

Product Line Software Engineering

Tsuneo Nakanishi

Fukuoka University

About This Material

- ▶ The material is originally written in Japanese.
- ▶ The material is not so updated in these years maybe thank to maturity of the SPL discipline.
- ▶ The first version was described in 2008 or around.
- ▶ The first version was used in open seminars for senior engineers of Fukuoka local companies in the QUBE Project by Kyushu University under sponsorship of the Japanese Ministry of Education.
- ▶ The open seminar was used for SPL adoption by Fujitsu Kyushu Network Technologies.
- ▶ The first version was also used in other open seminars held in Japan.
- ▶ The first version was used in seminars by System LSI College (predecessor of College of System Development Technology) operated by Fukuoka Industry, Science and Technology Foundation (Fukuoka IST).
- ▶ The copyright of the first version was handed over to Fukuoka IST.
- ▶ The material was renovated in 2016.
- ▶ The current version was used for SPL adoption by Aisin Seiki.
- ▶ The current version has been distributed by author's collaborating companies.

What's SPL?

Software Product Lines as Discipline

- ▶ 20+ years history from late 1990s
 - ▶ SPL was launched independently in US and Europe
 - ▶ CMU/SEI is a key institute in US.
 - ▶ Europe researched by private initiative (Three projects by ITEA: ESAPS, CAFÉ, and Families).
 - ▶ SPL was well developed during 2000s.
 - ▶ SPL seems to be almost matured theoretically.
- ▶ Not so many novel inventions
 - ▶ Existing knowledge of software engineering was redefined from the perspective of multiple product development including *development process, requirements engineering, analysis, desgin, components, modeling, architecture, testing, metrics, model driven development*, ...
- ▶ **Variability modeling** is the most distinctive technology in software product lines.

Sofware Product Lines as Technology

- ▶ Software product lines as software reuse technology
 - ▶ SPL is a technology for **planned reuse** of software assets.
 - ▶ SPL is not a technology producing unexpected products.
 - ▶ SPL is a technology to **reusability from upper processes**. (We construct artefacts with envisioning present and future products from beginning as much as possible.)
 - ▶ SPL aims **top-down (architecture centric)** reuse rather than bottom-up (component oriented) reuse.
 - ▶ SPL reuses not only concrete artefacts (components) but also **abstract artefacts (models)**.
 - ▶ SPL is a technology to define a reuse process.
 - ▶ SPL increases reusability of artefacts by **disciplined reuse**.
- ▶ Software product lines as software composition technology
 - ▶ SPL is a technology composing software based on description of high abstraction levels.
 - ▶ SPL increases reusability of descriptions by increasing their abstraction levels.

Popular Misunderstandings of SPL

- ▶ There is a concrete development process named as *SPL*.
- ▶ SPL is a development methodology that can deliver products that we have not intended.
- ▶ SPL is too academic to satisfy industrial need.
- ▶ SPL requires total change of the software development process established in the company.
- ▶ SPL must be applied to the entire process of software development.
- ▶ SPL requires engineers to learn many things for its practice.
- ▶ SPL is useless in companies doing contract development that are not responsible for specifications.

Definition of the Software Product Line

CMU/SEI's Definition

*SPL is a set of software-intensive systems **sharing a common, managed set of features** that satisfy the specific needs of **a particular market segment or mission** and that are developed from **a common set of core assets** in a prescribed way.* (Clements, 2001)

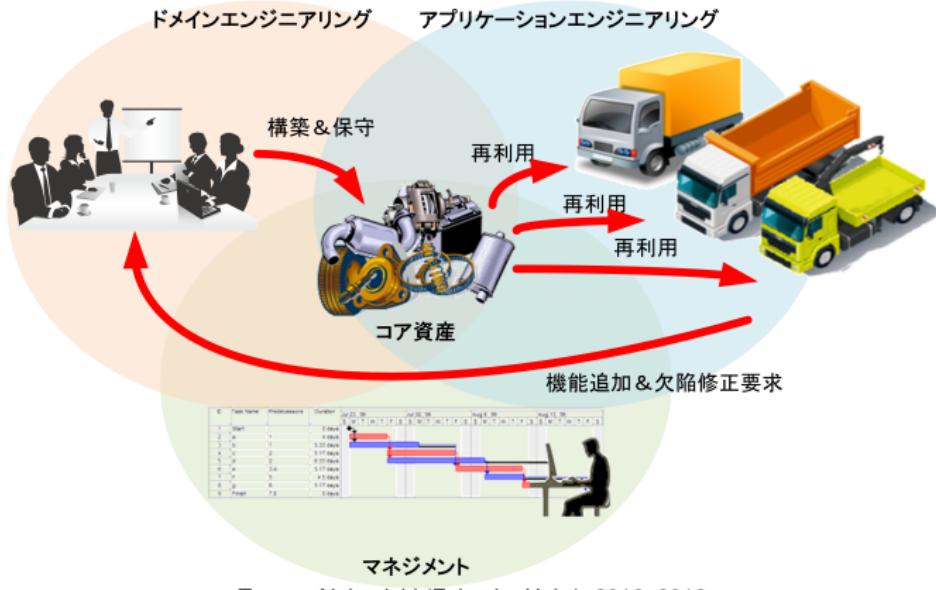
Key Points

- ▶ The product line is a set of products with common and managed set of features.
- ▶ The product line is a set of products satisfying needs of a particular market segment.
- ▶ The product line is a set of products produced from core assets by a prescribed way.

SPL as a Paradigm

SPL Paradigm: SPL is a **paradigm** that produce a set of products by comprehending **commonality and variability** among the products, constructing **core assets** shared by the products, and **reusing** the core assets in a defined process.

Three Essential Activities in SPL



SPL Paradigm

SPL is a paradigm rather than a technology!

Important! SPL Paradigm

- ▶ Separation of commonality and variability
- ▶ Architecture centric development
- ▶ Separation of domain engineering (core asset construction) and application engineering (core asset reuse)
- ▶ Separation of the problem space and the solution space
- ▶ Traceability from variabilities
- ▶ Controlled modification of variation points

It can be safely said that a development is SPL if the above-mentioned ideas are introduced.

SPL and Existing Reuse Technologies

SPL is not application of the simple adoption of the reuse technologies below:

- ▶ Library
- ▶ Component based development
- ▶ Reconfigurable architecture
- ▶ Derivative development from the closest product (Clone & Own)
- ▶ Technology standard

