CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Software Engineering Principles for Complex Systems

TDA594

Domain Engineering & Feature Modelling

Dr. Sam Jobara
jobara@chalmers.se
Software Engineering Division
Chalmers | GU

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Dr. Sam Jobara
Chalmers | University of Gothenburg

I have a Ph.D. in Computer Science and Engineering, USF, USA

I have research interests in testing and fault modeling, computer architecture, information security, and product line engineering. I am also interested in learning and cognitive theories.

My teaching covers a multitude of courses in Computer architecture, Information security, DIT824 Software engineering for Data Intensive AI Applications, DIT834 Startups and Industrial Software product Management, and DIT192 Agile Development Processes.

My industrial experience spans over 18 years as an IT Consultant in Telecommunication, Information Security consulting, and Supply Chain Management.

jobara@chalmers.se

Learning Objectives

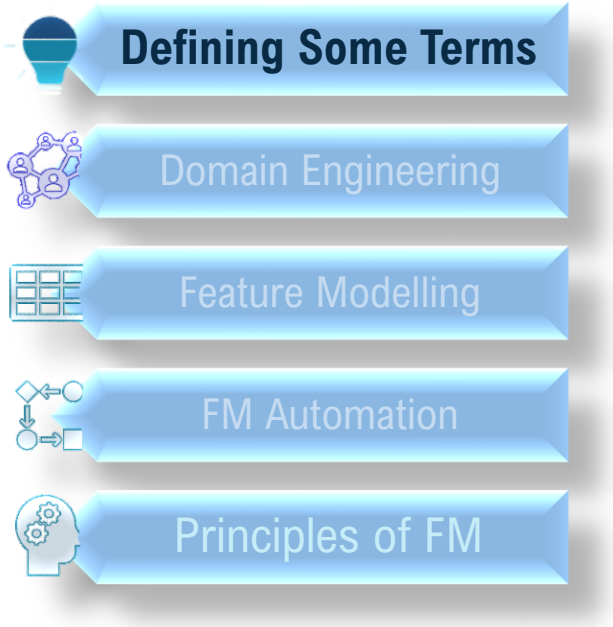
- Define feature, feature selection, feature dependency, product, domain.
- Understand what drives scoping decisions
- Translate feature diagrams to propositional formulas
- Modal features and feature dependencies by means of feature models
- Identify some industrial automation tools for Feature Modelling

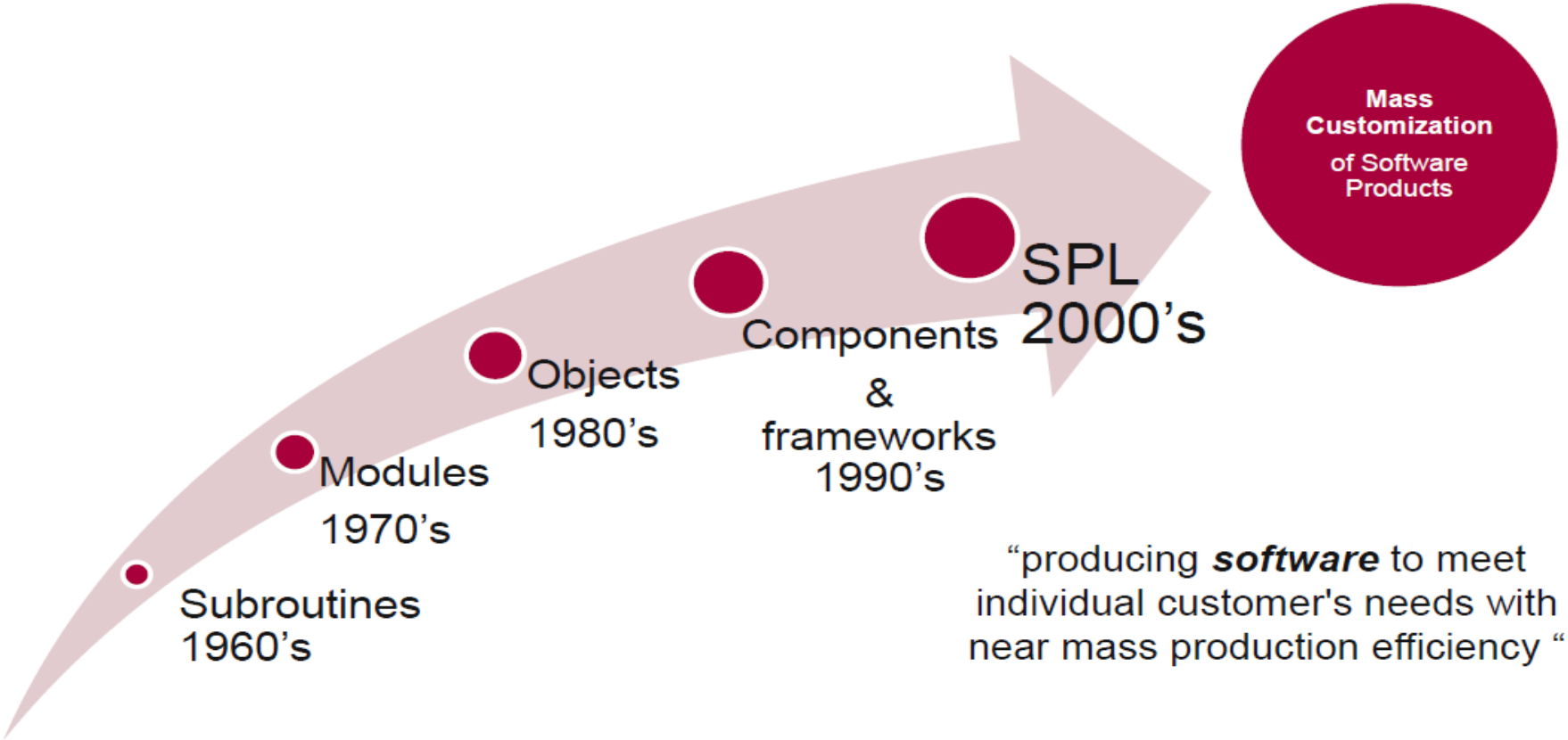
Main Reference:

Feature-Oriented Software Product Lines: Concepts and Implementation,
Sven Apel • Don Batory •Christian Kästner • Gunter Saake
Springer-Verlag Berlin Heidelberg 2013, ISBN 978-3-642-37521-7

Other publications (see slides for references)

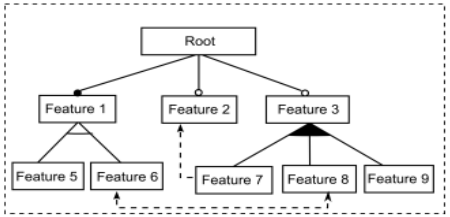
Agenda


Defining Some Terms
Domain Engineering
Feature Modelling
FM Automation
Principles of FM

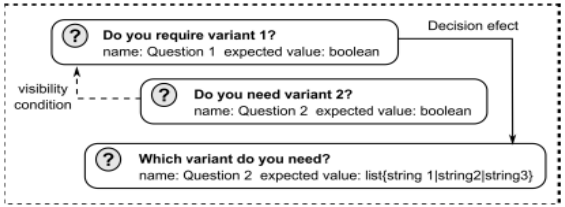


"producing ***software*** to meet individual customer's needs with near mass production efficiency "

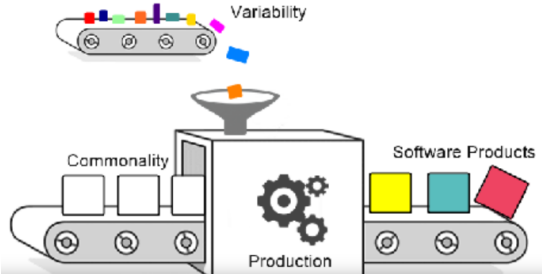
Variability Modelling Methods



Feature modelling



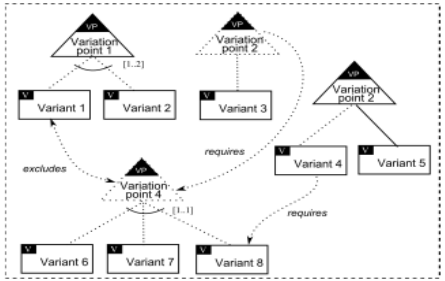
Decision modelling



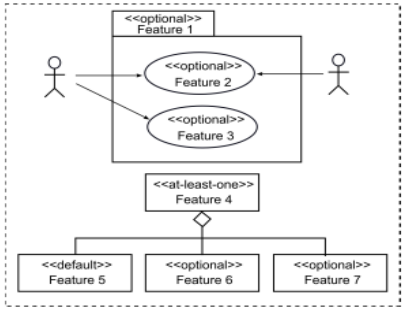
Techniques

Ad-hoc solutions: tables, textual docs, ...

