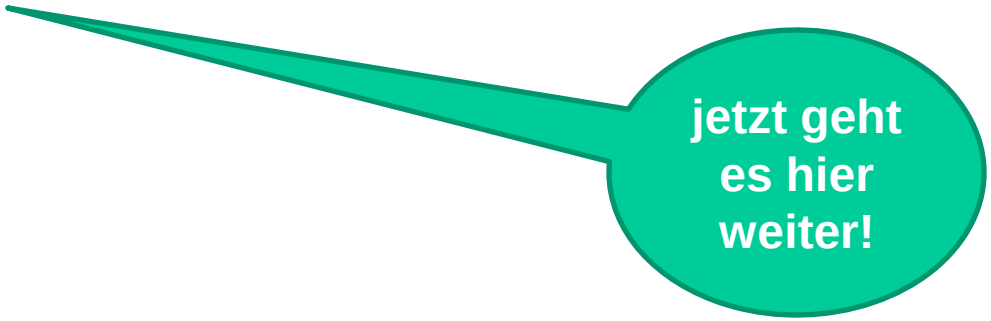
# Modellarbeit I: Entwurfsgerechte Klassenmodellierung

# Vom Analysemodell zum Entwurfsmodell

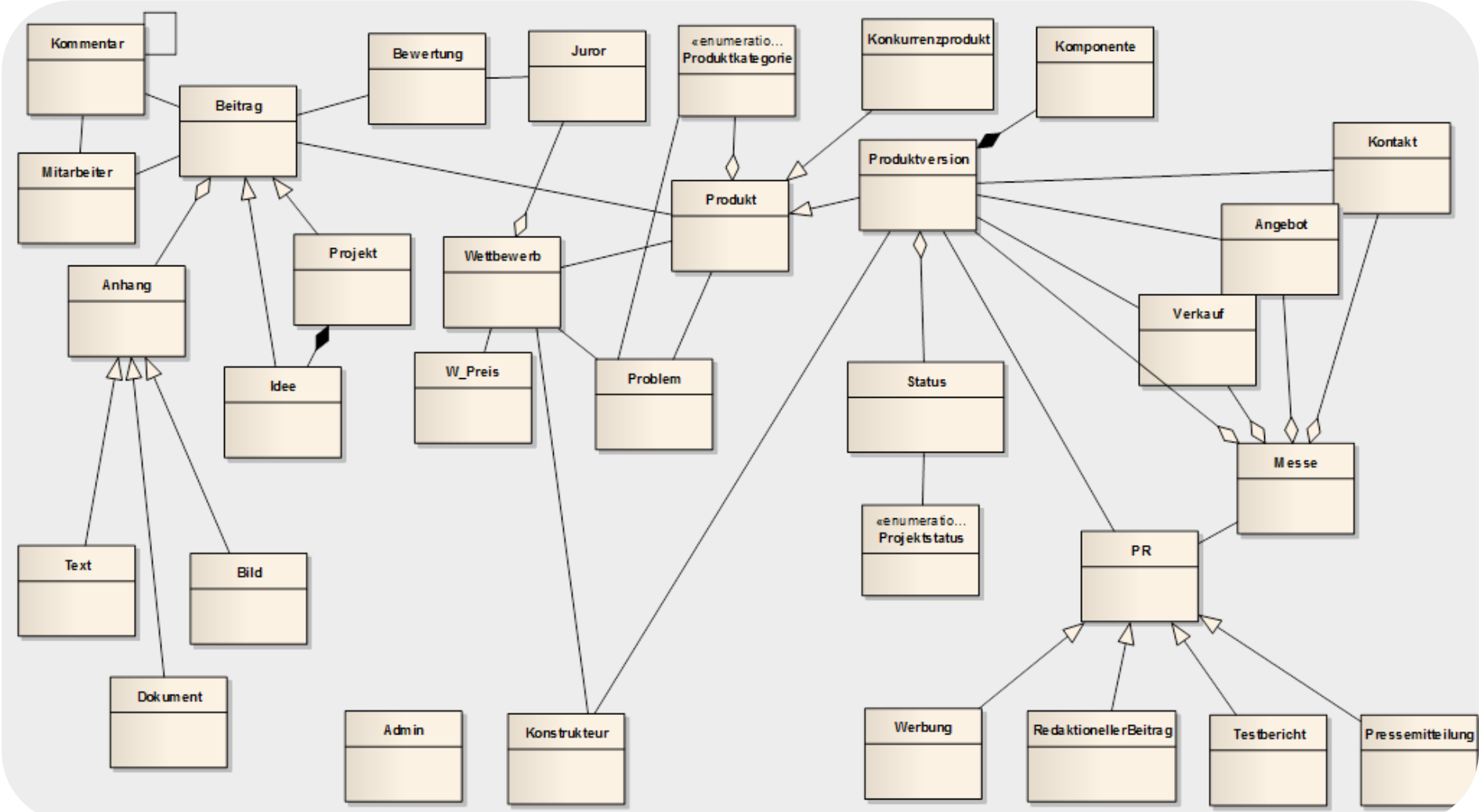
Nach der Etablierung der Techologien:

- Überarbeitung des Fachlichen Modells zu einem geeigneten Entwurfsmodell
  - Navigationen
  - Datenhaltung, Listen
  - Paketierung
  - Entwurfsmuster

