

Betriebssystemtechnik

Softwaretechnik: Programmfamilie

Wolfgang Schröder-Preikschat

Lehrstuhl Informatik 4

6. Juni 2011

Gliederung

- 1 Einleitung
- 2 Betriebssystemfamilie
 - Grundlagen
 - Fallbeispiel
 - Entwurfsprinzipien
- 3 Softwareproduktlinie
 - Grundlagen
 - Phasen und Aktivitäten
- 4 Merkmalmodellierung
 - Grundlagen
 - Variantenformulierung
 - Fallbeispiel
- 5 Zusammenfassung

Motiv: Adaptierbarkeit von Systemsoftware [6]

Klagen über Softwaresysteme [7] — nach wie vor von hoher Aktualität...

We were behind schedule and wanted to deliver an early release with only a proper subset of intended capabilities, but found that that subset would not work until everything worked.

We wanted to add simple capability, but to do so would have meant rewriting all or most of the current code.

We wanted to simplify and speed up the system by removing unneeded capability, but to take advantage of this simplification we should have had to rewrite major sections of the code.

Our SYSGEN was intended to allow us to tailor a system to our customers' needs but it was not flexible enough to suit us.

Systemsoftware im Wandel von Anwendungsfällen

Konsequenzen aus den in [6, 7] genannten Szenarien erwarten Methoden, Konzepte und Techniken zum...

- 1 Bereitstellen von Vorversionen
- 2 Hinzufügen von (einfachen) Funktionen
- 3 Entfernen von unnötigen Funktionen
- 4 Maßschneidern der Systemsoftware

Punkte 1. – 3. zielen ab auf eine *hierarchische Struktur* des Systems vor allem in Form einer **Programmhierarchie**

- inkrementeller Entwurf mehrschichtig organisierter Softwaresysteme

Punkt 4. deutet hin auf **Werkzeuge**, die den Entwicklungsprozess von Systemsoftware unterstützen

- zur **Adaptierung** vorgefertigter Softwarebestandteile des Systems

Adaptierbarkeit $\neq$ Adaptivität

adaptiv ist ein System, wenn es sich selbständig an veränderte Bedingungen anpassen kann $\leadsto$ Selbstorganisation

- Fähigkeiten zur Reflektion/Introspektion *zur Laufzeit*
 - Programme kennen ihre eigene *Struktur* und können diese vorausahnend oder bei Bedarf *verändern*
 - ggf. Sprachunterstützung für *reflektive Programmierung*
- baut auf Funktionen für **dynamisches Laden/Binden**

adaptierbar ist ein System, wenn es durch externe Eingriffe an veränderte Bedingungen angepaßt werden kann

- Maßnahmen zur Konfektionierung *vor/zur Laufzeit*
 - dynamisch** $\models$ Laufzeit
 - statisch** $\models$ Kompilier-, Assembler-, Binde-, Ladezeit
- erf. Werkzeuge/Dienstprogramme zur **Konfigurierung**

Beachte: *adaptierbar zu sein bedeutet, adaptiv sein zu können...*

Systemfähigkeit zur Adaptivität setzt Adaptierbarkeit voraus

Adaptierbarkeit $\leadsto$ Adaptivität

Fähigkeiten des Systems zur Adaptivität setzt auf Programmstrukturen, die bei Adaptierbarkeit bereits vorliegen (müssten)

- man könnte sagen: Adaptivität „benutzt“ Adaptierbarkeit
 - adaptiv** zu sein erfordert auf Maschinenprogrammebene wohl abgegrenzte Programmbestandteile
 - physische Modulstruktur
 - adaptierbar** zu sein impliziert, oberhalb dieser Ebene für derartige Abgrenzungen gesorgt zu haben
 - logische und ggf. auch physische Modulstruktur
- manifestiert sich in Entwurfsentscheidungen der Systemsoftware

Grundlage: Funktionale Hierarchie [4, S. 267]

- In a functional hierarchy where functions may actually be macros, a sequence of function calls may result in a single machine instruction (or possibly none at all) when the system is compiled.
- It ist the system *design* which is hierarchical, not its implementation.