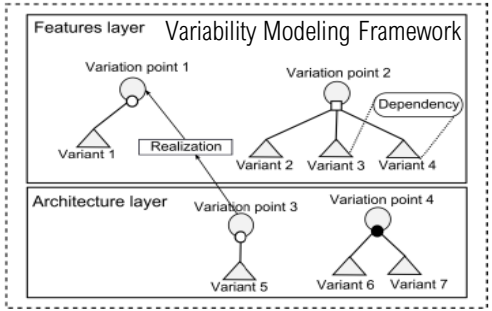
Orthogonal variability modelling



UML-based



COVAMOF



Features and Products

Features are the concerns of stakeholders and their primary interest in product-line engineering.
A *Feature* is "a logical unit of behavior specified by a set of functional and non-functional requirements"

Features are used in product-line engineering to specify and communicate commonalities and differences of the products between stakeholders, and to guide structure, reuse, and variation across all phases of the software life cycle.

The *product portfolio* of a product line is defined by its features and their relations.
A specific *product* is identified by a subset of features, called a *feature selection*.

Product Line := a family of products designed to take advantage of their common aspects and predicted variabilities [Weiss, Lai]

SPL Domain

A key success factor of product-line development is to set a proper focus on a particular, well-defined and well-scoped domain.

A *domain* is an area of knowledge that:
is scoped to maximize the satisfaction of the requirements of its stakeholders, includes a set of concepts and terminology understood by practitioners in that area and includes the knowledge of how to build software systems.

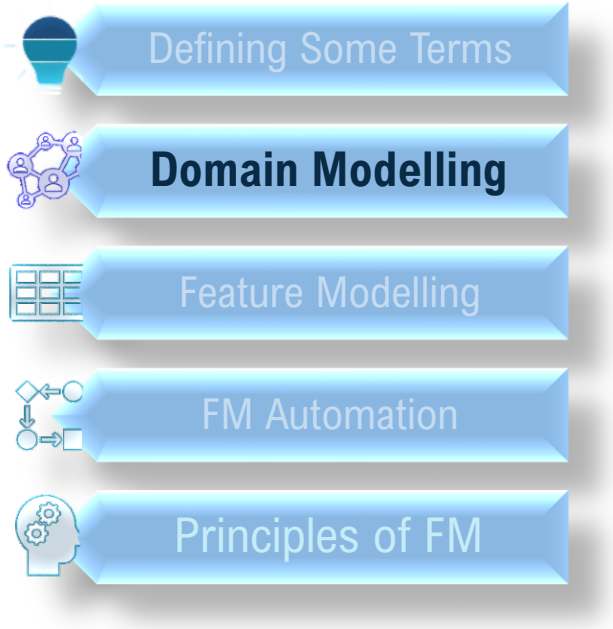
Domain Engineering

Domain engineering is the process of analyzing the domain of a product line and developing reusable artifacts. Domain engineering does not result in a specific software product, but prepares artifacts to be used in multiple, if not all, products of a product line.

Domain engineering is the life-cycle that is further responsible for scoping the product line and ensuring that the platform has the variability that is needed to support the desired scope of products. It targets development for reuse.

Domain engineering results in the common assets that together constitute the product line's platform.

Agenda

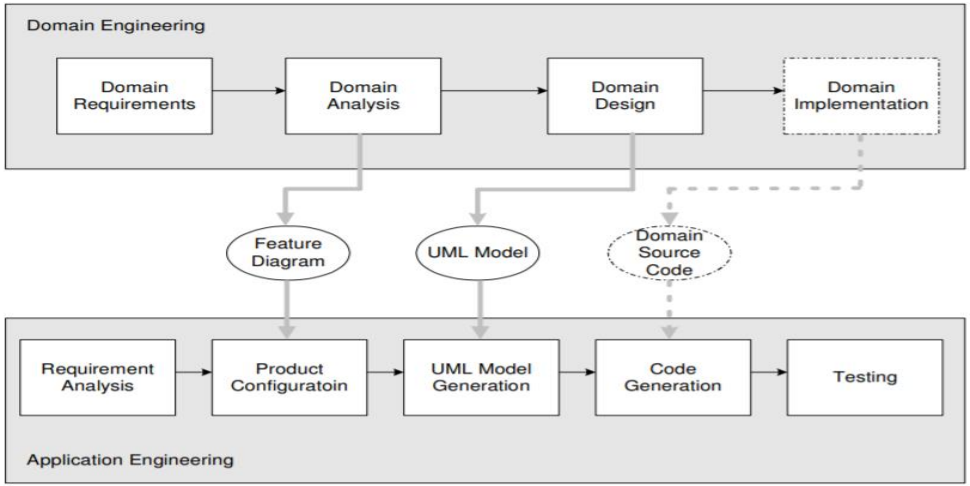

Defining Some Terms
Domain Modelling
Feature Modelling
FM Automation
Principles of FM

Domain Modelling

Domain modelling captures and documents the commonalities and variabilities of the scoped domain.
Typically, commonalities and differences between desired products are identified and documented in terms of features and their mutual dependencies under *feature models.*

Domain Modelling

The **FeatureUML method** aims to define a systematic way of developing software product line. The focus of this method is the domain analysis phase and part of the application engineering phase. The modeling techniques used in this method include feature diagrams and UML models.

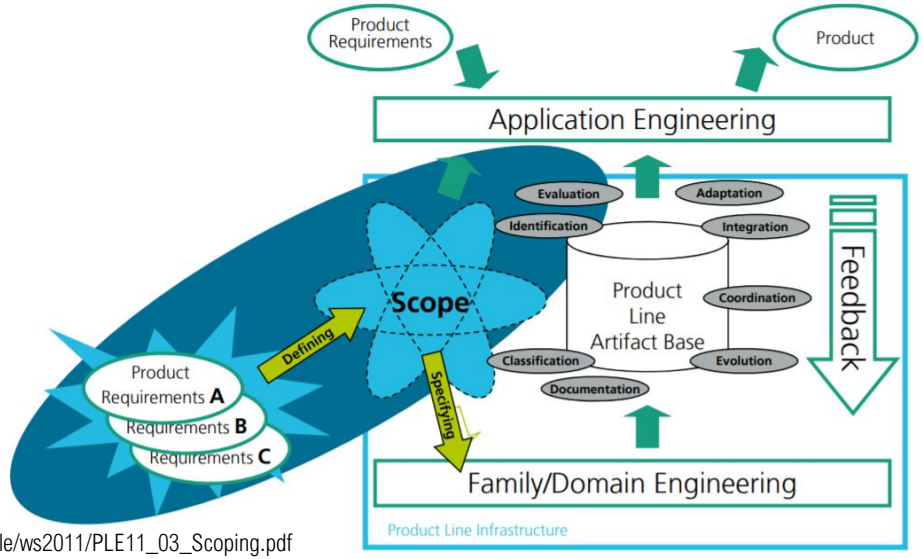


Product line system development process from FeatureUML.

Domain Modelling

Domain Analysis: Scoping

Domain scoping is the process of deciding on a product line's extent or range.
The scope describes desired features or specific products that should be supported.



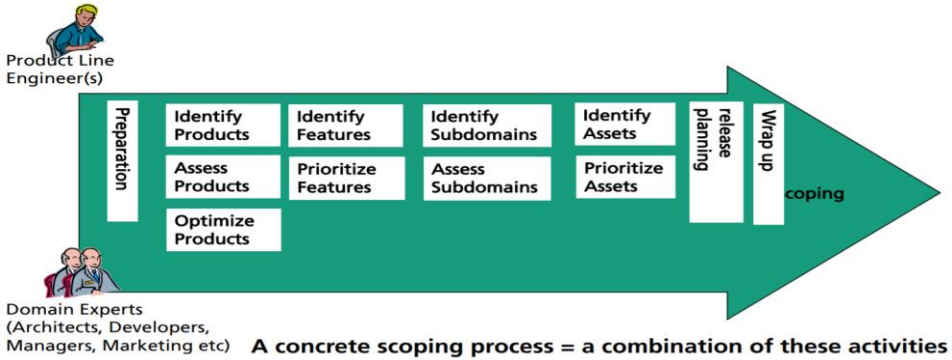
http://wwwagse.informatik.uni-kl.de/teaching/ple/ws2011/PLE11_03_Scoping.pdf

Domain Modelling

Domain Analysis: Scoping

Product lines with a small scope are easier to develop and maintain, as they target a well-defined domain of very similar products with few variations and much reuse.

There is a trade-off between implementation effort and potential use of the product line. The trade-off requires careful business consideration, including determining prospective revenue, potential customers, and costs of additional features.



A concrete scoping process = a combination of these activities

Domain Analysis: Scoping

Common concepts/questions of all scoping approaches

Products:

Which products do I want in my product line? What is their market, when will they be released?

Domains:

Which subdomains will my product line have? Which information do they carry? What are reasonable domains for the product line (in terms of knowledge, stability etc)?