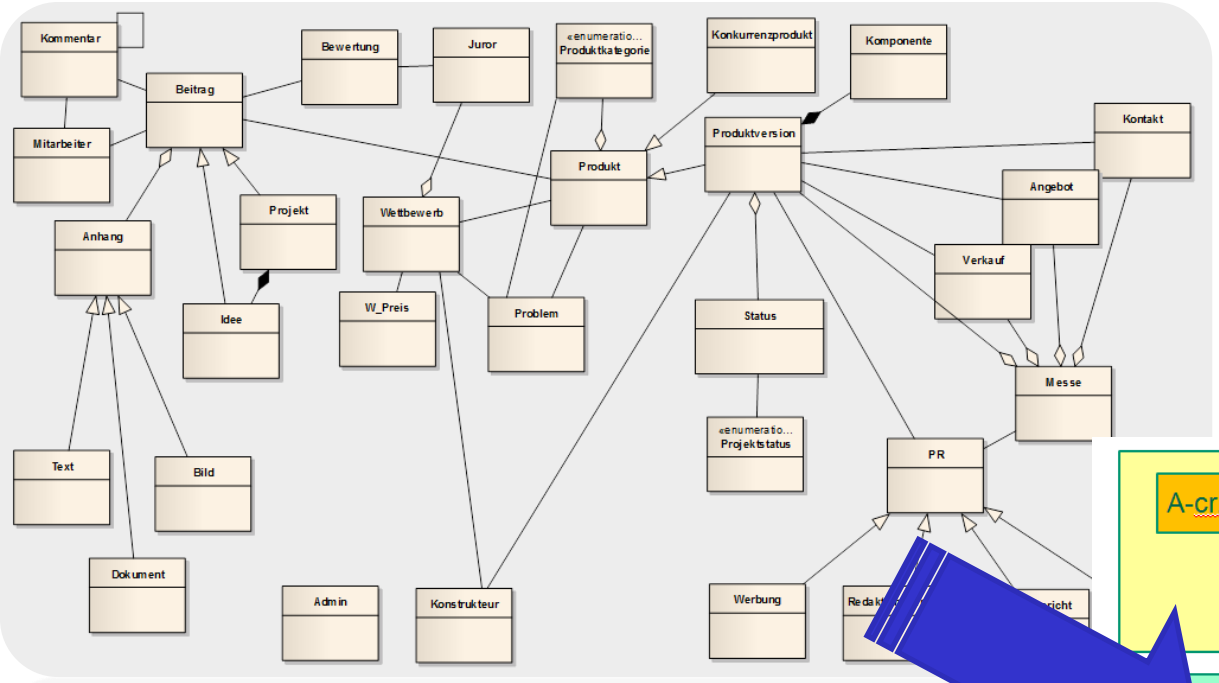


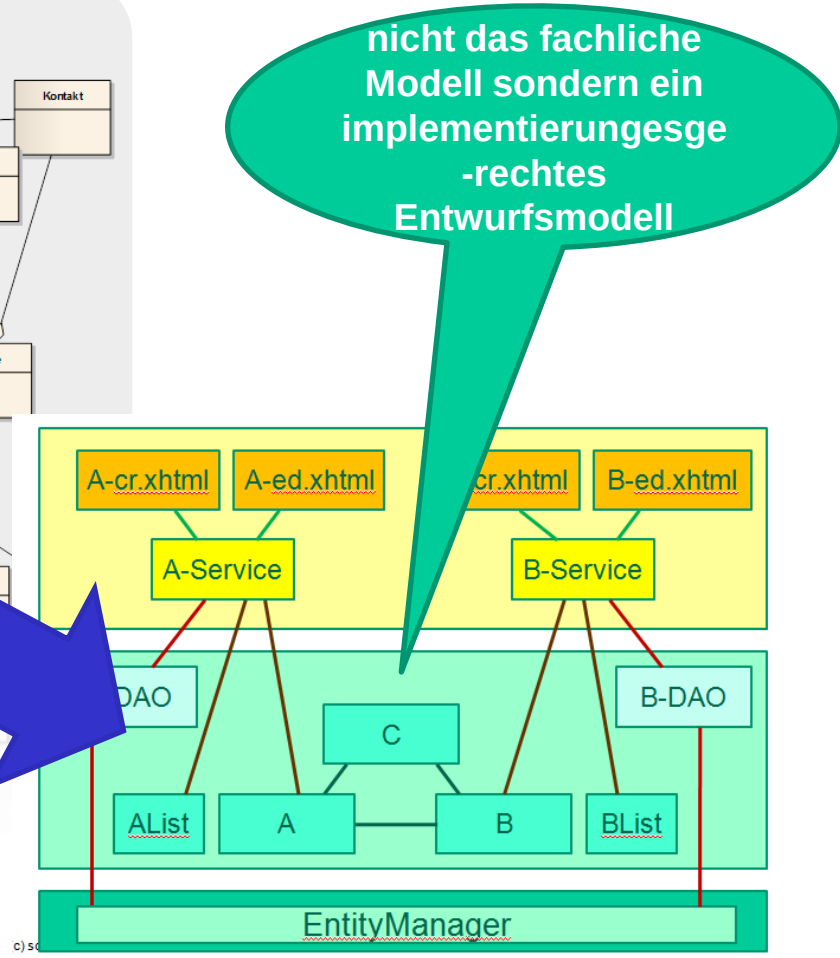

- Einbindung des Entwurfsmodells in eine Gesamtarchitektur
  - mit JSF für die Präsentation
  - mit JPA-Persistenz

