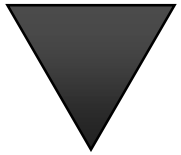
Produktsysteme im Versicherungsbereich

OOP 2000

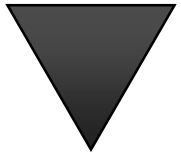
München, den 21. Januar. 2000

Jens Coldewey
Coldewey Consulting
Uhdestraße 12
D-81477 München
jens_coldewey@acm.org



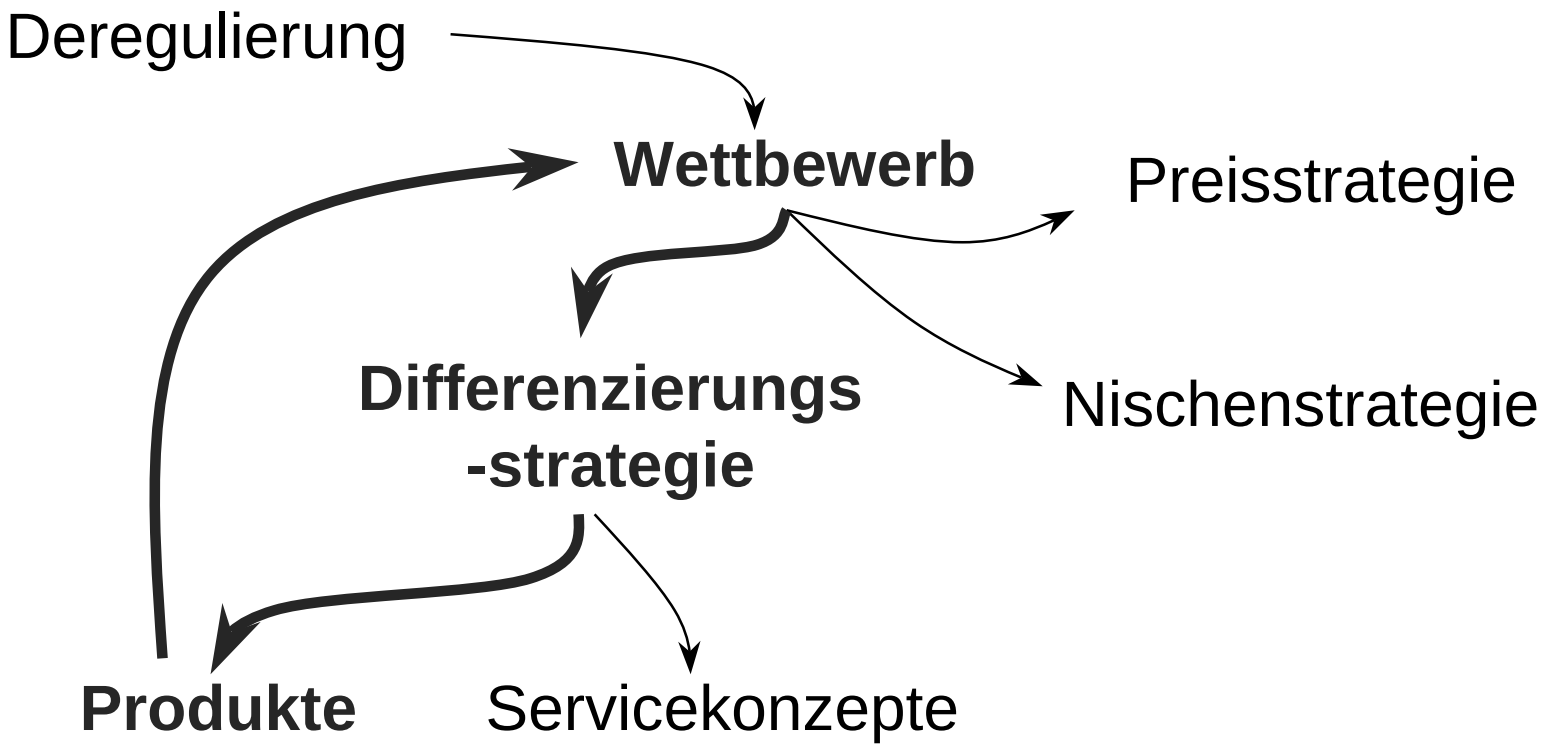
Der Fahrplan durch die nächsten Neunzig Minuten

- Der fachliche Hintergrund
- Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - Meta-System
- Zusammenfassung

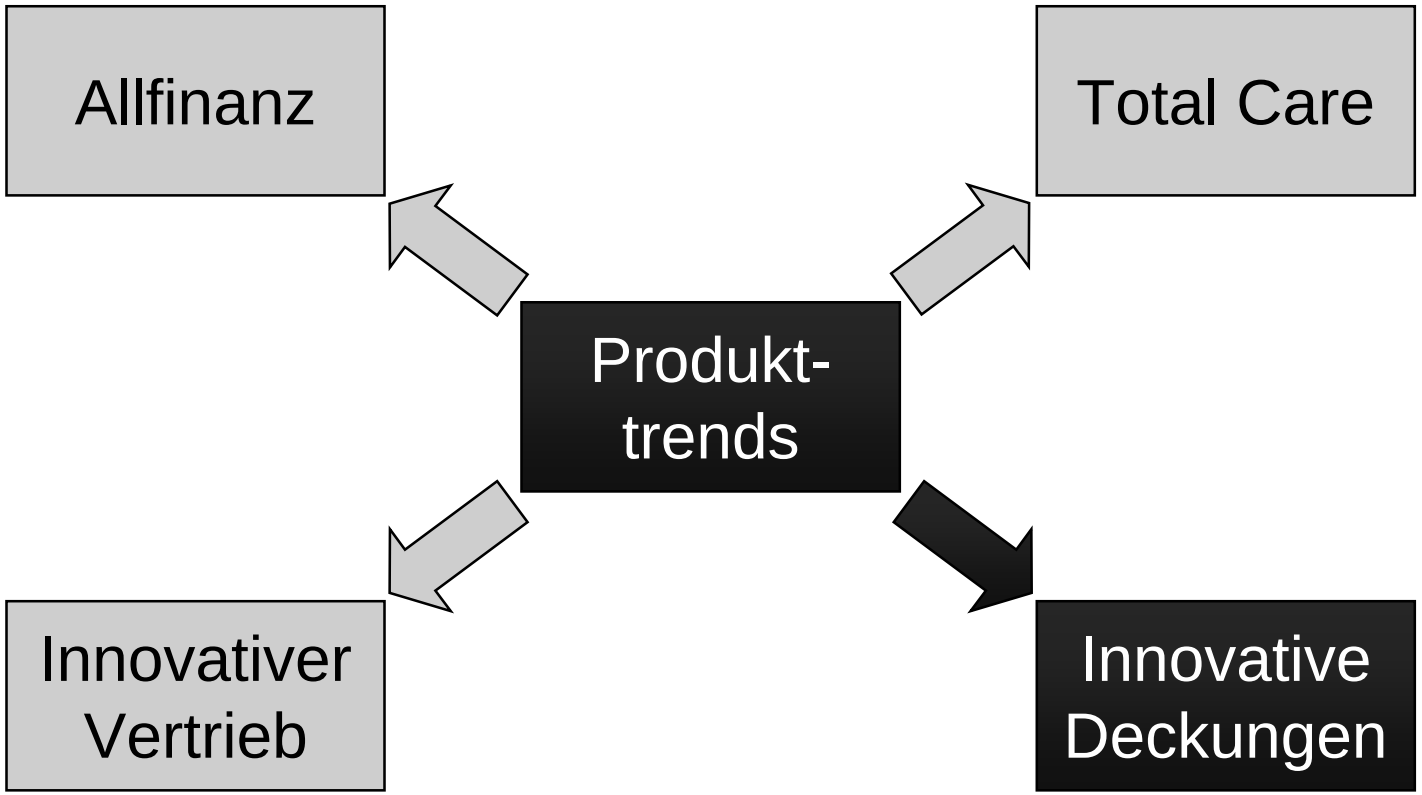


- ☞ Der fachliche Hintergrund
- Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - Meta-System
- Zusammenfassung

