Statische Skalierbarkeit mit Software-Produktlinien

Daniel Lohmann
Wolfgang Schröder-Preikschat
Olaf Spinczyk

Lehrstuhl für Informatik IV
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander Universität Erlangen-Nürnberg

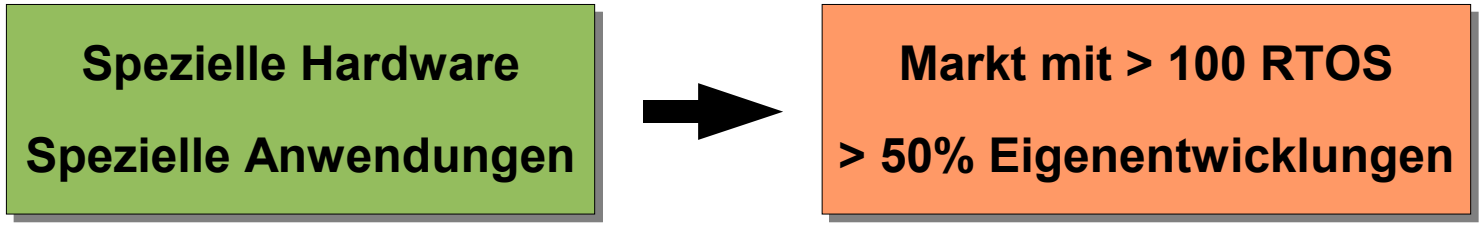
lohmann@cs.fau.de
http://www4.cs.fau.de/~lohmann

Agenda

- Software-Produktlinien: Stand der Kunst
  - Konzepte
  - Methoden
  - Werkzeuge
- Herausforderungen für die Zukunft
  - Automatische Konfigurierung
  - Beherrschbarkeit nicht-funktionaler Eigenschaften
  - Kombination und Integration

Motivation

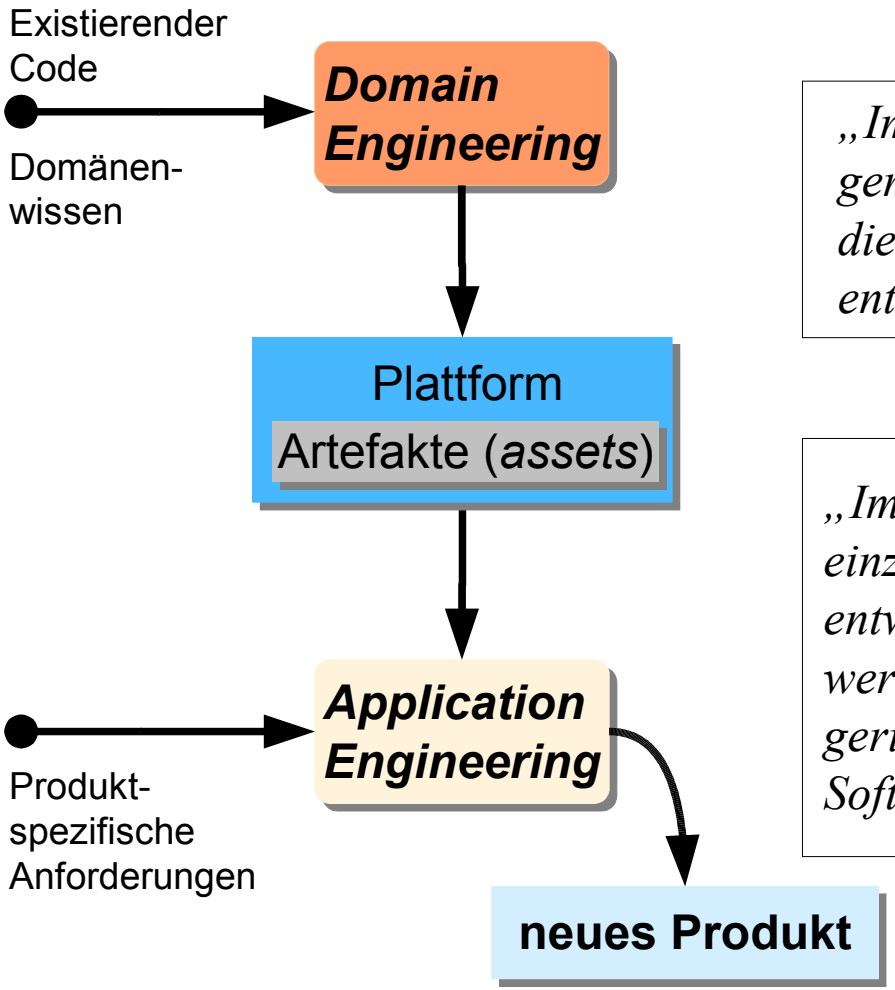
Betriebssysteme für eingebettete Systeme



- „das Rad wird neu erfunden“
  - auch die selben Fehler werden wiederholt
- oftmals bietet **ein** BS Hersteller **mehrere** Systeme an
  - mit getrennter Code-Basis
  - getrieben durch die speziellen Anforderungen seiner Kunden

Software-Produktlinien (Software-PL)

Organisierte Wiederverwendung durch die **aktive** Gestaltung einer gemeinsamen Plattform für aktuelle und künftige Produkte