# Unser Übungsmodell:



# Schritte zum Entwurfsmodell



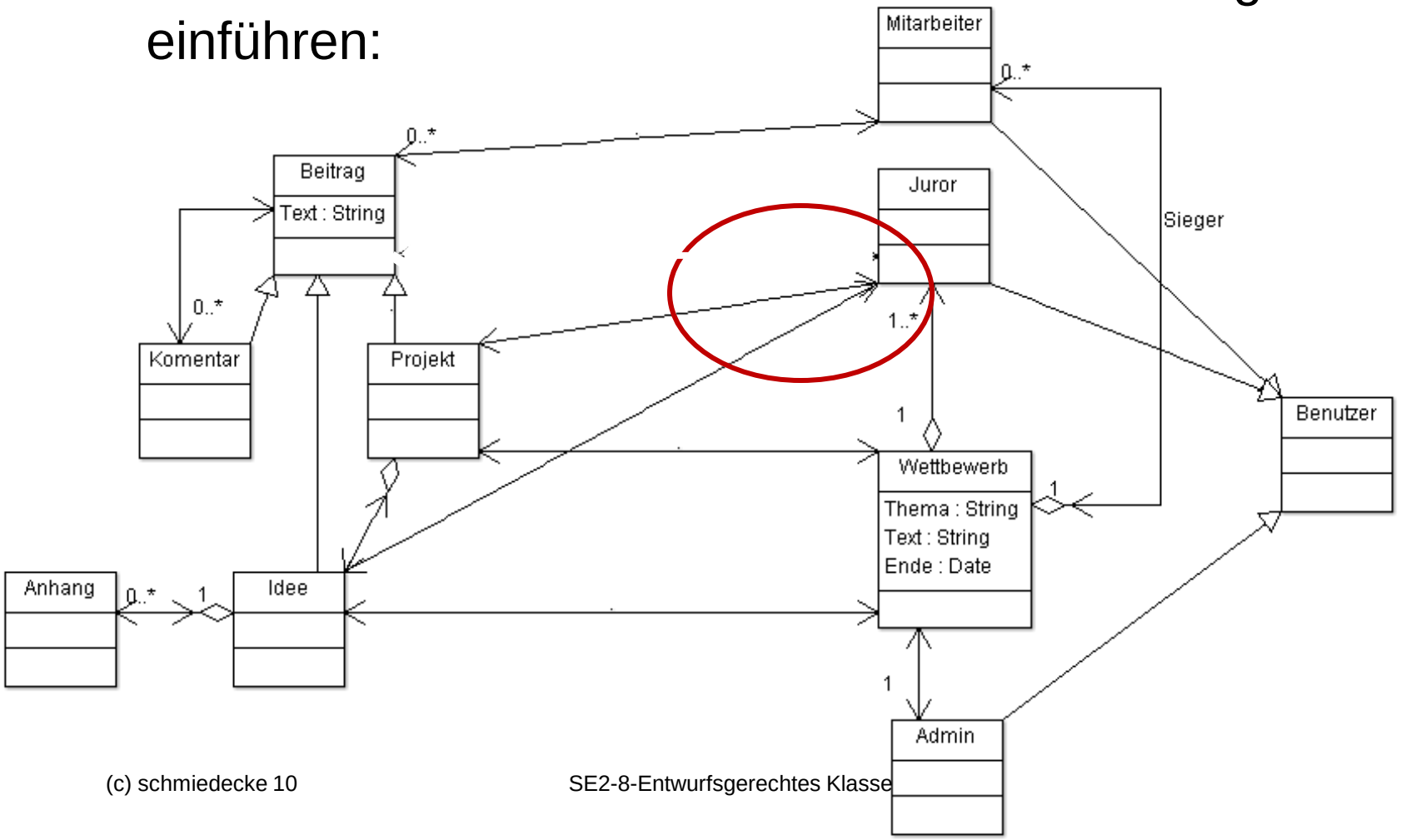
# Das Entwurfsmodell - 1:1-Vorlage für die Implementierung

**Anforderungen:**

- Implementierbare Struktur (Zielsprache Java)
  - keine Mehrfachvererbung (außer Interfaces)
  - keine assoziativen Klassen
  - zulässige Paket-, Klassen-, Attribut- und Methodennamen
  - korrekte Signaturen
  - keine "zufällig" gleichen Namen (Polymorphie!)
- Berücksichtigung von Konventionen
  - private Attribute mit Gettern und Settern (Properties)
  - Namenskonventionen (z.B. vorgegebene Präfixe, englische Namen)
- Problemgerechte Paketierung
- Effiziente Struktur
  - keine unnötigen Klassen oder Hierarchien
  - problemgerechte Navigation
- Wartbare Struktur
  - Lose Kopplung, klare Schnittstellen, **Entwurfsmuster**
- Konkretisierung der Datenverwaltung