Gliederung

- 1 Einleitung
- 2 Betriebssystemfamilie
 - Grundlagen
 - Fallbeispiel
 - Entwurfsprinzipien
- 3 Softwareproduktlinie
 - Grundlagen
 - Phasen und Aktivitäten
- 4 Merkmalmodellierung
 - Grundlagen
 - Variantenformulierung
 - Fallbeispiel
- 5 Zusammenfassung

Systemsoftware als Programmfamilie

Spezialisierung

$\hookrightarrow$ [7]

Some users may require only a subset of the services of features that other users need. These "less demanding" users may demand that they are not be forced to pay for the resources consumed by the unneeded features.

- auch Universalsysteme sind spezialisiert: für den Vielzweck...

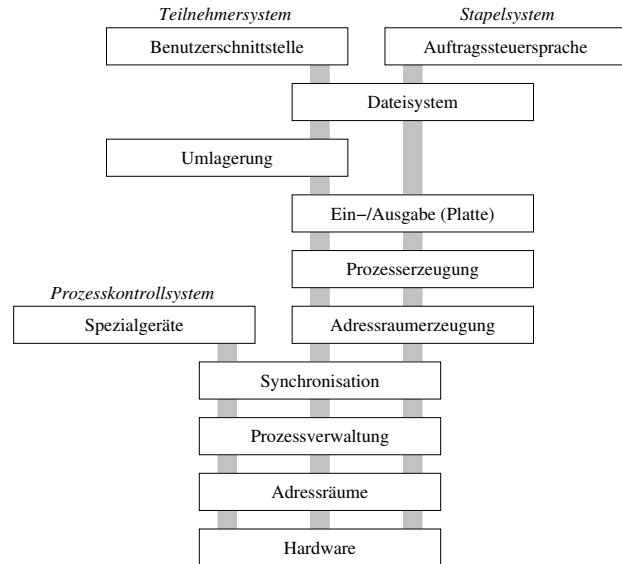
Wiederverwendung

$\hookrightarrow$ [7]

We consider a set of programs to be a program family if they have so much in common that it pays to study their common aspects before looking at the aspects that differentiate them.

We want to exploit the commonalities, share code, and reduce maintenance costs.

FAMOS [4]



FAMOS: Merkmale

Gemeinsamkeiten

total $\Rightarrow$ minimale Basis

- Adressräume, Prozessverwaltung, Synchronisation

partiell $\Rightarrow$ minimale Systemerweiterungen

- Adressraum- und Prozesserzeugung
- Ein-/Ausgabe (Platte), Dateisystem

Unterschiede $\Rightarrow$ Rechnerbetriebsart

- Benutzerschnittstelle, Auftragssteuersprache, Spezialgeräte

Variabilitäten $\Rightarrow$ Spezialisierung

- Dateisystem fähig/unfähig zur Umlagerung von Programmen

Prinzip: Zurückstellung von Entwurfsentscheidungen

$\hookrightarrow$ [4]

- *decisions which restrict the family are postponed as far as possible*

Strikt schrittweises Vorgehen

minimale Basis $\models$ minimale Teilmenge von Systemfunktionen

- umfasst zum Aufbau spezialisierter Systeme nützliche Funktionen
- besteht aus einer „kleinen Anzahl“ generalisierter Funktionen
- fokussiert viel mehr auf Mechanismen, weniger auf Strategien

minimale Systemerweiterungen

- entstehen schrittweise und aufbauend (engl. *bottom-up*)
 - sind dabei allerdings verfeinernd (engl. *top-down*) ausgelegt
- spezialisieren/individualisieren wiederverwendbare Bausteine
 - durch **anwendungsfallsspezifische funktionale Anreicherungen**
- kapseln strategische, anwendungsbezogene Entwurfsentscheidungen

Prinzip: Inkrementeller Systementwurf

$\hookrightarrow$ [4]

- iterativ, stufenweise, in kleinen Schritten: **Mut zur Vielschichtigkeit**
- von unten (Generalisierung) nach oben (Spezialisierung)

Funktionale Anreicherung

Erweiterung einer vorhandenen Grundmenge von Systemfunktionen um zusätzliche Funktionen, strikt anwendungsfallspezifisch

wiederverwenden (engl. *reuse*)

- auf bestehende Implementierungen aufsetzen

umarbeiten (engl. *rewrite*)

- in bestehende Implementierungen eingreifen

ersetzen (engl. *replace*)

- mit bestehenden Implementierungen austauschen

Beachte: Kombinationen in der Vorgehensweise sind gang und gäbe

- Vererbung von Implementierungen/Schnittstellen (OOP)
- Einweben von Aspekten (AOP)

Funktionale Anreicherung: Programmhierarchie

Methode der kleinen Schritte: resultiert in eine nicht selten sehr stark ausgeprägte **Hierarchie abstrakter Maschinen** [7]

Rather than write programs that perform the transformation from input to output data, we design software machine extensions that will be useful in writing many such programs.

