

DSM-TP 2016

Modeling Variability

Andrzej Wąsowski

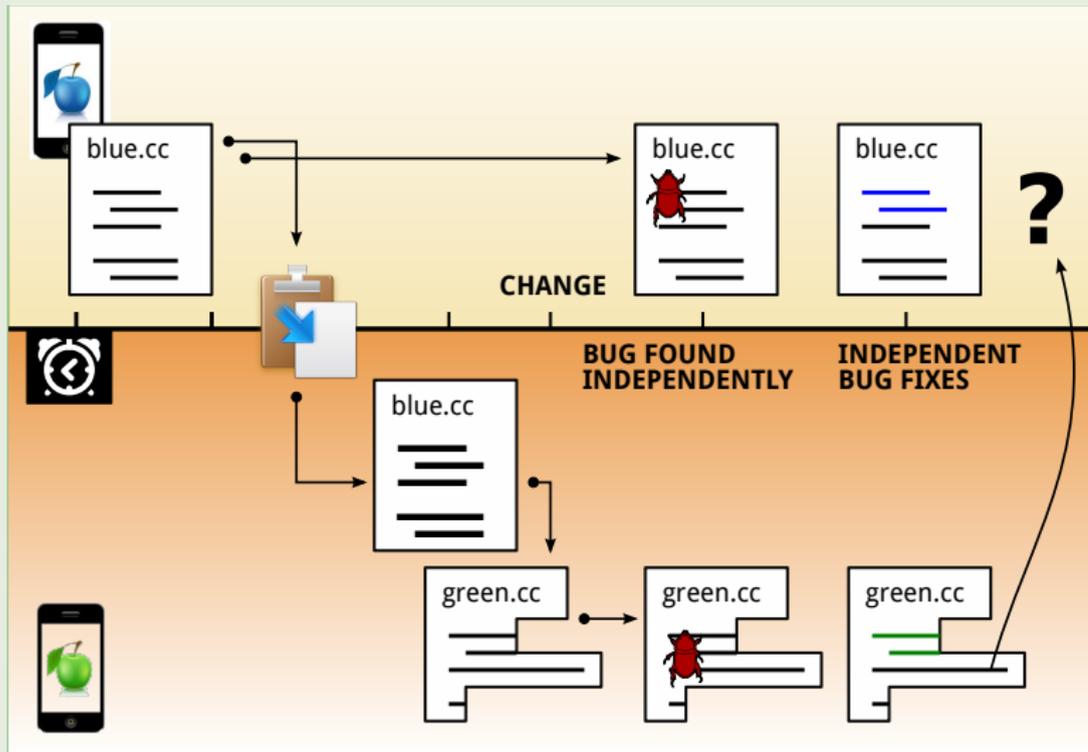
VARIETE, DFF SAPERE AUDE

VARIES

 COST
COOPERATION IN SCIENCE AND TECHNOLOGY

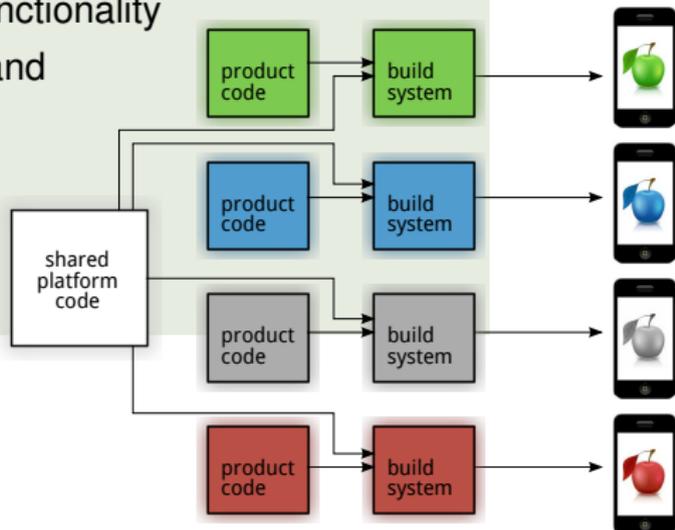
 MPM4CPS

Drowning in Clone-And-Own



Opportunistic Reuse Does Not Work

- ▶ Common scenario:
 - version the code, reuse when **opportunity** appears
- ▶ If the file to be reused needs change, **copy** it
 - *You clone-and-own* it
- ▶ **Benefit** from **quickly** available functionality
- ▶ But have to test, debug, change and evolve the file **yourself**
- ▶ **Product specific** code grows
- ▶ Platform code **diminishes and degrades**



SUCCESSFUL REUSE IS

**PROACTIVE
PLANNED
MANAGED**

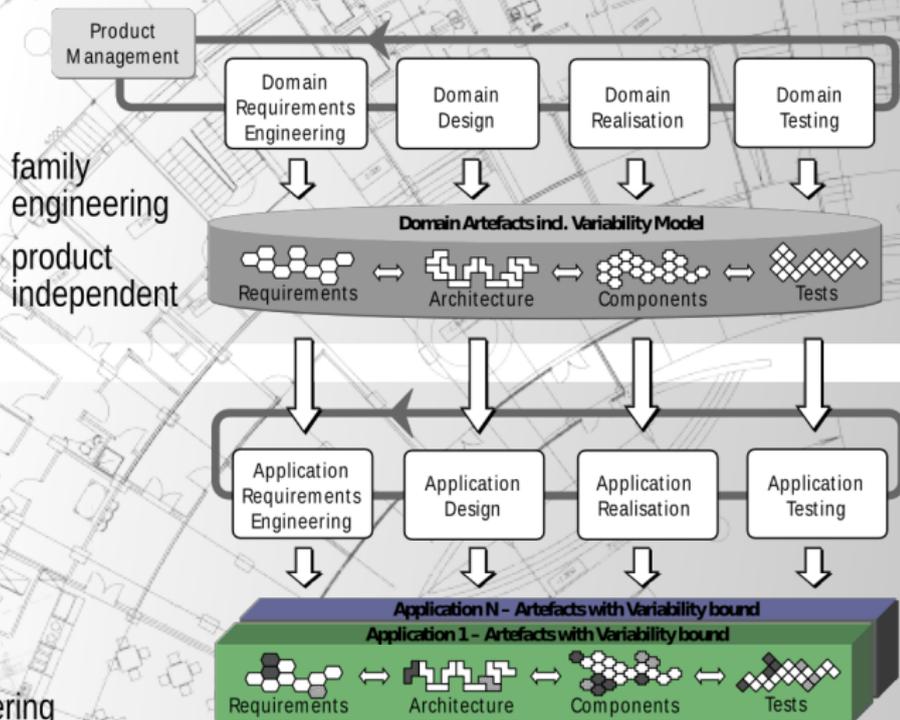
- 
- ▶ SPL Method, **Architecture**
 - ▶ Variability **Implementation Spectrum**
 - ▶ Variability Abstraction: **Feature Modeling**
 - ▶ Variability Modeling in **Practice**
 - ▶ Variability **Realization**