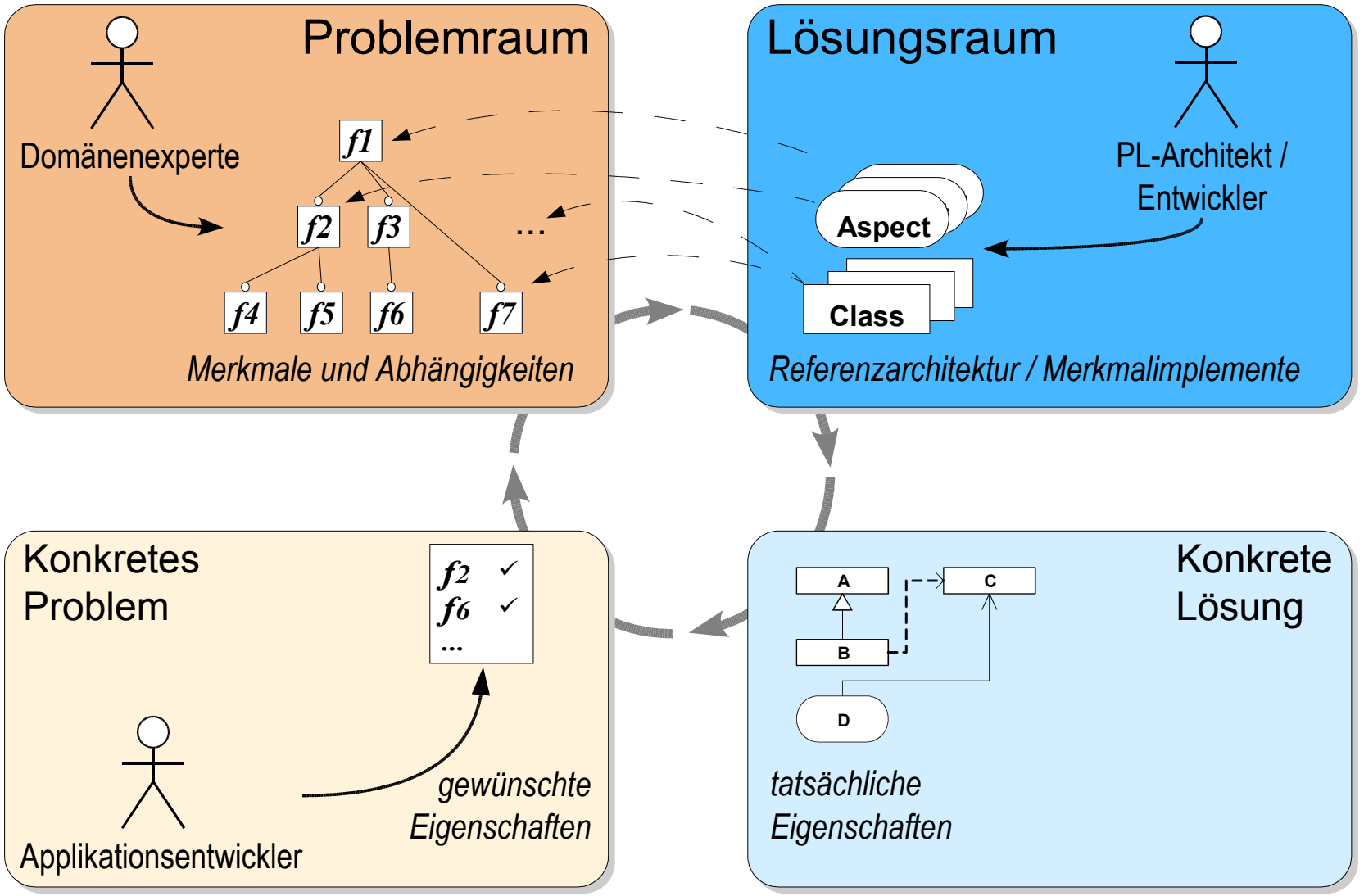
*„Im **Domain Engineering** werden die gemeinsamen und variablen **Artefakte**, die Bestandteile der **Plattform,** entwickelt.“*

[1]