- dabei ist eine Offenlegung der Gesamtstruktur nicht zwingend
 - zuweilen kann zu tiefer Einblick auch hier noch abschreckend wirken
- Grad der Notwendigkeit dazu hängt ab von der Interessensgruppe
 - transparent** für den Laien
 - will anwenden, muss das Innenleben nicht verstehen
 - intransparent** für den Experten
 - will/muss verstehen, wie das System aufgebaut ist
- die Methode unterstützt Experten — und behindert Laien nicht...

Programmfamilie $\models$ Softwarefamilie bzw. -produktlinie

Produktlinie (Software) $\mapsto$ mehrere, individuelle Ausprägungen eines Softwareprodukts, erstellt auf Basis einer gemeinsamen Plattform [9]

- wesentliches Bestimmungsmerkmal ist dabei ihre **Variabilität**
 - technisch unterlegt durch ausgewiesene **Variationspunkte**, an denen Entwurfs-/Implementierungsentscheidungen offen bleiben
 - um Produkte abzuleiten, werden die dazu benötigten **Varianten gebunden**, genauer: konfiguriert, ersetzt oder weggelassen
- geht meist einher mit Einschränkung auf eine bestimmte **Domäne**
 - um **Diversität** der Produkte handhabbar(er) zu gestalten

Eine Programmfamilie ist eine Softwareproduktlinie, in der die Programme die individuellen Produkte sind oder sein können.

Gliederung

- 1 Einleitung
- 2 Betriebssystemfamilie
 - Grundlagen
 - Fallbeispiel
 - Entwurfsprinzipien
- 3 Softwareproduktlinie
 - Grundlagen
 - Phasen und Aktivitäten
- 4 Merkmalmodellierung
 - Grundlagen
 - Variantenformulierung
 - Fallbeispiel
- 5 Zusammenfassung

Domäne (frz. *domaine*; Herrschaft, Herrschaftsbereich)

Wissensbereich („an area of knowledge“ [3])

- abgegrenzt, definiert Konzepte, gibt eine Terminologie vor
- liefert Kenntnisse über Prozesse zum Bau konkreter Systeme
- definiert sich durch alle, die ein Interesse an der Domäne haben
 - Management, Ein-/Verkauf, Marketing
 - Forschung, Entwicklung, Verwaltung, Fachverbände
 - Hersteller, Vertragspartner, Investoren, Kunden, Endnutzer
 - ...

Beispiele

- Hochleistungsrechnen (engl. *high-performance computing*, HPC)
 - Numerik, Graphik; Parallelverarbeitung (SIMD, MIMD)
- eingebettete Systeme
 - Motorsteuerungen, Wetterstationen, Spielgeräte, Sensornetze
- Informationssysteme, Transaktionssysteme, Datenzentren, ...

Softwarefamilie $\longleftrightarrow$ Organisierte Wiederverwendung

Domänenentwicklung (engl. *domain engineering*)

Entwicklung der gemeinsamen und variablen Artefakte als Bestandteile der Plattform

Applikationsentwicklung (engl. *application engineering*)

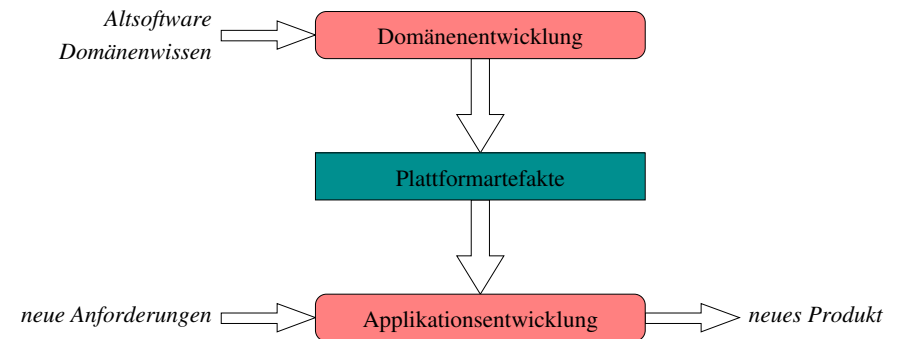
Ableitung einzelner Produkte der Produktlinie