Die Flexibilität der Produkte ist ein strategischer Wettbewerbsvorteil



Die Trends am Versicherungsmarkt gehen in mehrere Richtungen

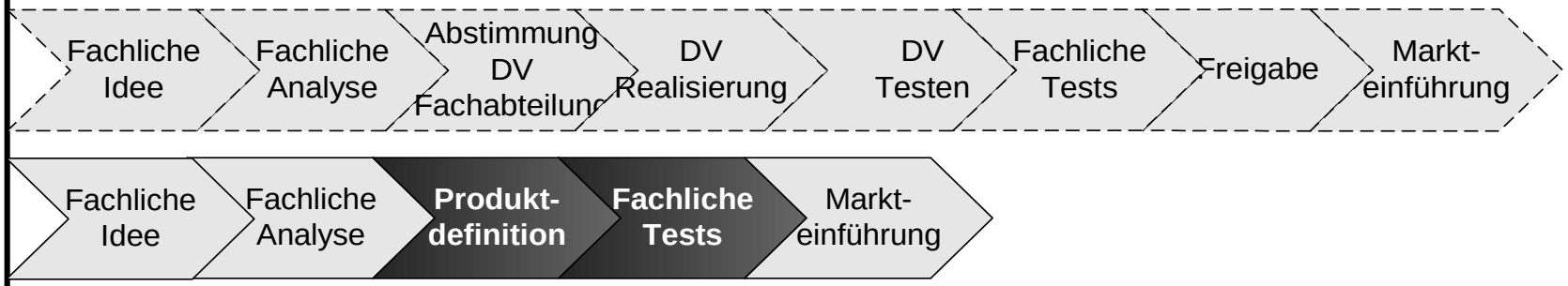


Der typische Produktzyklus bietet noch reichlich Beschleunigungspotential

- Lange Zeiten zur Markteinführung
- Hoher Abstimmungsaufwand
- Teuer

⇒ **Schlechte Wettbewerbsposition**

Ein Produktsystem trägt sein Schärflein zur Beschleunigung bei



- Produktdefinitionen durch Fachabteilung ohne Änderung an bestehender Software
- Verringerter Wartungsaufwand
- Schnellere Reaktionsfähigkeit

⇒ **Bessere Voraussetzungen für gute Wettbewerbsposition**

Wir unterscheiden zwischen Produkt*findung* und Produkt*definition*

Produktfindung

- Analyse der bestehenden Produkte
- Statistische Auswertungen Markttrends
- Wettbewerbsstrategie
- Simulationen

$\Rightarrow$ BWL

Produktdefinition

- Detaillierte Beschreibung
- Umsetzbar in DV
- Test der Definition

$\Rightarrow$ Grenzbereich BWL$\leftrightarrow$DV

Jedes System hat andere Anforderungen an das Produktsystem...

Angebot

- Schnelle Tarifierung
- Vergleichbarkeit von Produkten
- „Argumentationshilfen“
- Cross-Selling
- Internet-Fähigkeit

Bestand

- Tarifierung
- Annahmerichtlinien
- Historisierung
- Produktwechsel
- Statusführung
- Transaktionsfähigkeit

...die zu unterschiedlichen Ansätzen führen können

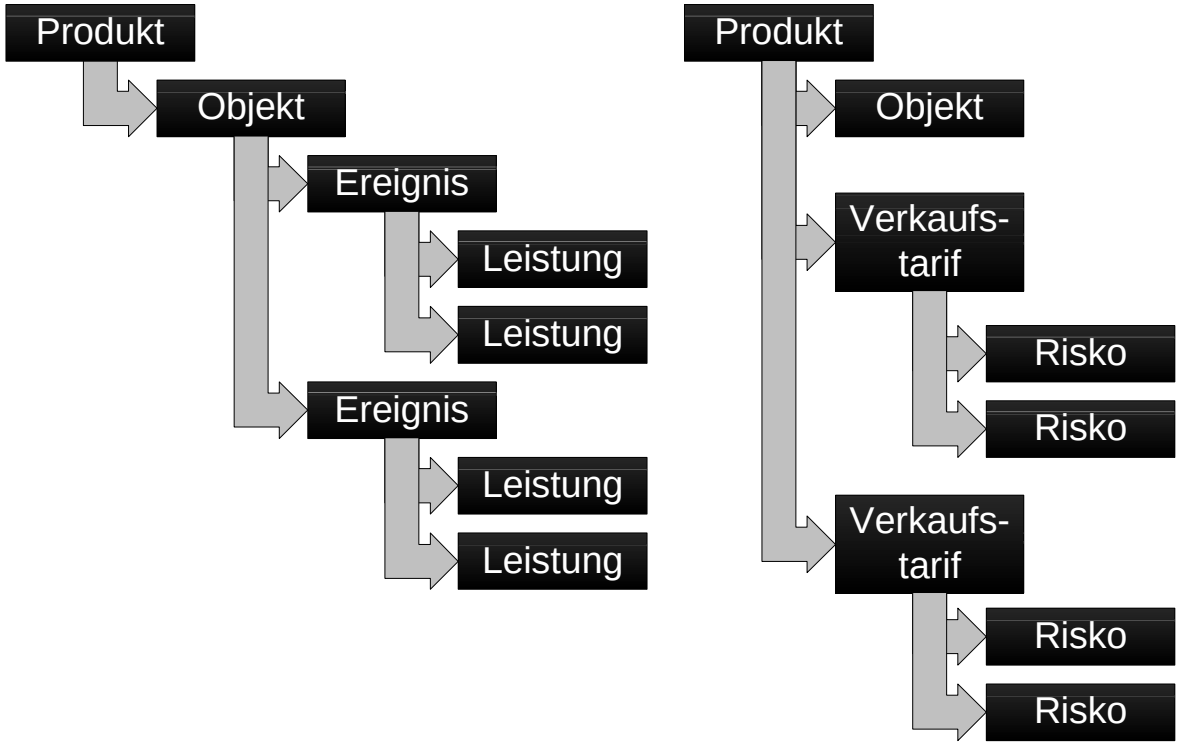
Schaden/Leistung

- Deckungsumfang
- Identifikation von Schnelläufern

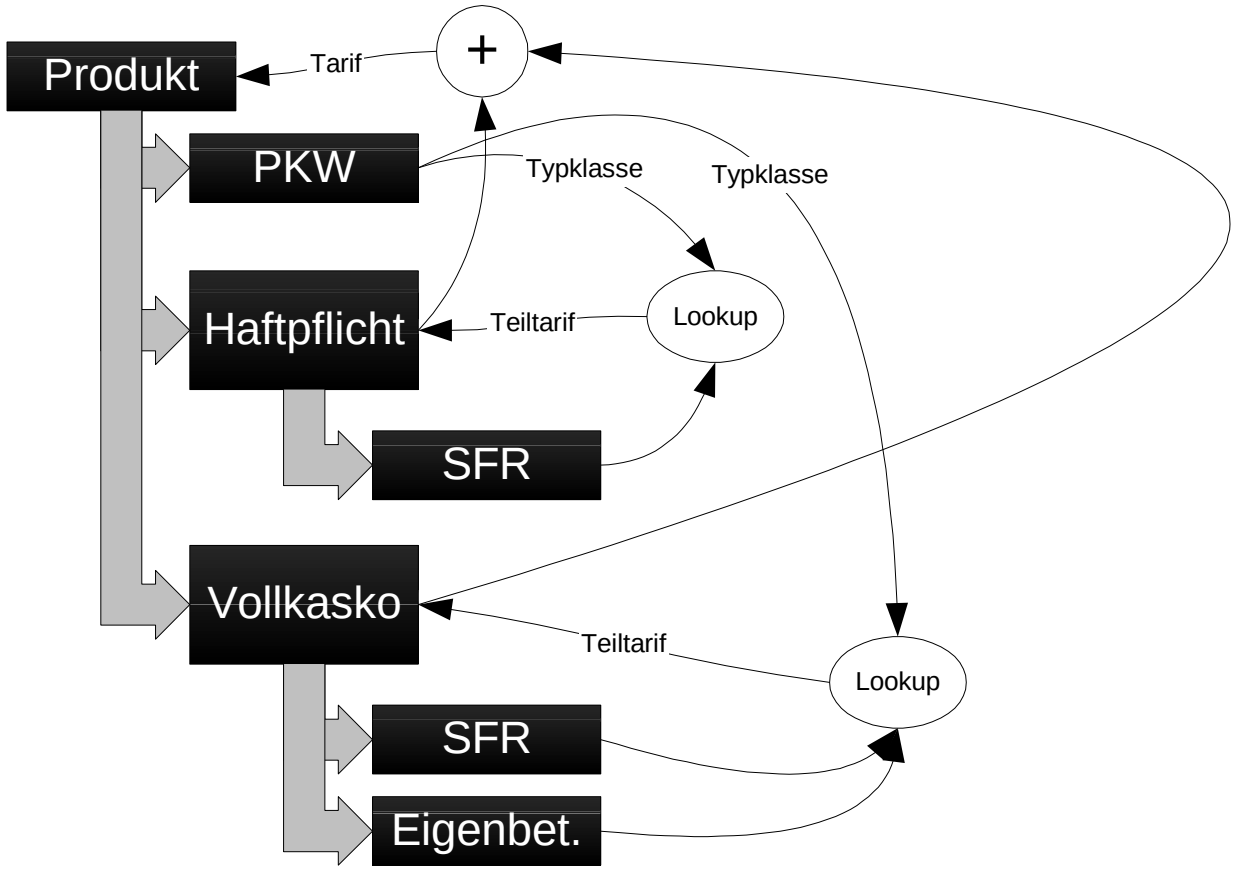
Statistik

- Vergleich verschiedener Produkte
- Freier Zugriff auf Bestandsdaten
- Keine definierten Geschäftsprozesse