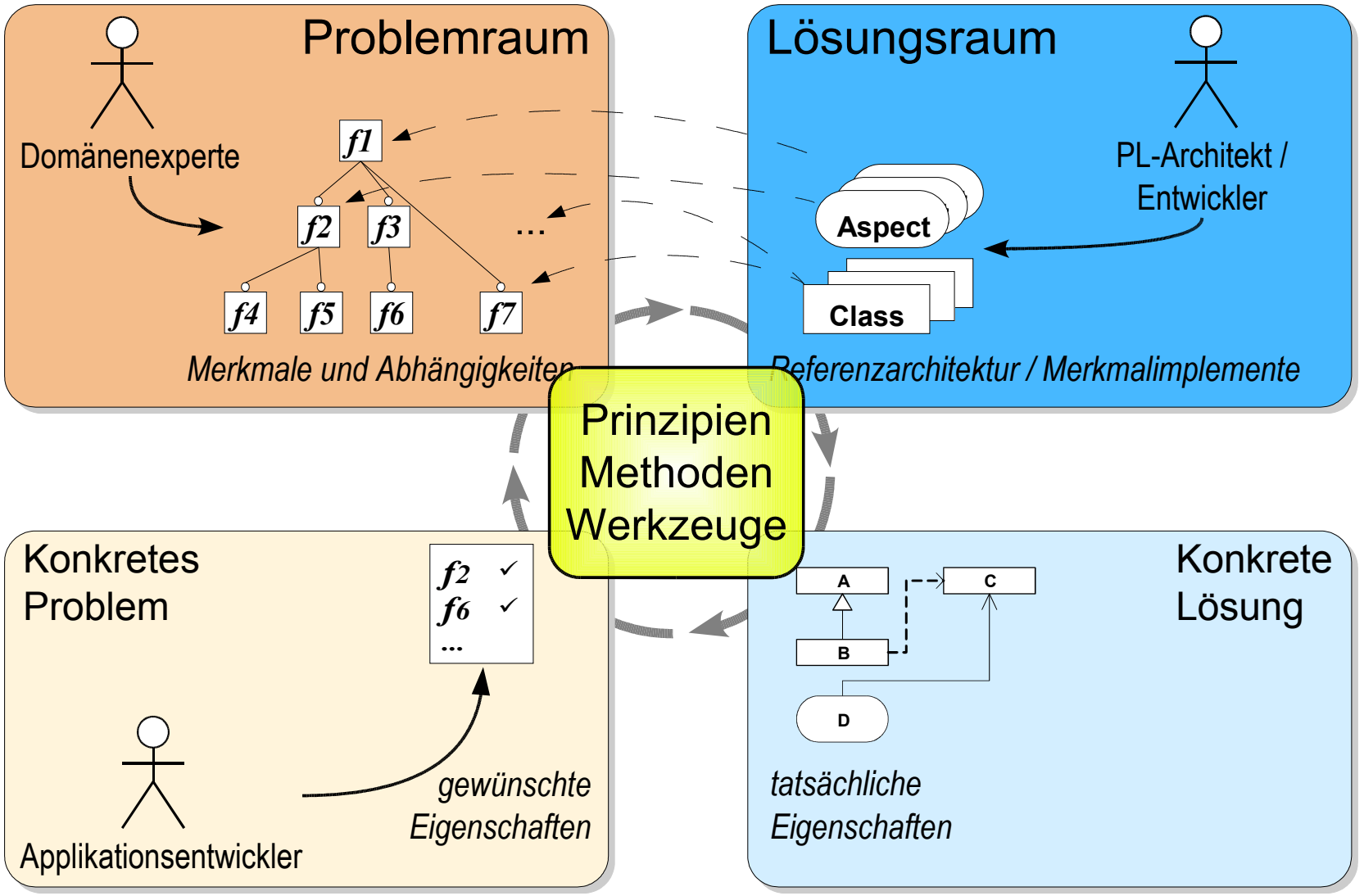
*„Im **Application Engineering** werden einzelne **Produkte** der Produktlinie entwickelt bzw. abgeleitet. Die Produkte werden [...] konfiguriert, so dass nur in geringem Maße produktspezifische Softwareentwicklung notwendig wird.“*

[1]

Überblick: Software-Produktlinien

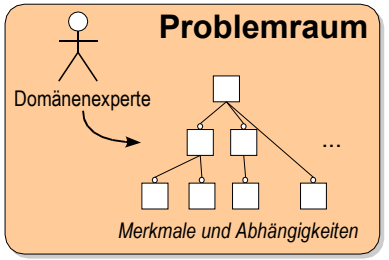


Überblick: Software-Produktlinien

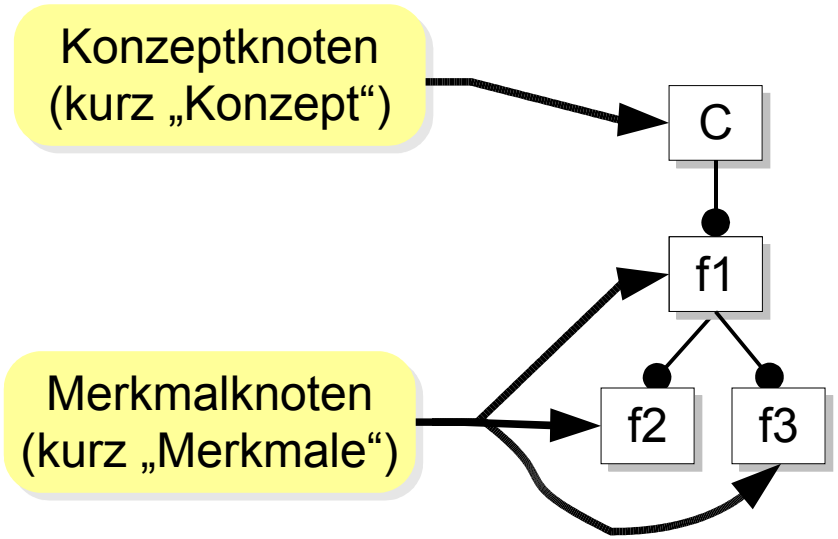


Beschreibung des Problemraums

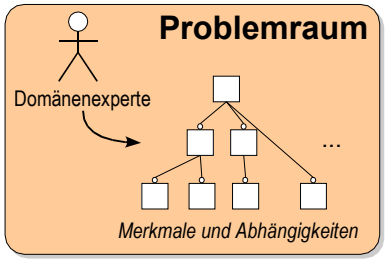
Merkmalmodell (*feature model*) [2]