- diese werden so weit wie möglich aus den Plattformartefakten zusammengefügt
 - konfiguriert
- so dass produktspezifische Entwicklung nur noch in gerigem Maße notwendig ist

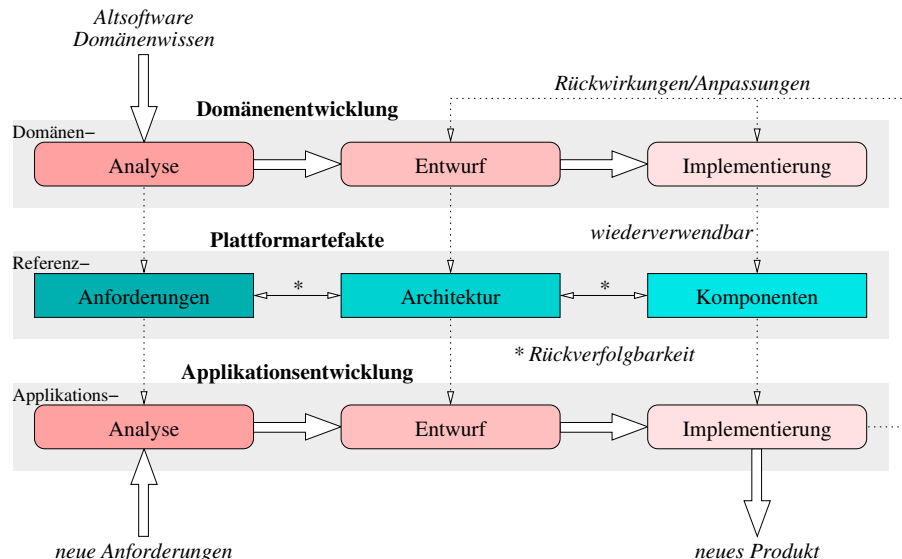
Softwareproduktlinie (engl. *software product line*) $\hookrightarrow$ [8, 1]

Im Vordergrund steht die aktive Gestaltung einer gemeinsamen Plattform für aktuelle und künftige Produkte.

Organisierte Wiederverwendung: Prozess



Organisierte Wiederverwendung: Prozess (PRAISE [5])



Produktlinienentwicklung

Domänenentwicklung

Analyse bestimmt den Zweck, dem die Familie dienen soll

Entwurf entscheidet über die in der Plattform benötigten Komponenten

Implementierung erstellen und erwerben von Komponenten und einer (die Produktfertigung) tragenden Infrastruktur

Applikationsentwicklung

Analyse bestimmt, was das für ein Produkt sein soll

Entwurf wählt die zur Herstellung des Produktes geeigneten Komponenten aus

Implementierung verbindet die Komponenten mittels einer Infrastruktur und ggf. weiteren, produktspezifischen Code

Beachte: Wichtige Problemkreise in diesem Kontext

Rückverfolgbarkeit (engl. *traceability*) $\mapsto$ Konfigurations-/Versionsverw.

Rückwirkung (engl. *feedback*) $\mapsto$ lenkbarer Prozess

Produktlinienentwicklung (Forts.)

