Considering Feature Interactions in Product Lines

Towards the Automatic Derivation of Dependencies Between Product Variants

Andreas Metzger, Stan Bühne, Kim Lauenroth, Klaus Pohl

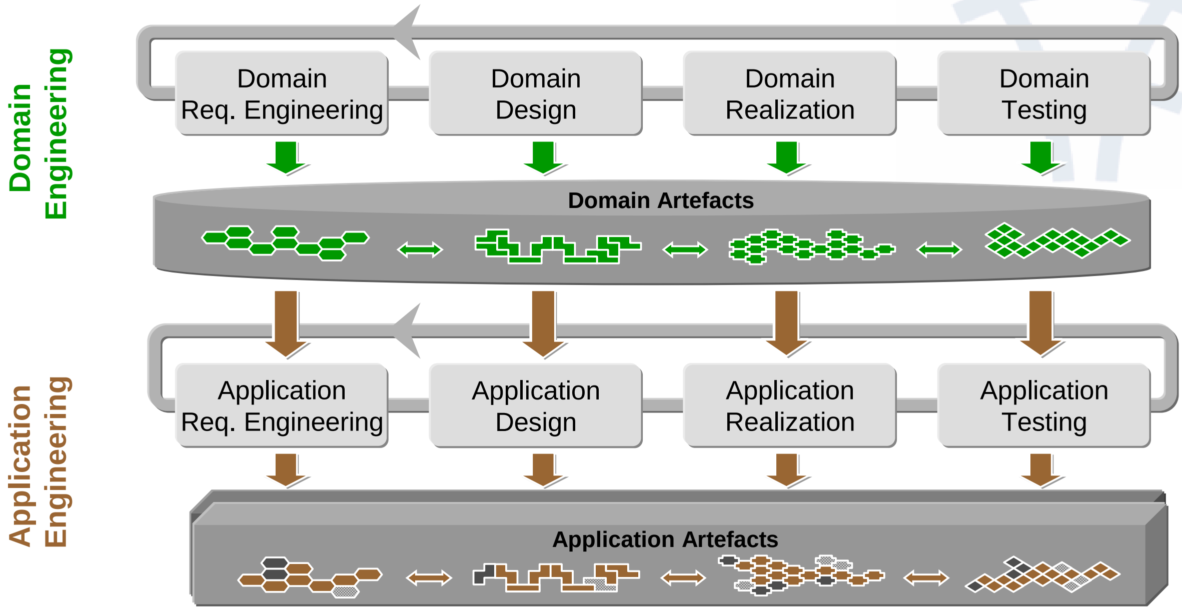
Software Systems Engineering
Institute for Computer Science and Business Information Systems
University of Duisburg-Essen, Germany

Outline

- **Introduction**
- **Variability in Software Product Lines**
- **Deriving Dependencies between Product Variants**
- **Conclusion**

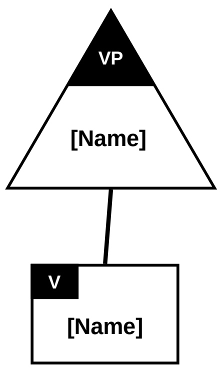
Introduction:
Software Product Line Engineering

- Development approach for customer specific software
  - **Systematic reuse**
    → Reduction of development cost and time; Increase in Quality
- **Two development processes**