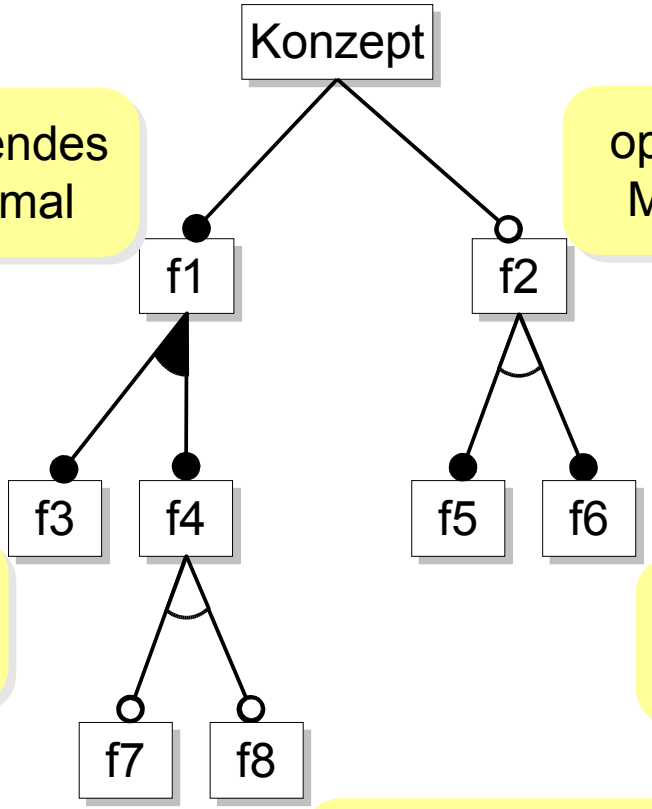


- Gerichteter, azyklischer Graph
- Beschreibt die möglichen ***Ausprägungen*** eines ***Konzepts*** anhand von ***Gemeinsamkeiten*** und ***Unterschieden***



Arten von Merkmalen [2]





zwingendes Merkmal

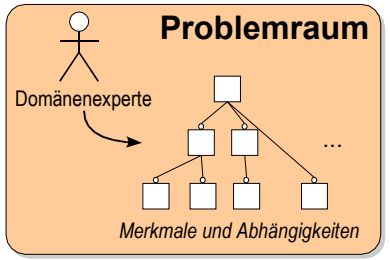
optionales Merkmal

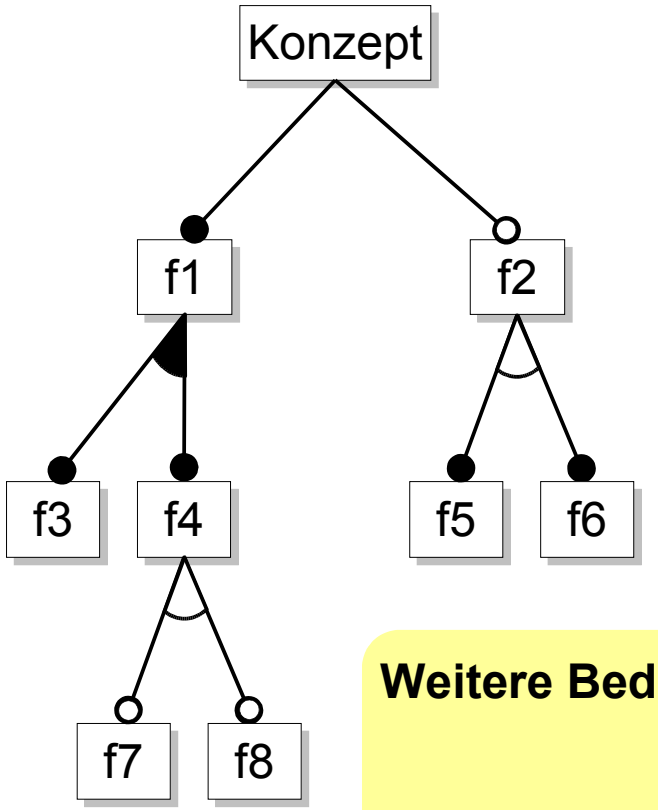
kumulative Gruppe (1 oder mehrere)

alternative Gruppe (genau 1)

optional alternative Gruppe (0 oder 1)

Arten von Merkmalen [2]