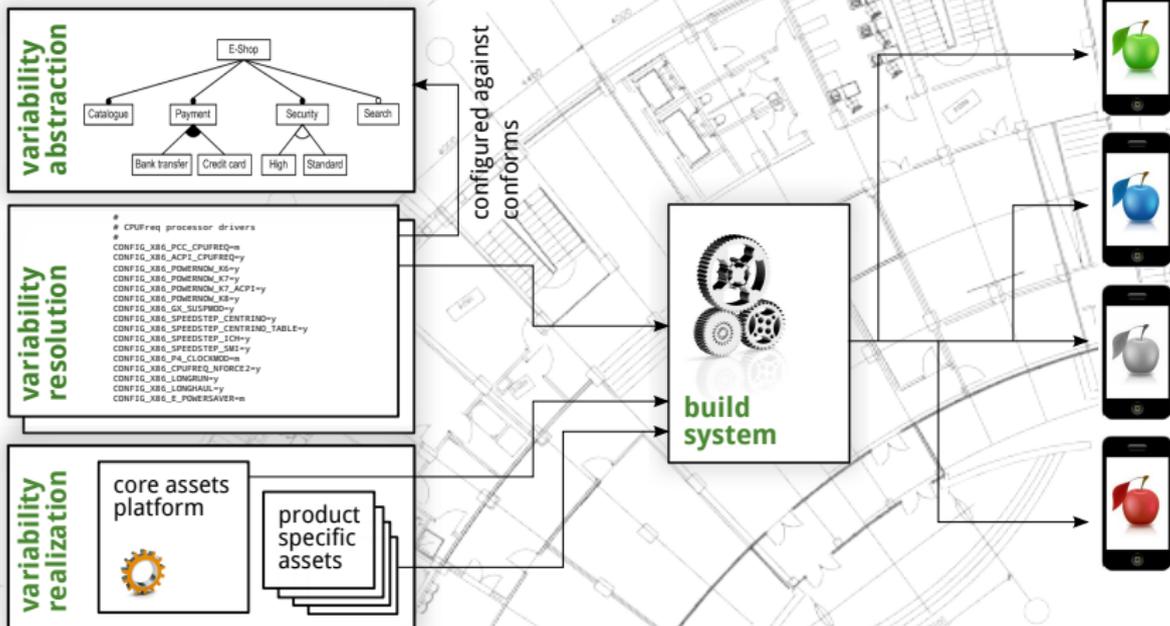
AGENDA

Domain vs Application Processes

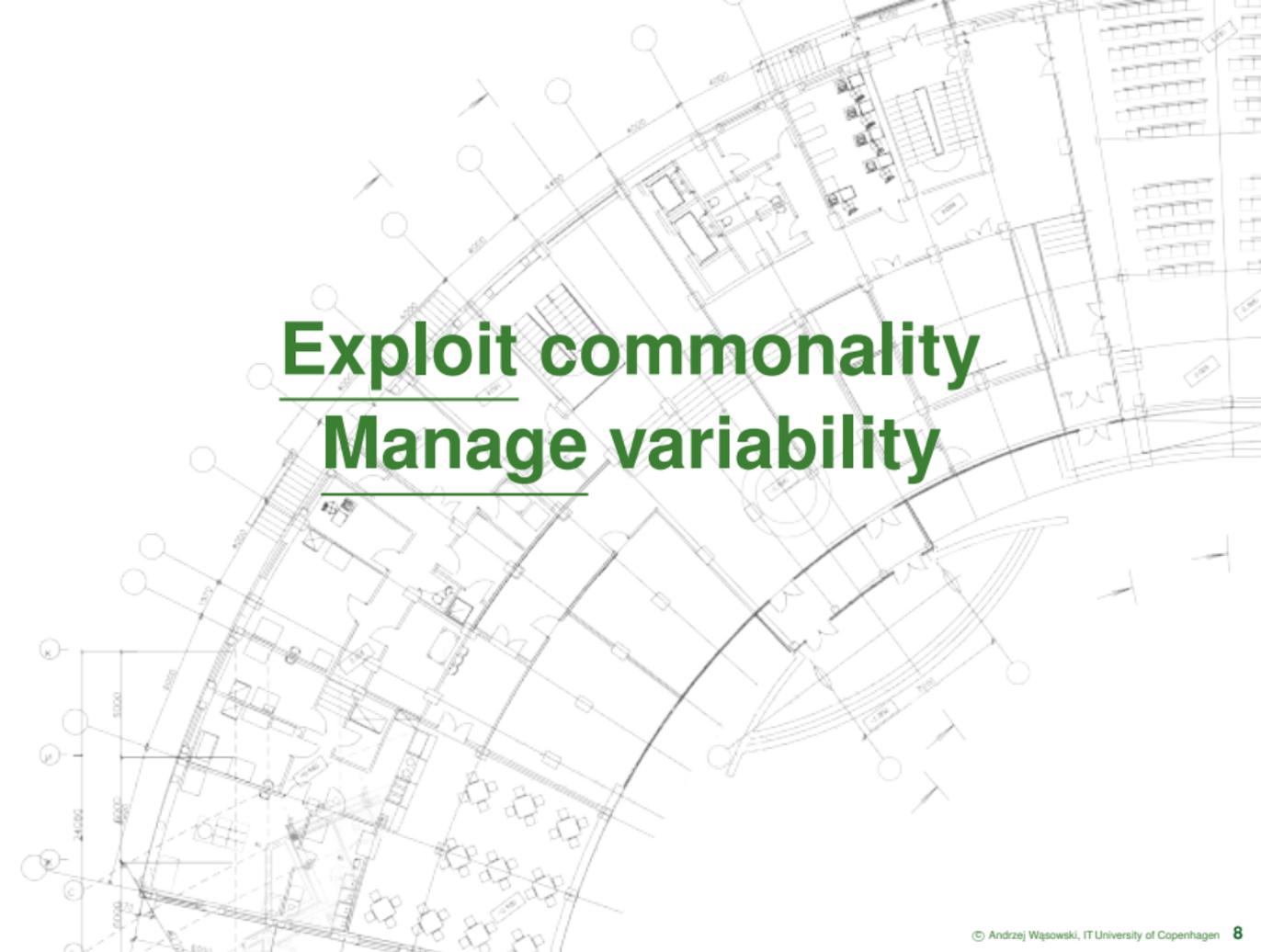
Pohl et. al. Software Product Line Engineering



A Simple Product Line Architecture



- ▶ Less product specific = more reuse: development/tests/debugging/build
- ▶ Model of *commonality* and *variability*.
- ▶ Scope under control. Explicit feature life cycle

The background is a detailed architectural floor plan of a building, showing various rooms, corridors, and structural elements. The plan is rendered in black lines on a white background. The text is overlaid on the central part of the plan.