Core Assets

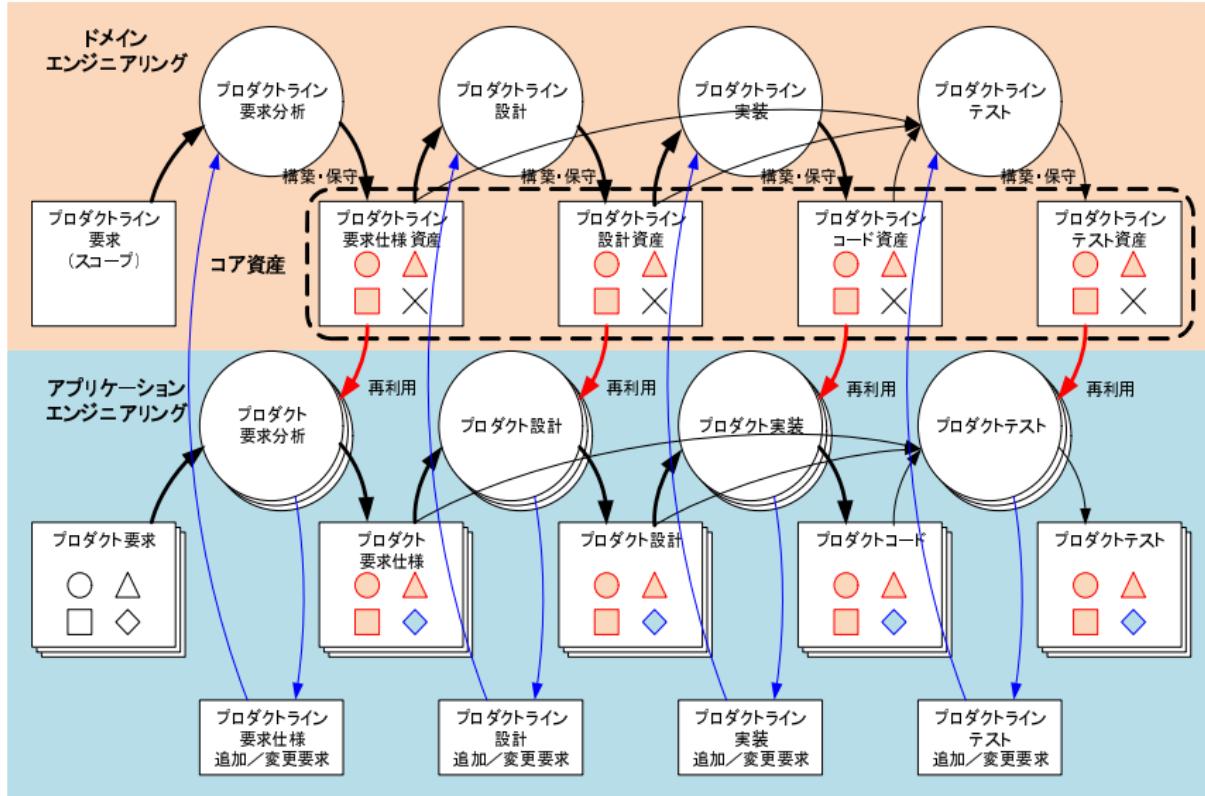
Core Assets: Artefacts shared among products of the product line under management and reused in the defined process.

Artefacts that Can Be Included in the Core Assets

- ▶ Software architecture
- ▶ Code assets (or components)
- ▶ Testing assets
- ▶ Analysis models
- ▶ Requirements-specifications
- ▶ Performance models
- ▶ Development schedule
- ▶ Metrics
- ▶ Schedule
- ▶ Process description etc.

Keep in mind that core assets **can include** various artefacts from various processes, from various viewpoints, and in various abstraction levels, but they must be managed with **maintenance cost**.

Process Model of SPL Development



Scoping (1)

Scoping: Activities to define what are included or not in the product line to maximize benefits of SPL development.

- ▶ No defined scope: Unsteady product line development; Broken architecture
- ▶ Broadly defined scope: Excessive development and maintenance cost of core assets; Excessive software scale, depressed performance, and costly hardware requirements due to excessive abstraction
- ▶ Narrowly defined scope: Limited derivation of products; Endless additional requirements to the product line
- ▶ Miss defined scope: Limited derivation of products; Fruitless core assets

Scoping (2)

Means of Scoping

- ▶ Envisioning a product road map
- ▶ Surveying existing products, customers, industry trends, and intellectual properties
- ▶ Constructing a product-attribute matrix
- ▶ Conducting feature modeling
- ▶ Having a session

Scoping (3)

DOs in Scoping

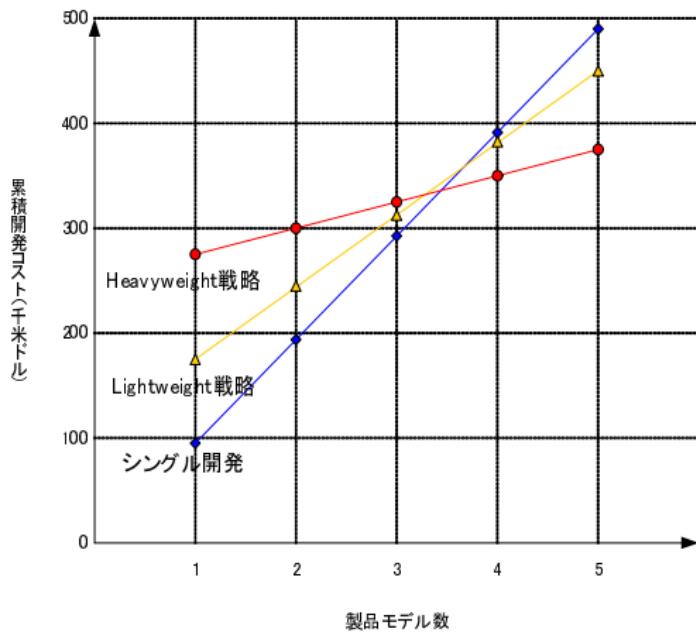
- ▶ Scoping should be conducted with various stakeholders.
 - ▶ Engineering
 - ▶ Marketing
 - ▶ Intellectual Property Department
 - ▶ Management
 - ▶ Customer
- ▶ *as-is* and *to-be* of the scope should be reviewed periodically.

Approaches for Migration to SPL Development (1)

- ▶ **Proactive Approach:** Product line development from initiation of the product line
 - ▶ Defines the scope of the product line, constructs core assets including a software architecture, and derive product variants.
 - ▶ Requires higher initiation cost and lower product derivation cost
 - ▶ Can adopt if we will have a limited product variants or can envision future products well
- ▶ **Reactive Approach:** Product line development after releases of some product variants
 - ▶ Separates common and variable parts of the artefacts of existing products, mines core assets, defines a software architecture, and migrates to product line development
 - ▶ Migrates to product line development gradually when we develop new product variants
 - ▶ Requires lower initiation cost and higher product derivation cost
 - ▶ Can adopt as a development process improvement activity
 - ▶ Can adopt if we hold many existing products with many artefacts or cannot envision future products well

Approaches for Migration to SPL Development (2)

Cost Model of Migration Strategies (McGregor, 2002)



It is empirically demonstrated that SPL pays off at the third product derivation.

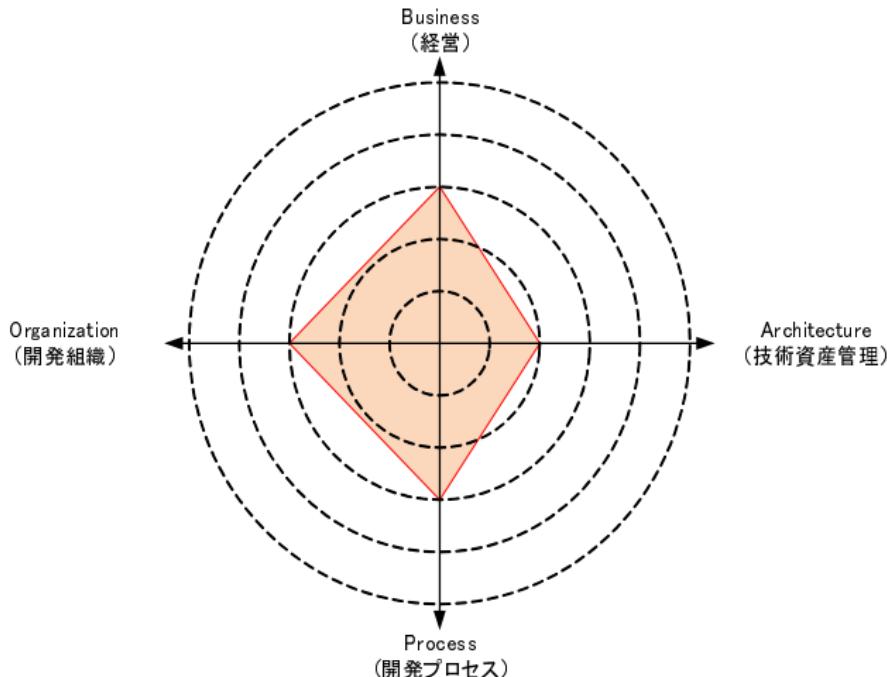
BAPO Model

BAPO Model: Four concerns that should be addressed for adoption and sustention of SPL. (Linden, 2007)

- **Business:** Business and profit affairs on SPL development
- **Architecture:** Technical means to build the software
- **Process:** Process affairs to build the software
- **Organization:** People and organizational affairs to build the software

FEF: Family Evaluation Framework

Family Evaluation Framework: A *report card* to evaluate organizational maturity of SPL development. The maturity is evaluated from four concerns of the BAPO model in five-grade.



