# Vorlesung
# Software aus Komponenten

## 2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2015/16

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

## Komponenten als sozio-technische Artefakte

Eine Software-Komponente ist eine Einheit der Komposition mit durch Kontrakt spezifizierten Schnittstellen und nur expliziten Kontext-Abhängigkeiten. Eine Software-Komponente kann unabhängig verteilt werden und zur Komposition durch Dritte verwendet werden.

- Definition wurde erstmals so 1996 auf der „European Conf. on OO Programming“ gegeben [Szyperski, Pfister]
- technische Seite: Unabhängigkeit, Schnittstellen-Kontrakt, Zusammenbau
- soziale Seite: Dritte, Verteilung
- Diese Verbindung ist typisch für jeden tragfähigen Komponenten-begriff nicht nur im Software-Bereich

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

# Schnittstellen-Kontrakt

- Muss die Verwendung der Komponente in einem Produktiv-System genau beschreiben
- Technische Aspekte:
  - Schnittstellen im engeren Sinne
  - Entpackung, Konfiguration, Installation der Komponente
  - Instanziierung und Beschreibung des Verhaltens der durch die Komponente erzeugbaren Objekte durch ihre Schnittstellen (Laufzeitverhalten)
  - Beschreibung der Ressourcenanforderungen der Komponente sowie der Anforderung an die Lokalisierungs-Umgebung
    - auch als Kontext-Abhängigkeit bezeichnet
  - enthält:
    - Komponenten-Modell = Spezifikation der Kompositions-Regeln
    - Komponenten-Plattform = Spezifikation der Regeln für Entpackung, Installation und Aktivierung von Komponenten

- Schnittstellen-Spezifikation als Kontrakt zwischen
  - Nutzer der **Funktionalität** einer Schnittstelle und
  - Anbieter der **Implementierung** dieser Schnittstelle
- verbreiteter Zugang auf technischer Ebene: durch Vor- und Nachbedingungen (Hoare-Kalkül: {V} P {N})
  - Nutzer sichert die Vorbedingungen V
  - Anbieter sichert dann Nachbedingung N
  - Problem: Sichert funktionale Eigenschaften, aber weder Performanz von P noch Termination überhaupt
- heute üblich: auch nicht-funktionale Aspekte im Kontrakt erfassen
  - Beispiel: Service Level Agreement
    - enthält Qualitätsaussagen für den Betrieb wie Verfügbarkeit, Fehlerrate, Datensicherheit etc.
  - Konsequenzen im Einsatz sind ähnlich gravierend wie funktionale Fehler

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

# Direkte und indirekte Schnittstellen

- Direkte Schnittstelle: Schnittstelle der Komponente selbst
  - meist prozeduraler Natur
- Indirekte Schnittstelle: Schnittstelle von Objekten, die in der Komponente erzeugt werden
  - meist objektorientierter Natur
- Vereinheitlichung durch Einführung eines statischen Objekts möglich
  - typischer Ansatz von OO Sprachkonzepten
- Überladung indirekter Schnittstellen und späte Bindung
  - Provider des Dienstes hängt vom Objekt ab
  - Derselbe Dienst kann über dieselbe Schnittstelle innerhalb desselben Komponenten-Kontexts von unterschiedlichen Anbietern kommen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

# Schnittstellen und Versionen

- Problem: Schnittstellen können ihr Verhalten zwischen Versionen wechseln
- Management traditionell über Versionsnummern
  - Komponente als unteilbare Einheit => Versionsnummern nur für ganze Komponenten
- Versions-Information in Import- und Exportschnittstellen
  - direkte Schnittstellen: Abfrage zur Bindungszeit, also (nur) vor dem ersten Schnittstellenaufruf
  - indirekte Schnittstellen: Abfrage vor jedem Aufruf erforderlich
    - Alternative: Integration ins Management der Objektidentität
- Problem der Versions-Information, wenn eine Objektreferenz Komponentengrenzen überschreitet
  - Objekt bietet eigene Dienste an
    - etwa Rückgabe in einem bestimmten Format
  - Objekt nutzt Dienste anderer => dynamische Versionskontrolle