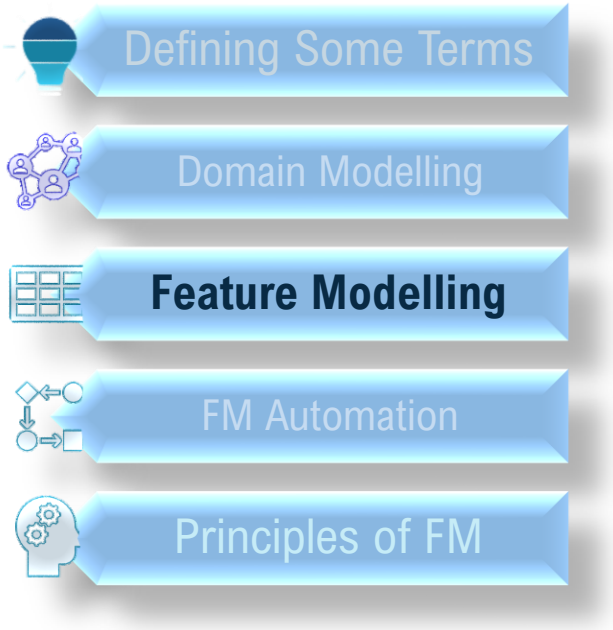
Features:

Which features will my product line have? Which product will have what kind of features? Which are easy, which are risky features?

Assets:

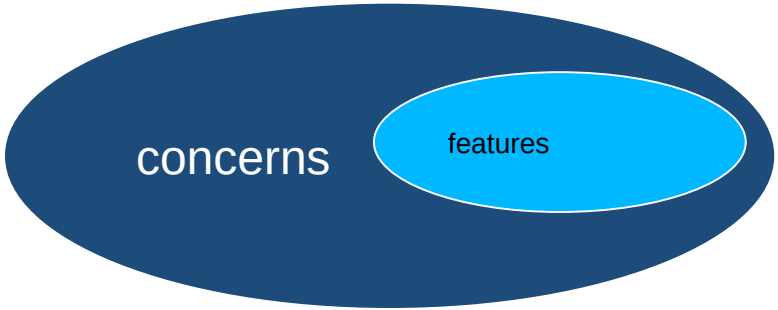
Which assets do I have in my product line? Which components, documentation etc exists already in a reusable form, which ones do I have to (re-)implement?

Agenda


Defining Some Terms
Domain Modelling
Feature Modelling
FM Automation
Principles of FM

What is a Feature?

- Concept in a domain
- Can be seen as a high-level requirement
- Features represent commonalities and variabilities in a product line
- Unit of communication among stakeholders
- A specific set of features determines a product variant
- feature configuration as input to product derivation
- A feature is a kind of concern



What is a Feature?

Feature-Based Configuration involves:

- Checking for satisfiability of FMs
- Enforcing correct configurations
- Counting possible configurations
- Enumerating valid configurations
- Propagating configuration decisions
- Merging distributed configurations
- ...

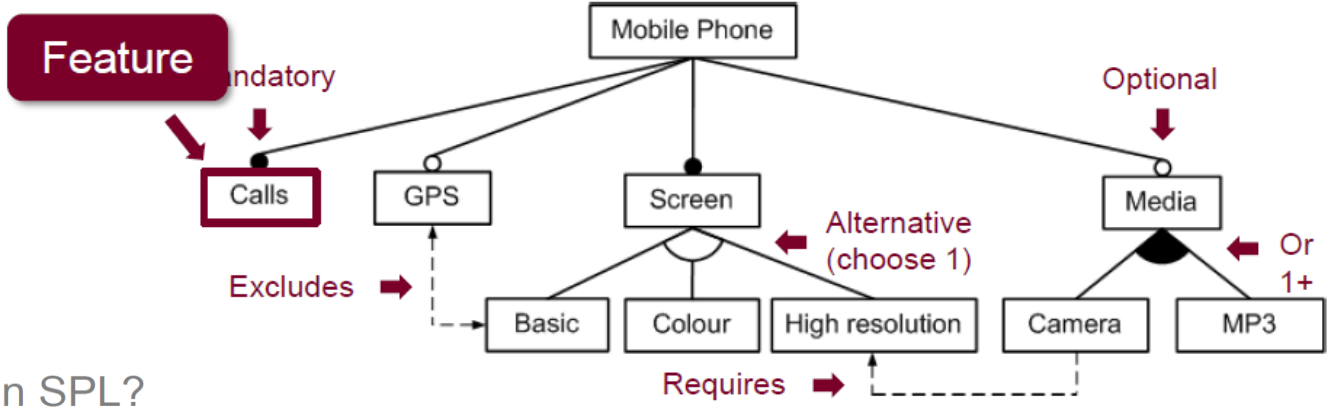
good feature

- popular with customers
- popular with developers
- well implemented
- error-free
- thoroughly tested
- architecture-conform
- distinct functionality

bad feature

- customer complaints
- duplicate features
- workaround (“hack”)
- defect features
- test challenges
- optional feature
- highly volatile

Feature Modelling



Feature models

How to specify an SPL?

"Feature Model: A hierarchically arranged set of features to represent all possible products of an SPL"

Feature Modelling

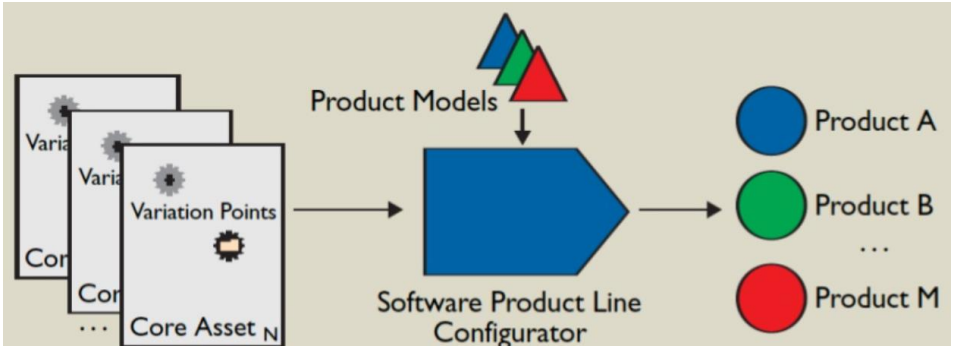
Feature modeling is a notation for modeling commonality and variability in product families. In their basic form, feature models contain mandatory/optional features, feature groups, and implies and excludes relationships.

It is known that such feature models can be translated into propositional formulas, which enables the analysis and configuration using existing logic-based tools.

Feature Modelling

Feature modeling is to express variability in terms of common and optional features.

Feature modeling takes place in domain analysis, but its results play a central role in other phases of product-line development such as requirements analysis and product derivation. In short, a feature model documents a product line's variability.

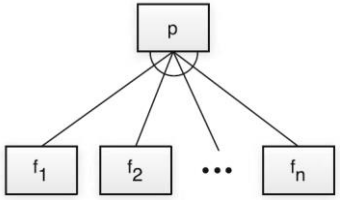


Feature Modelling

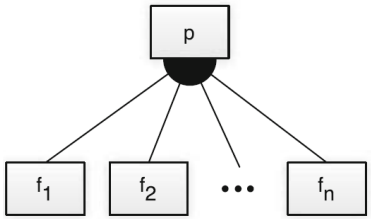
Feature Diagrams

A **feature diagram** is a graphical notation to specify a feature model. It is a tree whose nodes are labeled with feature names. Different notations convey various parent–child relationships between features and their constraints:

Only One-out-of many choice. This choice corresponds to a generalized XOR operator



Some-out-of-many choice. This choice corresponds to the logical OR operator



Feature Modelling

Feature Diagrams

Figure 2.5 shows an example of a feature diagram that, in its complete form, covers 65 features. Nodes shaded in gray are folded subtrees.