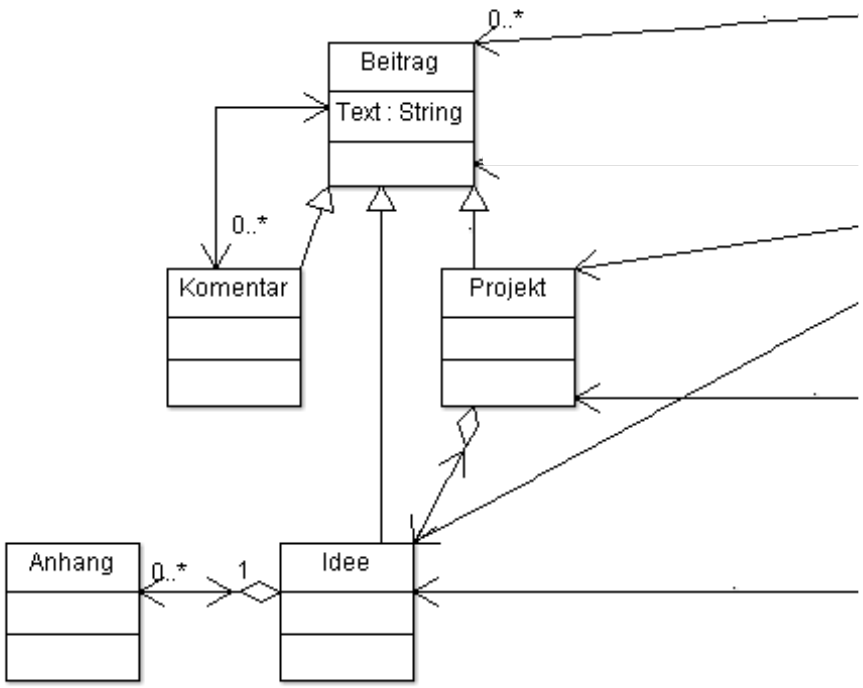
# 1. Vererbungsstruktur prüfen

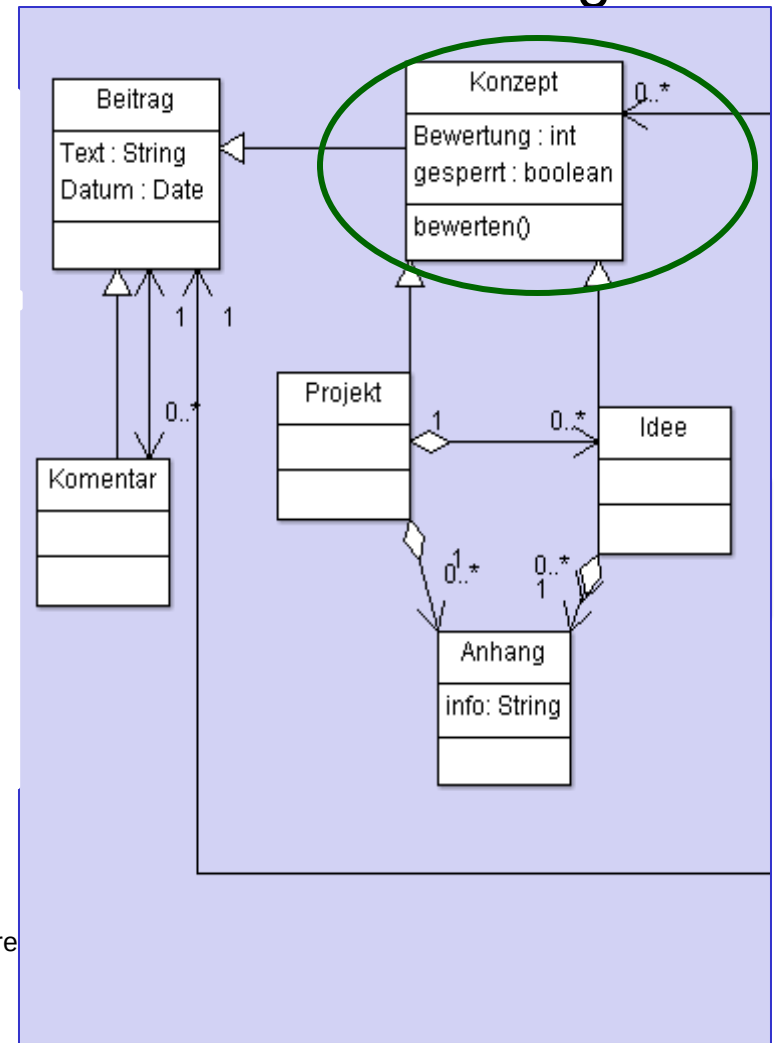
1. Oberklassen oder Interfaces zur Vereinfachung einführen:



# 1. Vererbungsstruktur prüfen

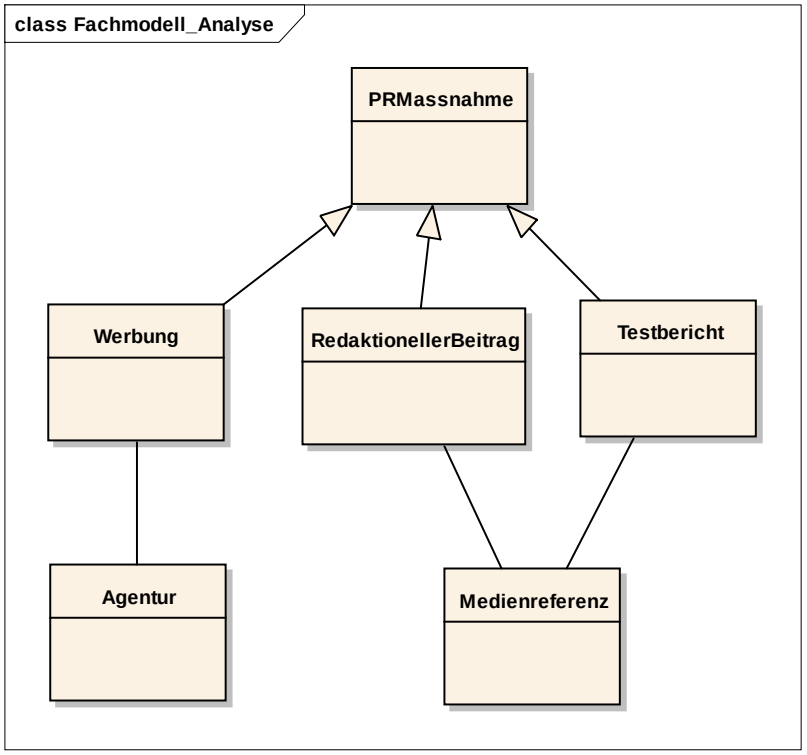
1. Oberklassen oder Interfaces zur Vereinfachung einführen:

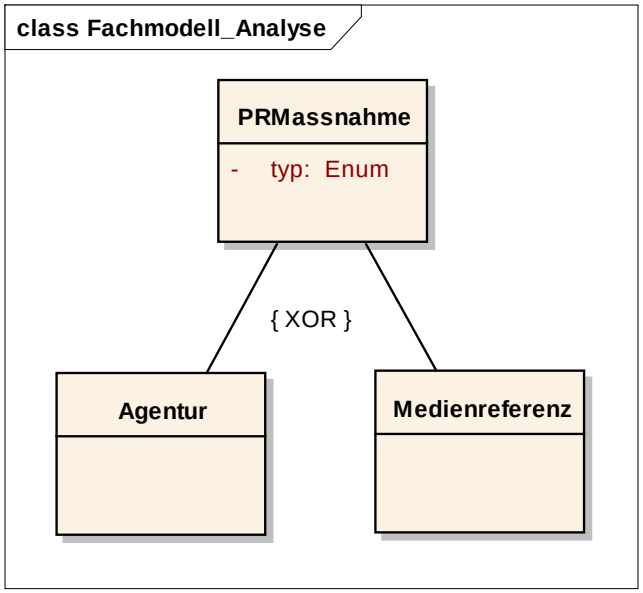


# 1. Vererbungsstruktur prüfen

2. ODER-Strukturen evtl. dynamisch lösen (komprimieren):

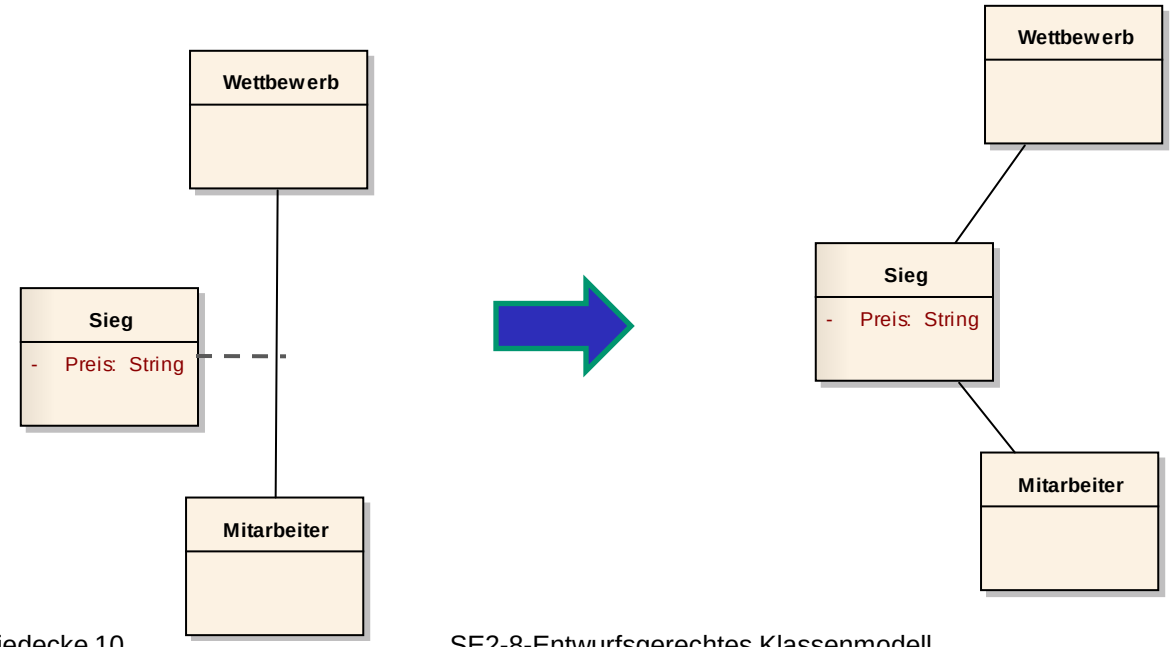



3. Typhierarchien evtl. durch generische Klassen ersetzen.
4. Zwischen Interfaces, abstrakten und implementierten Klassen unterscheiden

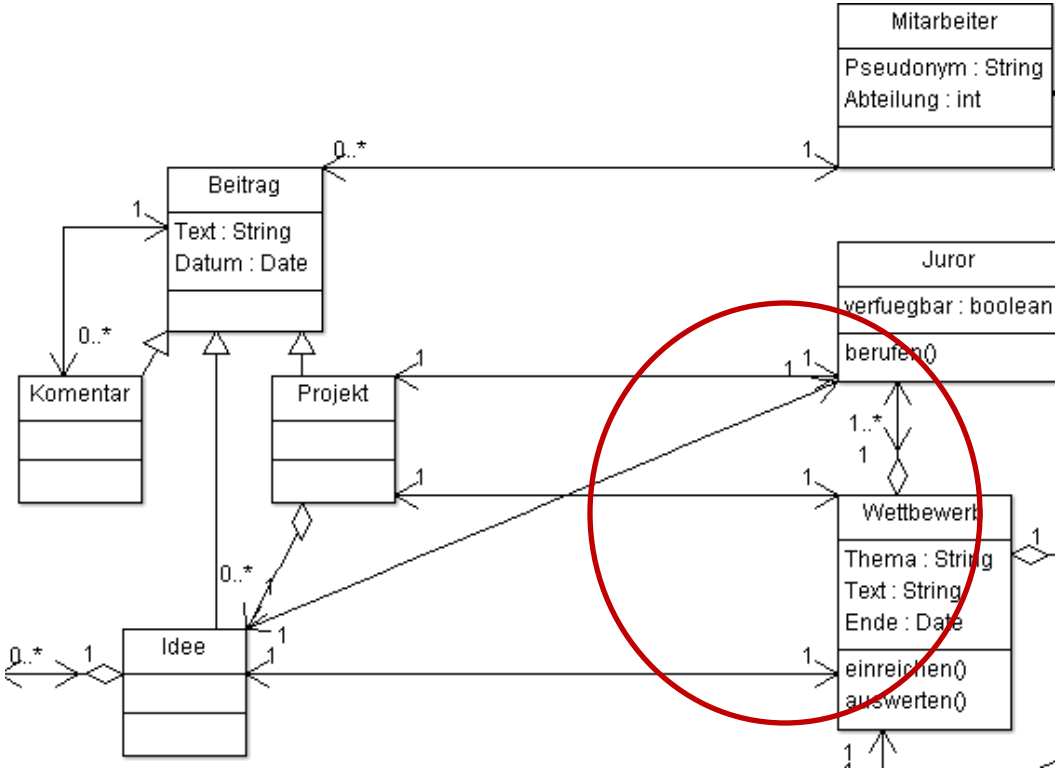
# 2. Klassen und Attribute an die Zielsprache anpassen