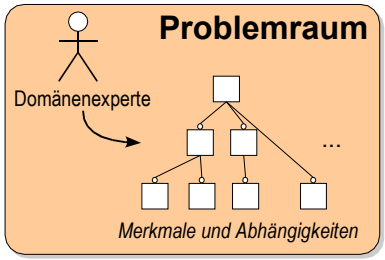


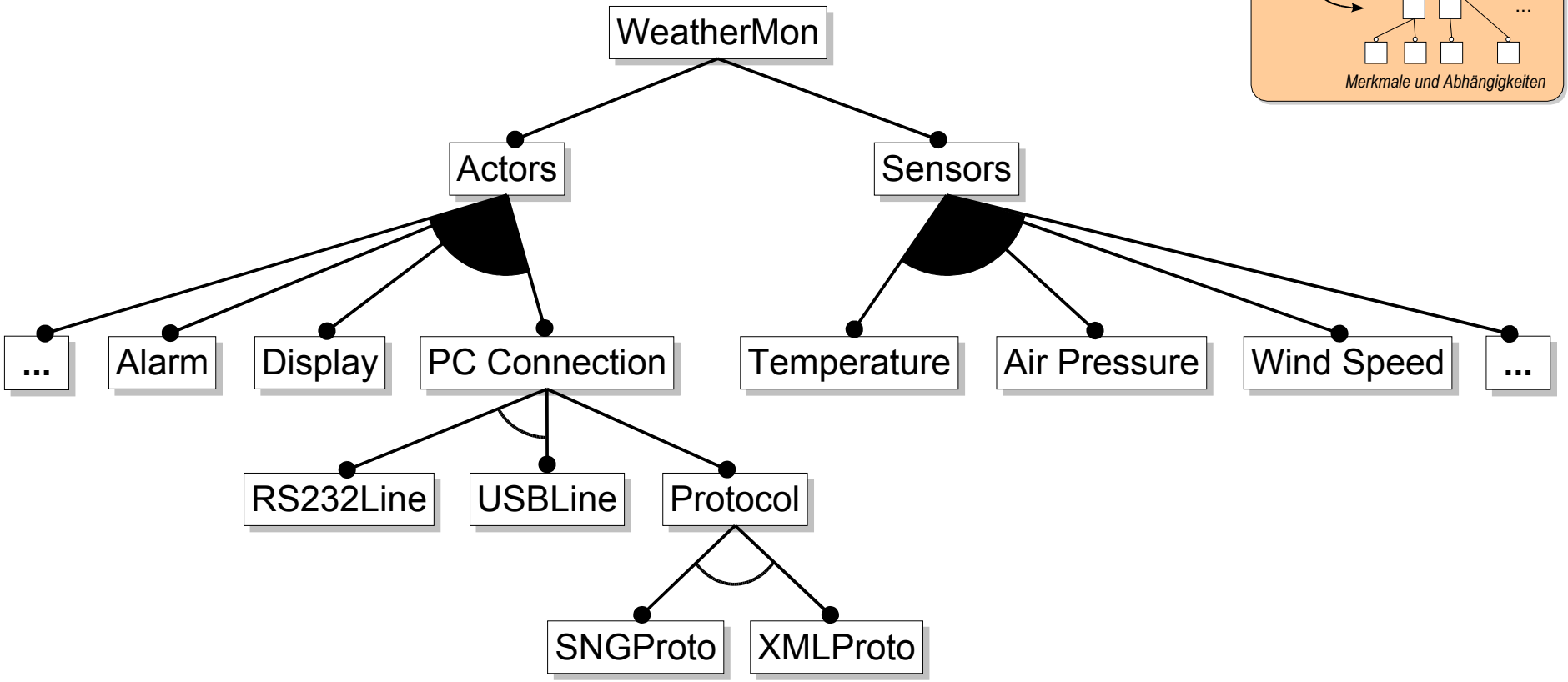


Weitere Bedingungsangaben möglich:

f5 **requires** f3
f7 **conflicts** f6
...

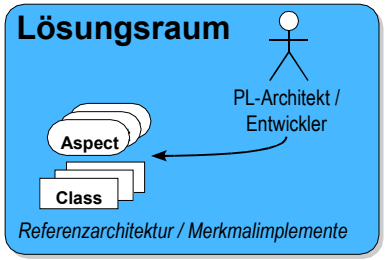
Beispiel (Wetterstation)





Beschreibung des Lösungsraums

Plattform



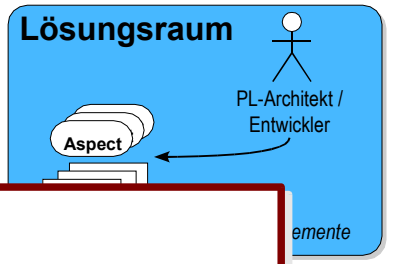
- Komponentenlager / -framework
- ***Implementierung*** der Konzepte und Merkmale

Familienmodell

- ***Abbildung*** aus Komponentenlager in den Problemraum (von Artefakten auf Merkmale):
  $A \rightarrow M$
- Eventuell über zusätzliche ***Transformationen***:
  $(A \times T) \rightarrow M$