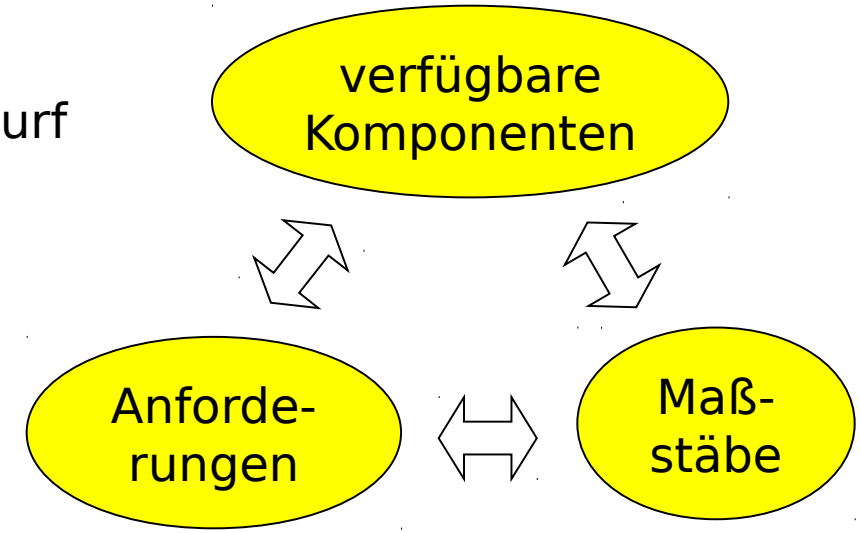
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Schnittstellenversionen müssen klar als kompatibel oder klar als veraltet (deprecated) deklariert werden können
- Ansatz der unveränderlichen Schnittstellen-Spezifikation (immutable interfaces)
  - Neue Versionen nur als neue Schnittstellen
  - Veraltete Schnittstellen werden einfach nicht mehr unterstützt
  - Beispiel: COM = Component Object Model von MicroSoft
- Veränderbare Schnittstellen-Spezifikationen:
  - klares Kompatibilitäts-Konzept erforderlich
  - Installation verschiedener Komponentenversionen in derselben Umgebung kann erforderlich sein.
  - Unterscheidung zwischen Komponenten, die immer in der aktuellsten Version verwendet werden können und Komponenten, die nur in der ursprünglich installierten Version verwendet werden können
    - wird im Rahmen der CLR = Common Language Runtime verfolgt

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

## Komponenten abgrenzen

Problem: Wie ist ein kompletter Entwurf in Komponenten zu partitionieren?




Prinzipien:

- Modularität und Kapselung
- Abhängigkeiten zwischen K. sind expliziert
- Mehrere Abstraktionsebenen, hierarchische Strukturierung
- natürliche Zuordnung von Verantwortlichkeiten
- Migrations-Erfordernisse vorab berücksichtigen

# 2.5. Komponenten abgrenzen
## Komponenten-Gewicht

- Wie „fett“ soll eine Komponente sein?
  - optimal, aber unreal: „richtige Schnittstellenmenge“ und keine Kontext-Abhängigkeit
  - maximal: „fette“ Komponente, die alle benötigten Dienste mitbringt (=Applikation, grobkörnig)
  - minimal: Auslagern aller bis auf die zentrale Funktionalität (=Klasse, feinkörnig)
    - „Maximizing reuse minimizes use.“
    - Grund: Explodierende Kontext-Abhängigkeit
    - würde nur unter statischen Entwicklungsbedingungen funktionieren
    - Beispiel: Linux-Probleme mit Bibliotheksversionen
  - praktisch ist hier ein je ausgewogenes Mittel zu finden
- Je detaillierter Normierung und Standardisierung, desto schlankere Komponenten sind möglich
  - Standardisierung ist in vertikalen Marktsegmenten (funktional) eher möglich als in horizontalen, aber wegen der geringen Marktgröße schwieriger