Commonality and Variability

Commonality among Products

SPL begins with identification, naming, and description of variability.

Temporal and Spatial Variabilities

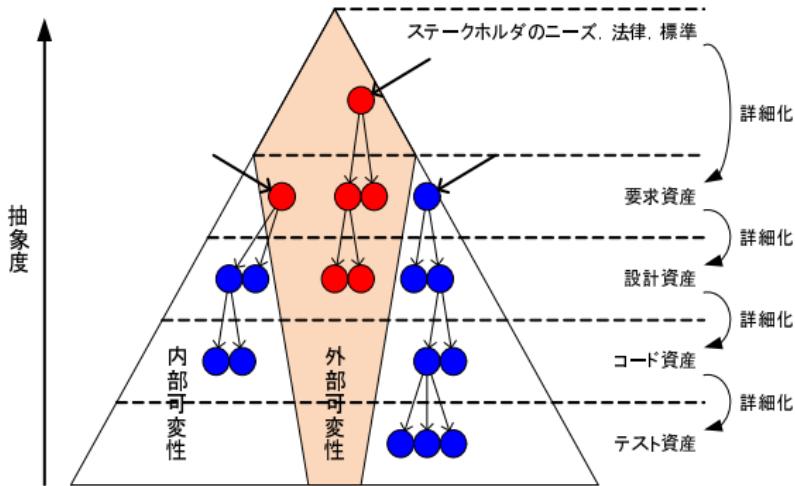
- ▶ **Temporal Variability:** Variability among products released at different times; namely, evolution of the product line.
- ▶ **Spatial Variability:** Variability among products released at the same time

External and Internal Variabilities

- ▶ **External Variability:** Variabilities visible to customers
- ▶ **Internal Variability:** Variabilities invisible to customers

Variability Pyramid

Variability Pyramid: A model representing abstraction levels, volume, and proportion external to internal variabilities.



- ▶ There exist dependences among variabilities across abstraction levels.
- ▶ There exist more variabilities in artefacts of later sub processes, which represent variabilities relating to design and implementation decisions.

Commonality/Variability Analysis (1)

Commonality/Variability Analysis: Separately representing common and variable aspects of products in the product line in functions, structures, behaviors, non-functional characteristics, etc.

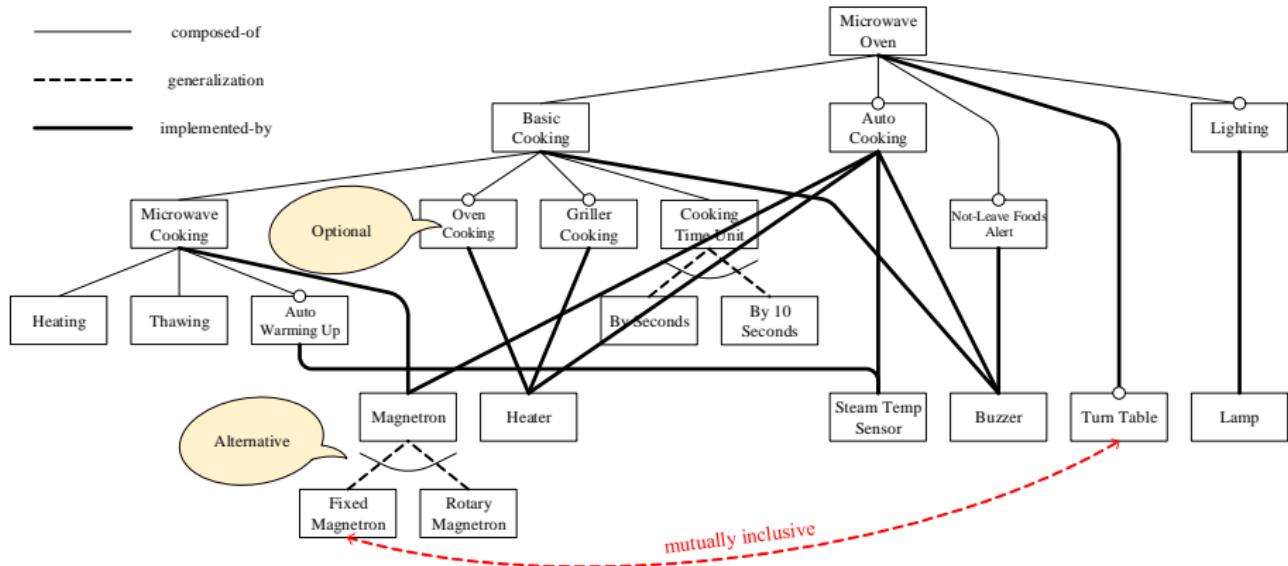
Feature Model: A tree-form model capturing commonality and variability of products in the product line as **features** and representing relationship among them. (Kang, 1990; Kang 1998)

What the Feature Model Represents

- ▶ Name
- ▶ Abstraction Level
- ▶ Semantic Relationship among Features: composed-of, generalization, implemented-by
- ▶ Constraints on Feature Selection: mandatory, optional, alternative, and others

Commonality/Variability Analysis (2)

Example Feature Model: Microwave Oven Product Line



Commonality/Variability Analysis (3)

Semantic Parent-Child Relationships between Features

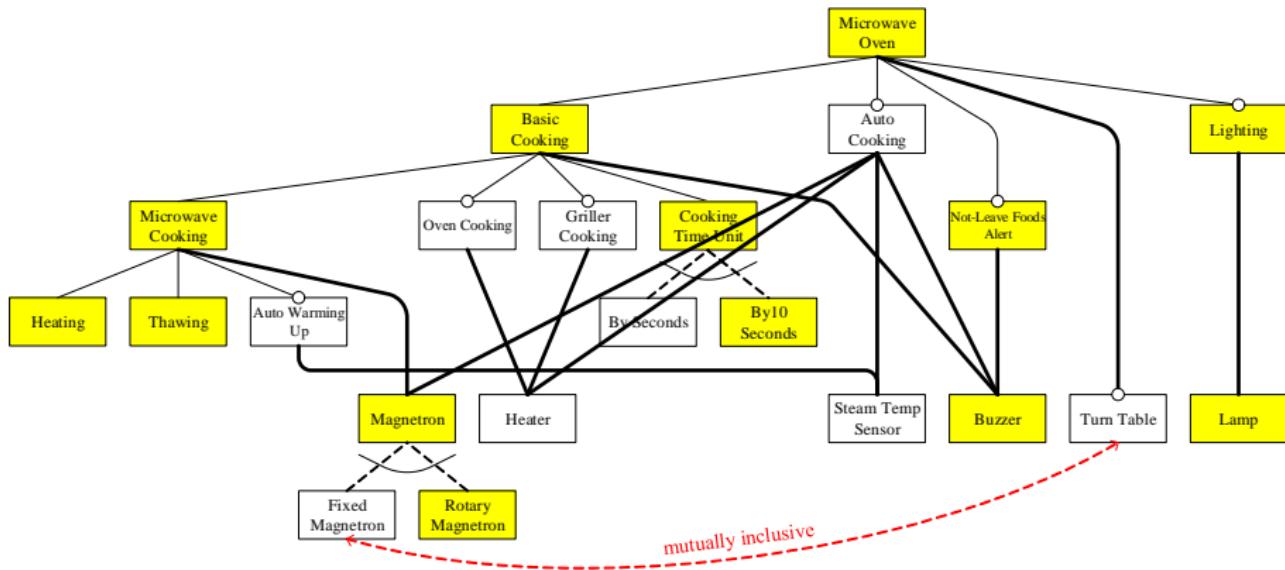
- ▶ composed-of
- ▶ generalization
- ▶ implemented-by