Beschreibung des Lösungsraums

Plattform

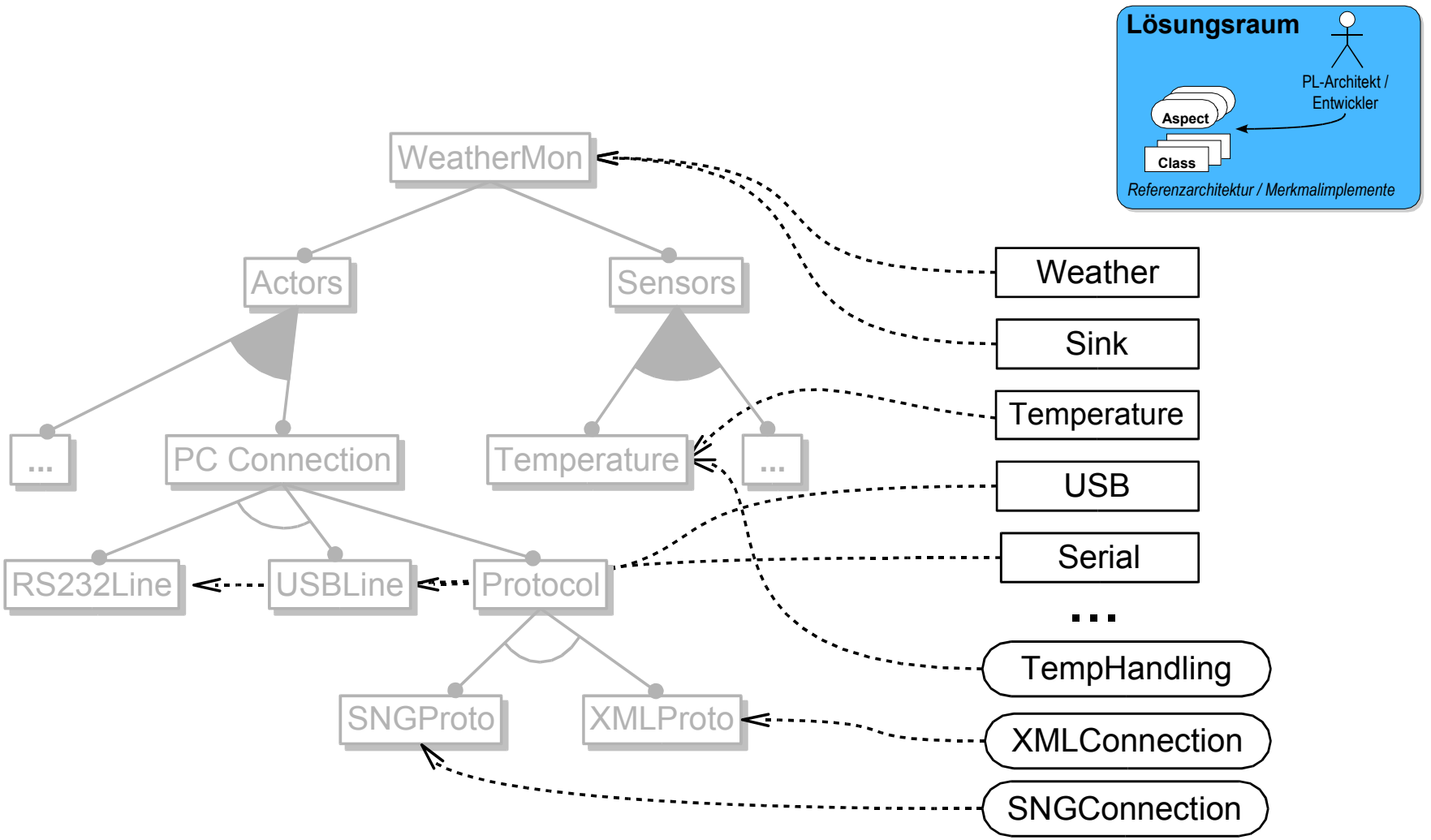


- Komponentenlager / -framework
-
- F
-
- Eventuell über zusätzliche ***Transformationen***:
  $(A \times T) \rightarrow M$

Problem: Trennung der Belange

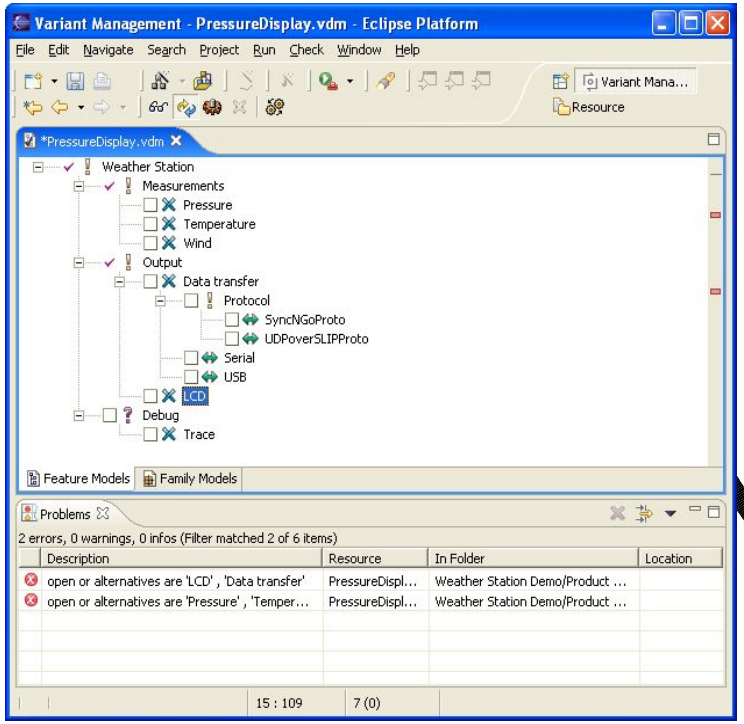
- **Ideal:** Jedes Artefakt ist genau einem Merkmal zugeordnet: $A \rightarrow M$ ist rechtseindeutig
- **Realität:** „`#ifdef`-Hölle“
- **Gute Erfahrungen mit AOP und C++**

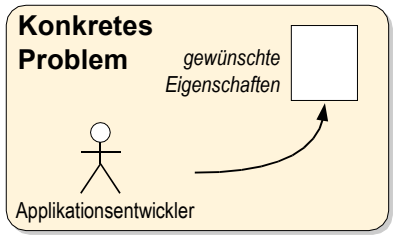
Beispiel (Wetterstation)