Exploit commonality
Manage variability

- 
- ▶ SPL Method, **Architecture**
 - ▶ Variability **Implementation Spectrum**
 - ▶ Variability Abstraction: **Feature Modeling**
 - ▶ Variability Modeling in **Practice**
 - ▶ Variability **Realization**



AGENDA

Spectrum of Variability Architectures

Stay as close to the left as possible

only product specific code (no reuse)

frameworks + framework completion code

domain specific languages + code generation

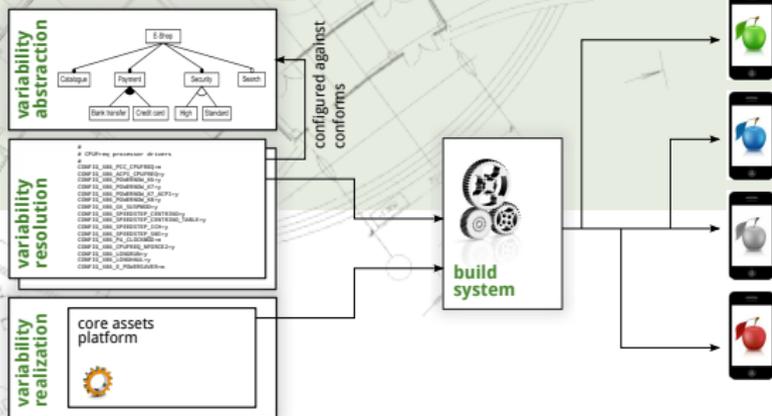
feature models + product specific code

feature models + build system

property & configuration files + build system

Implementation Technologies

- ▶ Variability abstraction: FMs, DSLs, or **none**
- ▶ Variability resolution:
 - XML property file
 - FM configuration
 - Domain specific model (DSM)
- ▶ Variability realization:
 - general purpose code
 - w/ variability techniques
 - code generators
 - model transformers
 - parts may use DSLs
 - etc.



- 
- ▶ SPL Method, **Architecture**
 - ▶ Variability **Implementation Spectrum**
 - ▶ Variability Abstraction: **Feature Modeling**
 - ▶ Variability Modeling in **Practice**
 - ▶ Variability **Realization**

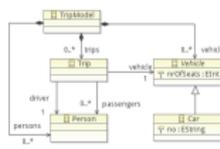
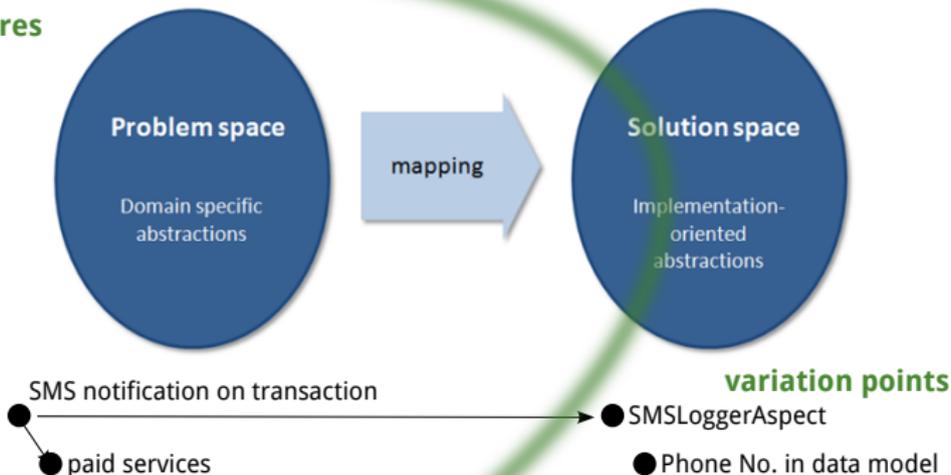


AGENDA

Problem Space

Solution Space

features



CVL Architecture for Dummies

The degree of coupling can be controlled by moving the mapping

Variability Abstraction

| Feature/Decision Models

Variability Realization

| Feature Mapping

Base (model)

| Source Code

Feature Modeling (I)

feature:

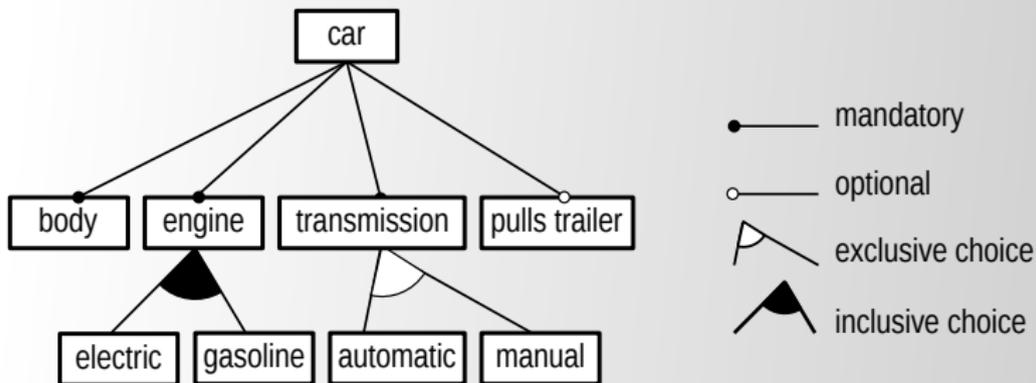
a single variability increment in the problem domain (decision)

variation point:

a single variability increment in the solution space

Feature Modeling (II)