Feature Selection Constraints

- ▶ **Mandatory:** Feature that must be selected (if its parent feature is selected)
- ▶ **Optional:** Feature that can be selected optionally (if its parent feature is selected)
- ▶ **Alternative:** Set of features such that one of them can be selected alternatively (if their common parent feature is selected)

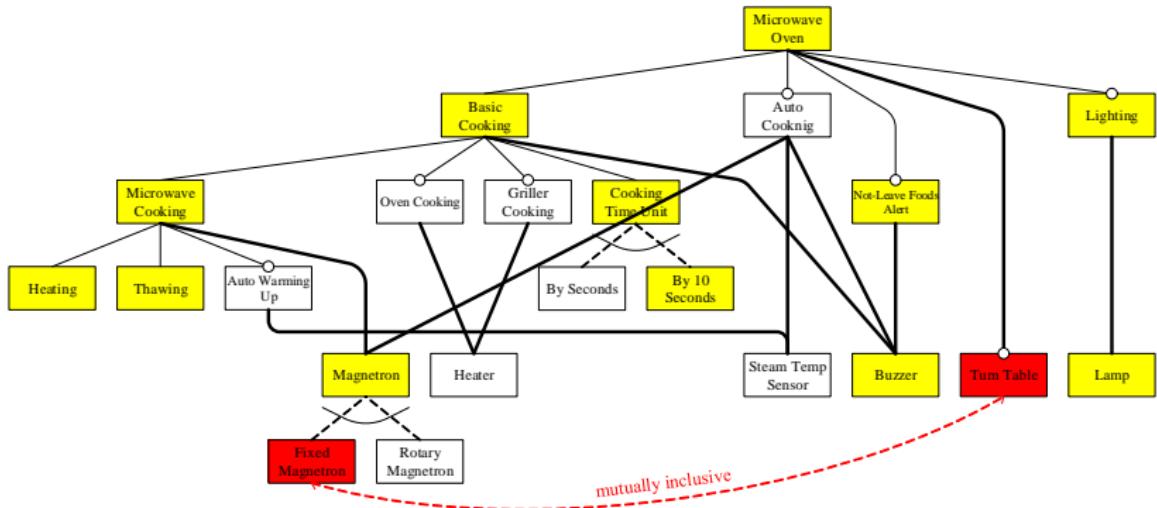
Commonality/Variability Analysis (4)

Product Representation: A valid selection of the features on the feature model represents a single product in the product line.



Commonality/Variability Analysis (5)

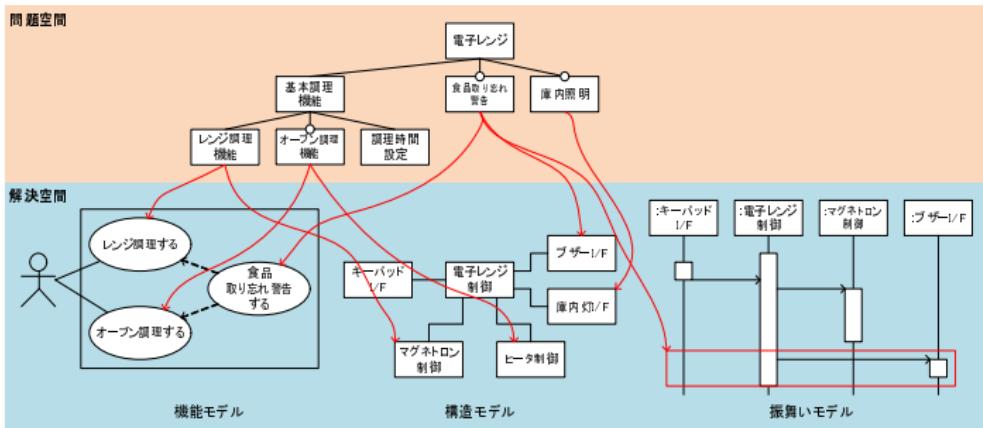
Dependency on Feature Selection: Constraints on feature selection between features that are not in semantic parent-child relationship are represented by dependence arcs in the feature model.



In the above example, *Fixed Magnetron* and *Turn Table* must be selected mutually inclusively.

Separation of Problem and Solution Spaces

- ▶ **Problem Space:** A document space that stores artefacts representing variability among products.
- ▶ **Solution Space:** A document space that stores artefacts contributing to composition of products.



Variability can be used as indices for artefacts on composition in different abstraction levels by traceability from the problem space to the solution space.

Realizing Variabilities

Ways of Variability Realization (Anastasopoulos, 2001; Gomaa, 2004)

- ▶ Conditional Compilation
- ▶ Parameters
- ▶ Information Hiding
 - ▶ Static Libraries
 - ▶ Dynamic Libraries
 - ▶ Dynamic Class Loading
 - ▶ Overload
- ▶ Inheritance
- ▶ Aspects

Feature binding times are restricted depending on the way of variability realization.

Feature Binding Time: Time that the feature is activated or deactivated. That can be compile-time, link-time, load-time, run-time, etc.

Traceability from Variabilities

Ways of Guaranteeing Traceability

- ▶ Tag
- ▶ Traceability Matrix
- ▶ Attached Process

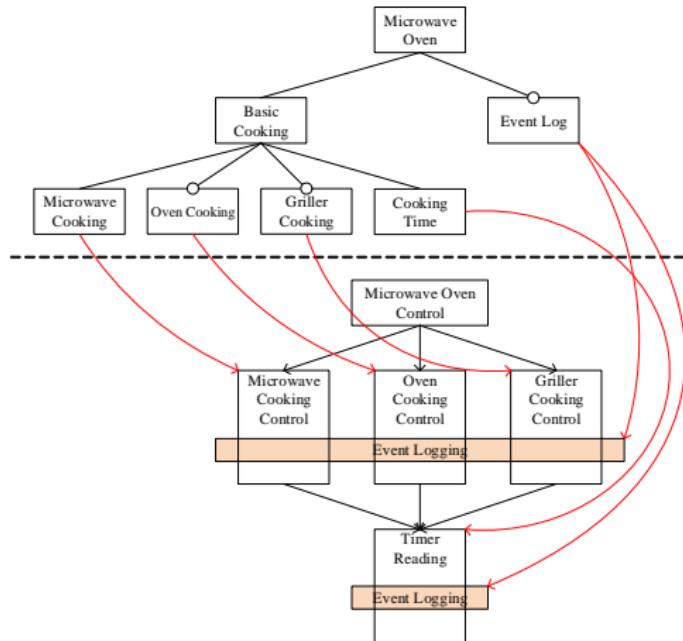
Balancing Traceability and Granularity

- ▶ Excessively fine traceability (to the statement, for example) increases description and maintenance cost for tags and traceability matrices.
- ▶ Coarse traceability (to the file, for example) is allowed if smooth product derivation at acceptable quality, cost, and delivery time is guaranteed.
- ▶ Fine traceability is not indispensable unless we do automatic product derivation.
- ▶ It is possible to have minimum traceability at the granularity that a single engineer can manage and control product derivation by the attached process.