Domänenentwicklung

Problemraum $\models$ Variabilität

- funktionale, nichtfunktionale Eigenschaften
- Modell von Merkmalen

Lösungsraum $\models$ Implementierung

- verfügbare Komponenten
 - Software und Hardware
- Auswahlregeln

Applikationsentwicklung

Problemraum $\models$ Auswahl

- Beschreibung des jew. zu lösenden Problemfalls
- spezifisches Problem

Lösungsraum $\models$ Variante

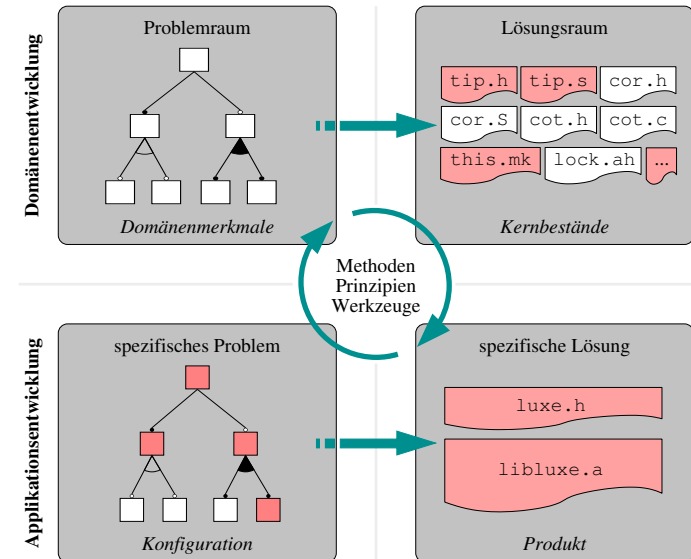
- integrierte Komponenten
 - Plattform und ggf. Appl.
- spezifische Lösung

Beachte: Änderungswünsche (engl. *change requests*) $\leadsto$ Risiko

Je Einzelfall ist zu entscheiden, ob nachträgliche Änderungen (a) in die Plattform einfließen oder (b) in der Applikation umgesetzt werden sollen

- Softwarefreigabedauer (a) vs. Wiederverwendungspotential (b)

Produktlinienentwicklung: Prozess



Gliederung

- 1 Einleitung
- 2 Betriebssystemfamilie
 - Grundlagen
 - Fallbeispiel
 - Entwurfsprinzipien
- 3 Softwareproduktlinie
 - Grundlagen
 - Phasen und Aktivitäten
- 4 Merkmalmodellierung
 - Grundlagen
 - Variantenformulierung
 - Fallbeispiel
- 5 Zusammenfassung

Merkmale der Systemfamilie bestimmen: Domänenanalyse

Domänenabgrenzung (engl. *domain scoping*)

- Auswahl und Aufbereitung von Wissen über die Domäne
- Festlegung von Einschränkungen zur...
 - Begrenzung der durch Diversität verursachten Komplexität
 - Maximierung der Wiederverwendbarkeit von Komponenten

Domänenmodellierung (engl. *domain modeling*)

- Auswertung des angesammelten Wissens
- Aufstellung einer Taxonomie

Ergebnis der Domänenanalyse ist das **Domänenmodell**, d.h.:

Domänendefinition legt Bereich und Sparte der Systemfamilie fest

Domänenlexikon gibt das verwendete Vokabular und Begrifflichkeit vor

Konzeptmodelle zeigen Ziele, daraus abgeleitete Umsetzungsstrategien

Merkmalmodelle erfassen Anforderungen an die Systemfamilie

Domänenmodell $\Rightarrow$ Programmfamilie

Domänenmodell $\supset$ Merkmalmodell $\hookrightarrow$ [3]

- A domain model is an explicit representation of the **common** and the **variable** properties of the system in a domain, the semantics of the properties and domain concepts, and the dependencies between the variable properties.
- Feature modeling is the activity of modeling the common and the variable properties of concepts and their **interdependencies** and organizing them into a coherent model referred to as feature model.

Merkmalmodell $\models$ Programmfamilie $\hookrightarrow$ [7]

- We want to exploit the **commonalities**, share code, and reduce maintenance costs.

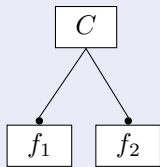
Modellierungsprozess

Kontinuierlicher, iterativer Prozess zum Finden und Dokumentieren von Merkmalen

Aktivitäten, die in kleinen, schnellen Schritten wiederholt durchgeführt werden („micro-cycles“ [2]):

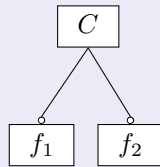
- 1 Gemeinsamkeiten von Entitäten notieren
 - diese benennen, ihnen gemeinsame Merkmale zuschreiben
- 2 Unterschiede zwischen Entitäten notieren
 - diese benennen, ihnen differenzierende Merkmale zuschreiben
- 3 Merkmale in **Merkmal diagrammen** organisieren
 - Kombinationen von, Interaktionen zwischen Merkmalen spezifizieren
- 4 Merkmalkombinationen und -interaktionen analysieren
 - Abhängigkeiten und Konflikte ermitteln
 - weitere, im Modell „versteckte“ Merkmale identifizieren
 - die ggf. erst durch eingehende Analyse erkennbar werden
- 5 Aufnahme zusätzlicher Informationen zu den Merkmalen
 - Beschreibung, Vor-/Nachteile, Prioritäten, Einschränkungen, ...