Ein gängiges Produktmodell für Sach-versicherungen ist der Produktbaum



Berechnungen im Baum bilden die Zusammenhänge ab



Für Leben und Personenversicherungen trägt dieses Modell jedoch nicht

Sachversicherungen

- Differenzierung über Deckung
- Einheitliche Verarbeitung
- Keine individuellen Rückstellungen

⇒ Unterschiedliche Produktstruktur

Lebens- und Personen-versicherungen

- Differenzierung über Rückstellungen
- Komplexe Verarbeitung
- Individuelle Rückstellungen

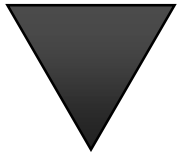
⇒ Unterschiedliche Produktdynamik

Wichtig ist die fachliche Abgrenzung

Im Grenzbereich der Produktdefinition liegen Definitionen von:

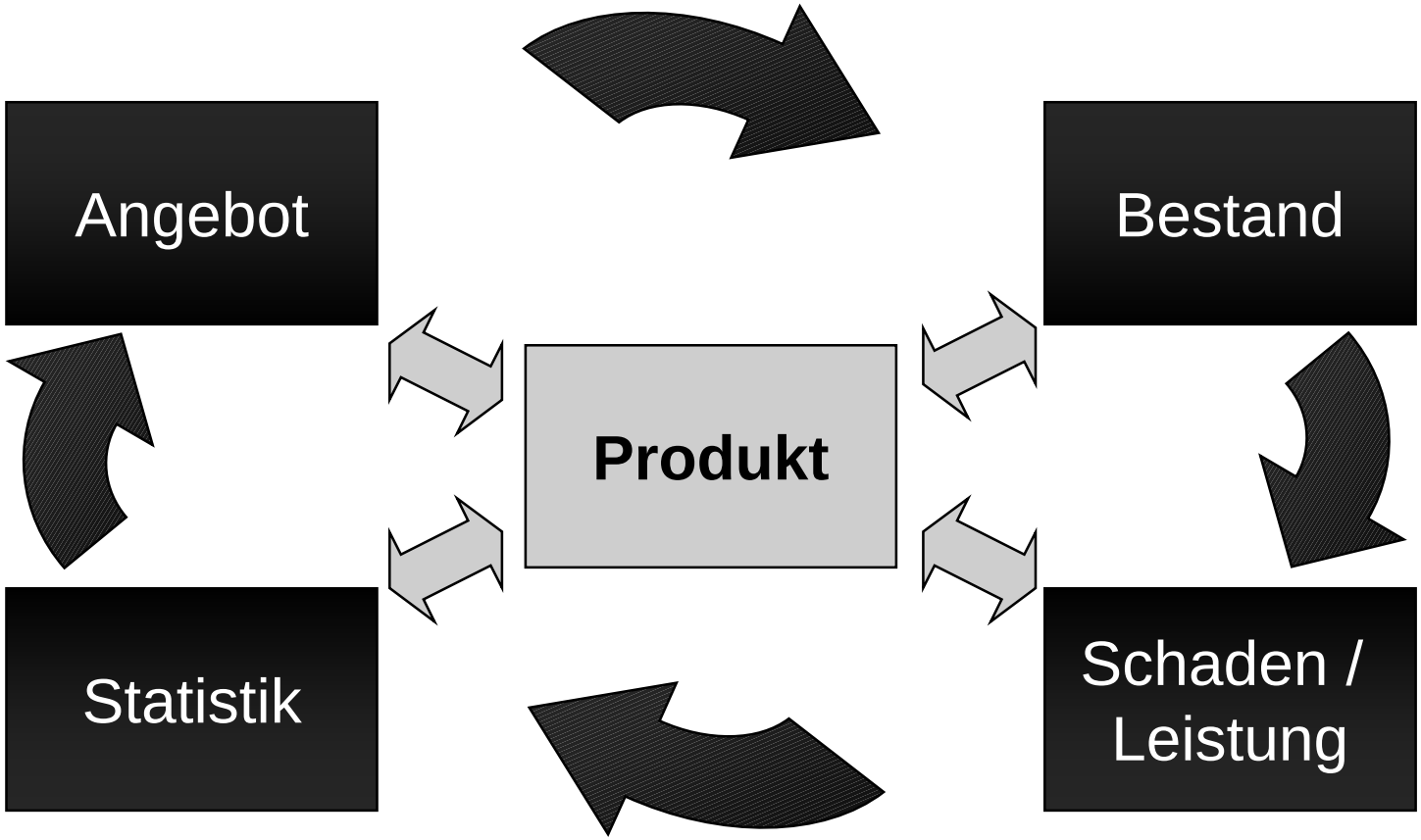
- versicherbaren Objekten und Partnerdaten
- Dokumenten
- Workflow
- Rückversicherung
- Vertrieb und Provision

$\Rightarrow$ Wichtig ist die pragmatische Einpassung in die existierende Landschaft

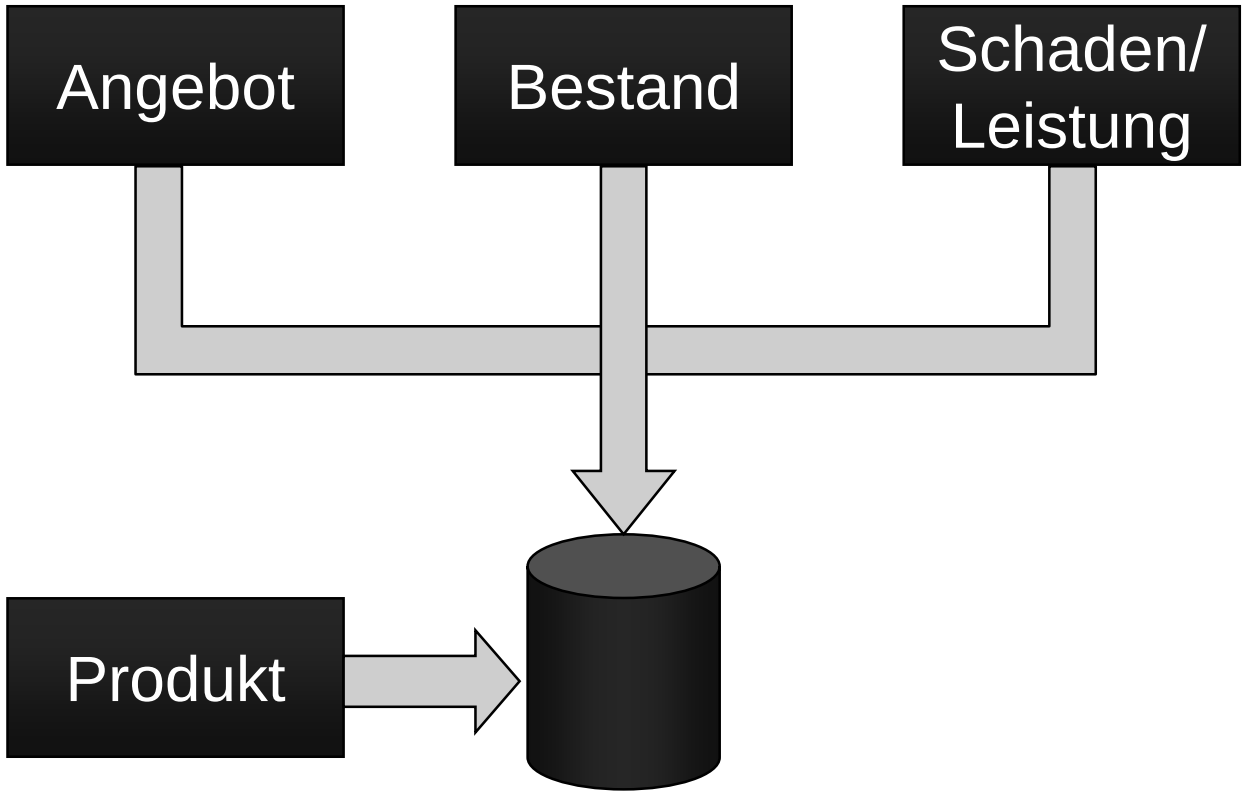


- Der fachliche Hintergrund
- ☞ Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - Meta-System
- Zusammenfassung