Objects Implementing Functional Features

Objects Implementing Functional Features

- ▶ Modules
- ▶ Aspects



Handling Non-Functional Features

Examples of Handling Non-Functional Features

- ▶ Non-Functional Features Realized by Functions : Establish traceability to functional features that can realize a non-functional feature based on the result of Quality Function Deployment (QFD).
- ▶ Non-Functional Features Realized by Processes: Establish traceability to the processes in the process description that can realize a non-functional feature.

Feature Modeling

Objectives of Feature Modeling

- ▶ Describing variability among products
- ▶ Improving communication among stakeholders
 - ▶ Communication within the project (especially between domain and application engineering teams)
 - ▶ Communication with management and business sectors
 - ▶ Communication with customers
- ▶ Planning the scope of the product line
 - ▶ Planning the range to where SPL development is applied
 - ▶ Finding new functions
- ▶ Capturing changeable requirements
- ▶ Pre-designing the architecture
 - ▶ Abstraction
 - ▶ Separation of concerns
 - ▶ Interface design

Process of Feature Modeling

Process of Feature Modeling (Czarnecki, 2000)

1. Identify features from various sources including stakeholders, domain experts, existing systems, etc.
2. Organize features in a hierarchical form
3. Analyze constraints on feature selection
4. Describe supplemental information on features

Iterate the above processes and improve the feature model continuously until its maturity

Points to Remember

- ▶ Review the feature model every moment in the development process (since refinement may increase variability)
- ▶ Include not only present features but also possible features in future
- ▶ Describe features in a little bigger scope than the scope of actual development

Finding Feature Candidates

Sources of Feature Candidates (Czarnecki, 2000)

- ▶ Domain experts and documents
- ▶ Present and future stakeholders
- ▶ Documents for existing products

Points to Remember

- ▶ Think variability first than commonality
- ▶ Do divide-and-conquer if you have too many products (that is, divide the products to groups and think variability within a group one by one)

Organizing Features (1)

Organizing Features in a Tree Form

- ▶ Lay features in the order corresponding to their abstraction levels
- ▶ Define semantic parent-child relationship
 - ▶ Generalization: Make features with a common concept of the same abstraction level and granularity be brothers and define their parent feature corresponding to their maximum common concept.
 - ▶ Composed-of: Make a feature corresponding to a part of the concept be a child.
 - ▶ Implemented-by: Make a feature corresponding to a mean be a child.
- ▶ Think opposing concepts (in-out, open-close, continuous-discrete etc. or same axis concepts (visual-aural-tactile etc. to find brother features and increase coverage.

Organizing Features (2)

Points to Remember

- ▶ Do not think details in where there is no variability
- ▶ Put abstract features into shape until you find no variability
- ▶ Ask yourself “for what” to find super features
- ▶ Ask yourself “how to” to find sub features

Organizing Features (3)

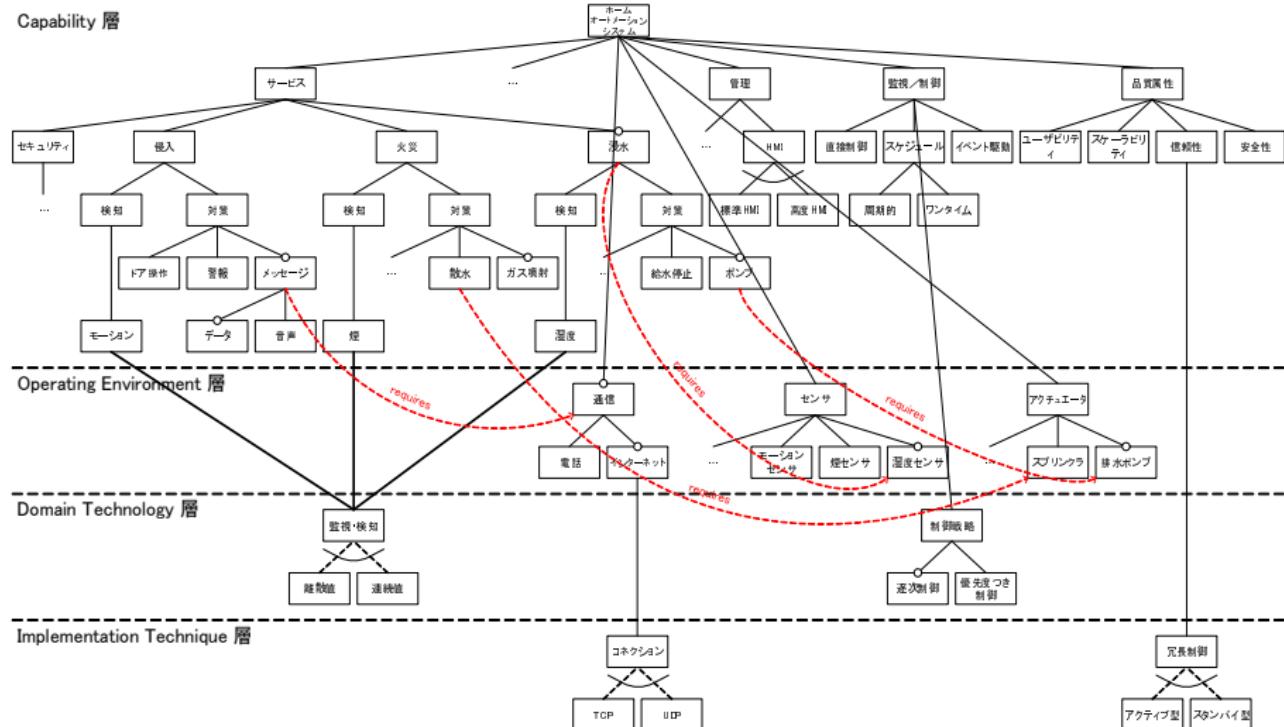
FORM's Feature Layer Model (Kang, 2002; Lee, 2002)

Capability	プロダクトラインが提供するサービス面のちがい	
	サービス	システムサービス
	オペレーション	システムオペレーション
	非機能特性	品質属性, コスト, 見栄え, 制約
Operating Environment	実行環境面のちがい	
	SW/HW インターフェース	ソフトウェア API, デバイスドライバ
	SW/HW プラットフォーム	ミドルウェア, OS, プロセッサ
Domain Technology	ドメイン固有のちがい	
	ドメイン固有手法	ドメイン理論, 法律, 標準, 推奨
Implementation Technique	ドメイン非依存の実装面のちがい	
	設計決定	アーキテクチャスタイル, プロセス分割
	実装決定	アルゴリズム, 通信プロトコル, 実現手法

Organizing Features (4)

Example Feature Model Based on FORM's Model: Home Automation System (Kang, 2002)

Capability 層

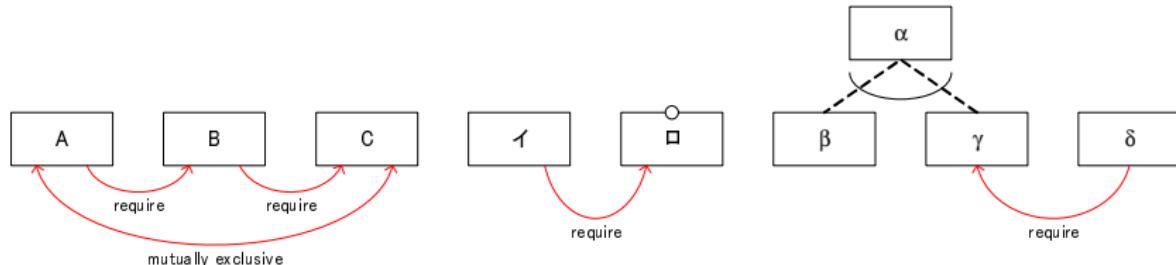


Describing Constraints on Feature Selection (1)

