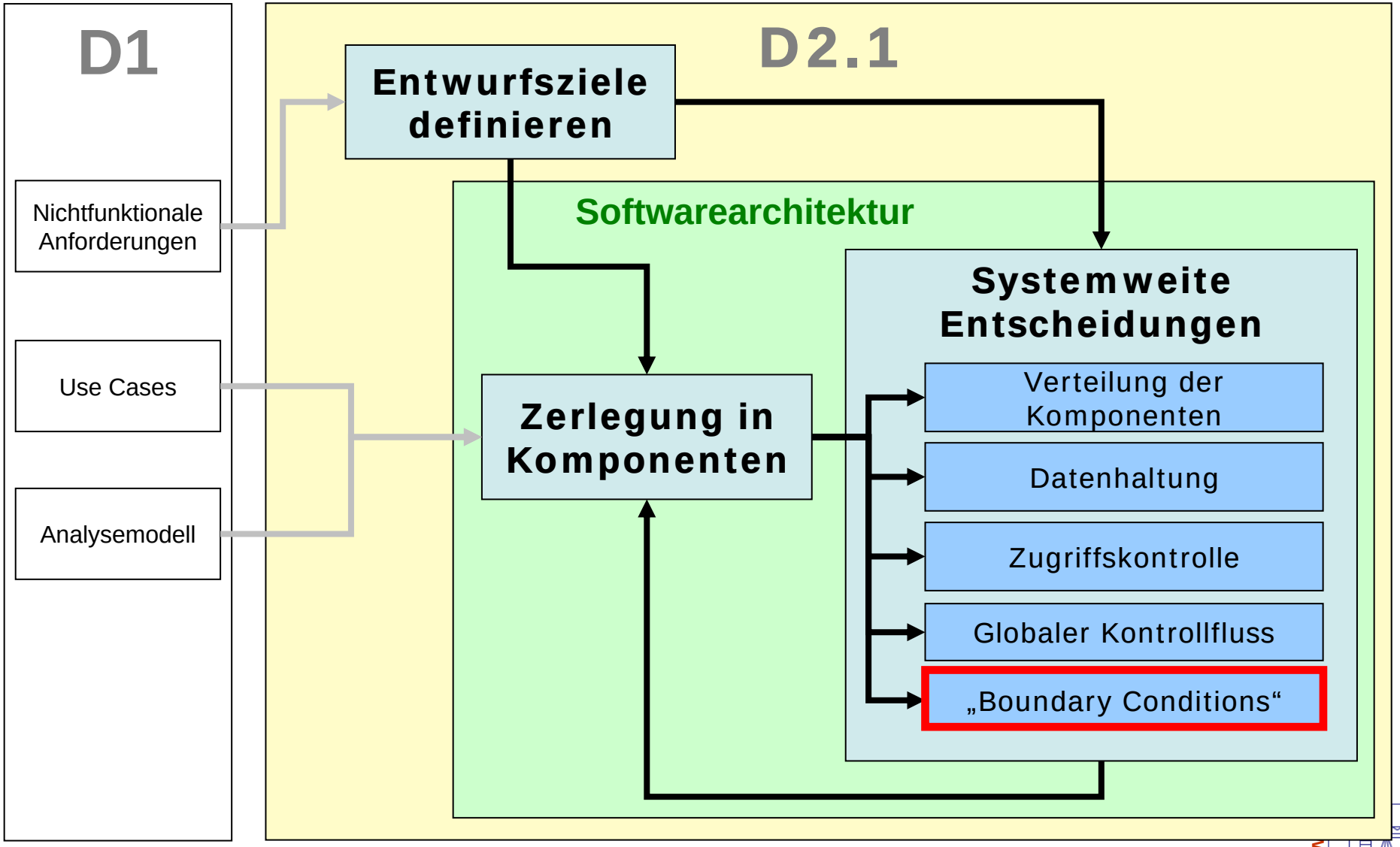
Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