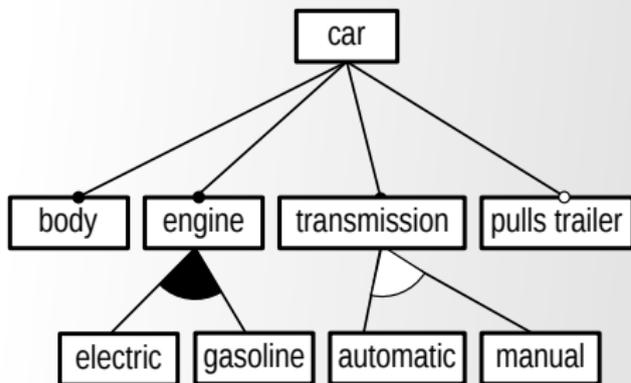
Example from Czarnecki'02



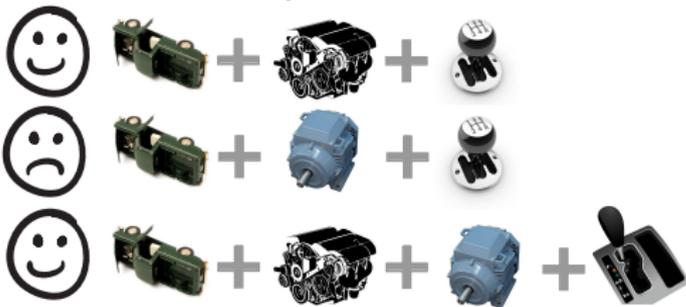
- ▶ **Hierarchy** constraints, for example:
 - manual requires transmission (each child node requires its parent node)
- ▶ **Groups** constraints: engine is electric or gas driven or both
- ▶ Not all constraints in hierarchy & groups, cross-tree constraints in text:
 - electric *requires* automatic
- ▶ Attributes are added like to classes (eg. engine volume)

Feature Modeling (III)

Configuration

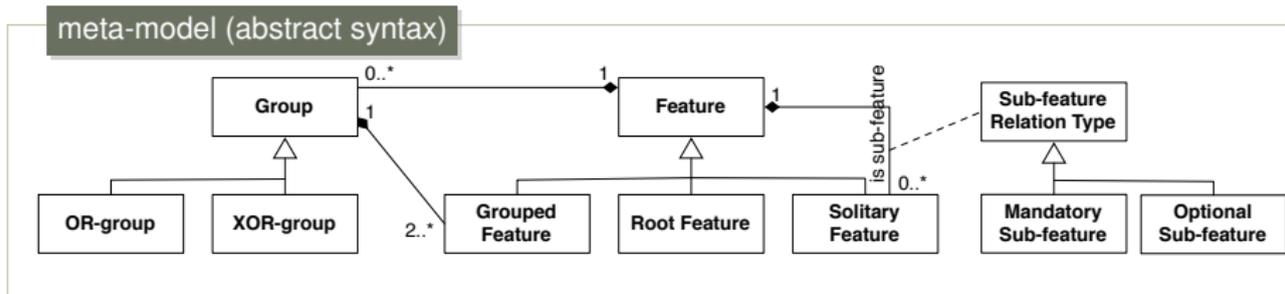


electric requires automatic



Feature Models (IV)

An example meta-model from Janota'08



- ▶ Note a single generic kind of relations: **subfeature**
- ▶ No distinction between **kind-of** (inheritance) and **part-of** (containment), like class modeling does
- ▶ A characteristic feature of **configuration and constraint languages** (as opposed to structural modeling languages)
- ▶ Clafer (as a structural modeling language) supports the distinction, but so do other feature modeling languages

Feature Modeling and FODA

Feature Oriented Design and Analysis by Kang et al. 1990

- ▶ **FODA succeeds for its simplicity**
- ▶ Probably best intro in Czarnecki's *Generative Programming* (Chpt. 4)
- ▶ **3950+** citations, **never formally published**

Google

feature oriented analysis and design

About 1,890,000 results (0.16 sec)

Scholar

Articles

Legal documents

[Feature-oriented domain analysis \(FODA\) feasibility study](#)
KC Kang, SG Cohen, JA Hess, WE Novak... - 1990 - DTIC Document

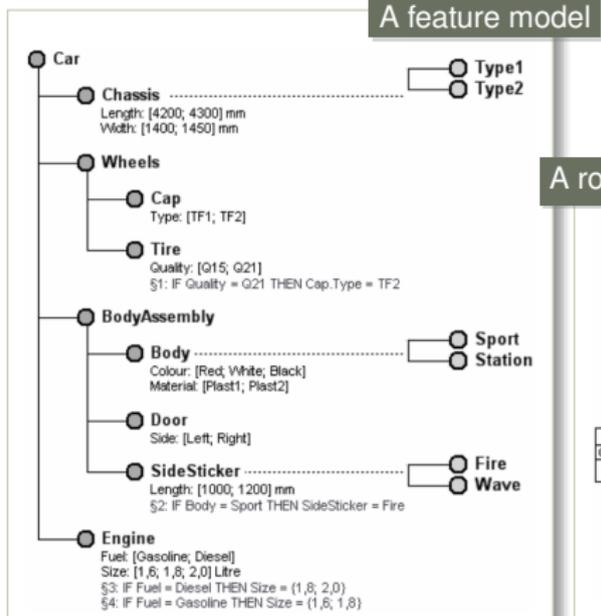
... 2.2.1. The Genesis System 13 2.2.2. MCC Work 15 2.2.2.1. The DESIRE Design Recovery Tool
15 2.2.2.2. Domain Analysis Method 1f 2.2.3. CTA Work 16 2.2.4. SPS Work 18 3. Overview
of the Feature Oriented Domain Analysis (FODA) Method 21 3.1. Method Concepts 21 ...
Cited by 2451 - Related articles - All 24 versions

[Product line engineering](#)
Software, IEEE, 2002 - ieexplore.ieee.org

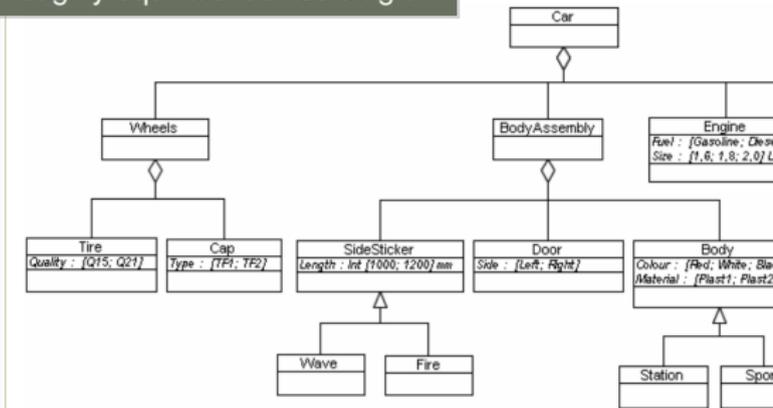
Feature-Oriented Reuse Method (FORM) not only to support
development but also to in-corporate a

Feature Modeling vs Class Modeling

A feature model in *Product Variant Master Notation* (Hvam)



A roughly equivalent class diagram



More on this: Bąk. Czarnecki. Wąsowski. *Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled*. SLE 2010

Above models from: Haug. Degn. Poulsen. Hvam. *Creating a documentation system to support the development and maintenance of product configuration systems*

Applications of Feature Models

Design & Management



domain
modeling

product line scoping
product line mngmt



code generation
driving build system

driving
testing



Development & Test

How To Build Feature Models?