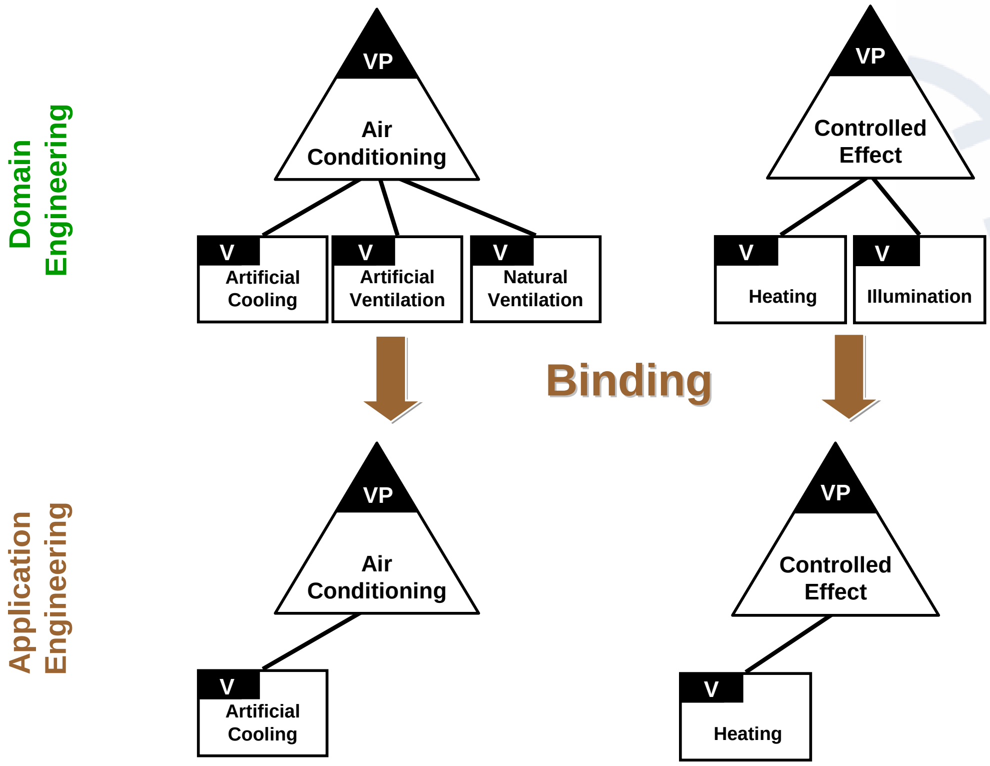
Introduction:
The Concept of Variability (1)

- **Variability**
  - "Ability of an artefact to be adaptable"
- Employing variability
  - **Domain engineering:**
    - **Modelling** of "generic" **domain artefacts**
  - **Application engineering:**
    - **Binding** the variability of the domain artefacts
- **Variation point (VP)**
  - Point at which an artefact can vary
- **Variant (V)**
  - Concrete instances or alternatives for variable parts
  - associated to one VP



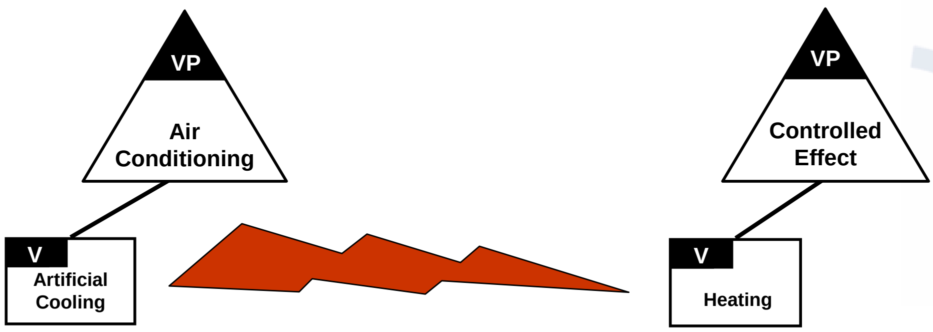
Introduction:
The Concept of Variability (2)

- **Example: “Small Building Control System Familiy”**



Introduction:
Dependencies between Variants (1)

- **Problem: Variants can influence each other**



→ **Consider dependencies** during application requirements engineering
  – Example: artificial cooling ***hinders*** heating

  → customer is **aware** that “artificial cooling” will impact “heating”
  → **choice of alternatives** is possible (e.g., natural ventilation, integrated HVAC, …)

- **Problem:** Dependencies have to be known **before** application requirements engineering
  – otherwise: frustration of customers

Introduction: Dependencies between Variants (2)

- **Solution: Identify dependencies in domain engineering**
  - All possible applications have to be considered

- **Problem:** Number of possible combinations of variants
  - Small example: 5 variants → (3+2+1) * (2+1) = **18 applications**
  - Large example: 14 variants → **639 applications**
  - In practical contexts: > 100 variants → **>> 1000 applications**

  → Manual identification of dependencies does not scale!