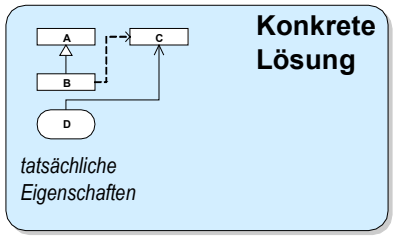


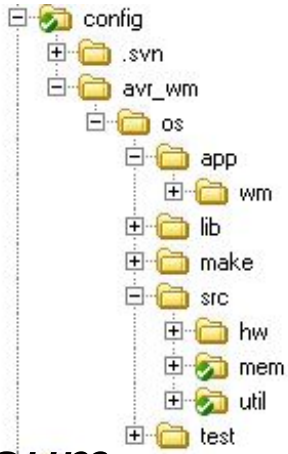
Konkretes Problem, konkrete Lösung

Merkmalauswahl









Generierter Quellcode-Baum

Fazit: Stand der Kunst Software-PL

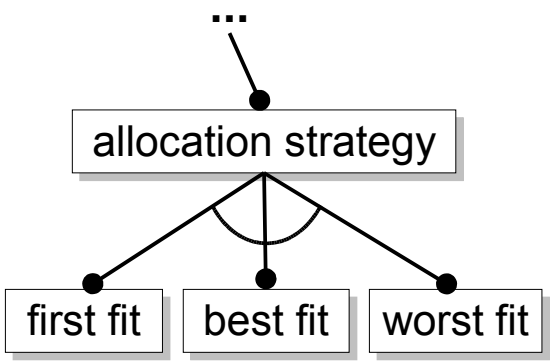
- **Insgesamt: durchaus erfolgreich**
  - Betriebssysteme, Middleware-Plattformen, Datenbanken
  - Zahlreiche Anwendungen: Mobiltelefone, Steuergeräte, ...
- **Es bleiben viele Herausforderungen**
  - **(Automatisches) Finden** der optimalen Variante
  - Konfigurierung **nicht-funktionaler Eigenschaften**
  - **Komposition** von Produktlinien

Herausforderung: Finden der optimalen Variante

- Gegeben
  - **Anwendung** (z.B. Steuergerät)
  - Feingranular konfigurierbare **Produktlinie** (z.B. PURE: **~250** konfigurierbare Merkmale)
- Gesucht
  - **Optimale Variante** (=Konfiguration)
  - Entsprechend den Anforderungen der Anwendung

Merkmale der PL oft begrifflich „zu weit entfernt“ von den Anforderungen der Anwendung

Unterschiede nur für „Insider“ erkennbar



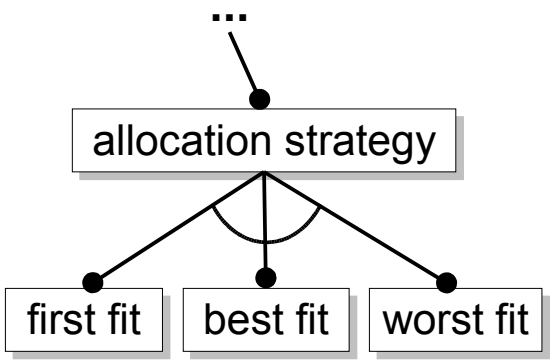
Herausforderung: Finden der optimalen Variante

- Gegeben
  - **Anwendung** (z.B. Steuergerät)
  - Feingranular konfigurierbare **Produktlinie** (z.B.
- Gesu
  - **Optir**
  - Entsp

Ziel: Automatische Merkmalauswahl

Merkmale der PL oft begrifflich „zu weit entfernt“ von den Anforderungen der Anwendung

Unterschiede nur für „Insider“ erkennbar



Herausforderung: nicht-funktionale Eigenschaften

Als **nicht-funktional** werden jene **Eigenschaften** eines Softwaresystems bezeichnet, die nicht den eigentlichen Funktionsumfang betreffen, jedoch beim **Betrieb der Software beobachtbar sind.**

- Performanz
- Ressourcenverbrauch
- Skalierbarkeit
- Vorhersagbarkeit
- Latenz
- ...

Nicht-funktionale Eigenschaften sind...

- **erfolgsbestimmend**
  - TollCollect 1 *funktionierte* prima...
  - dominieren oft funktionale Eigenschaften
- **emergent**
  - nicht sichtbar auf der Ebene einzelner Komponenten
  - ergeben sich „plötzlich“ aus der Gesamtkomposition
  - ***nicht trennbar*** von der Implementierung funktionaler Belange
  - ***Effekte des Lösungsraum*** ohne Entsprechung im Problemraum
- **nicht direkt konfigurierbar**

Nicht-funktionale Eigenschaften sind...