Two strategies, but only one good :)

top-down

- ▶ **Big-bang** adoption
- ▶ Perform **careful domain analysis**
- ▶ Document **concepts, abstractions and relations** between them in a FM

bottom-up

- ▶ Identify a **cloned component**
- ▶ Find the **patches** that describe differences
- ▶ Translate **diffs to variation points**
- ▶ Organize **variation points into features**, and a hierarchy
- ▶ Works well with **incremental adoption**
- ▶ See SPLC07 paper by Danfoss

- ▶ SPL Method, **Architecture**
- ▶ Variability **Implementation Spectrum**
- ▶ Variability Abstraction: **Feature Modeling**
- ▶ Variability Modeling in **Practice**
- ▶ Variability **Realization**



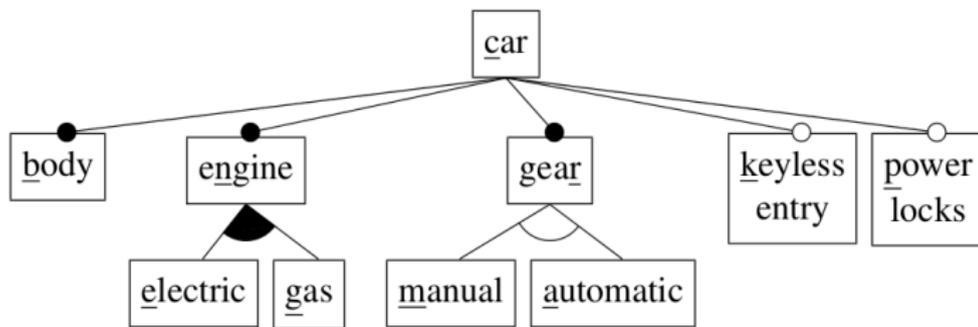
AGENDA

Variability Modeling is The Success Story of Modeling





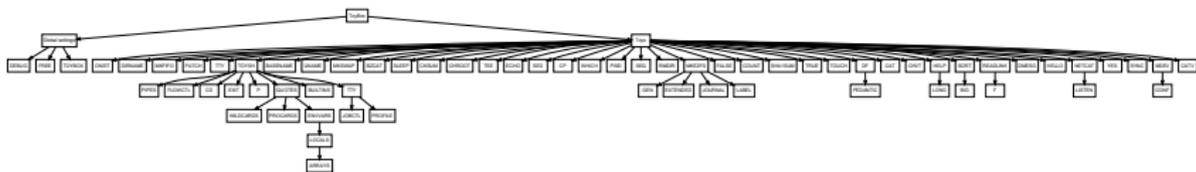
A Laboratory Feature Model



keyless_entry → *power_locks*

A Healthy Wild Feature Model Cub

ToyBox project, 71 features



**The Linux Kernel has 6-12K features, depending how you count!
But maximum depth is 8, most leaves are at 4!**

↓ this is the Linux kernel model fit to the slide width ↓

Is FODA special? Not really!

eCos configurator

The screenshot shows the eCos configurator interface. On the left is a tree view of configuration options. The 'Compress data' option is selected, and its sub-options are also checked. On the right is a table showing the current values for various properties and macros.

Item	Conflict	
CYGPKG_POSIX_CLOCKS	Unsatisfied	R
CYGHWR_IO_FLASH_DEVICE	Unsatisfied	R
CYGPKG_FILEIO_FNMATCH	Unsatisfied	R

Property	Value
URL	reffileio.html
Macro	CYGOPT_FS_JFFS2_COMPRESS
File	/home/shshe/Dev/ecos/ecos-3.0/
Enabled	True
Flavor	bool
Define	JFFS2_COMPRESSION
DefaultValue	1

Compression and decompression are entirely handled by the file system and are fully transparent to applications. However, selecting this option increases the amount of RAM required and slows down read and write operations considerably if you have a slow CPU.

- ▶ **Linux kernel** and **eCos operating system** use similar configurator, controlled by **textual variability models**
- ▶ **Trees become unwieldy** very fast
- ▶ Many tools used **linearized trees**, like above
- ▶ Nice **trees are good for PowerPoint**, whiteboard and brainstorming

It is easy to design a textual syntax

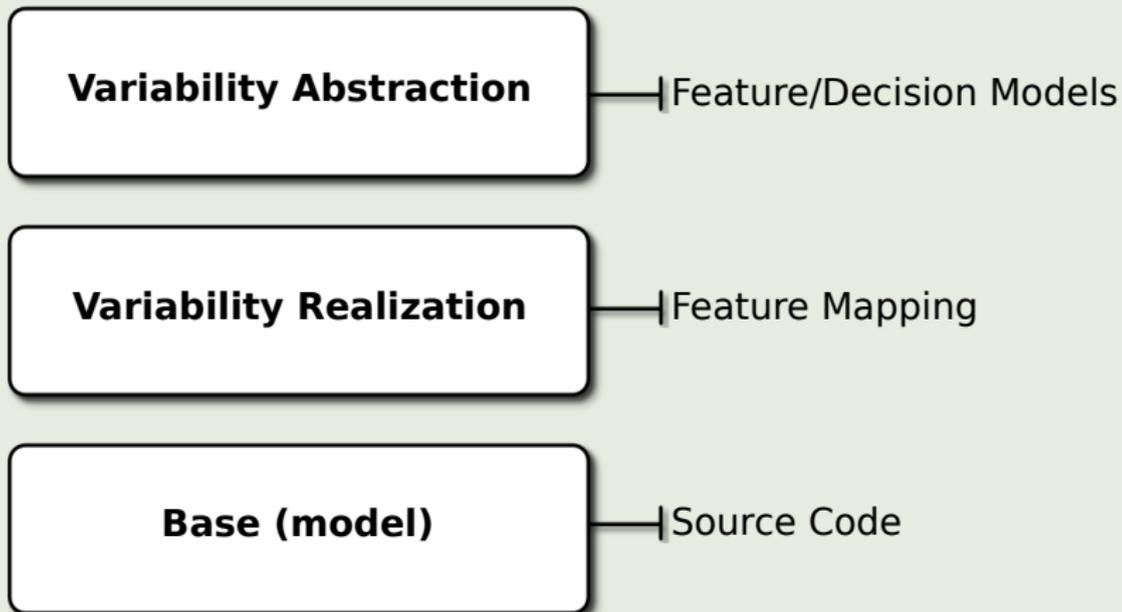
Kconfig of Linux kernel, designed by non-experts, with no tools
CDL of eCos, designed by non-experts, with no tools

```
k-1 menuconfig MISC_FILESYSTEMS
k-2     bool "Miscellaneous filesystems"
k-3
k-4     if MISC_FILESYSTEMS
k-5
k-6         config JFFS2_FS
k-7             tristate "Journalling Flash File System" if MTD
k-8             select CRC32 if MTD
k-9
k-10
k-11
k-12
k-13         config JFFS2_FS_DEBUG
k-14             int "JFFS2 Debug level (0=quiet, 2=noisy)"
k-15             depends on JFFS2_FS
k-16             default 0
k-17             range 0 2
k-18             --- help ---
k-19                 Debug verbosity of ...
k-20
k-21
k-22         config JFFS2_FS_WRITEBUFFER
k-23             bool
k-24             depends on JFFS2_FS
k-25             default HAS_IOMEM
k-26
```