Merkmal diagramme: Elemente



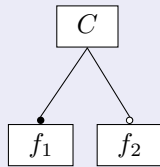
- Verpflichtungen
- $f_1 \cdot f_2$

zwingende Merkmale
 f_1 und f_2 müssen einbezogen werden, wenn C ausgewählt wird



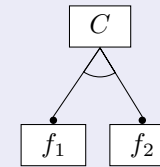
- Optionen
- $\emptyset, f_1, f_2, f_1 \cdot f_2$

wahlfreie Merkmale f_1 und f_2 können einbezogen werden, wenn C ausgewählt wird



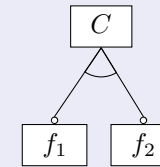
- Zusatzoption
- $f_1, f_1 \cdot f_2$

kombinierte Merkmale
 f_1 muss, f_2 kann einbezogen werden, wenn C ausgewählt wird



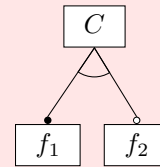
- Alternative
- f_1, f_2

alternative Merkmale
 f_1 oder f_2 müssen einbezogen werden, wenn C ausgewählt wird



- Alternativoption
- $\emptyset, f_1, f_2$

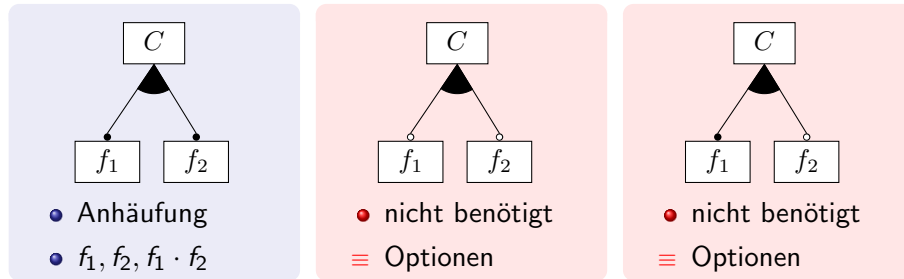
wahlfrei alternative Merkmale
 f_1 oder f_2 können einbezogen werden, wenn C ausgewählt wird



- nicht benötigt
- Alternativoption

Merkmal diagramme: Gruppen von Alternativen

Merkmal diagramme: Gruppen von Anhäufungen



kumulative Merkmale (auch: Oder-Merkmale)

weingstens eins von f_1
und f_2 muss einbezogen
werden, wenn C
ausgewählt wird

Kardinalitäten der Merkmalstypen

Merkmal	Varianten	Konfigurationen
zwingend	1	1
wahlfrei	2^n	2^{n-1}
kombiniert	$1 + 2^n$	$1 + 2^{n-1}$
alternativ	n	n
wahlfrei alternativ	$n + 1$	$n + 1$
kumulativ	$2^n - 1$	2^{n-1}

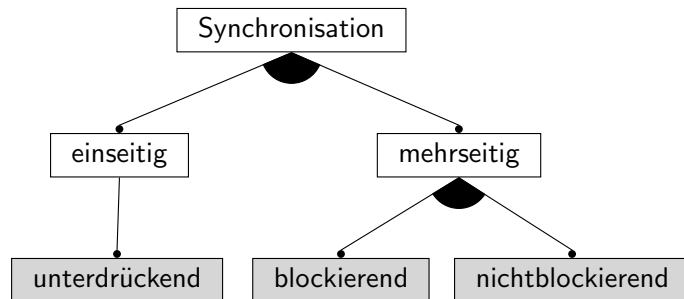
Varianten $\equiv$ Anzahl der nicht redundanten Kombinationen