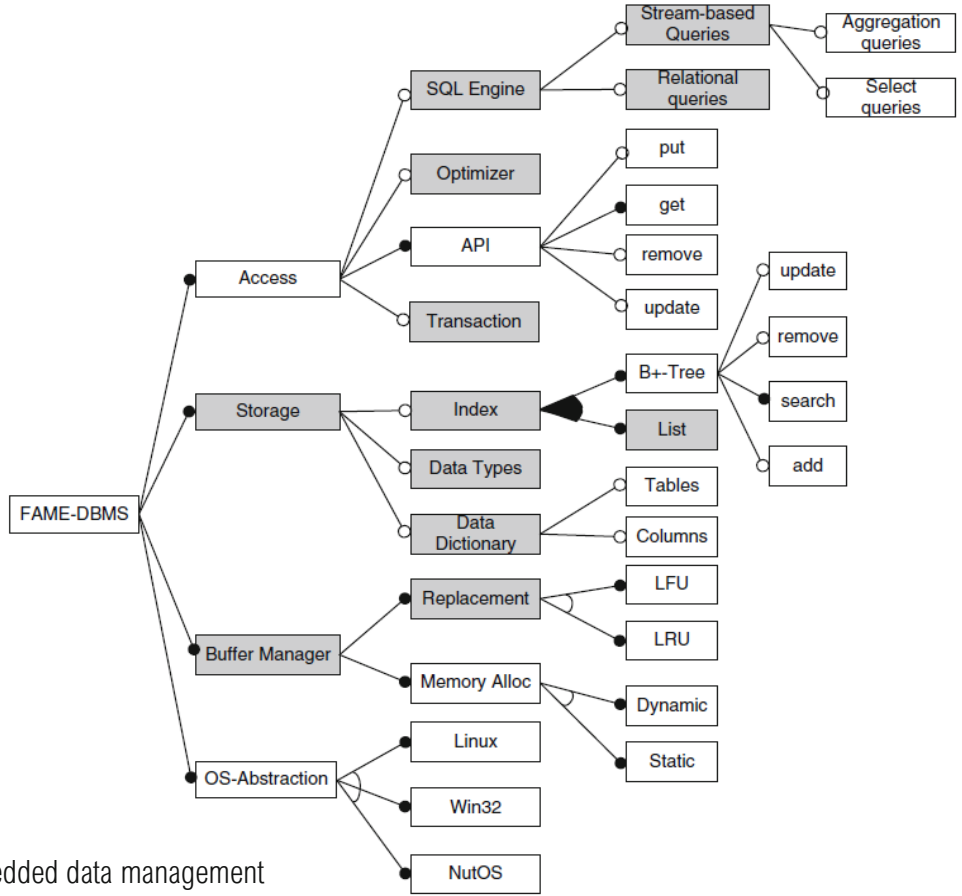


Fig. 2.5 Sample feature diagram for embedded data management

Feature Modelling

Formalization in Propositional Logic

Feature diagrams can be directly mapped to propositional formulas, thereby defining a formal semantics of feature diagrams. A set F of feature names are interpreted as propositional variables, and p, f and fi are members of F.

A mandatory feature definition between a parent feature p and a child feature f:

$$\mathbf{mandatory(p,f) \equiv f \Leftrightarrow p}$$

Denote solid bullet, if the parent feature is selected, then the child must be selected, and vice versa

An optional feature states that the parent p may be chosen independently from f, but the child f can only be chosen if p is selected:

$$\mathbf{optional(p,f) \equiv f \Rightarrow p}$$

Denoted by an empty bullet

Feature Modelling

Formalization in Propositional Logic

Mapped to propositional logic, this is a ***disjunction***, in which, at least, one child feature is selected when the parent is chosen. It is an XOR, one-out-of-many choice:

$$\texttt{alternative}(p, \{f_1, ..., f_n\}) \equiv ((f_1 \vee \ldots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$$

An unrestricted **choice** denoted by a filled arc in feature diagrams. Mapped to propositional logic, the selection of p is equivalent to a disjunction of the child features. It is an OR, some-out-of-many choice:

$$\texttt{or}(p, \{f_1, \ldots f_n\}) \equiv (f_1 \vee \ldots \vee f_n) \Leftrightarrow p$$

Propositional logic enables us to use automated tools to test interesting properties, such as checking validity of feature models and feature selections, and detect dead features

Feature Modelling

Formalization in Propositional Logic

We use the product line of graph libraries to illustrate feature diagrams formalization. We use the feature diagram to illustrate the mapping to a propositional formulas:

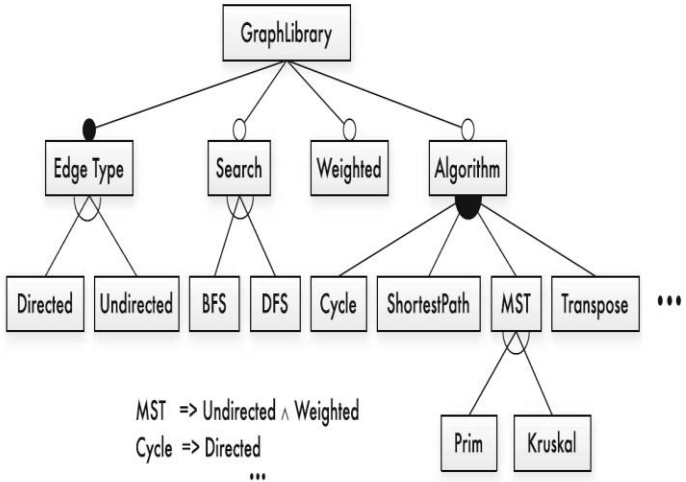


Fig. 2.6 A possible feature diagram of the graph library

$$
\begin{aligned}
&\texttt{root(GraphLibrary)}\\
\wedge\ &\texttt{mandatory(GraphLibrary,EdgeType)}\\
\wedge\ &\texttt{optional(GraphLibrary,Search)}\\
\wedge\ &\texttt{optional(GraphLibrary,Weighted)}\\
\wedge\ &\texttt{optional(GraphLibrary,Algorithm)}\\
\wedge\ &\texttt{alternative(EdgeType,\{Directed,Undirected\})}\\
\wedge\ &\texttt{or(Search,\{BFS,DFS\})}\\
\wedge\ &\texttt{or(Algorithm,\{Cycle,ShortestPath,MST,Transpose\})}\\
\wedge\ &\texttt{alternative(MST,\{Prim,Kruskal\})}\\
\wedge\ &(\texttt{MST} \Rightarrow \texttt{Weighted})\\
\wedge\ &(\texttt{Cycle} \Rightarrow \texttt{Directed})\\
\wedge\ &(\cdots)
\end{aligned}
$$

Any coment over the formula of Search node?

$$\wedge\ \texttt{alternative(Search,\{DFS,BFS\})}$$

Feature Modelling

Formalization in Propositional Logic

After expanding the feature constraints, we arrive at the following formula:

$$
\begin{aligned}
&\texttt{root(GraphLibrary)} \\
\wedge\ &\texttt{mandatory(GraphLibrary, EdgeType)} \\
\wedge\ &\texttt{optional(GraphLibrary, Search)} \\
\wedge\ &\texttt{optional(GraphLibrary, Weighted)} \\
\wedge\ &\texttt{optional(GraphLibrary, Algorithm)} \\
\wedge\ &\texttt{alternative(EdgeType, \{Directed, Undirected\})} \\
\wedge\ &\texttt{or(Search, \{BFS, DFS\})} \\
\wedge\ &\texttt{or(Algorithm, \{Cycle, ShortestPath, MST, Transpose\})} \\
\wedge\ &\texttt{alternative(MST, \{Prim, Kruskal\})} \\
\wedge\ &(\texttt{MST} \Rightarrow \texttt{Weighted}) \\
\wedge\ &(\texttt{Cycle} \Rightarrow \texttt{Directed}) \\
\wedge\ &(\cdots)
\end{aligned}
$$

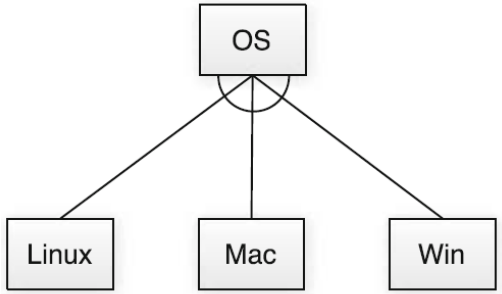
$$
\begin{aligned}
&\texttt{GraphLibrary} \\
\wedge\ &(\texttt{EdgeType} \Leftrightarrow \texttt{GraphLibrary}) \\
\wedge\ &(\texttt{Search} \Rightarrow \texttt{EdgeType}) \\
\wedge\ &(\texttt{Weighted} \Rightarrow \texttt{EdgeType}) \\
\wedge\ &(\texttt{Algorithm} \Rightarrow \texttt{EdgeType}) \\
\wedge\ &(((\texttt{Directed} \vee \texttt{Undirected}) \Leftrightarrow \texttt{EdgeType}) \wedge \neg(\texttt{Directed} \wedge \texttt{Undirected})) \\
\wedge\ &((\texttt{BFS} \vee \texttt{DFS}) \Leftrightarrow \texttt{Search}) \\
\wedge\ &((\texttt{Cycle} \vee \texttt{ShortestPath} \vee \texttt{MST} \vee \texttt{Transpose}) \Leftrightarrow \texttt{Algorithm}) \\
\wedge\ &(((\texttt{Prim} \vee \texttt{Kruskal}) \Leftrightarrow \texttt{MST}) \wedge \neg(\texttt{Prim} \wedge \texttt{Kruskal})) \\
\wedge\ &(\texttt{MST} \Rightarrow \texttt{Weighted}) \\
\wedge\ &(\texttt{Cycle} \Rightarrow \texttt{Directed}) \\
\wedge\ &(\cdots)
\end{aligned}
$$

$$\wedge\ ((\texttt{BFS} \vee \texttt{DFS}) \Leftrightarrow \texttt{Search}) \wedge \neg(\texttt{BFS} \wedge \texttt{DFS}))$$

Feature Modelling

Formalization in Propositional Logic

To illustrate the transformation to propositional logic for 3-children situations, we use an additional example to ensure all combinations are counted for:



$$OS \Leftrightarrow (Linux \vee Win \vee Mac) \wedge \neg(Linux \wedge Win) \wedge \neg(Linux \wedge Mac) \wedge \neg(Win \wedge Mac)$$

Feature Modelling

Formalization in Propositional Logic

In the literature, there are many variations of feature diagrams including:

1. Some cross-tree constraints can be modeled graphically. Arrows can denote implications or mutual exclusion, as exemplified in Fig. 2.8a.