1. Java-gerechte Klassennamen (Großschreibung!)
2. Java-gerechte Attributnamen (Kleinschreibung!)
3. Java-gerechte Typen
4. Mehrfachvererbung auflösen (Interfaces nutzen)
5. Assoziative Klassen auflösen:



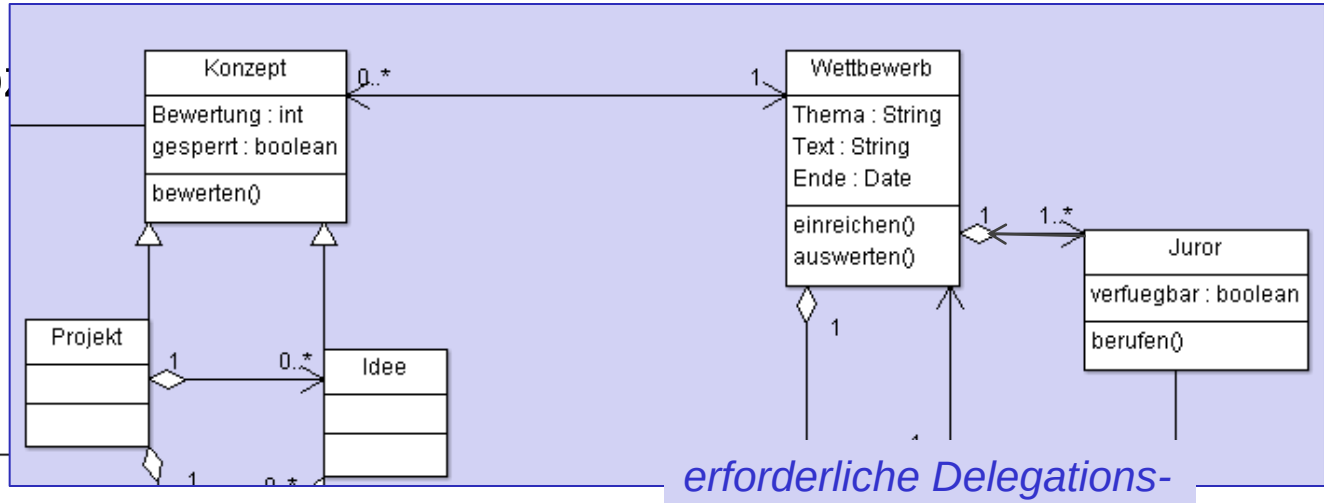
# 3. Attribute und Assoziationen präzisieren