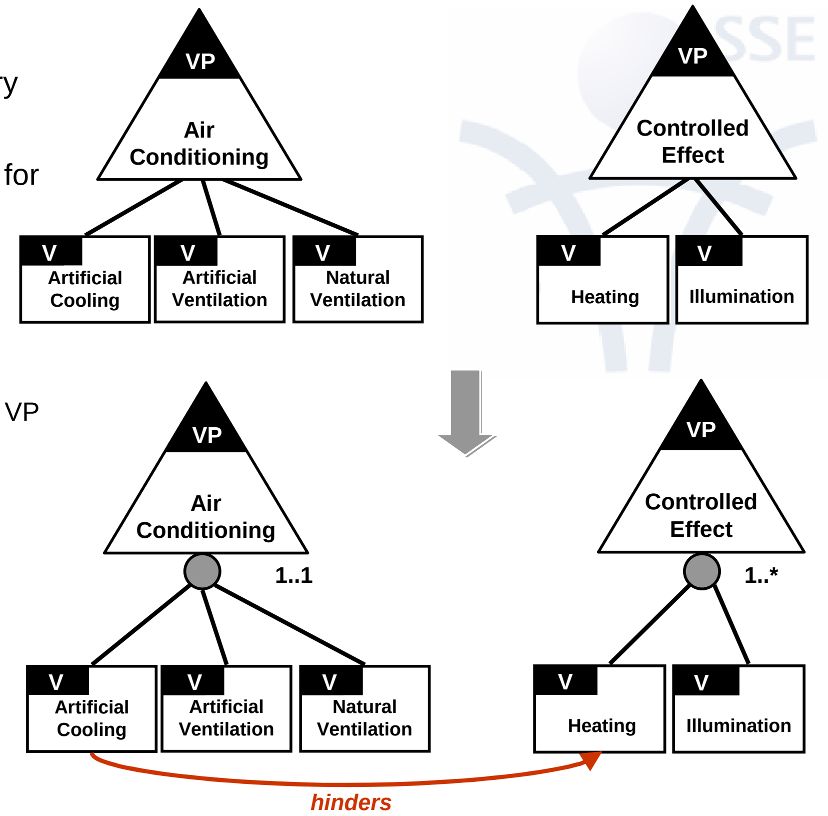
- **Our solution:** Semi-automatic approach based on feature interaction detection
  - **Feature interaction between V_j and V_k → dependency between V_j and V_k**

Outline

- **Introduction**
- **Variability in Software Product Lines**
- **Deriving Dependencies between Product Variants**
- **Conclusion**