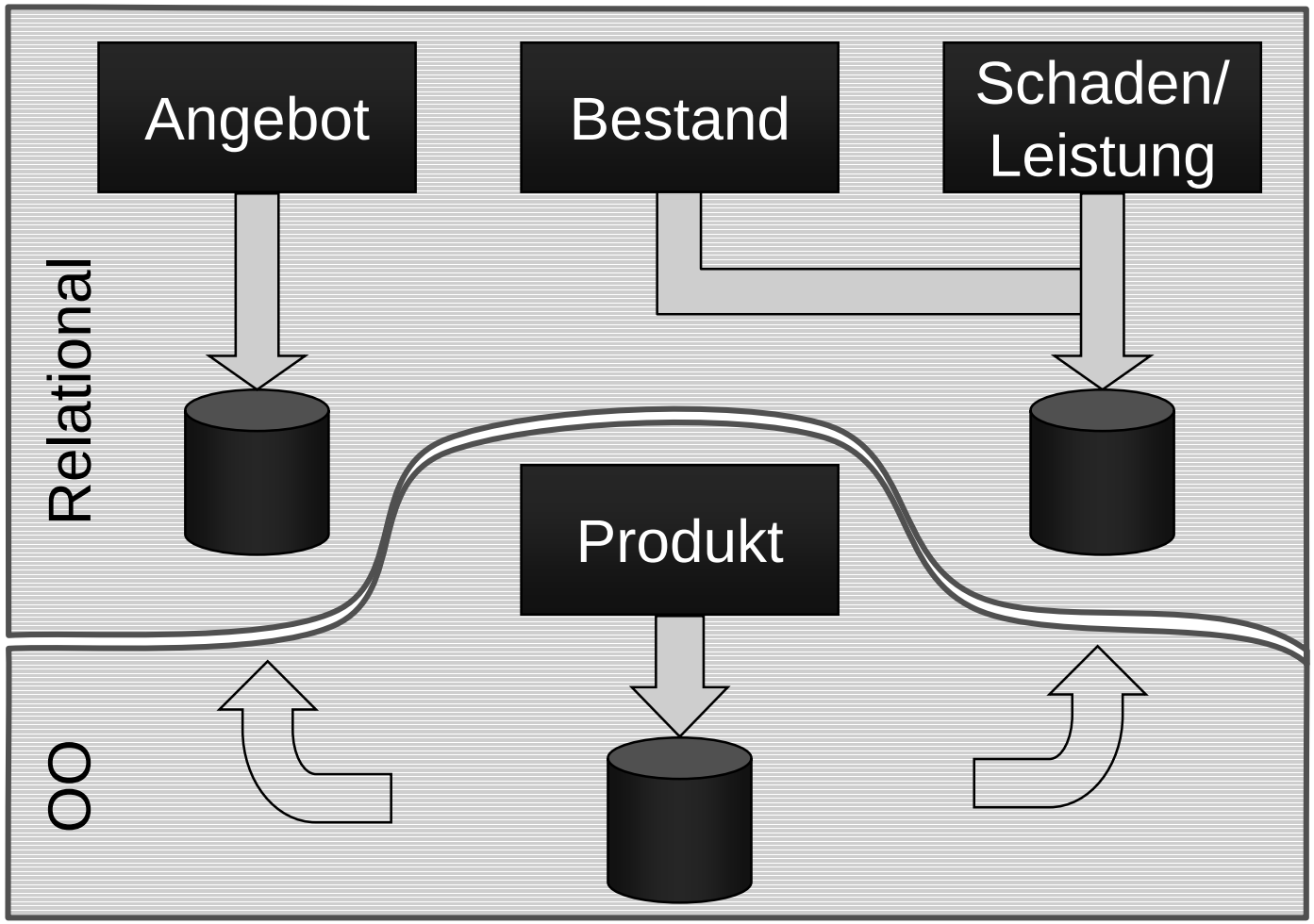
Ein isoliertes Produktsystem ist eine Brücke in der Landschaft



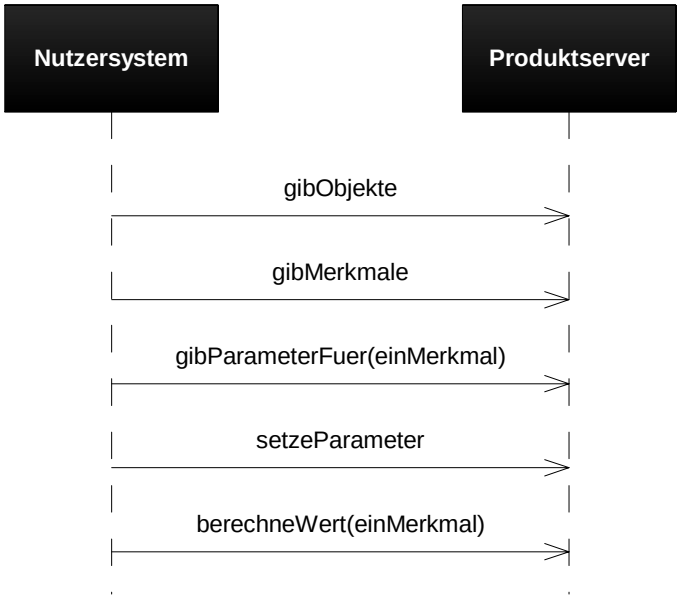
Der naive Ansatz wird den unterschiedlichen Anforderungen nicht gerecht



Statt dessen ist eine klare Trennung in Buildtime und Runtime meist günstiger



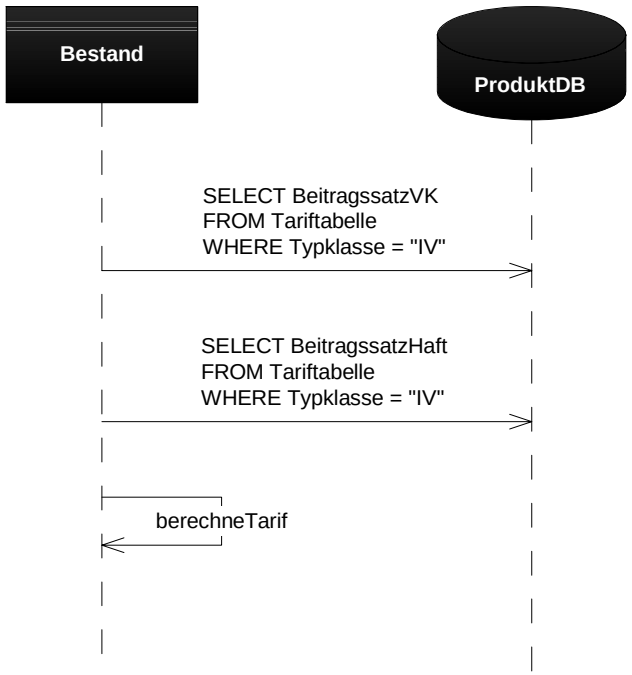
Eine reflektive Schnittstelle erlaubt lose Kopplung zwischen Nutzer- und Produktsystem



z.B.: VP/MS (PMS micado)

- \+ Schmale Schnittstelle
- \+ Flexibel
- \- Setzt Wissen über Struktur voraus
- \- Kritisch im Transaktionsbetrieb
- \- Nicht geeignet, um Bestände zu steuern
- \- Inperformant bei Massenverarbeitung
- $\Rightarrow$ Gut geeignet für Angebotssysteme

Konfigurationsschnittstellen sind gut geeignet, um Altsysteme zu steuern



z.B.: SST (FJA)