- ▶ Describe reuse categories for brother features (mandatory, optional, alternative)
- ▶ Describe dependency of feature selection by arcs for non-brother features
 - ▶ **requires**: The sink feature must be selected if the source feature is selected.
 - ▶ **mutually exclusive**: The source and sink features must not be selected at the same time.
 - ▶ **mutually inclusive**: The source and sink features must be selected and unselected together.
- ▶ Complicated dependency can be described logical expressions, mathematical expressions, natural languages, and so on.
(Composition Rules)

Describing Constraints on Feature Selection (2)

Anomaly of the Feature Model (Maßen, 2004)



- ▶ Sometimes appear in the complicated feature model
- ▶ Examine yourself whether you recognize features with ambiguity if you find anomaly of the featur model
- ▶ Anomaly in the feature model should be detected by the modeling tool

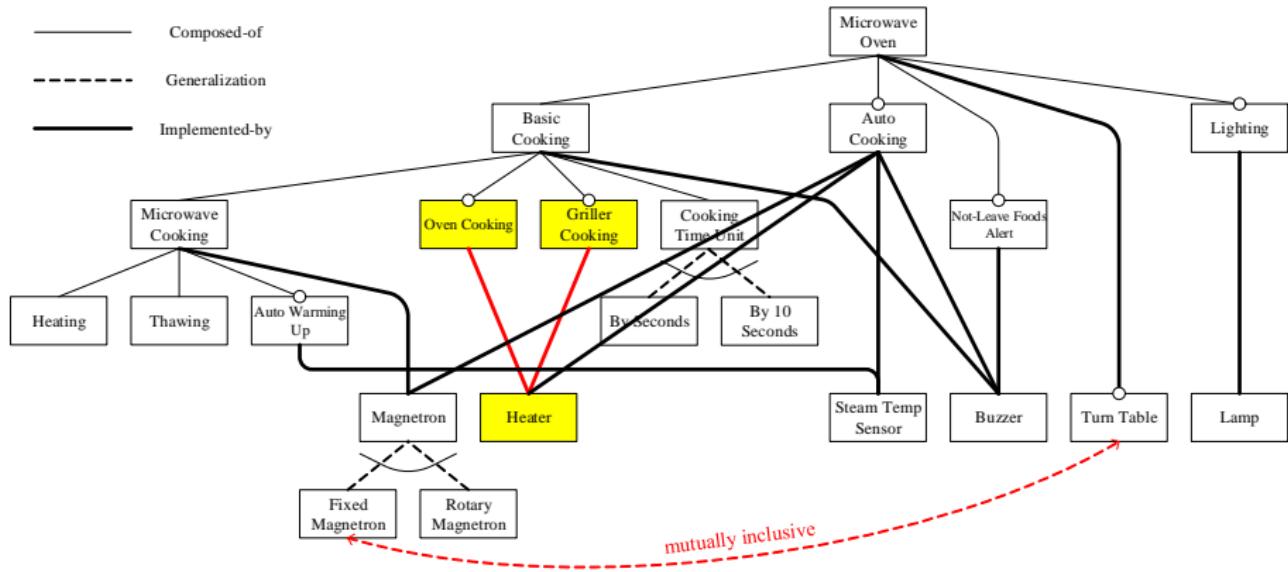
Describing Supplemental Information on Features

Supplemental Information of Features (Czarnecki, 2000)

- ▶ Meaning
- ▶ Categories: Functional/Non-Functional, Functional Category, Non-Functional Characteristics (Performance, Cost, etc.)
- ▶ Stakeholders
- ▶ Open/Closed
 - ▶ Opened Features: Sub features are allowed.
 - ▶ Closed Features: Sub features are not allowed.
- ▶ Binding Time: Timing when the feature is activated (compile-time, installation-time, run-time, etc.)
- ▶ Priority of Realization

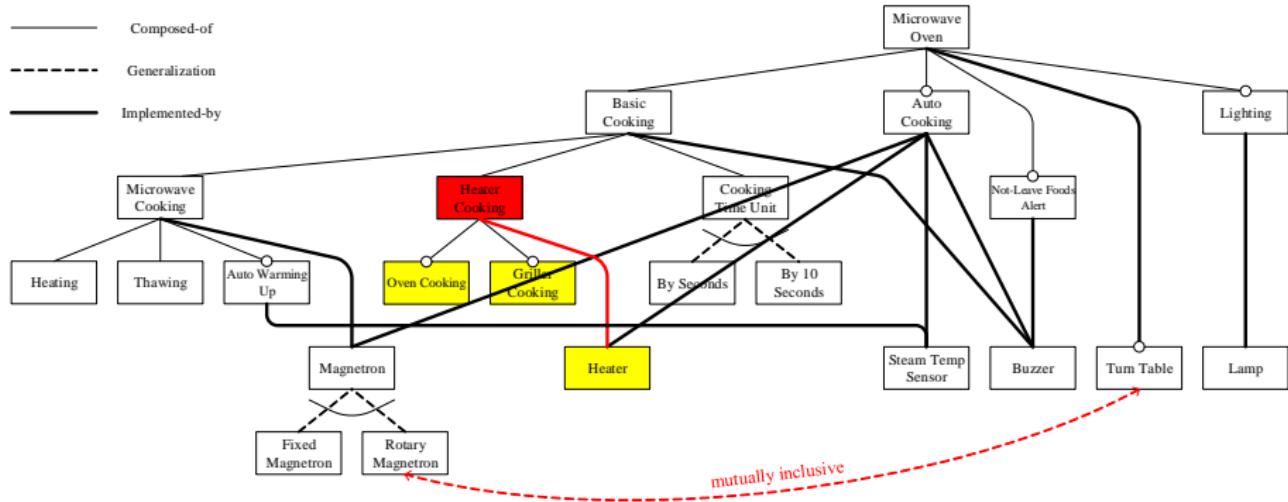
Improving the Feature Model (1)

Deriving intermediate concepts by abstraction (before)



Improving the Feature Model (2)

Deriving intermediate concepts by abstraction (after)



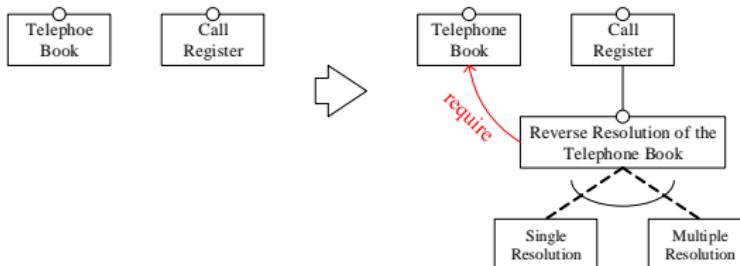
Possible Effects

- ▶ Abstracted modules, Derivation of common interface
- ▶ Loosely coupled modules
- ▶ Simplification of the feature model

Improving the Feature Model (3)