Konfigurationen $\equiv$ Anzahl des Auftretens eines Merkmals f_i

Beachte: Kombinatorik und Komplexität

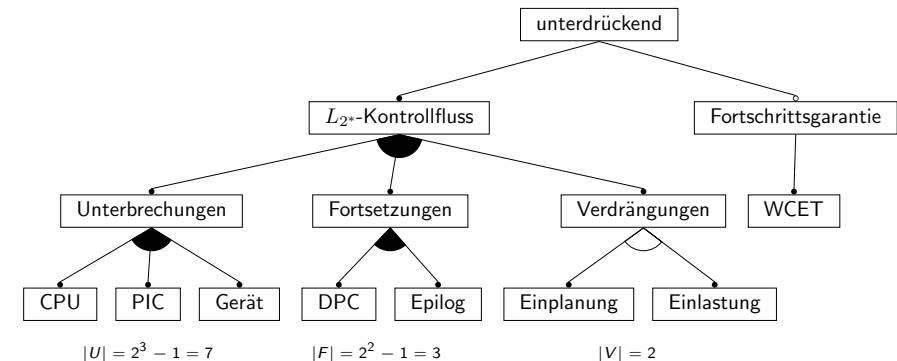
Die Zahl möglicher Anordnungen oder Auswahlen von unterscheidbaren Konfigurationen (ohne Beachtung der Reihenfolge von Merkmalen) hängt ab vom Aufbau des Merkmaldiagramms.

Fallstudie: Synchronisation



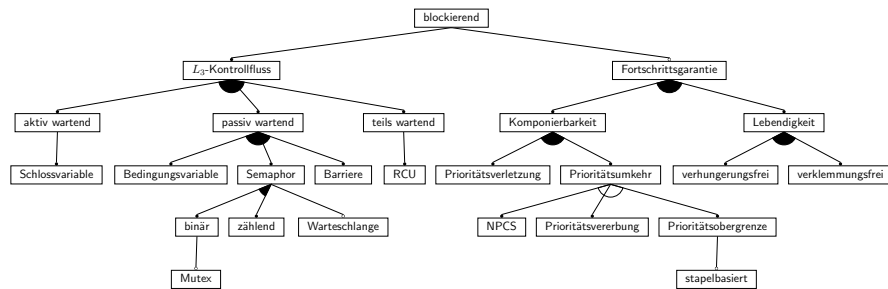
$$|S| \geq 2^3 - 1 = \text{weit mehr als 7 mögliche Konfigurationen}$$

Fallstudie: Synchronisation (Forts.)



$$\begin{aligned}
 |u| &= 2 \times |L2^*K| \\
 &= 2 \times (|U| + |F| + |V| + |U \cdot F| + |U \cdot V| + |F \cdot V| + |U \cdot F \cdot V|) \\
 &= 2 \times (7 + 3 + 2 + 21 + 14 + 6 + 42) \\
 &= 190 \text{ mögliche Konfigurationen}
 \end{aligned}$$

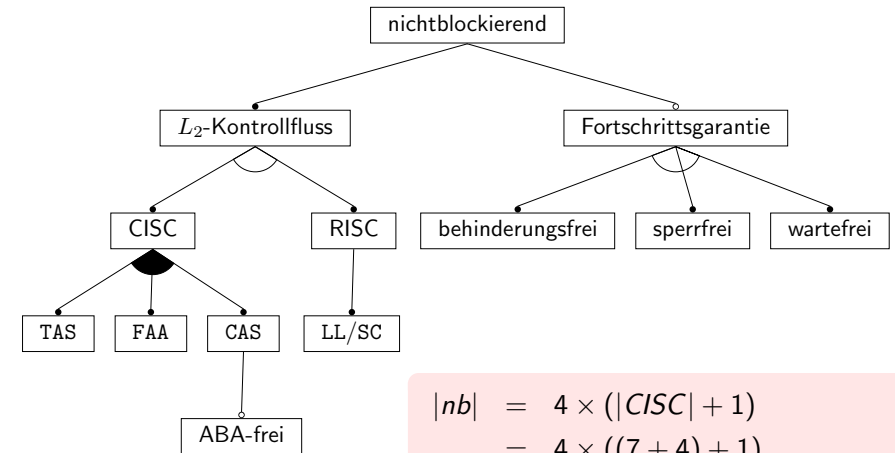
Fallstudie: Synchronisation (Forts.)