2. Some notations distinguish *abstract* from concrete features. Abstract features are used for structuring and documentation purposes only and are not bound to implementation artifacts. (such as features EdgeType and Search in Fig. 2.6). In Fig. 2.8b, abstract features are denoted by gray boxes.

3. Some notations support multiple group types under the same feature. For example, in Fig. 2.8c, features I, J, and K share the same parent even though they belong to different groups.

4. Some notations permit the mixing of mandatory and optional features with alternative and choice groups, as also illustrated in Fig. 2.8c.

Feature Modelling

Formalization in Propositional Logic

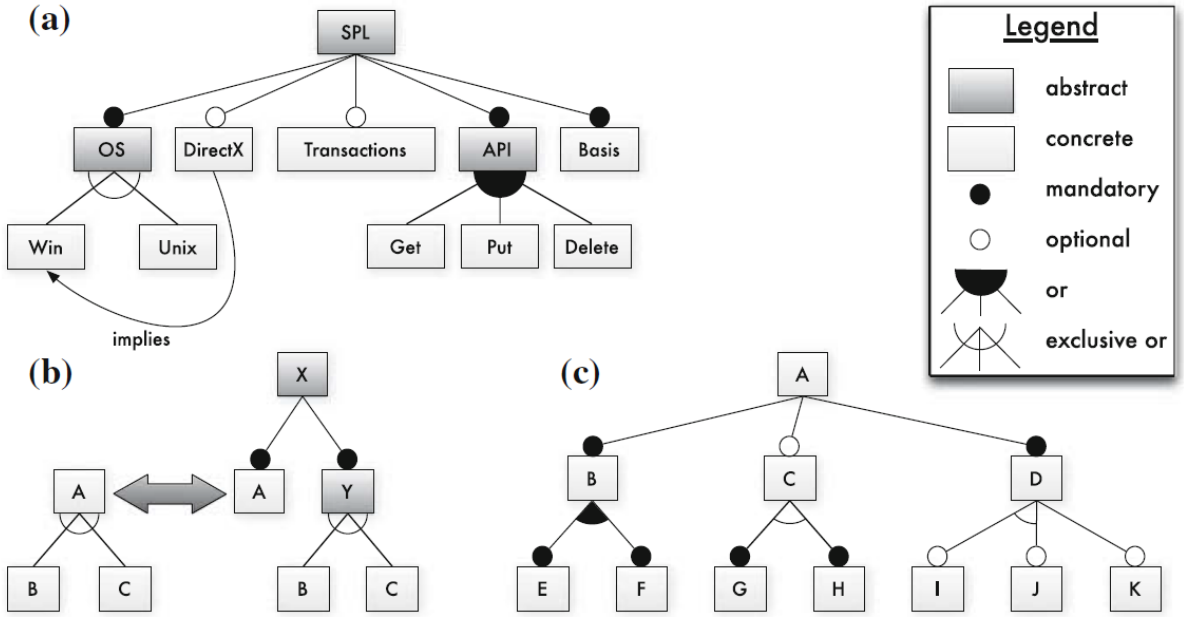
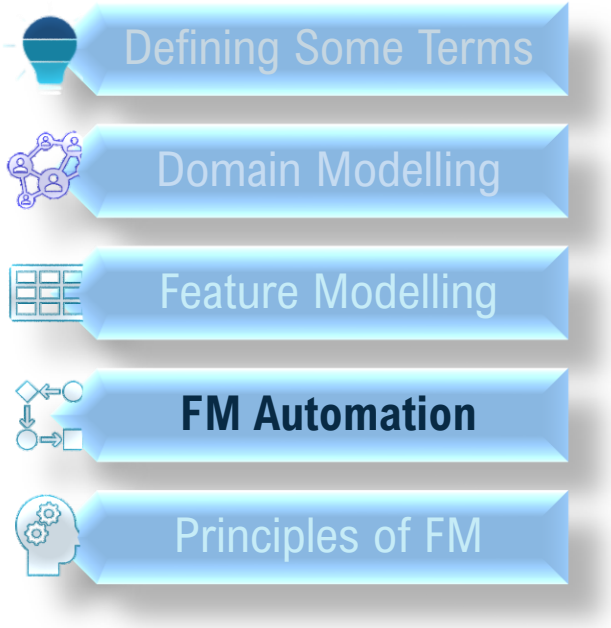


Fig. 2.8 Some variations of feature diagrams: **a** cross-tree constraints, **b** transformation toward abstract inner features, **c** mixing optionality and group constraints

Agenda


Defining Some Terms
Domain Modelling
Feature Modelling
FM Automation
Principles of FM

FM Automation

Feature Modeling in Auto industry*

In Automotive industry suppliers analyze the systems requirements and include applicable domain specific guideline requirements.
Form a master feature list as shown in Figure 1. Domain specific guideline encompasses best practices and certification related requirements pertaining to the required features.
Feature models show the common and variable features of a Product Line.
Feature Models could be created using several tools both commercial and open source. Here we show Feature IDE tool for this examples.

*Jaisimha, S., Rajan, M., Kanagaraj, P., and Rajendran, S.K., "Managing Product Development through Feature Modeling," SAE Technical Paper 2018-01-0089, 2018, doi:10.4271/2018-01-0089

FM Automation

Feature Modeling in Auto industry

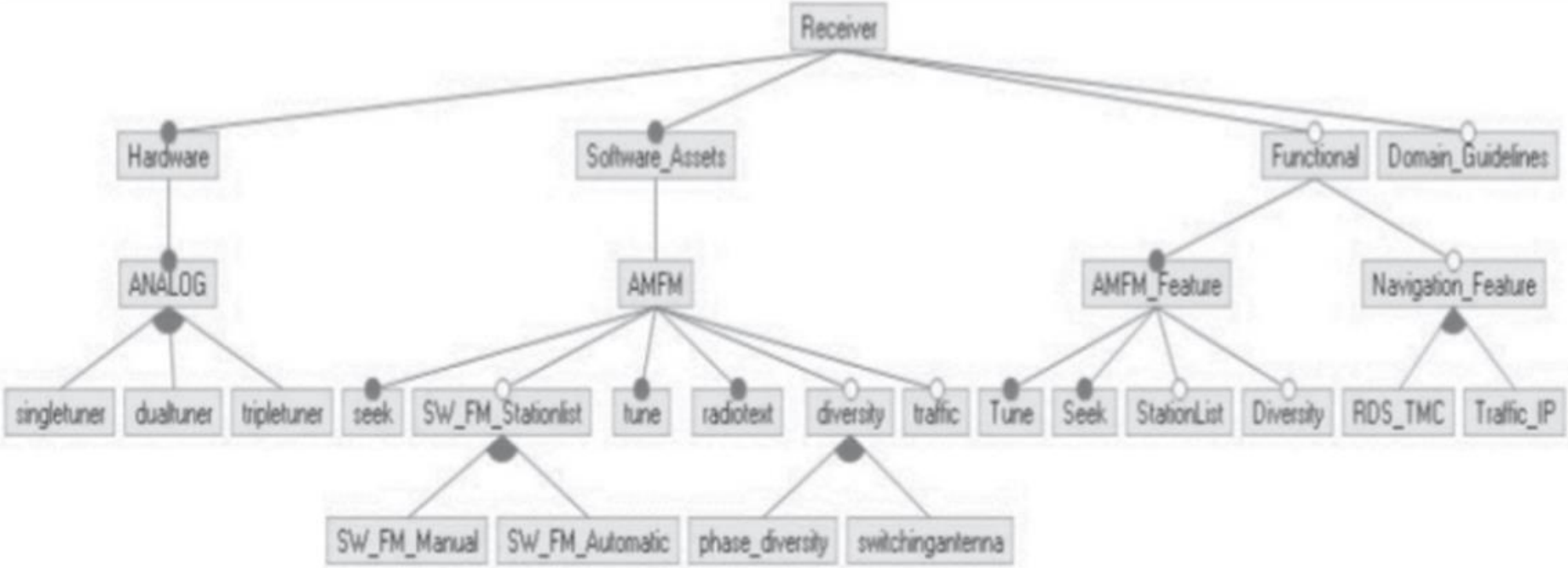
The following steps show how to create a master model and use it to derive different product variants:
Step 1: Incorporate all inputs stemming from functional, non-functional, domain guidelines and form a feature model.
Step 2: Iterate across this feature model until we have the correct level of feature levels leading to an atomic tangible feature that can be tested.
Step 3: Include the feature model with Hardware, Software, and Mechanical model options to realize the feature sets.
Step 4: Map the feature constraint relationship to form a master feature Model.

$$SW\_FM\_Manual \Rightarrow singletuner$$
$$phase\_diversity \Rightarrow dualtuner$$
$$RDS\_TMC \Rightarrow \neg singletuner$$
$$SW\_FM\_Automatic \Rightarrow \neg singletuner$$
$$RDS\_TMC \Rightarrow traffic$$
$$Diversity \Rightarrow phase\_diversity$$
$$phase\_diversity \Rightarrow \neg singletuner$$

FM Automation

Feature Modeling in Auto industry

Master Feature model example exhibit with model constraints defined



FM Automation

Feature Modeling in Auto industry

Step 5: Review the product variations for the different vehicle segments and regions needed for the OEM. Using the Master Feature Model, derive respective product configuration or product profiles. Figures 2c and 2d shows the product variants or profiles derived from the master feature model.