- **erfolgsbestimmend**
  - TollCollect 1 *funktionierte* prima...
  - do
- **em**
  - ni
  - er
  - ***ni***
  - ***Ei***
- **nic**

Ansatz: *Indirekte Beinflussung*

- über Merkmale mit **bekannt positiven** Auswirkungen auf nicht-funktionale Eigenschaften
- über **konfigurierbare Architektureigenschaften**

Rückabbildung in den Problemraum

- Metriken und Heuristiken erforderlich

Herausforderung: Komposition von Produktlinien

Existierende Produktlinien sind Insellösungen

- eng abgesteckte Domäne
- implementierungsnahe Merkmale
- eigene Modellierungssprachen und -konzepte
- eigene Konfigurierungswerkzeuge

Herausforderung: Komposition von Produktlinien

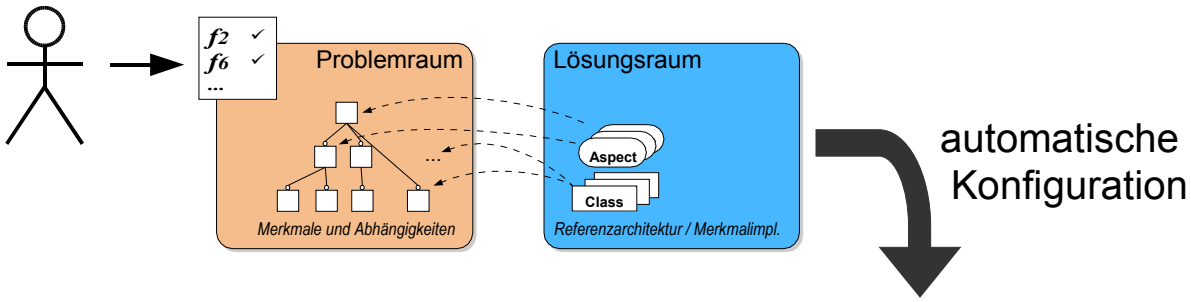
Existierende Produktlinien sind Insellösungen

- eng abgesteckte Domäne
- imple
- eigen
- eigene Konfigurierungswerkzeuge

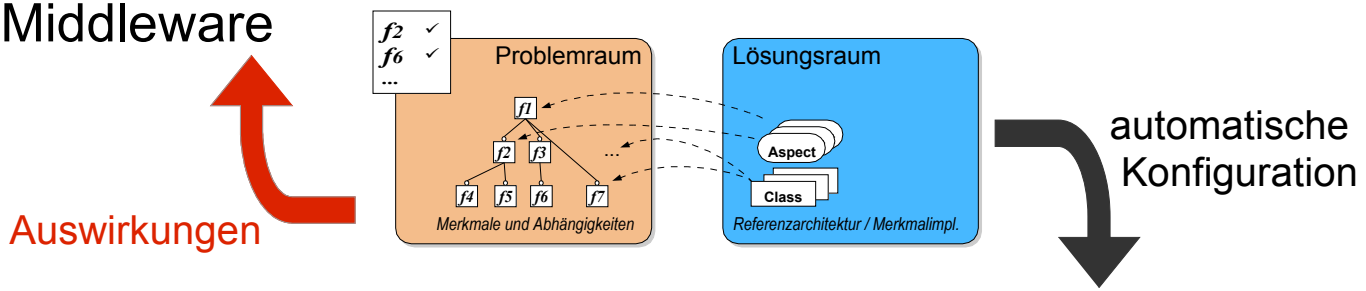
Ziel: Kombinierbare Produktlinien

Szenario: Komposition von Produktlinien

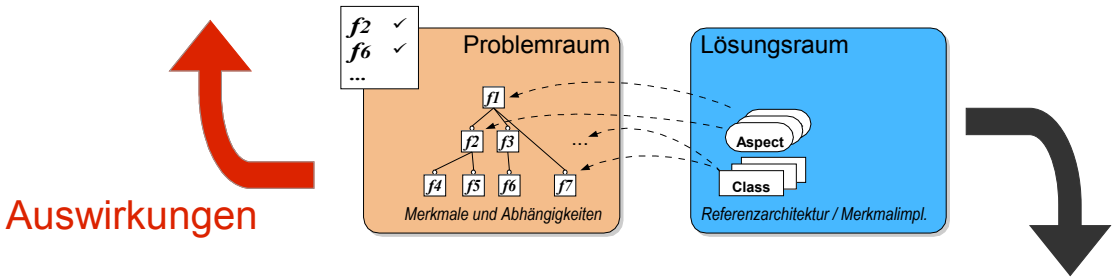
Schicht 3: Anwendungen



Schicht 2: Middleware



Schicht 1: Betriebssystem



Schicht 0: Hardware

Zusammenfassung: Software-PL

- Organisierte Variabilität und Wiederverwendung
  - Explizite Modellierung der Variabilität im **Problemraum**
  - Wiederverwendung durch gemeinsamen **Lösungsraum**
- Anwendung im Kleinen: Erfolgreich
- Anwendung im Großen: Viele offene Fragen
  - Variantenvielfalt
  - Beherrschbarkeit nicht-funktionaler Belange
  - Integration und Komposition unterschiedlicher PL
  - ...

Methodisch steht die PL-Entwicklung erst am Anfang!

Literatur

[1] G. Böckle, P. Knauber, K. Pohl, K. Schmid (Hrsg.). *Software-Produktlinien*. dpunkt.verlag, 2004. ISBN 3-89864-257-7.

[2] K. Czarnecki und U.W. Eisenecker. *Generative Programming – Methods, Tools, and Applications*. Addison-Wesley, 2000. ISBN 0-201-30977-7.

[3] pure-systems GmbH. *Variantenmanagement mit pure-variants*. Technical White Paper, http://www.pure-systems.com/.