Globaler Kontrolfluss

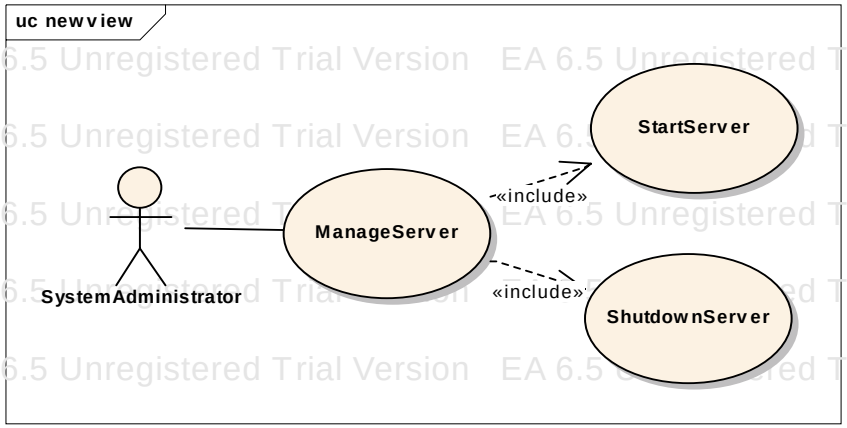
- Kontrollfluss: Sequentialisierung der Aktionen eines Systems
  - Im Feinentwurf: Für einzelne Methoden, …
  - Jetzt: **Zusammenspiel der Komponenten**
- In der Analysephase nicht relevant: Alle Objekte sind „aktiv“
- Welche Threads und Prozesse gibt es? Wer initiiert wen?
- Event-getriebene Architektur?
- Kurzbeschreibung in Textform genügt
  - Alternativ: UML Sequenzdiagramme, Zustandsautomaten

Überblick


D1
D2.1
Entwurfsziele definieren
Nichtfunktionale Anforderungen
Use Cases
Analysemodell
Softwarearchitektur
Zerlegung in Komponenten
Systemweite Entscheidungen
Verteilung der Komponenten
Datenhaltung
Zugriffskontrolle
Globaler Kontrollfluss
„Boundary Conditions“

„Boundary Conditions“

- Das **Verhalten des Systems in Sonderfällen** ist meist auch global
  - System starten
  - System herunterfahren
  - Schwere Fehler, Ausnahmen
    - Daten korrupt, keine Serververbindung, usw.
- Das Verhalten in diesen Situationen wird in zusätzlichen **Boundary Use Cases** beschrieben



Use Case Name	StartServer
Entry Condition	1. Der `SystemAdministrator` ist angemeldet
Flow of Events	1. `SystemAdminstrator` ruft `StartServer` auf 2. ... 3. Alle Kurse werden eingelesen und `CourseCacheManager` initialisiert 4. ...
Exit Condition	Der Server steht für Anfragen zur Verfügung

Review des Systementwurfs

- Ist der Systementwurf…
  - **… korrekt?**
    - Gibt es für jede Komponente einen entsprechenden Use Case oder eine NFA?
    - Gibt es für jedes Entwurfsziel eine entsprechende NFA?
  - **… vollständig?**
    - Alle systemübergreifenden Entscheidungen gefällt?
    - Werden alle Use Cases umgesetzt?
    - Wird jede NFA beachtet?
    - Gibt es für jeden Aktor eine access polic
  - **… konsistent?**
    - Sind widersprüchliche Entwurfsziele priorisiert?
    - …
  - **… realistisch?**
    - Verwendung neuer Technologien und Komponenten?
    - …
  - **… und lesbar?**
    - Sinnvolle Namen für Komponenten gewählt?
    - Überall gleiches Abstraktionsniveau?

Zusammenfassung

- Grobentwurf ist:
  - Definition der **Entwurfsziele**
    - Auswahl und Priorisierung
  - Beschreibung der **Softwarearchitektur**
    - Zerlegung in Entwurfskomponenten
      - Verwendung von Architekturstilen
    - Festlegung der Schnittstellen
    - Treffen systemweiter Entscheidungen

Das Systementwurfsdokument

1. Einführung
   1. Zweck des Systems
   2. **Entwurfsziele**
   3. Definitionen, Abkürzungen, etc.
   4. Literaturverweise
   5. Überblick
2. (Derzeitige Architektur)
3. Vorgeschlagene Architektur
   1. Überblick
   2. **Zerlegung in Komponenten und Beschreibung der Schnittstellen**
   3. **Hardware/Software mapping**
   4. **Management persistenter Daten**
   5. **Zugriffsrechte und –kontrolle**
   6. **Globaler Kontrollfluss**
   7. **Boundary Conditions**
4. Glossar

- **Abgabe Entwurf (Grob- und Feinentwurf)**
  - 31.5. 12^{00} Uhr
  - Template für Gesamtdokument nächstes Mal

Literaturhinweise

- Bruegge, Dutoit. **Object-oriented Software Engineering. Using UML, Patterns and Java.** 2004.
  - Kapitel 6 und 7

- Architekturstile:
  - Shaw, Garlan. Software Architecture. **Perspectives on an emerging discipline.** 1996.
    - Erster Katalog für Architekturstile
  - Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. **Pattern-Oriented Software Architecture. A System of Patterns.** 1996

- Gamma, Helm, Johnson, Vlissides. **Design Patterns.** 1995
  - Für **Facade** und **Observer**
  - … und alle anderen Entwurfsmuster im nächsten Teil