Step 6: Perform model checks to ensure that there is no risk in achieving the feature requirement. Any issues found during the model check will require a common review with the System Architecture Team

Variant Matrix

Feature	USA	Europe
Tune	Yes	Yes
Seek	Yes	Yes
Phase Diversity	No	Yes
RDS TMC	Not Needed	Not Needed
Station list	Not needed	Yes

FM Automation

Feature Modeling in Auto industry

Feature Modelling, using Feature IDE tool to drive Product Configuration:

FIGURE 2C USA Product Configuration derived from Master Feature model

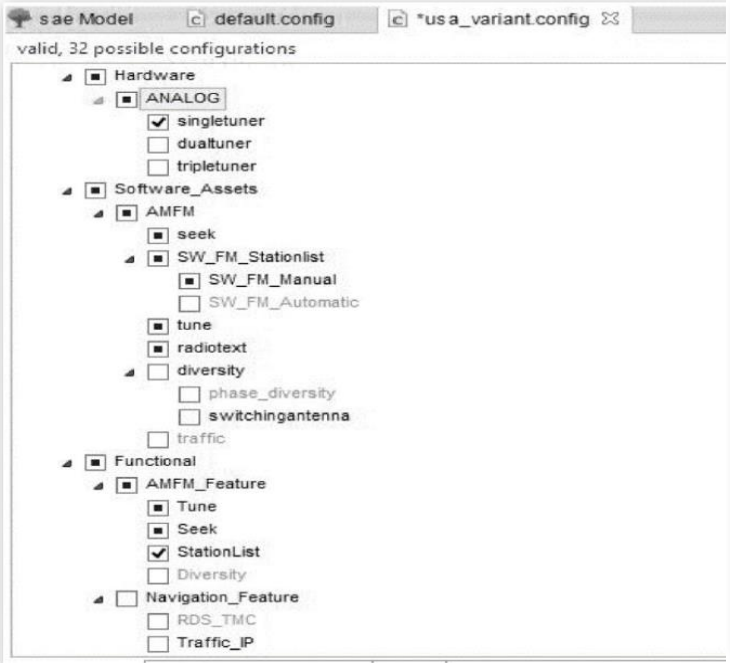


FIGURE 2D European Product Configuration derived from Master Feature model

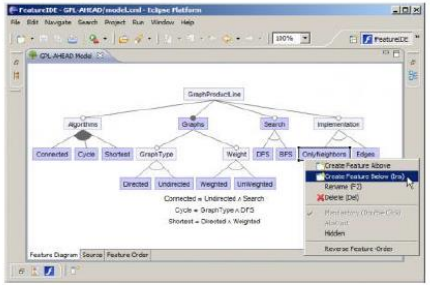
FM Automation

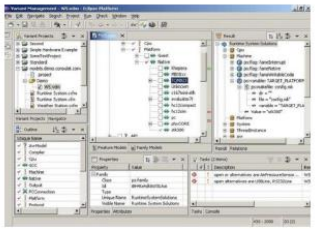
Automation Tools

pure-systems

IDE Feature

BigLever Software, Inc.





FM Automation

Pure::variants*

Pure::variants provides a set of integrated tools to support each phase of the software product-line development process.

The problem domain is represented using hierarchical Feature Models.

The solution domain, i.e. the concrete design and implementation of the software family, is implemented as Family Models.

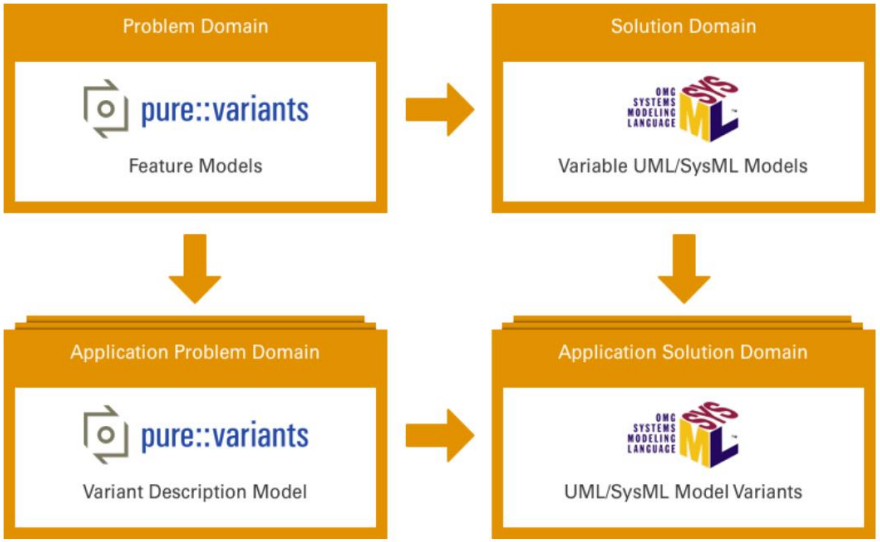
The Application Problem Domain, has the Variant Description Model (VDM), containing the selected feature set, represents a single problem.

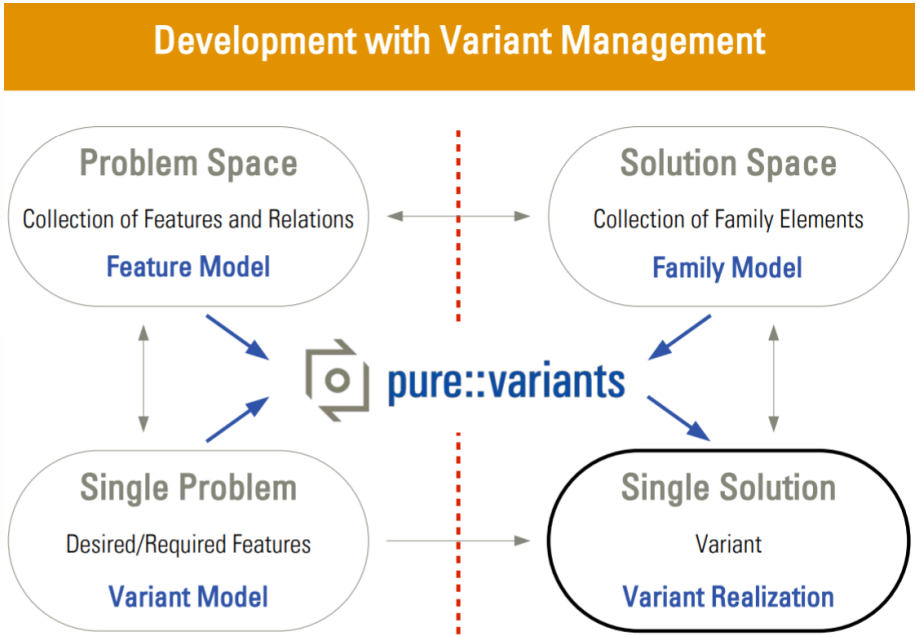
The Variant Result Model describes a single concrete solution drawn from the solution family.