\+ Geringer Anlayseaufwand

\+ Wenig Eingriff in bestehende Systeme

\+ Gute Performance

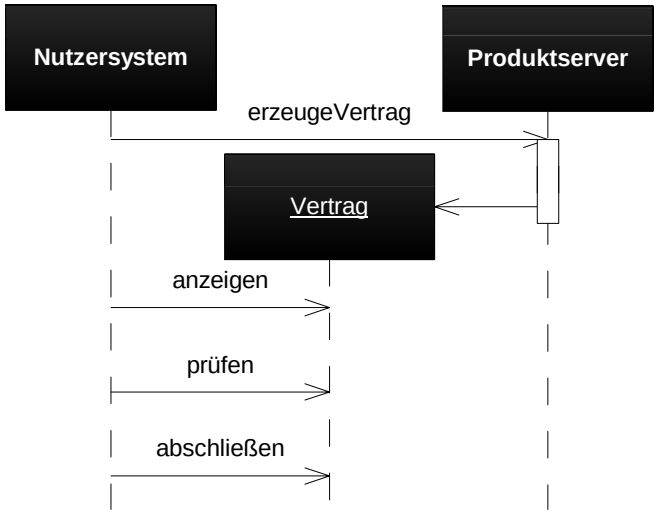
\+ Transaktionsfähig

\- Unflexibel

\- Eingeschränkte Modellierung

$\Rightarrow$ Gut geeignet als Migrationsstrategie

Produktkomponenten ermöglichen gute Performance bei hoher Flexibilität



z.B.: PDFS Metaruntime (Generali)

\+ Freigabe per Knopfdruck möglich

\+ Saubere Schnittstellen

\+ Gute Performance

\+ Transaktionsfähig

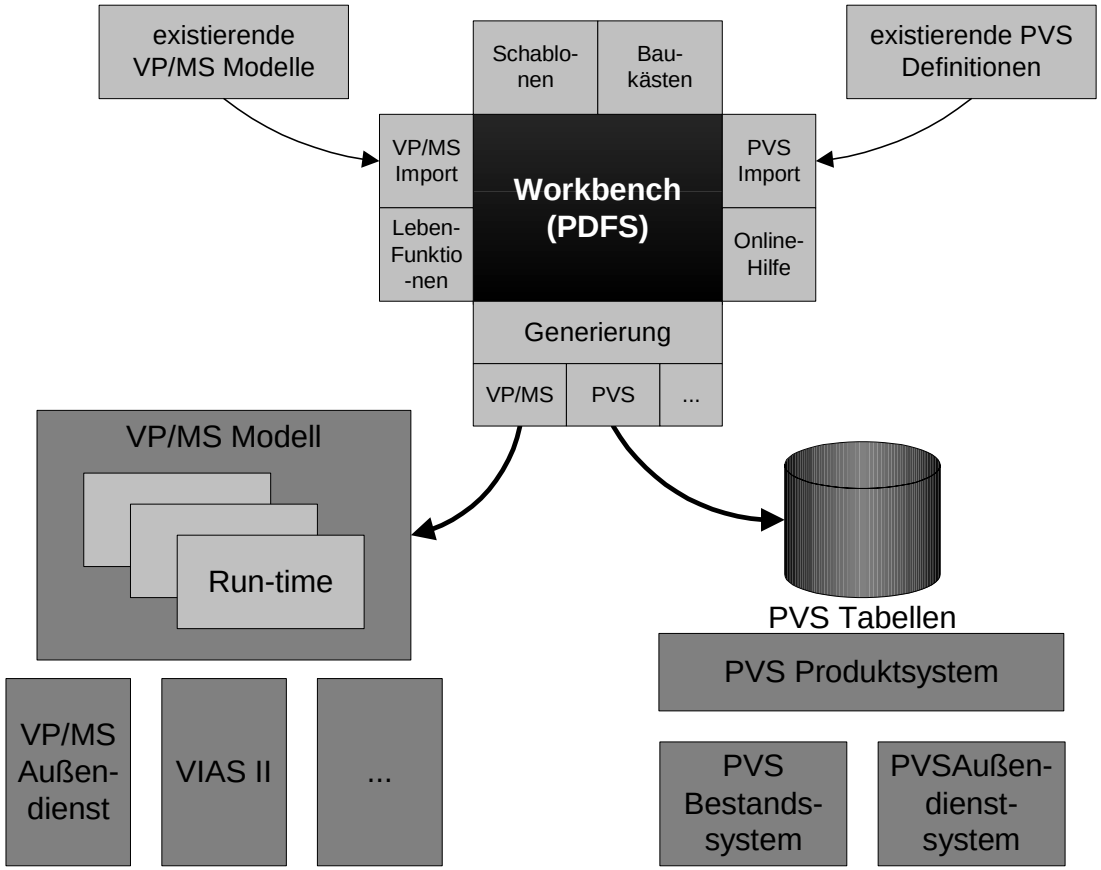
\- Aufwendig zu bauen

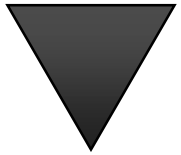
\- Hohe Processorlast

\- Schwer mit Statistik zu koppeln

$\Rightarrow$ Geeignet für Backoffice

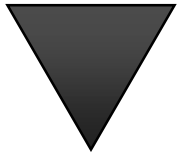
In großen Systemen ist es daher sinnvoll, die Ansätze zu kombinieren





- Der fachliche Hintergrund
- Technische Einbettung
- ☞ Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - Meta-System
- Zusammenfassung