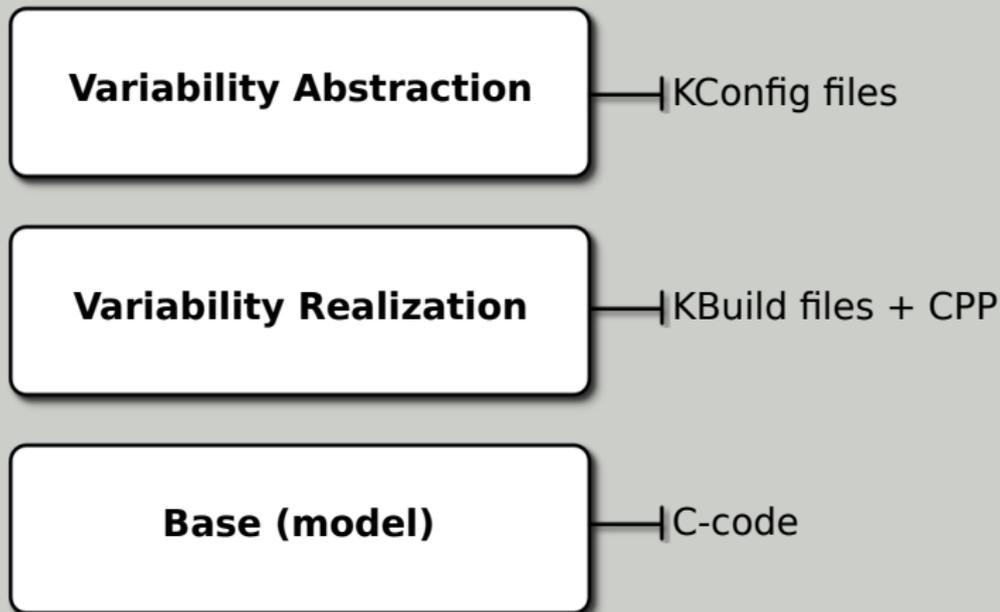
```
c-1 cdl_component MISC_FILESYSTEMS {
c-2     display "Miscellaneous filesystems"
c-3     flavor none
c-4     ↓
c-5
c-6     cdl_package CYGPKG_FS_JFFS2 {
c-7         display "Journalling Flash File System"
c-8         requires CYGPKG_CRC
c-9         implements CYGINT_IO_FILEIO
c-10        parent MISC_FILESYSTEMS
c-11        active_if MTD
c-12
c-13        cdl_option CYGOPT_FS_JFFS2_DEBUG {
c-14            display "Debug level"
c-15            flavor data
c-16            default_value 0
c-17            legal_values 0 to 2
c-18            define CONFIG_JFFS2_FS_DEBUG
c-19            description "Debug verbosity of..."
c-20        }
c-21
c-22        cdl_option CYGOPT_FS_JFFS2_NAND {
c-23            flavor bool
c-24            define CONFIG_JFFS2_FS_WRITEBUFFER
c-25            calculated HAS_IOMEM
c-26        }
```

CVL Architecture for Dummies

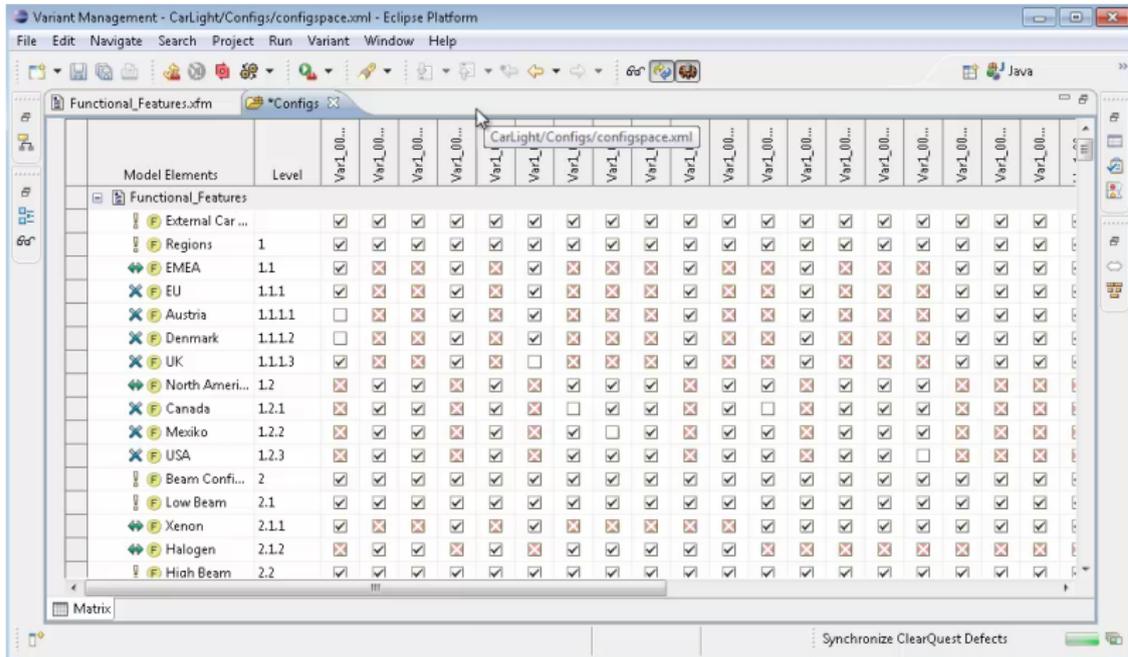
The degree of coupling can be controlled by moving the mapping