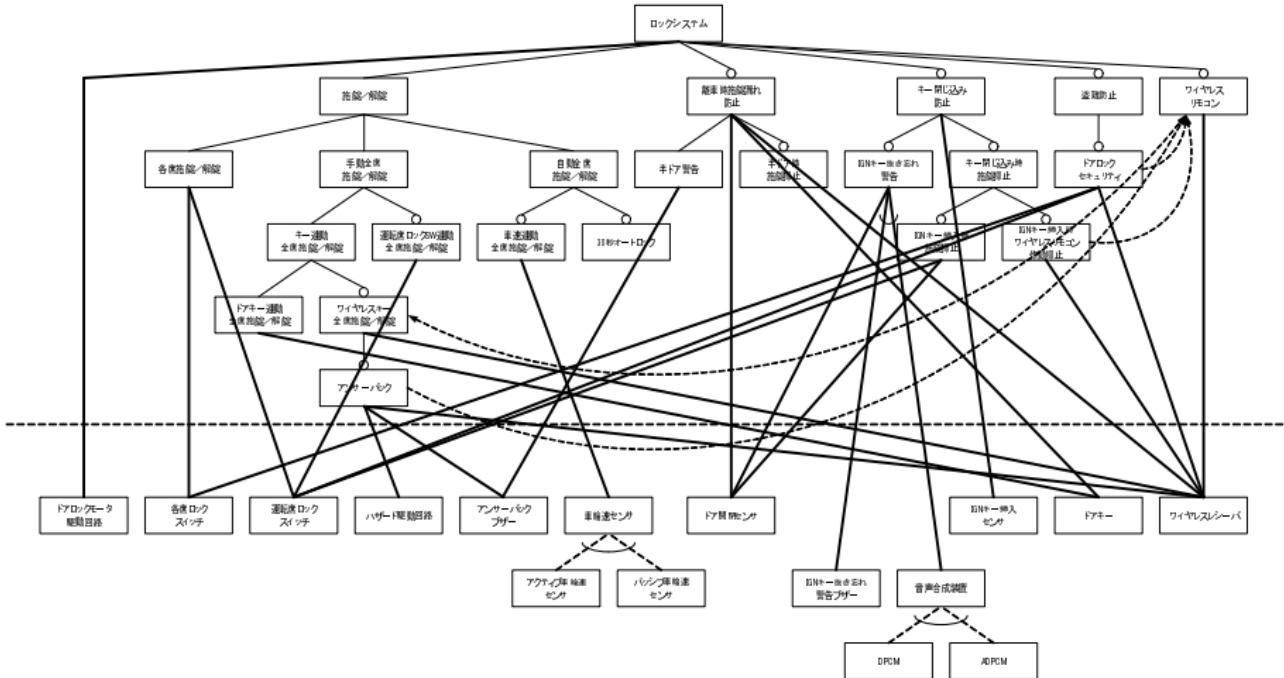
Separating a partial function interferred by another function

- ▶ Telephone Book: Register names and zero or more phone numbers of individuals. One can specify the recipient of the call by his/her name, instead his/her the phone number.
- ▶ Call Register: Display the phone numbers of the received calls in the history.
 - ▶ Some telephones with Telephone Book displays names, instead of phone numbers, in the history for persons registered in the telephone book.
 - ▶ If two or more persons are registered for a phone number in the telephone book, the telephone displays either one name of them or all the names of them in the history.



Example Feature Model

Automotive Door Lock



Ref.) CQ Publishing, *Interface* (『インターフェース』), Feb. 2008, pp.104–105. (Figure by the author)

Why are we attracted by the feature model?

Charm of the Feature Model

- ▶ Well drawn feature model **brushes away cobwebs** on the vision of the system.
- ▶ The feature model lets us feel **good for something**.
- ▶ However, we are often caught in cobwebs when we construct a feature model.

What we are doing in feature modeling: We do something that software engineers will do on models or codes somewhere at higher abstraction levels in advance.

- ▶ Cobwebs → abstraction, separation of concerns
- ▶ Good for something → system decomposition, interface design, etc.

The feature model shows a concise view of the system to be refined into structures and behaviors.

Predesigning the Architecture

Refinement to Solution Space Models from the Feature Model

Functional Features	Coarse Granularity	Function: use cases; Structure: sub systems, classes (themselves, operations, attributes), processes, modules, data stores, aspects; Behavior: whole the sequence diagram/communication diagram/state chart
	Fine Granularity	Structure: class (operations, attributes), conditional compile, macros; Behavior: portion of the use case description, portion of the behavioral model, operations of the class, portion of the conditional compilation, macros
Non-Functional Features	Process description; Task structures; (Dependency to functional features)	
Quantitative Features	Parameters	
Generalization	Generalization of classes, common interfaces	
Composed-of	Associations, aggregation, or composition of classes; Module invocations	

Anti-Patterns of Feature Modeling

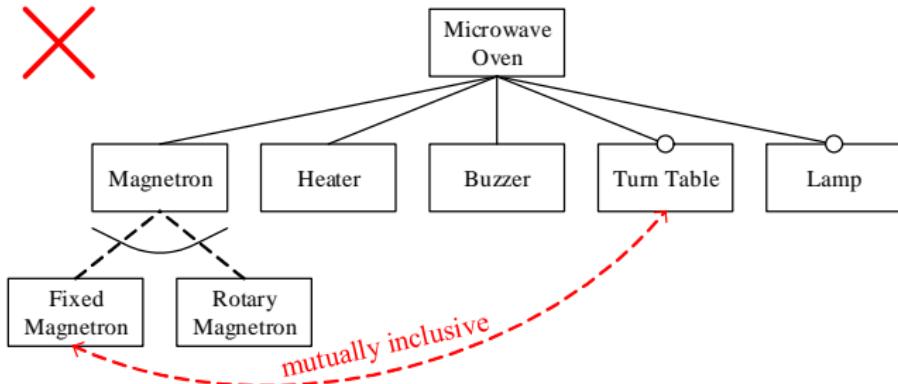
Anti-Patterns of Feature Modeling

Anti-patterns to achieve separation of abstraction levels and separation of concerns

- Representing only the physical structure of the product line
- Turning ends and means upside down
- Separating variability insufficiently
- Miss-recognizing an instance of configuration as a feature
- Confusing product variability and run-time variability
- Mismatching conceptual hierarchy
- Representing module invocation rather than variability

Representing Only the Physical Structure of the Product Line

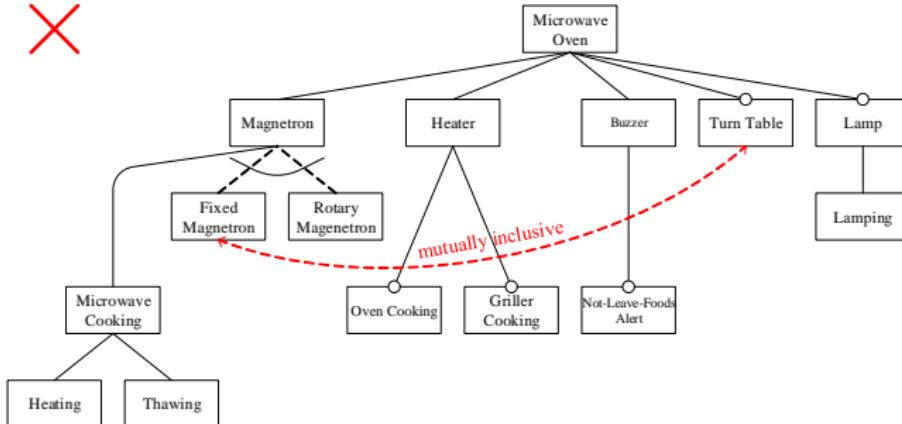
Microwave Oven Example



- ▶ Variability of the physical structure can be easily identified.
- ▶ Feature of the physical structure should appear in lower layers of the feature model.
- ▶ Think the purpose of the physical feature to identify a service feature becoming its super feature.