* pure::variants User's Guide pure-systems GmbH

FM Automation

Pure::variants*

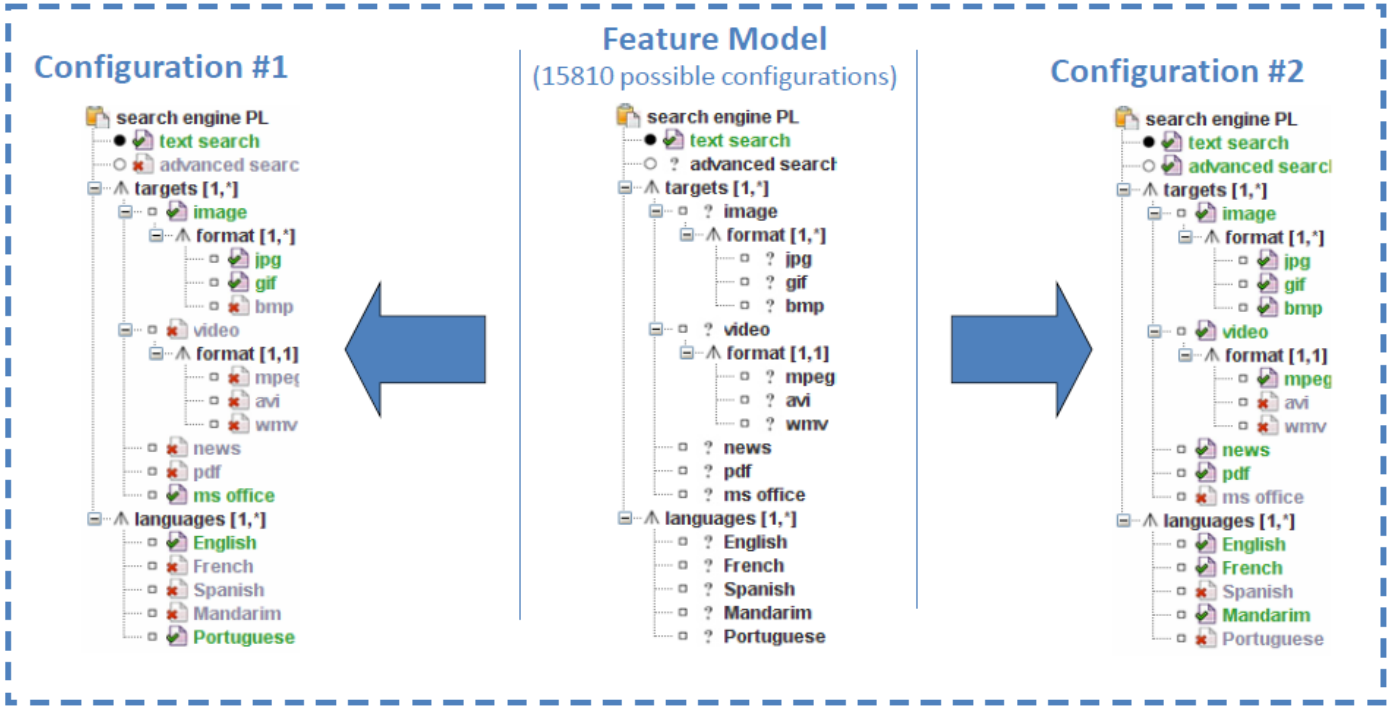




FM Automation

Pure::variants

Several support approaches & tools: FMP, XFeature, AHEAD, Gears, pure::variants, …

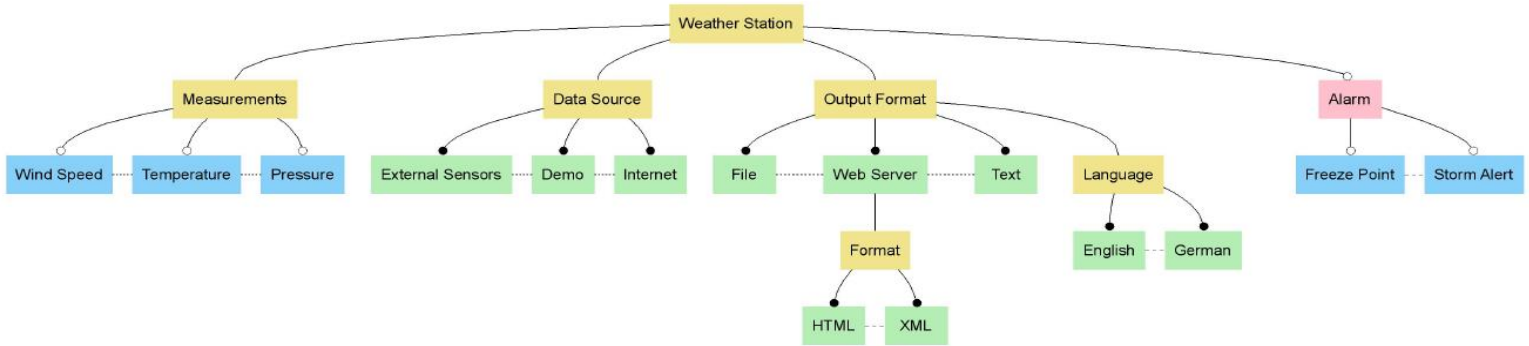


FM Automation

Pure::variants: Feature Model

The problem space can be described with Feature Models, or with a Domain Specific Language (DSL). Feature models are, hierarchical models that capture the commonality and variability of a Product Line.

In this representation both color and box connector are used independently to indicate the type of group. In the figure, Each Feature Model has a root feature. Beneath this are three mandatory features – "Measurements", "Data Source“ and "Output Format". Mandatory features are not variable in the true sense.



FM Automation

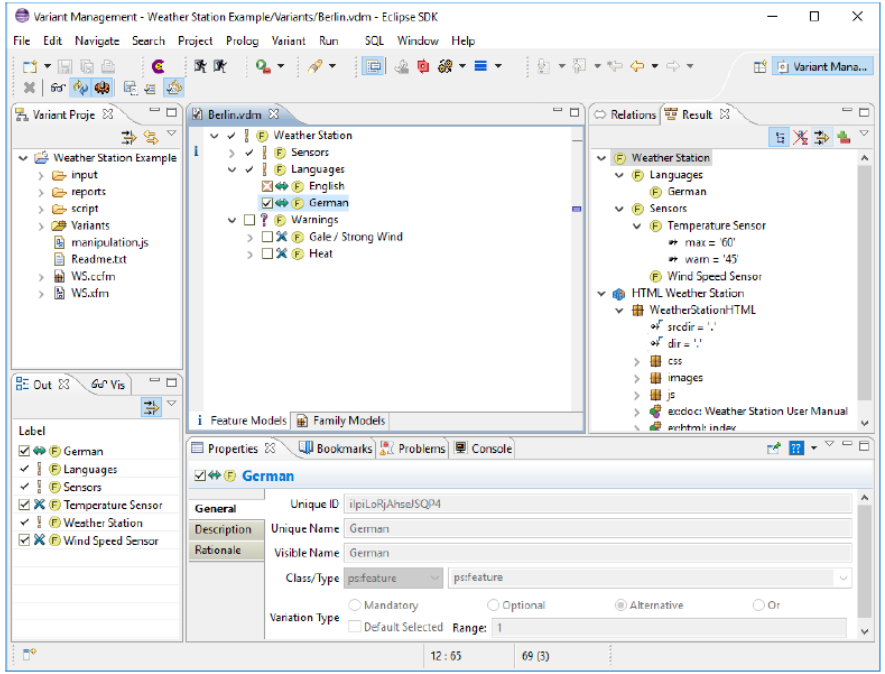
Pure::variants: Using Feature Models

When you select the Variant Projects view in the upper left side of the Eclipse window.

Create an initial standard project using the context menu of this view and choose New->Variant Project.

Once the standard project has been created, three editor windows will be opened automatically:
one for the **Feature model**, one for the Family Model and one for the VDM.

Initial layout of the Variant Management Perspective



FM Automation

Pure::variants: Using Feature Models

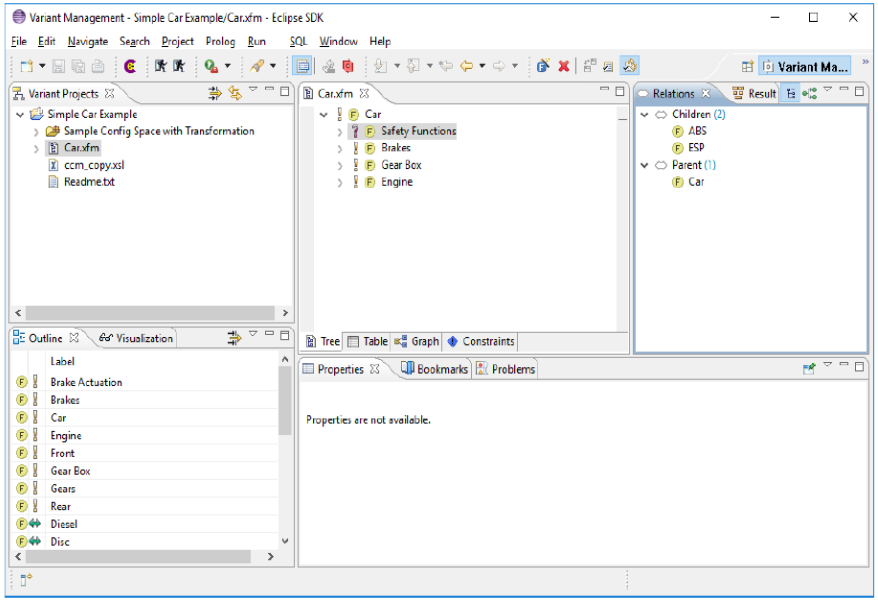
The figure here shows an example of Feature Model for a car. The Outline view (lower left corner) shows configurable views of the selected Feature Model.

The Properties view in the lower middle of the Eclipse window shows properties of the currently selected feature.

The Table tab of the Feature Model Editor (shown in the lower left part) provides a table view of the model. It lists all features in a table.

The Constraints tab contains a table with all constraints defined in the model supporting full editing capabilities for the constraints.

A simple Feature Model of a car



FM Automation

FeatureIDE Introduction

Feature Models could be created using several tools both commercial and open source. In addition to Pure::variants, these known tools are:

- FeatureIDE
- Capela
- GEARS

I will provide some highlights on Feature-ID in this lecture.