CVL Architecture for Linux Junkies



Grid View in Pure•Variants

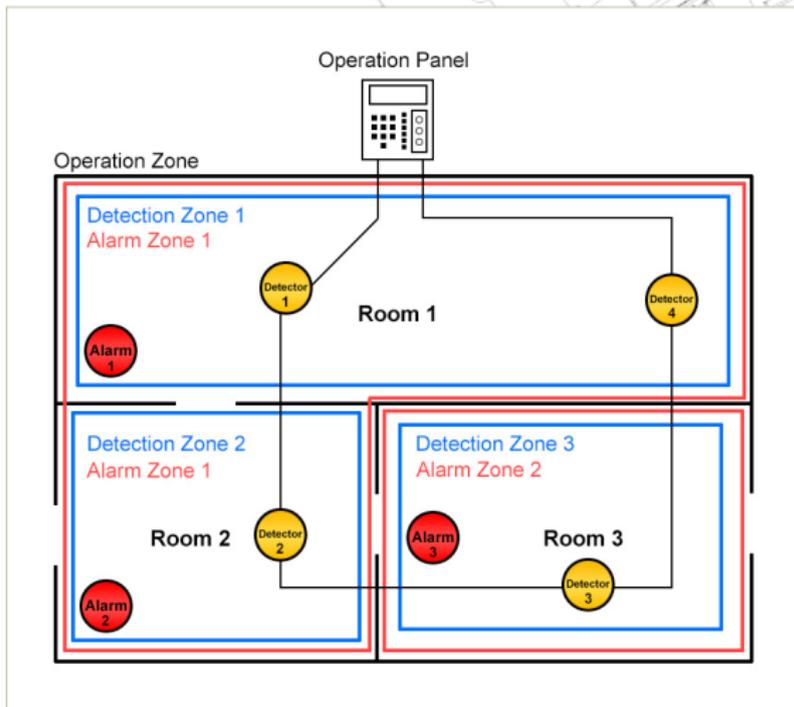


The screenshot shows the Eclipse Variant Management interface. The main window displays a grid view of model elements and variants. The grid has columns for 'Model Elements' and 'Level', followed by multiple columns for variants (labeled 'Var1_00...'). The 'Model Elements' column lists features like 'External Car ...', 'Regions', 'EMEA', 'EU', 'Austria', 'Denmark', 'UK', 'North Ameri...', 'Canada', 'Mexico', 'USA', 'Beam Confi...', 'Low Beam', 'Xenon', 'Halogen', and 'High Beam'. The 'Level' column shows hierarchical levels from 1 to 2.2. The grid cells contain checkboxes, some of which are checked (green) or unchecked (red). A tooltip is visible over one of the variant columns, showing the path 'CarLight/Configs/configspace.xml'. The interface includes a menu bar (File, Edit, Navigate, Search, Project, Run, Variant, Window, Help) and a toolbar with various icons. The status bar at the bottom right shows 'Synchronize ClearQuest Defects'.

- ▶ Commercial tools support **multiple views** of the same model
- ▶ Some vendors: Pure Systems (DE), Big Lever (US), most PLM tools
- ▶ Clafer also has the grid view

Are all variability models trees?

A Fire Alarm System



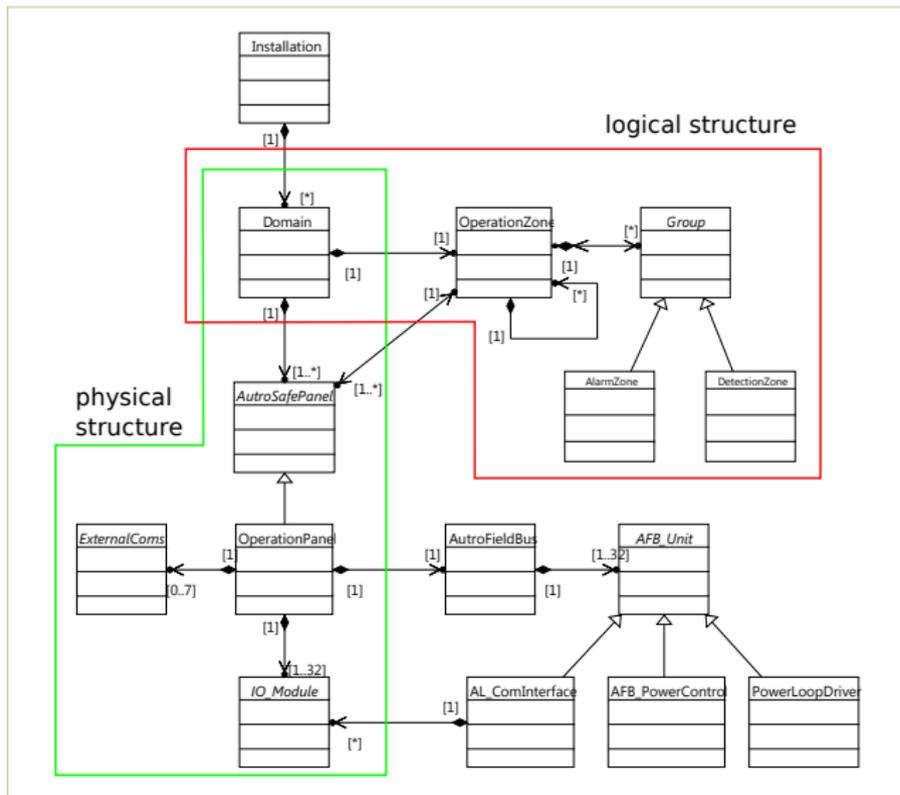
- ▶ detection zones
- ▶ alarm zones
- ▶ wiring
- ▶ three different structures
- ▶ Modeled as constrained class diagrams!
- ▶ Sometimes called **topological variability**

Berger, Stanculescu, Øgård, Haugen, Larsen, Wałowski. *To connect or not to connect: experiences from modeling topological variability*. SPLC 2014

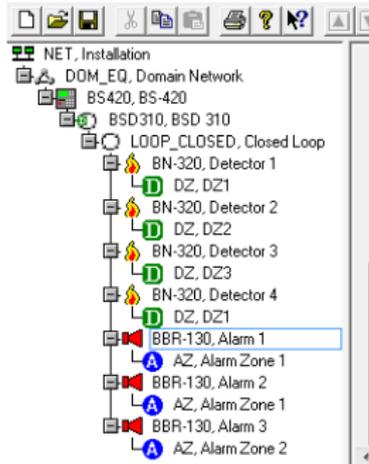
Fantechi. *Topologically configurable systems as product families*. SPLC 2013