Zunächst einmal eine kleine Demo

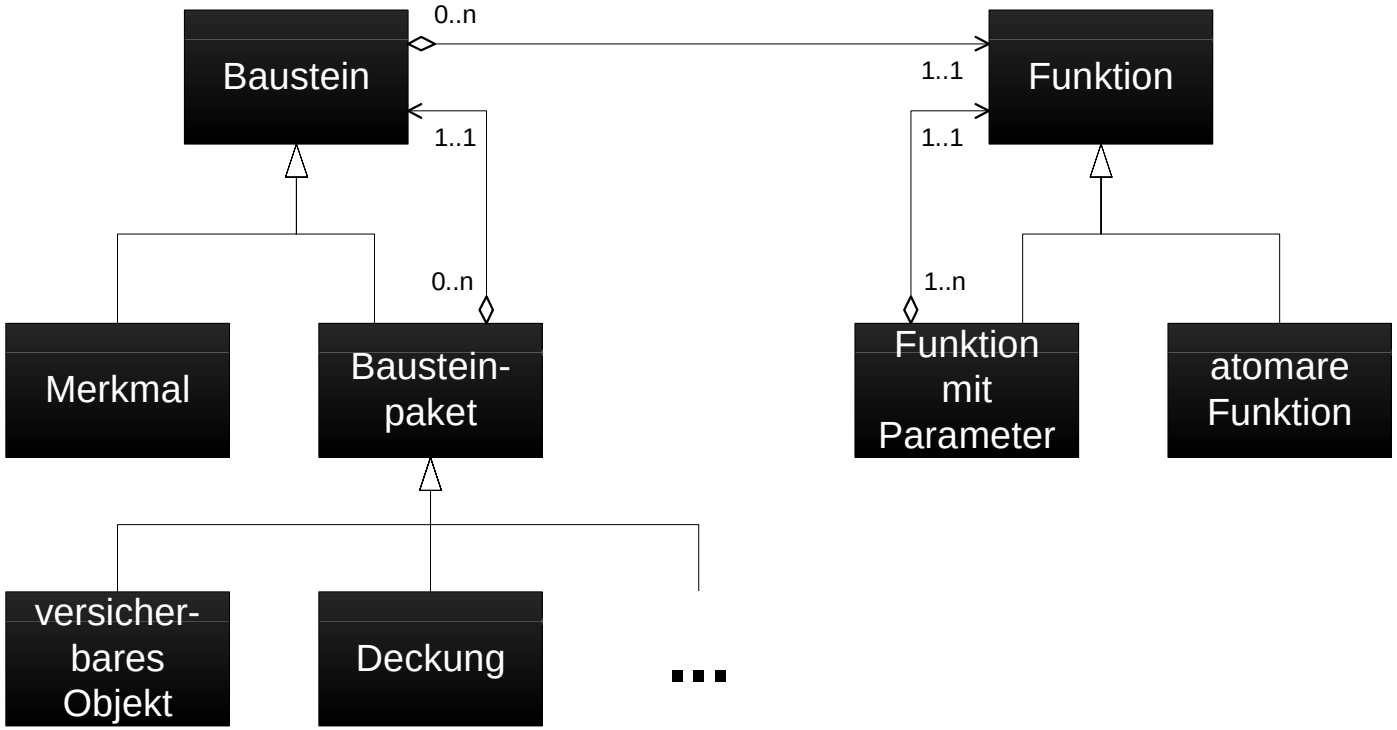


Die Anforderungen sprechen für eine objektorientierte Realisierung

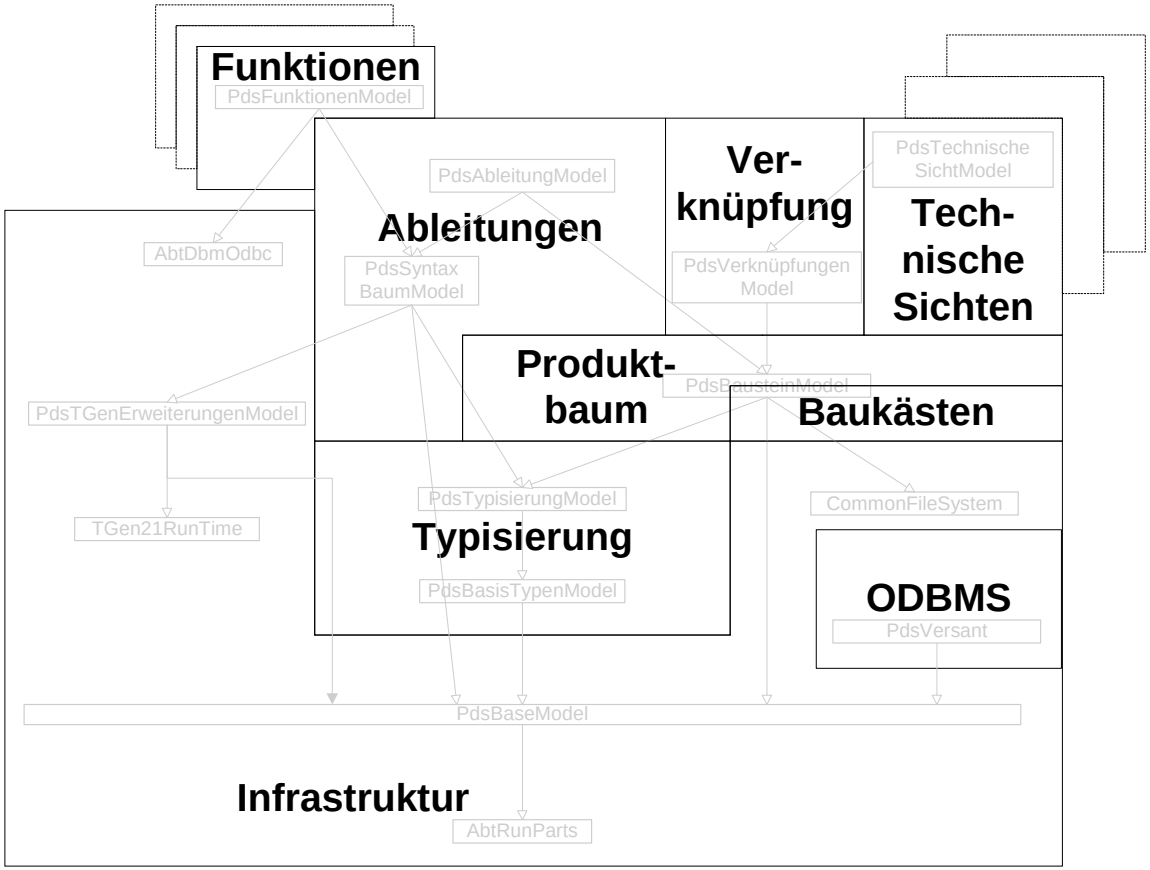
- Projektorientierte Oberfläche mit langlaufenden Transaktionen (CAD-style)
- Hohe Flexibilität notwendig
- Manipulation meist auf abstrakter Ebene (Baustein, Funktion, etc.)
- Enges Geflecht kleiner Klassen
- Bedienung arbeitet vor allem mit Navigation

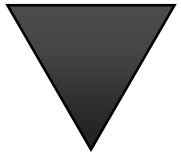
$\Rightarrow$ Objektorientierte Realisierung und DB

Das grundlegende Design ist ein attributierter Baum



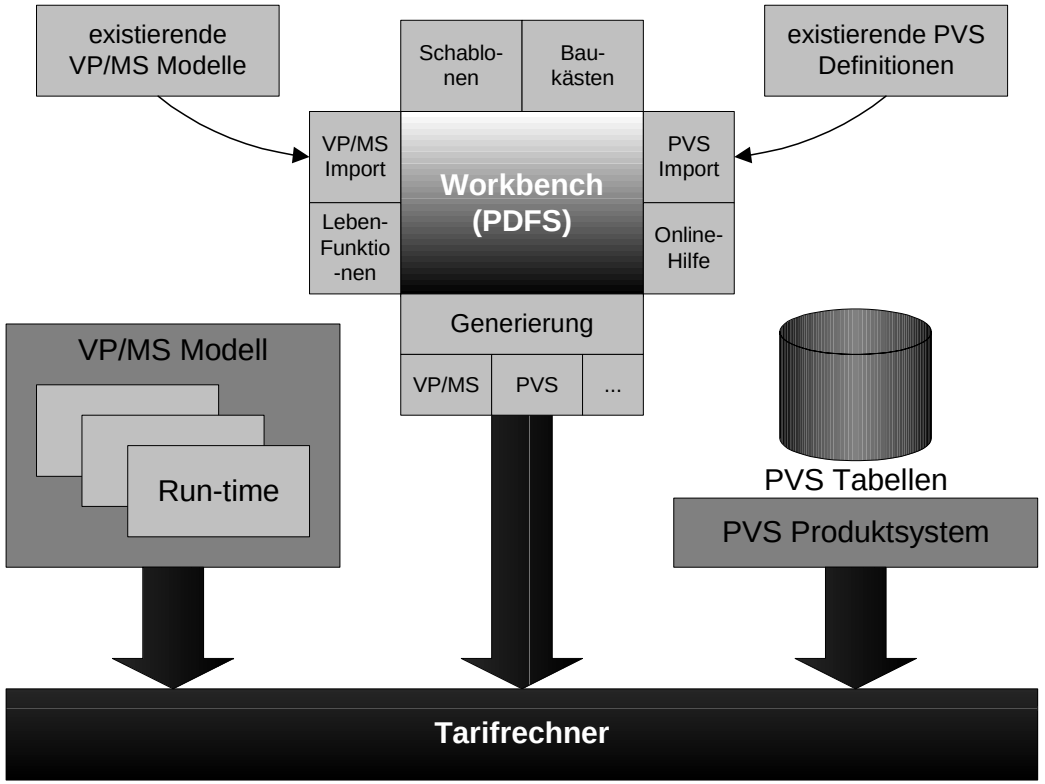
In der Praxis wird die Architektur etwas aufwendiger...

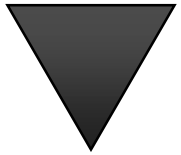




- Der fachliche Hintergrund
- Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - ☞ Tarifrechner
    - – Meta-System
- Zusammenfassung