1. Redundante Assoziationen und Attribute prüfen, evtl. entfernen

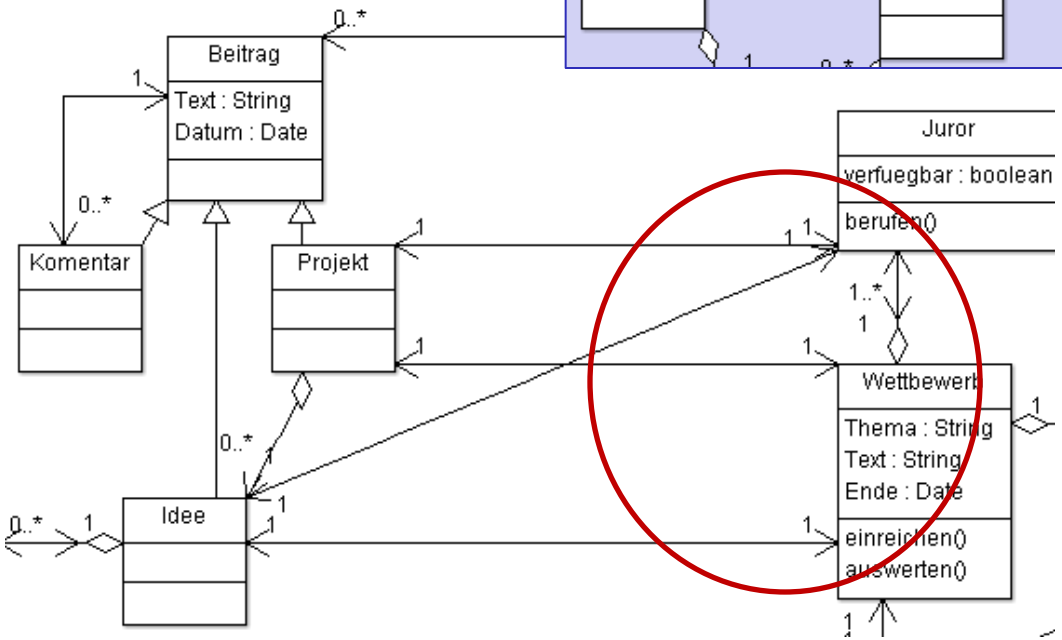


# 3. Attribute und Assoziationen präzisieren

1. Redundante Asso




*erforderliche Delegations-methoden einfügen!*



# 3. Attribute und Assoziationen präzisieren

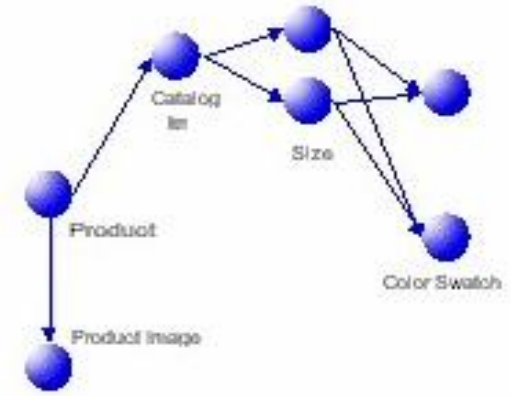
1. Properties kennzeichnen: <<Property>>
   *nicht Getter und Setter modellieren!*
2. Abgeleitete (berechenbare) Attribute durch Getter-Methoden ersetzen
   - Häufig Statistik – Benutzer online, letzter Zugriff, …
   - Ausnahme:
     Berechnung ist so aufwändig, dass Zwischenspeicherung sinnvoll ist.

3. Sichtbarkeit der Attribute
   - grundsätzlich *private*
   - Bedingungen kennzeichnen*:* *<<readonly>> <<not null>> …*

4. Constraints
   - {frozen} - Unveränderbare Objektbeziehung
   - {addOnly} - Objektbeziehungen dürfen nicht entfernt werden

# 4. Navigation und Kardinalitäten festlegen

- Navigationsstruktur beeinflusst
  - Datenbankschema
  - Zugriffseffizienz
- Grundsatz: restriktive Navigation
  - nur ausnahmsweise bidirektional
  - semantisch korrekte Navigation (nicht 1:n durch n:1 ersetzen)
  - Grund: Lazy Materialization!
- Aggregation:
  - immer in vom Aggregat her navigierbar



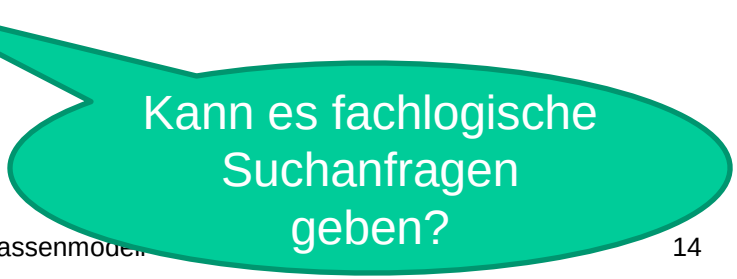

- Kardinalitiäten
  - auch festlegen ob optional (0..1, 0..n)
  - ggf. weitere logische Bedingungen als Constraints einfügen z.B.{XOR} in Folie 6, …

# 5. Operationen präzisieren

1. Für alle Operationen vollständige Signaturen angeben.

2. Entscheiden, ob Operation Fachlogik oder Service ist
   - Kriterium: Datenaufbereitung für die GUI oder fachlogische Datenverarbeitung
   - Beispiele für Fachlogik: *siegerErmitteln, bewertungBerechnen*
   - Beispiele für Service: *neuesteEintraegefinden, sortieren*

3. Suchaufgaben an DAO-Klassen delegieren
   - dort die entsprechenden Query-Methoden vorsehen

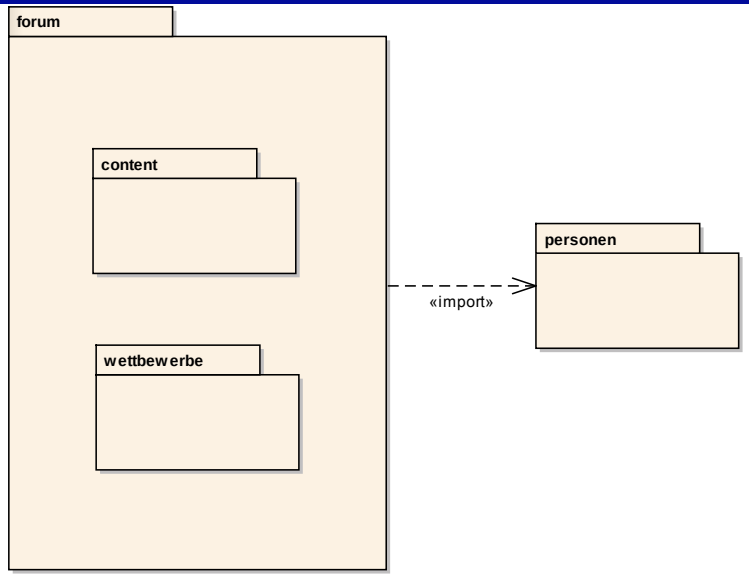


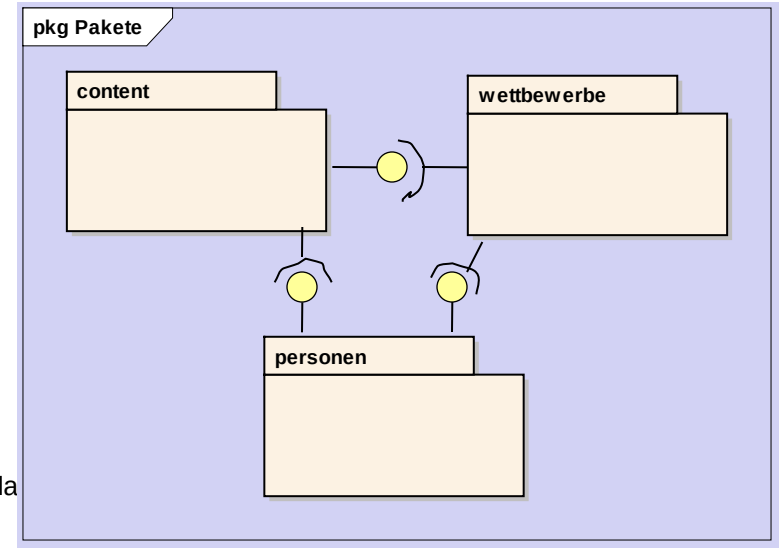
# 6. Containerklassen einfügen

1. Für listenartige Operationsergebnisse Containerklassen anlegen.
   - Containerklassen werden für Suchergebnisse etc. benötigt
   - Array, Collection, etc., oder spezifischer benutzerdefinierter Typ
   - Objekte dynamisch erzeugt
   - nicht vom Container verwaltet (keine ManagedBeans)
   - dynamisch gefüllt
   - ***nicht persistent***.