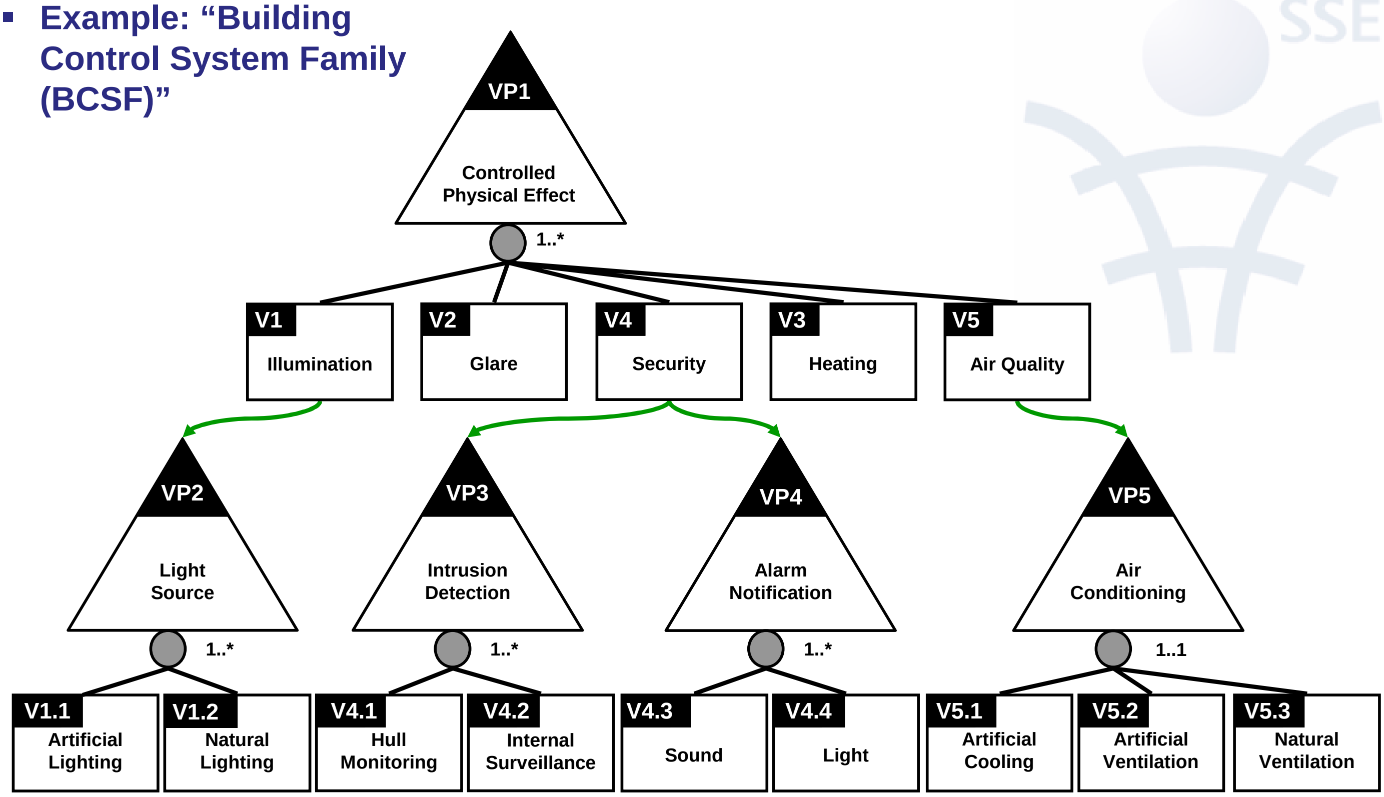
Variability in Software Product Lines: Variability Model (1)

- **Variation point (VP)**
  - Point at which the artefact can vary
- **Variant (V)**
  - Concrete instances or alternatives for variable parts
  - associated to one VP
- **VP-V Dependency**
  - **Variant Group (VG)**
    - Constrains selection of variants for VP
- **V-VP Dependency**
  - Allows hierarchical refinement
- **V-V Dependency**
  - Dependencies between variants
    - **hard:** excludes / requires
    - **subtle:** hints / hinders
- similarities with feature diagrams



Variability in Software Product Lines: Variability Model (2)

- **Example: "Building Control System Family (BCSF)"**



Outline

- Introduction
- Variability in Software Product Lines
- Deriving Dependencies between Product Variants
- Conclusion

Deriving Dependencies between Product Variants: Basic Approach

1. **Automatically** determine **feature interactions** for all possible applications

   1.1 Select **representatives** from all possible combinations

   → Tackling the problem of scale

   for each representative:

   1.2 Derive "application" by binding variants

   1.3 Detect feature interactions

   - using algorithm from single system development

2. **Manually** derive and model **dependencies**

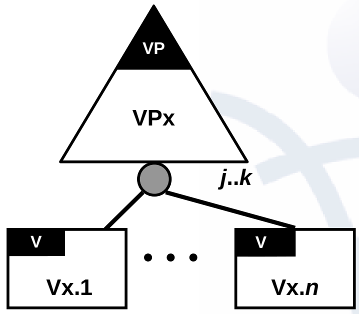
   2.1 Determine relevant feature interactions

   → Relevance cannot be derived from input models

   2.2 Model dependencies between variants

Deriving Dependencies between Product Variants: 1.1 Select Representatives (1)