Ein Tarifrechner stellt einheitliche Auswertungen für alle Systeme sicher



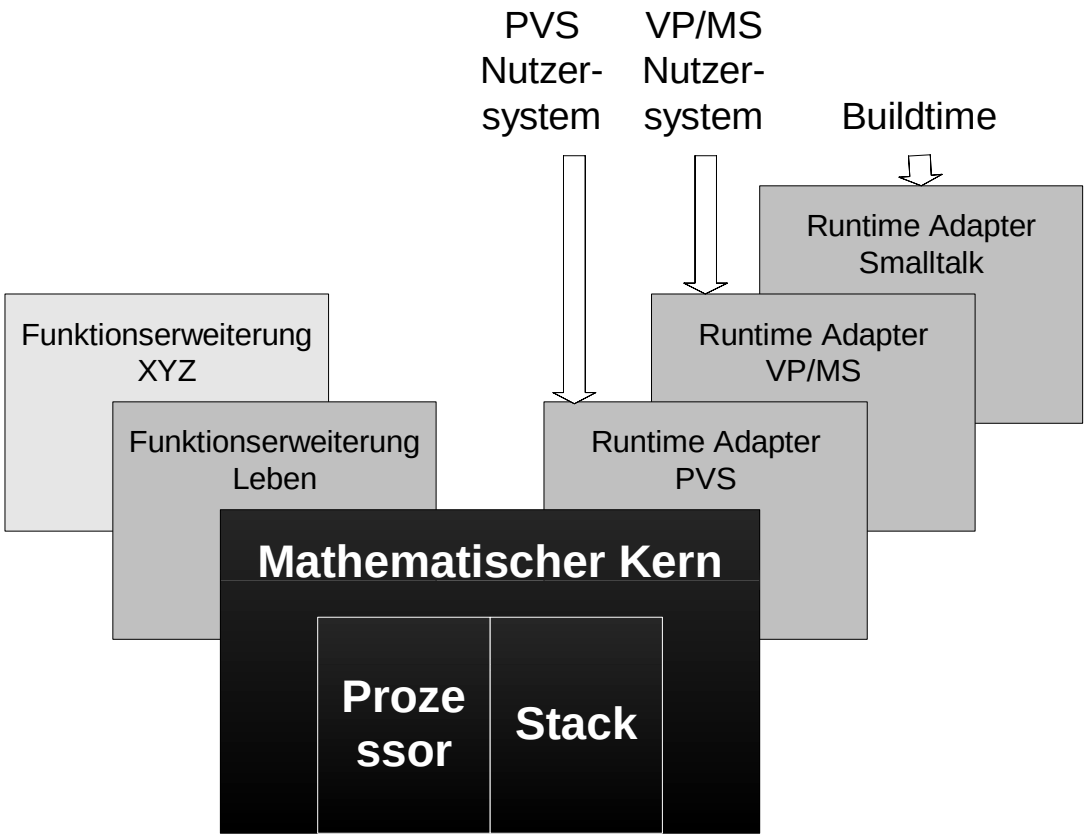


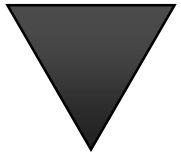
Ein Tarifrechner muß harten technischen Anforderungen genügen

- Kern codeidentisch auf allen Plattformen
- Hohe Performance
- Transaktions- und sessionfähig
- Schnittstelle sprach- und paradigmen-übergreifend
- leicht erweiterbar

$\Rightarrow$ C Realisierung mit plattformspezifischen Erweiterungen

Der Kern des Tarifrechners besteht aus einer Stackmaschine





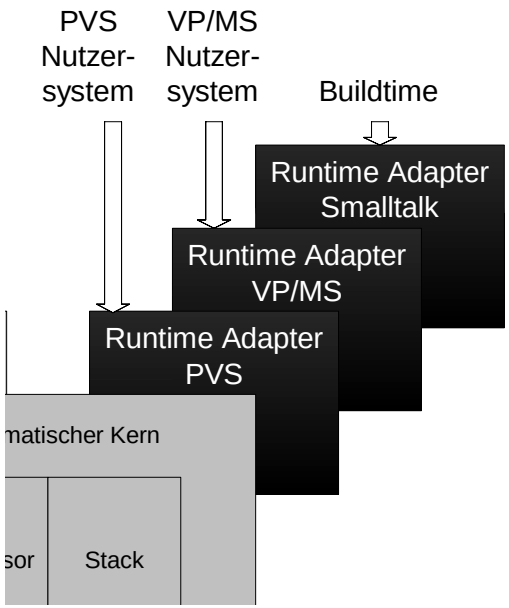
Der Kern stellt einige wenige Funktionen zur Verfügung

- sessionHnd startSession ()
- void setVar (sessionHnd, id, wert)
- wert compute (sessionHnd, byteCode)
- void closeSession (sessionHnd)

Diese Funktionen können in einer lib oder einer DLL sprachübergreifend zur Verfügung gestellt werden

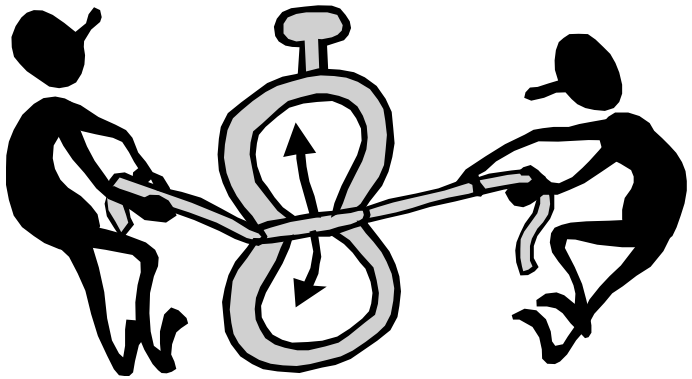
Plattformspezifische Funktionalitäten werden in Adaptern bereitgestellt

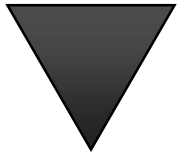
- Sprachanbindung
- Bereitstellen des produktspezifischen Bytecodes
- Bereitstellen der Variablen
  - Im voraus
  - Bei Bedarf
- Datenbank Lookup



Dieser Ansatz ist sehr gut geeignet für Angebotssysteme

- Performance: ca. 5µs/Operation
- Sehr leicht zu bedienende Schnittstelle
- Schwierig: Sessions über Transaktionsgrenzen hinweg
- Gut geeignet, um bereits bestehende Produkte in flexible Umsysteme einzuspeisen





- Der fachliche Hintergrund
- Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - ☞ Meta-System
- Zusammenfassung

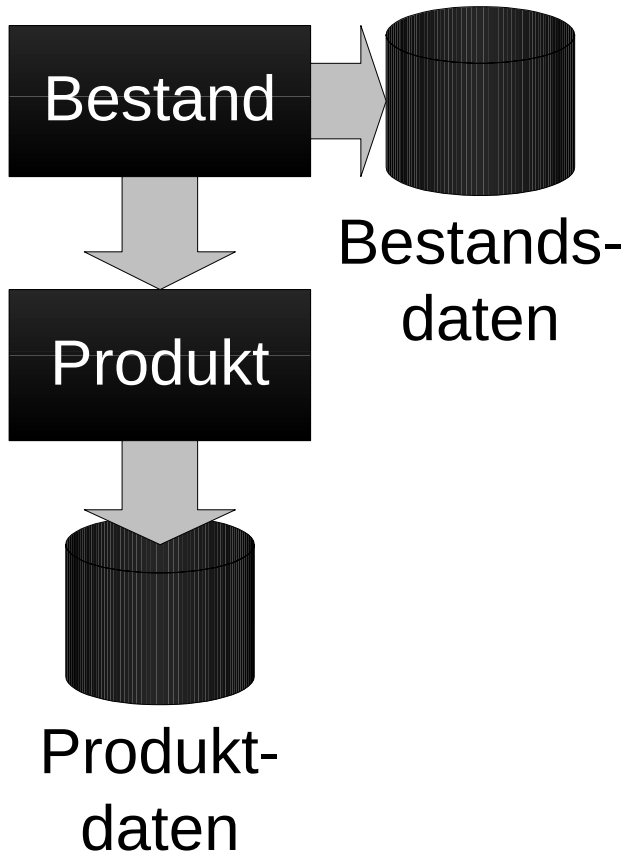
Eine Meta-Runtime ermöglicht flexible Produkte auch im Bestand

Bestand

- hält die Bestandsdaten
- sorgt für Historisierung
- implementiert Abwicklung (z.B. Inkassokopplung)

Produkt

- liefert Datenbank-schema für Bestand
- führt Prüfungen und Berechnungen auf Verträgen durch



Die nicht-funktionalen Anforderungen legen relationale Techniken nahe

- Transaktionsorienterte Oberfläche
- Genau definierte Geschäftprozesse
- Manipulation meist auf konkreter Ebene (Vertrag erfassen, ändern etc.)
- Große Anzahl ähnlicher Verträge
- Bedienung vor allem auf einzelnen Verträgen über Primärschlüssel
- Hohe Zuverlässigkeitsanforderungen

$\Rightarrow$ **Relationale DB - meist auch klassische 3GL Realisierung**

Sowohl die Datenhaltung als auch die Berechnungen müssen flexibel sein

Flexibilisierung der Datenhaltung

- Vertikale Datenhaltung
- Produktübergreifende Schlüssel zur Identifikation vergleichbarer Merkmale
- Sorgfältiger Betrieb