Modeled using class diagrams in Papyrus

A model view showing two model hierarchies



A home grown **configurator** used to instantiate models



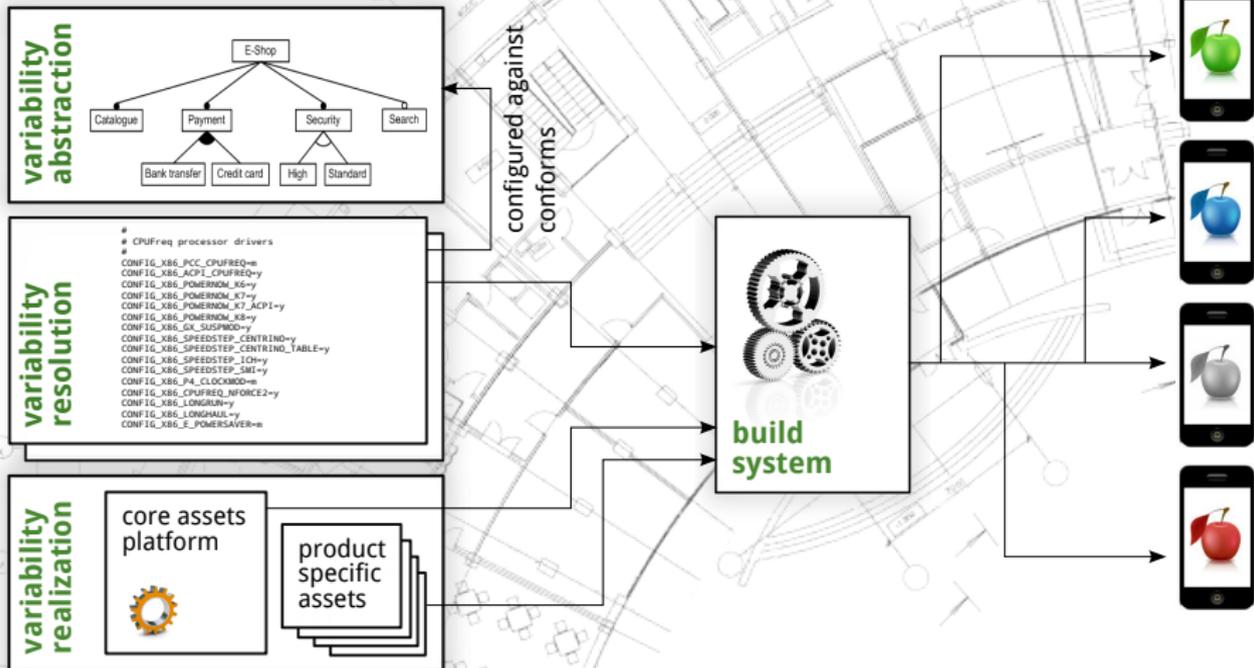
- ▶ SPL Method, **Architecture**
- ▶ Variability **Implementation Spectrum**
- ▶ Variability Abstraction: **Feature Modeling**
- ▶ Variability Modeling in **Practice**
- ▶ Variability **Realization**



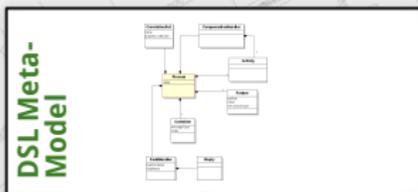
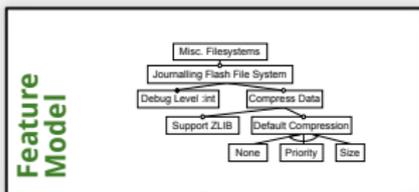
AGENDA

Connect Abstraction to Realization

Most of the school is about it :)

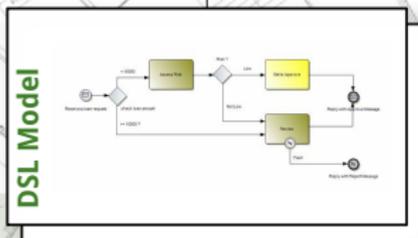
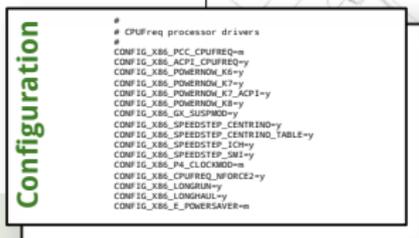


Feature Models vs DSLs



conforms to

conforms to



- ▶ Feature models are **ready and simple** (no design effort, deep insight)
- ▶ DSL requires **design effort**, but rewards with more **expressiveness**
- ▶ Effort also translates to **maintenance**
- ▶ FM effort is offset by **existing feature modeling tools**
- ▶ DSL development effort is offset by **language workbenches**

Advice on Realization

[Stahl and Völter]

- ▶ Choose **functional** domain **concepts** as features / DSL concepts
- ▶ Start a **small** domain model and **grow** it **iteratively**
- ▶ Keep the **build automatic** at all times
- ▶ Generate/synthesize **legible code/models**
- ▶ We follow these principles in the Clafer tutorial on railway stations

Generative Programming

Methods,



Tools, and



Applications



Krzysztof Czarnecki
Ulrich W. Eisenecker

 **WILEY**

TIMELY. PRACTICAL. RELIABLE.

Model-Driven Software Development

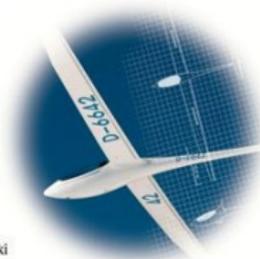
Technology,
Engineering,
Management

Thomas Stahl,
Markus Völter

with Jörn Bettin, Arno Haase
and Simon Helsen

Foreword by Krzysztof Czarnecki

Translated by Bettina von Stockfleth



KNOWLEDGE-BASED CONFIGURATION

FROM RESEARCH TO BUSINESS CASES



MK
MARTIN KOPPEL

EDITED BY
ALEXANDER FELFERNIG, LOTHAR HOTZ,
CLAIRE BAGLEY AND JUHA TIHONEN

Product Customization



Springer

Urheberrechtlich geschütztes Material

- 
- ▶ SPL Method, **Architecture**
 - ▶ Variability **Implementation Spectrum**
 - ▶ Variability Abstraction: **Feature Modeling**
 - ▶ Variability Modeling in **Practice**
 - ▶ Variability **Realization**



AGENDA