- Number of possible **variant combinations** for one VG



$$K(j,k,n) = \sum_{i=j}^{k} \binom{n}{i}$$

  – BCSF Example: 639 combinations

- Assumption: **no *m*-way feature interactions**
  – *m*-way feature interaction :=
    feature interaction that does not occur between $1 < r < m$ features but occurs among m features
  – interaction between features $F_1, \ldots, F_m$
    $\Rightarrow$ interactions between all features $F_{i1}, \ldots, F_{ir} \in \{F_1, \ldots, F_m\}$ with $1 < r < m$

➔ **Select** combination with **largest possible number of variants**:
  – "worst case": false positives → more interactions to check manually

$$K(j,k,n) = \binom{n}{k}$$

  – BCSF Example: 639 → 3

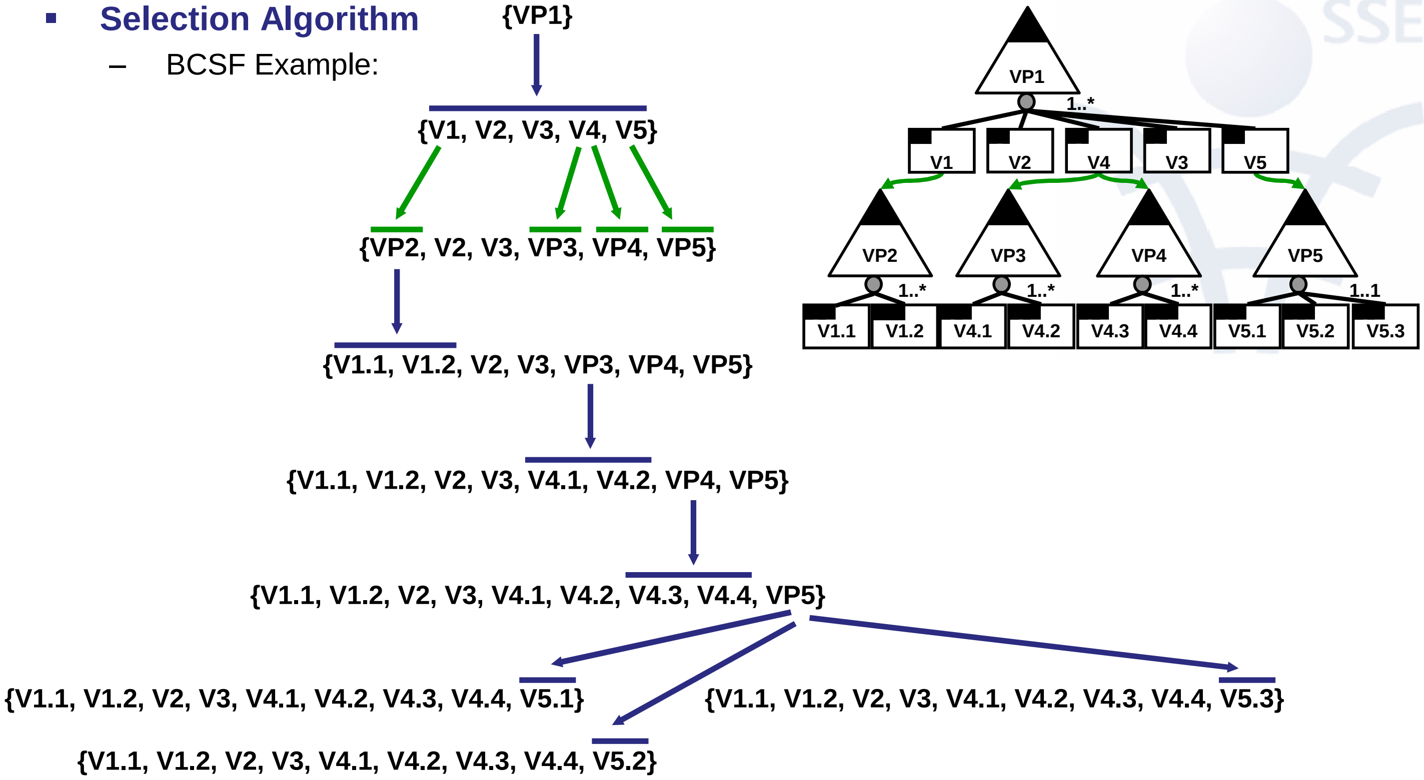
Deriving Dependencies between Product Variants: 1.1 Select Representatives (2)