$$\begin{aligned}
 |Sem| &= 2 \times ((2 \times 3) - 1) = 10 \\
 |pw| &= 3 + 4 \times |Sem| = 43 \\
 |L_3K| &= 3 + 4 \times |pw| = 175 \\
 |Pu| &= 4 \\
 |K| &= 1 + 2 \times |Pu| = 9 \\
 |L| &= 3 \\
 |F| &= |K| + |L| + |K \cdot L| \\
 &= 9 + 3 + 27 = 39
 \end{aligned}$$

$$\begin{aligned}
 |b| &= 2 \times |L_3K| + |F| \\
 &= 2 \times 175 + 39 \\
 &= 389 \text{ mögliche Konfigurationen}
 \end{aligned}$$

Fallstudie: Synchronisation (Forts.)



$$\begin{aligned}
 |nb| &= 4 \times (|CISC| + 1) \\
 &= 4 \times ((7 + 4) + 1) \\
 &= 48 \text{ mögliche Konfigurationen}
 \end{aligned}$$

Gliederung

- 1 Einleitung
- 2 Betriebssystemfamilie
 - Grundlagen
 - Fallbeispiel
 - Entwurfsprinzipien
- 3 Softwareproduktlinie
 - Grundlagen
 - Phasen und Aktivitäten
- 4 Merkmalmodellierung
 - Grundlagen
 - Variantenformulierung
 - Fallbeispiel
- 5 Zusammenfassung

Resümee

Betriebssystemfamilie ↔ Spezialisierung vs. Wiederverwendung

- Fallbeispiel (FAMOS): Gemeinsamkeiten, Unterschiede
- inkrementeller Systementwurf, funktionale Anreicherung

Softwareproduktlinie ↔ organisierte Wiederverwendung

- Domäne, Domänen- und Applikationsentwicklung, Prozess
- Domänen-/Applikationsanalyse, -entwurf, -implementierung
- Problem- und Lösungsraum

Merkmalmodellierung ↔ Domänenanalyse

- Domänenabgrenzung und -modellierung
- (Merkmalmodell $\models$ Programmfamilie) $\subset$ Domänenmodell
 - Domänendefinition und -lexikon, Konzept- und Merkmalmodelle
- Modellierungsprozess, Merkmaldiagramme
- Fallstudie: Synchronisation $|S| \gg 627$ mögliche Konfigurationen

- [1] BÖCKLE, G. ; KNAUBER, P. ; POHL, K. ; SCHMID, K. :
Software-Produktlinien: Methoden, Einführung und Praxis.
Heidelberg : dpunkt.verlag GmbH, 2004. –
ISBN 3–80864–257–7
- [2] CZARNECKI, K. :
Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models.
Ilmenau, Germany, Technische Universität Ilmenau, Diss., 1998
- [3] CZARNECKI, K. ; EISENECKER, U. W.:
Generative Programming. Methods, Tools and Applications.
Addison-Wesley, 2000. –
ISBN 0–20–13097–77
- [4] HABERMANN, A. N. ; FLON, L. ; COOPRIDER, L. W.:
Modularization and Hierarchy in a Family of Operating Systems.
In: *Communications of the ACM* 19 (1976), Mai, Nr. 5, S. 266–272
- [5] LINDEN, F. van d.:
Software Product Families in Europe: The Esaps & Café Projects.
In: *IEEE Software* 19 (2002), Jul./Aug., Nr. 4, S. 41–40

- [6] PARNAS, D. L.:
On the Design and Development of Program Families.
In: *IEEE Transactions on Software Engineering* SE-2 (1976), März, Nr. 1, S. 1–9
- [7] PARNAS, D. L.:
Designing Software for Ease of Extension and Contraction.
In: *IEEE Transactions on Software Engineering* SE-5 (1979), Nr. 2, S. 128–138
- [8] WEISS, D. M. ; LAI, C. T. R.:
Software Product-Line Engineering: A Family-Based Software Development Process.
Addison-Wesley, 1999. –
ISBN 0–201–69438–7
- [9] WIKIPEDIA FOUNDATION INC.:
Wikipedia, Die freie Enzyklopädie.
San Francisco, CA, USA : <http://de.wikipedia.org>,