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

# Komponenten und Systemdesign

**Komponenten und Analyse**

- Analyse komplexer Systeme erfolgt durch Zerlegung in handhabbare Einheiten, Tiefenanalyse der Einheiten und Zusammensetzung von Extrakten der Analyseergebnisse zum Gesamtsystem
- Ansatz: white box → black box
- Entwurfsexpertise lässt sich auf diese Weise kapseln und wiederverwenden (design expertise ready for use)
- Komponentenzuschnitt längs dieser Abstraktionsgrenzen erleichtert das Verständnis des Gesamtsystems

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

## Komponenten aus technischer Anbietersicht

**Komponenten und Compilierbarkeit**

- Compilierbarkeit kann aus Sicht der Auslieferung interessant sein
- Compilierbarkeit und Analyse sind eng verbunden
  - white box: Analyse
  - black box: Compilierbarkeit
- Compilierbarkeit und Optimierung
  - Optimierung ist ein globales Phänomen, deshalb Antwort 1: Einheiten möglichst groß wählen
- Antwort 2: Optimierung *zwischen* Komponenten, vielleicht sogar erst nach der Lokalisierung
  - Verfahren muss allerdings im Komponentenkonzept und in der Komponentenbeschreibung verankert sein

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

# Komponenten als Einheit der Packung

**Komponenten und der Auslieferungsprozess**

- Entpackung (deployment) = Prozess der Vorbereitung der Komponente auf den Einsatz in einer speziellen Umgebung (Lokalisierung)
  - wurde lange nicht als separater Schritt betrachtet
  - Prozess der Anbindung an eine spezielle Komponentenplattform
- Konfiguration = Einstellung spezieller Eigenschaften der Komponente für den konkreten Einsatz
- Installation = plattformspezifische Aktivität, mit der eine entpackte Komponente für die Nutzung in einer speziellen Hardware-Konfiguration verfügbar gemacht wird, die von der Plattform unterstützt wird.
  - Zeit, in der auch kritische Tests ausgeführt werden, die vor dem eigentlichen Betrieb erfolgen müssen (etwa Integritätstests)
- für alle drei Aktivitäten müssen entsprechende Beschreibungen erstellt werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

## Komponenten und Management

**K. als Einheit der (Auseinandersetzung um) Fehlersuche**

- Problem: Was ist (u.a. wer haftet?), wenn ein aus Komponenten zusammengebautes Produktivsystem fehlerhaft arbeitet?
- Problem der Lokalisierung von Fehlern (und damit Verantwortlichkeiten)
  - besonders schwierig wird es, wenn Objektreferenzen die Komponentengrenzen verlassen
- vitale Regel: Fehler müssen in den verursachenden Komponenten bleiben (bug containment)
  - typische nicht-lokalisierbare Fehler: Speicherzugriffsfehler
- Folge: Ausnahmebehandlungen müssen in der Regel innerhalb einer Komponente bleiben
  - Ausnahmen davon sind im Komponenten-Kontrakt zu fixieren
  - Komponente als Einheit der Fehlerbehandlung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

**K. als Auslieferungseinheit**

- Bündel der technischen und wirtschaftlichen Aspekte
- Management (Service, Wartung, Schulung, Updates, ...) treibt den Preis in die Höhe
- betriebswirtschaftliche Bedeutung jenseits der (geringen) Replikationskosten

**K. als Einheit der Kostenrechnung**

- wichtig in größeren industriellen Kontexten, um Projektentwicklungskosten verfolgen zu können

**K. als Einheit des Managements**

- oft zu klein, Management auf der Ebene von Subsystemen, die mehrere Komponenten zusammenfassen
- etwa auf der Ebene der Server