- **Selection Algorithm ("Outline")**

1. Start with root VP
2. Resolve VP by adding maximum number of variants
   - considering alternatives if $k < n$
3. Resolve hierarchical variants (replace by VP)
4. Repeat at 2. until no more VPs are contained

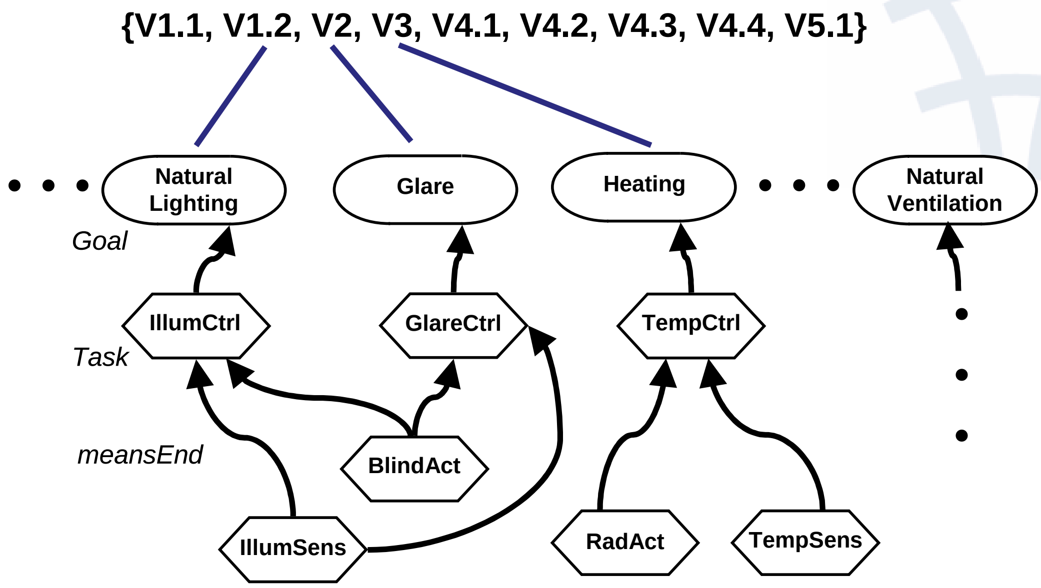
Deriving Dependencies between Product Variants: 1.1 Select Representatives (3)

- **Selection Algorithm**
  - BCSF Example:



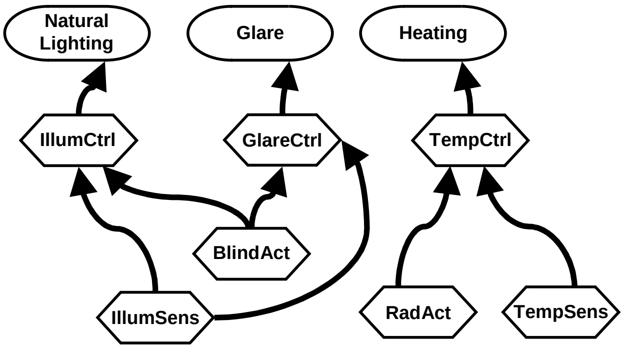
Deriving Dependencies between Product Variants: 1.2 Derive "Application" (2)

- **Bind variants in GRL (Goal Oriented Requirements Language) models**
  - Each variant is assigned to one goal
  - BCSF Example (Excerpt):



Deriving Dependencies between Product Variants: 1.3 Detect Feature Interactions

- **Detect points of interaction (cf. [Metzger et al. 2003], [Metzger 2004])**
  - Point of interaction := task that
    - contributes to the realization of more than one goal
    - has more than one direct parent
    - does not realize goals only
  - Recursive algorithm on GRL metamodel instance
  - BCSF Example:



  - Interaction between **NaturalLighting**, **Glare** ➔ Interaction between **V1.2**, **V2**
  - **Extension for embedded systems:**
    - Additionally consider environment

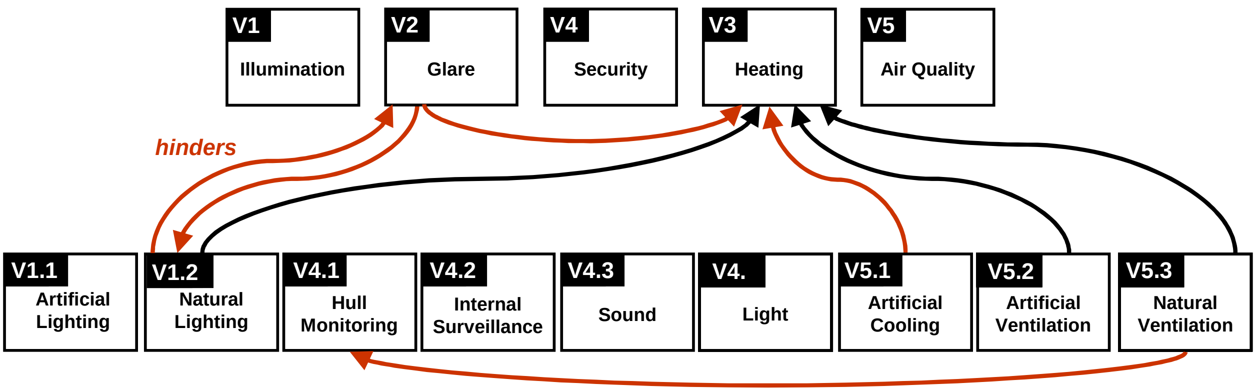
Deriving Dependencies between Product Variants: 2. Manually Derive and Model Dependencies

2.1 Determine relevant feature interactions

- BCSF example:
  1. **{V1.2, V2}** @ **Task**
  2. **{V1.1, V1.2, V2, V3, V4.2}** @ **Task**
  3. **{V1.1, V1.2, V2}** @ **Task**
  4. **{V1.2, V2, V3, V5.2}** @ **environment**
  5. **{V1.2, V2, V3, V5.1 }** @ **environment**
  6. **{V4.1, V5.3}** @ **environment**
  7. **{V1.2, V2, V3, V5.3}** @ **environment**

2.2 Model dependencies between variants

- BCSF example:



Conclusion

- **Semi-automatic approach**
  - Reduction of complexity/effort
    - Automatic selection of representatives: **639** → **3**
    - Automatic detection of interactions: **37** goals/tasks, **39** means-end-links → **7** interactions

- **Model-based approach**
  - Inputs/outputs are models
  - Model-based implementation of detection tool
    - Core: 175 manually implemented Java LOCs

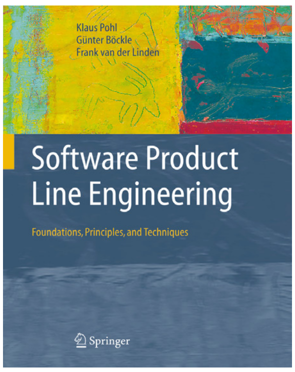
- **Generality of approach**
  - Variability model → feature diagrams could also be used
  - Requirements model (GRL) → other detection approaches are applicable

"Positive" use of Feature Interactions for Product Line Engineering
→ Selection of alternatives to **"avoid" undesired interactions**

Further Information …

… Text Book:

Pohl, Böckle, van der Linden:
Software Product Line Engineering – Foundations, Principles and Techniques.
Springer, 2005



… Contact:

Dr. Andreas Metzger

Software Systems Engineering
Schützenbahn 70
University of Duisburg-Essen
45117 Essen, Germany

metzger@sse.uni-essen.de
www.sse.uni-essen.de

… Questions?