# 7. Paketstruktur erzeugen

1. Pakete dienen der Strukturierung
2. Pakete sind Namensräume
3. Pakete sind Sichtbarkeitsgrenzen
   - Was nicht public ist, ist außerhalb seines Pakets unsichtbar
4. Pakete exportieren nur ihre öffentlichen Elemente
5. Zwischen Paketen besteht eine import-Beziehung
6. Pakete sollten über Interfaces miteinander verbunden sein



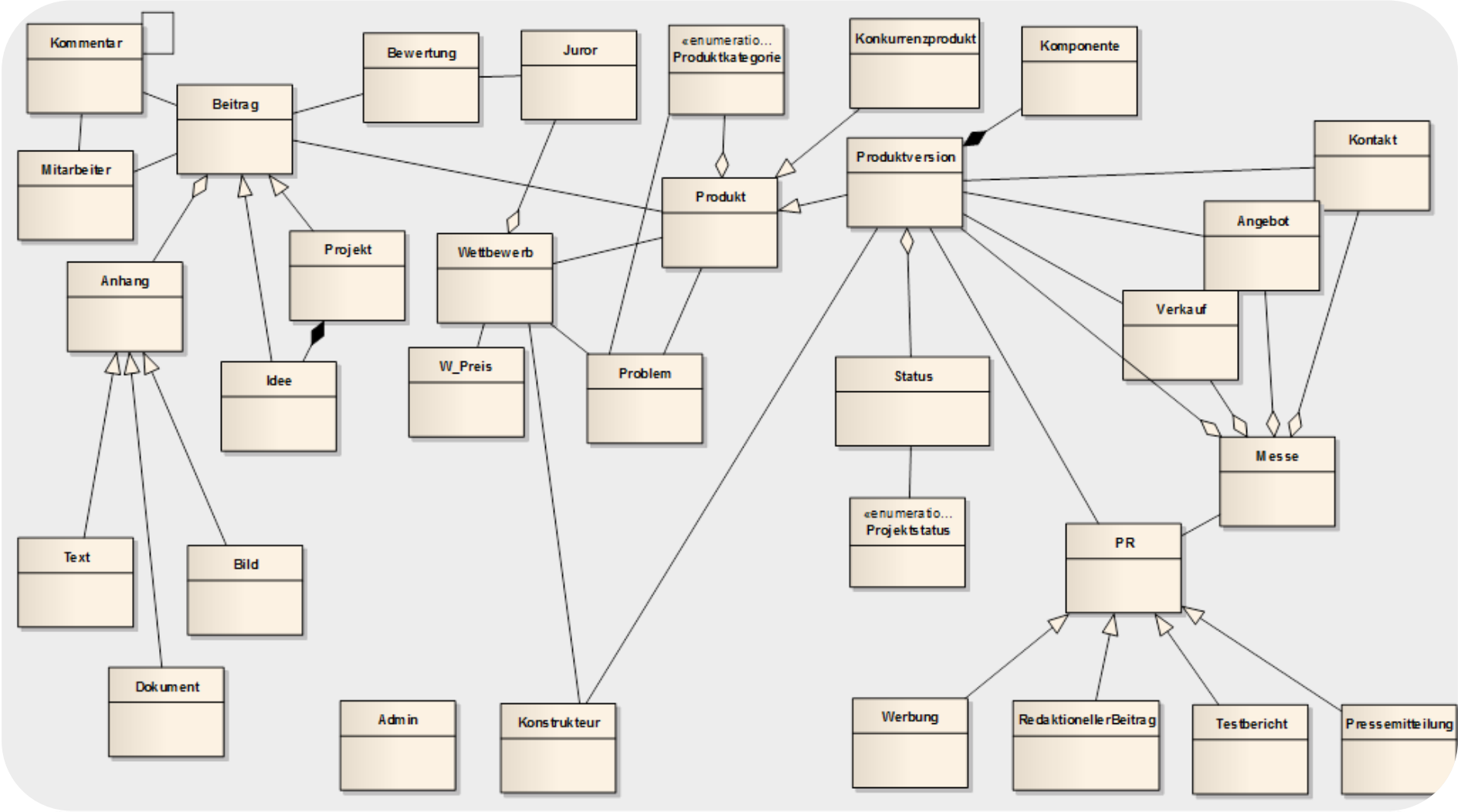
# 8. Entwurfsmuster nutzen

- … Unser Thema für die nächsten Wochen.
- Einige kennen Sie schon:

Proxy, Broker, DAO

aber das ist nur die Spitze des

# Viel Spaß beim Aufräumen ☺



**Nächstes Mal:**
**Strukturverbesserung durch Entwurfsmuster**

# Auswahl Entwurfsmuster: Welche kennen Sie schon?

1. Strategy
2. Observer
3. Factory
4. Abstract Factory
5. Singleton
6. Builder
7. Prototype
8. Decorator
9. Command
10. Adapter
11. Facade
12. Bridge
13. Template Method
14. Iterator
15. Composite
16. Flyweight
17. State
18. Proxy
19. Chain of Responsibility
20. Interpreter
21. Mediator
22. Memento
23. Visitor