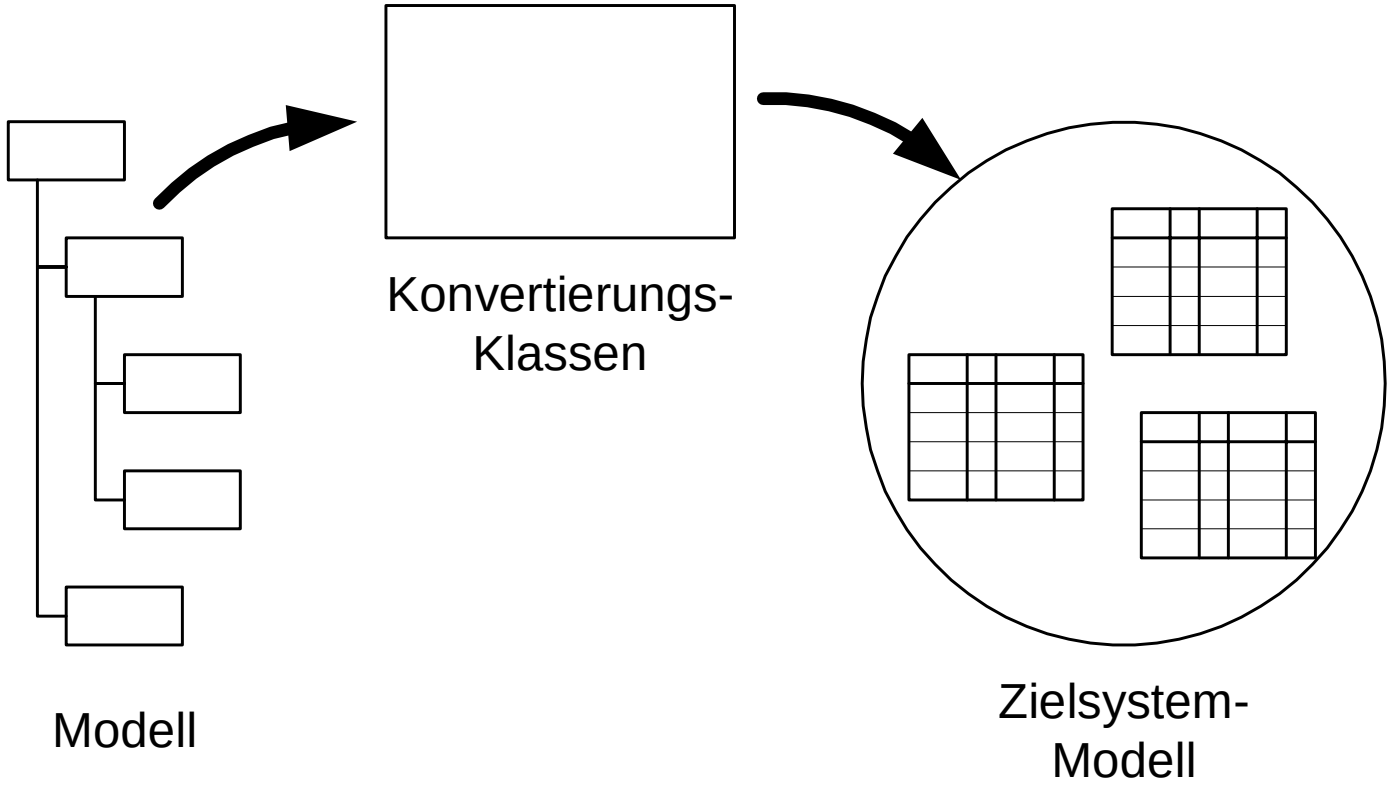
☞ ca. 100 Zugriffe/Vertrag

Flexibilisierung der Berechnungen

- Anschluß eines Tarifrechners
- Versionierte Speicherung des Bytecodes in der DB
- Bereitstellen der Daten paketweise

☞ Durchrechnen eines K-Vertrages < 1ms

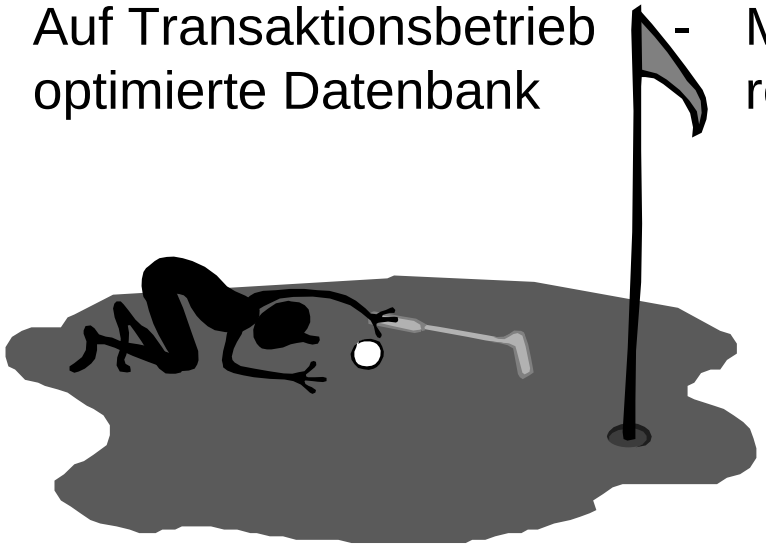
Der Export aus dem Definitionssystem erfolgt nach klassischem Compilerbau

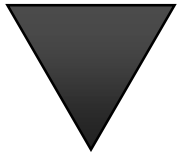


Eine Meta-Runtime ist teuer, bringt aber auch deutliche Vorteile

+ Hohe Flexibilität bei den Produkten
+ Mainframe und RDBMS bringen im Betrieb deutliche Vorteile
+ Auf Transaktionsbetrieb optimierte Datenbank

- Die Last steigt deutlich
- Export ist um so aufwendiger, je mehr Altlasten berücksichtigt werden müssen
- Mainframe und PC rechnen unterschiedlich



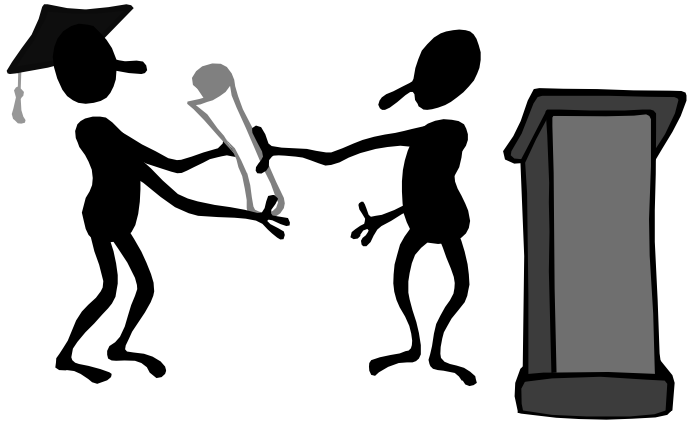


- Der fachliche Hintergrund
- Technische Einbettung
- Design der Buildtime
- Design der Runtime
  - Tarifrechner
  - Meta-System

☞ Zusammenfassung

Produktsysteme gehören zu den derzeit schwierigsten Herausforderungen

- Produktsysteme sind „bleeding edge“
- Ihre Umsetzung stellt besondere Anforder-ungen an das Team
- Geschickte Kopplung klassischer und moderner Techniken erlaubt schnelles Schließen der Wertschöpfungskette



Vielen Dank für die gute Zusammenarbeit und die hilfreiche Tips an

Team Produkt / Vertrag, Generali München:

- Rudolf Donko
- Michael Augustin
- Stefan Blüml
- Alwine Brem
- Elke Knall
- Peter Krömer
- Andrea Meier
- Michael Niebler
- Robert Sigl
- Heinz Weber

Sowie

- Wolfgang Keller, Generali Wien
- Rudolf Lewandowski, Generali Wien
- Team PMS, Kordoba
- Ruth Leuzinger, Zürich
- Ralph Johnson, UIUC
- Alistair Cockburn
- Jutta Eckstein
- Alexander Rudyj, msg

Die Verwendung von Projektunterlagen erfolgte mit freundlicher Genehmigung der Generali Office Service und Consulting AG

Weiterführende Referenzen

- Paul Schönsleben, Ruth Leuzinger: „Innovative Gestaltung von Versicherungsprodukten - Flexible Industriekonzepte in der Assekuranz“; Gabler Verlag, Wiesbaden
- Wolfgang Keller: „Some Patterns for Insurance Systems“, Conference on Pattern Languages of Programming 1998, http://www.objectarchitects.de/ObjectArchitects/papers/
- Jens Coldewey: „Choosing a Database Technology“, Object Expo 1998, http://www.coldewey.com/publikationen
- Jeff Oaks, Ralph Johnson: „User Defined Product Framework “; http://st-www.cs.uiuc.edu/users/johnson/DOM.html