FM Automation

FeatureIDE Introduction

FeatureIDE provides support for:

- Domain analysis using feature models specifying valid combinations of features
- Requirements analysis with configuration support in concert with the feature model
- Domain implementation for diverse implementation languages and tools
- Product derivation for several generation tools without IDE support or support for features
- Feature traceability to trace features in feature model, configurations, and code
- Quality assurance for building reliable software product lines

FM Automation

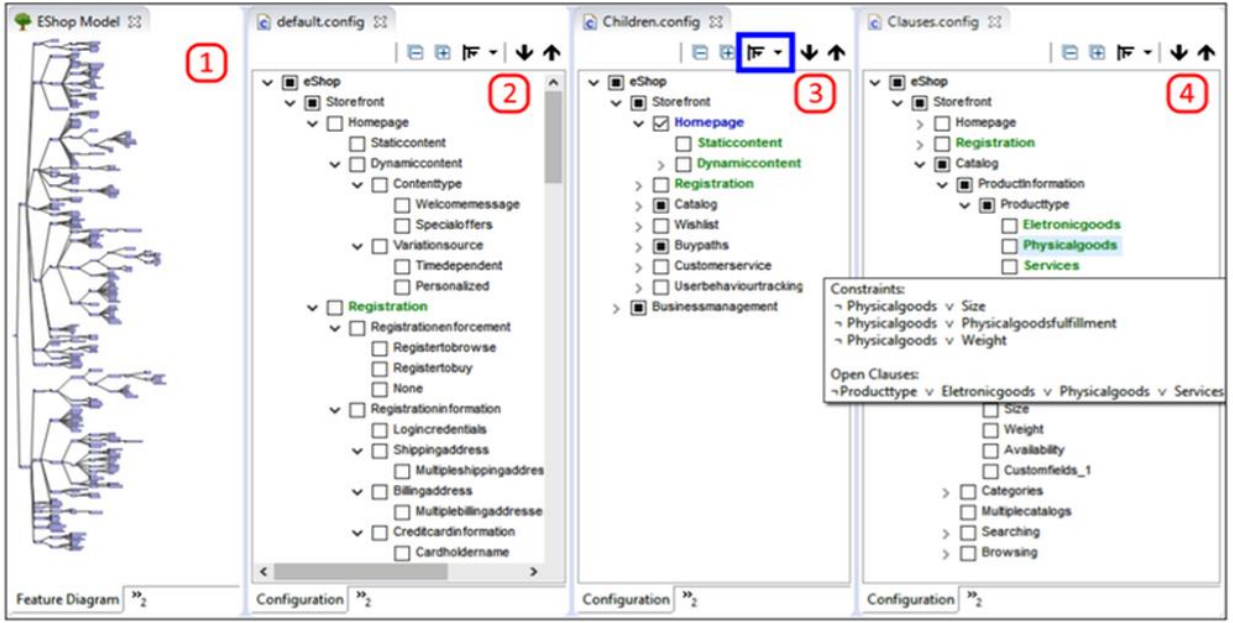
FeatureIDE Introduction

FeatureIDE supports all phases of feature-oriented software development for the development of SPLs: domain analysis, domain design, domain implementation, requirements analysis, software generation, and quality assurance.

Different SPL implementation techniques are integrated such as feature-oriented programming (FOP), aspect-oriented programming (AOP), preprocessors, and plug-ins.

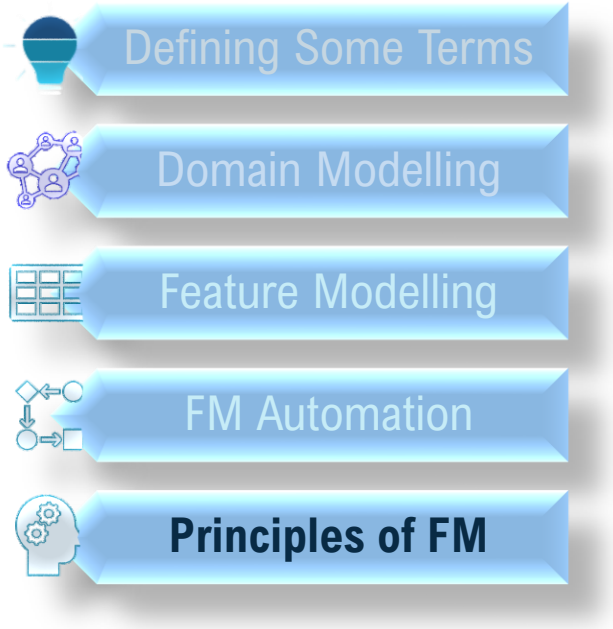
FM Automation

Feature-ID Introduction



An overview of FeatureIDE's configuration support: ① feature model editor, ②-④ configuration editor (② showing all features, ③ showing direct children, ④ finalizing configuration).

Agenda


Defining Some Terms
Domain Modelling
Feature Modelling
FM Automation
Principles of FM

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Principles of Feature Modeling*

* Damir Nešić, Jacob Krüger, Ștefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338906.3338974

Feature-Modelling Principles*

Feature models help developers to keep an overall understanding of the system, and also support scoping, planning, development, variant derivation, configuration, testing, and maintenance activities that sustain the system's long-term success.

In this research* a set of 34 principles, covering eight different phases of feature modeling, these principles provide practical, context specific advice on how to perform feature modeling, describe what information sources to consider, and highlight common characteristics of feature models.

These principles should enhance feature-modeling tooling, synthesis, and analyses techniques.

*http://www.cse.chalmers.se/~bergert/paper/2019-fse-fm_principles.pdf
"Principles of Feature Modeling" Damir Nešić, Jacob Krüger, Ştefan Stănciulescu, Thorsten Berger

Feature-modeling principles

1. Planning and Preparation

PP1: **Identify relevant stakeholders :** these stakeholders can include diverse roles (e.g., architects, application engineers, project managers, requirements engineers).

PP2: **In immature or heterogeneous domains, unify the domain terminology**:
To facilitate the modeling process and model comprehension, it is beneficial to unify the terminology used by stakeholders and provide descriptive terms

PP3: **Define the purpose of the feature model**: this can be divided into two categories. First, >FM can support design and management of a product line, second, feature models can support the actual product-line development.

Feature-modeling principles

1. Planning and Preparation

PP4: Define criteria for feature to sub-feature decomposition. it can represent relations such as part-of, and functionality decomposition. Defining how a feature should be divided into sub-features facilitates achieving a consistent model that provides a single perspective on the product line

PP5: Plan feature modeling as an iterative process. FM should iteratively alternate between domain scoping and modeling.

PP6: Keep the number of modelers low. Industrial practice shows that the number of stakeholders performing the modeling should be low, in some cases a single person. only architects and project managers are involved in the modeling process.

Feature-modeling principles

2. Model Organization

MO1: The depth of the feature-model hierarchy should not exceed eight levels. While rarely made explicit in experience reports, survey papers and most of our interviewees report that the feature-model hierarchy is typically between three to six levels deep

MO2: Features at higher levels in the hierarchy should be more abstract. We found that the higher a feature is in the feature-model hierarchy, the more it is visible to the customers or it represents a more abstract domain-specific functionality.

MO3: Split large models Several sources state that large feature models with thousands of features should be decomposed into smaller ones.

Feature-modeling principles

2. Model Organization

MO4: Avoid complex cross-tree constraints. Cross-tree constraints allow adding dependencies between subtree of a feature model. However, complex constraints, typically in the form of arbitrary Boolean formulas, hamper comprehension, maintenance, and evolution of model.

MO5: Maximize cohesion and minimize coupling with feature groups. A high cohesion within a group and low coupling to other groups (absence of cross-tree constraints) indicates that the features belong together, which will also promote higher reusability.

Feature-modeling principles

3. Modeling

M1: Use workshops to extract domain knowledge. Workshops are used extensively to initiate feature modeling; they are the most efficient way to start.

M2: Focus first on identifying features that distinguish variants. it is easier for most stakeholders to describe the features that distinguish variants from each other rather than focusing on the commonalities

M3: Apply bottom-up modeling to identify differences between artifacts. Different artifacts can be analyzed to identify the differences between existing variants. Source code files are typically the first artifacts to be analyzed, and the analysis can be done automatically by different tools.

Feature-modeling principles

3. Modeling

M4: Apply top-down modeling to identify differences in the domain. For a top-down analysis, “Top-down is successful with domain experts, more abstract features.” The features that emerge from the top down analysis typically represent commonalities or abstract features that help with feature model structuring

M5: Use a combination of bottom-up and top-down modeling. Due to the different results that can emerge from bottom-up and top-down analyses (M2), it is highly recommended to combine both strategies.

M6: A feature typically represents a distinctive, functional abstraction. While some works use feature models to represent non-functional properties (e.g., performance requirements or physical properties, such as color, majority emphasize that features should represent functional abstractions .

Feature-modeling principles

4. Dependencies

D1: If the models are configured by (company) experts, avoid feature-dependency modeling. In real-world feature modeling, identifying dependencies is expensive and if explicitly modeled, the maintenance of the feature model becomes complex , if explicitly modeled, around 50% of features would be involved in dependencies.

D2: If the main users of a feature model are end-users, perform feature-dependency modeling. Considering end-users instead of domain experts. If feature model configuration is done by end-users, or with large Principles of Feature Modeling, then feature-dependency modeling should be performed.