Turning Ends and Means Upside Down

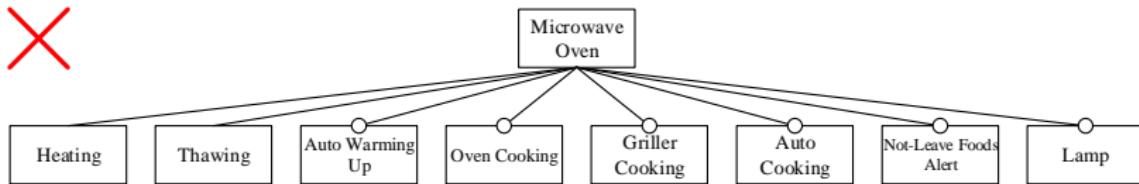
Microwave Oven Example



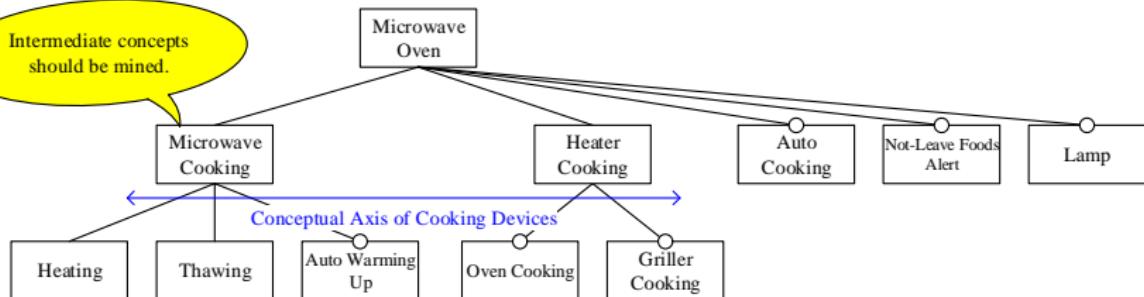
- Variability of services should be represented in upper layers of the feature model.
- Think means for realization to identify sub features for the service feature.
- Think the purpose to identify a super feature for the physical device feature.

Separating Variability Insufficiently

Cellular Phone Example



Intermediate concepts
should be mined.



- ▶ Limit the number of brother features.
- ▶ Align granularity and *conceptual axis* of brother features.

Miss-Recognizing an Instance of Configuration as a Feature (1)

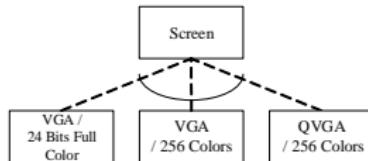
Car Navigation System Example

- ▶ A car navigation product line of different display capability
 - ▶ VGA (640x480), 24 Bits Full Colors
 - ▶ VGA (640x480), 256 Colors out of 24 Bits Colors
 - ▶ QVGA (320x240), 256 Colors out of 24 Bits Colors
- ▶ The products change visuals depending on display capability for legibility.
 - ▶ Widget layout, font size, and icon size are changed depending on display resolution
 - ▶ Icon colors and backgrounds are changed depending on color capability

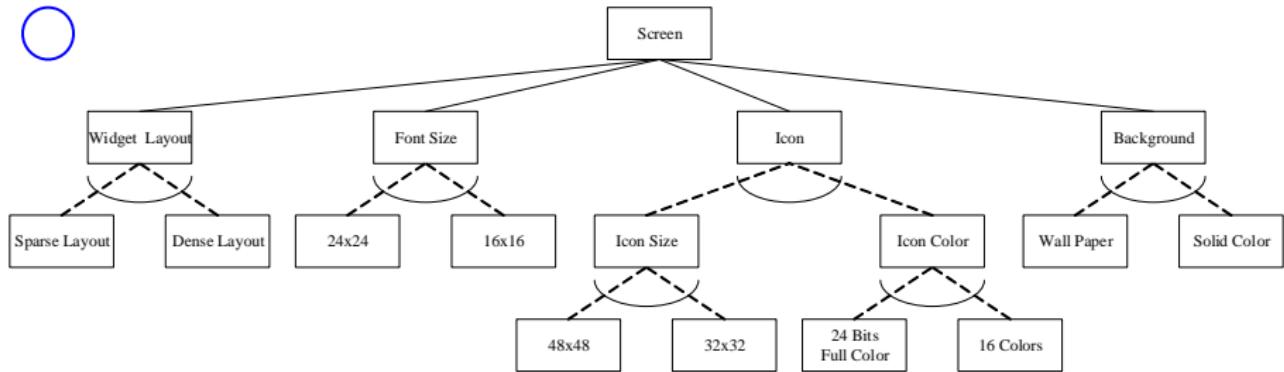
Items	VGA, 24 bits colors	VGA, 256 colors	QVGA, 256 colors
Widget Layout	Sparse Layout	Sparse Layout	Dense Layout
Font Size	24x24	24x24	16x16
Icon Size	48x48	48x48	32x32
Icon Colors	Full Colors	16 Colors	16 Colors
Background	Wall Paper	Solid Color	Solid Color

Miss-Recognizing an Instance of Configuration as a Feature (2)

✗



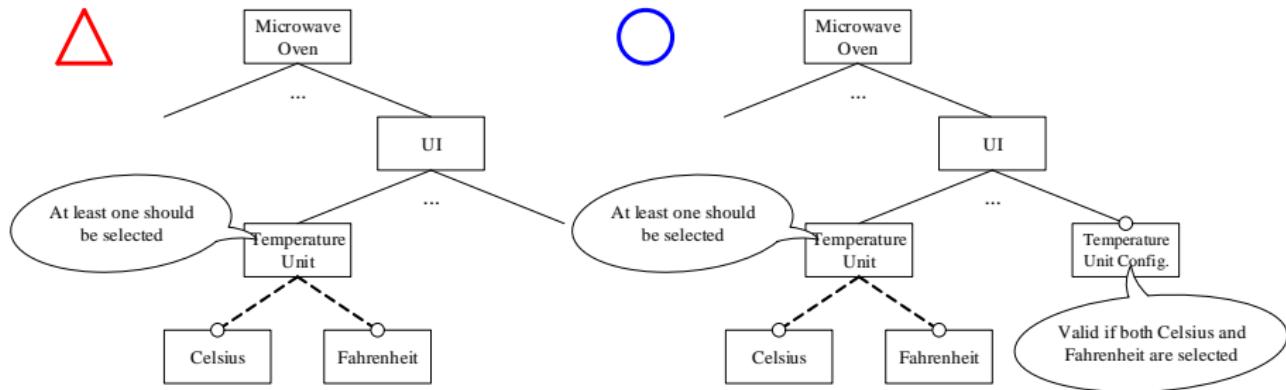
○



Confusing Product Variability and Run-Time Variability

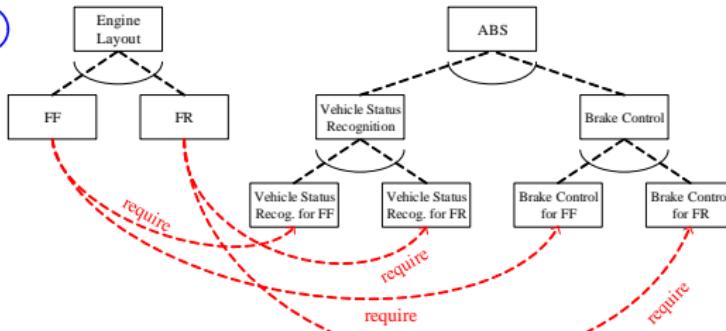
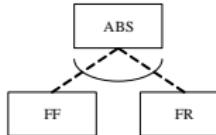
Microwave Oven Example

- ▶ A microwave oven product line with three variants
 - ▶ Displaying temperature in Celsius
 - ▶ Displaying temperature in Fahrenheit
 - ▶ Displaying temeprature in Celsius or Fahrenheit as configured by the user
- ▶ The left model does not represent the third variant well.



Mismatching Conceptual Hierarchy

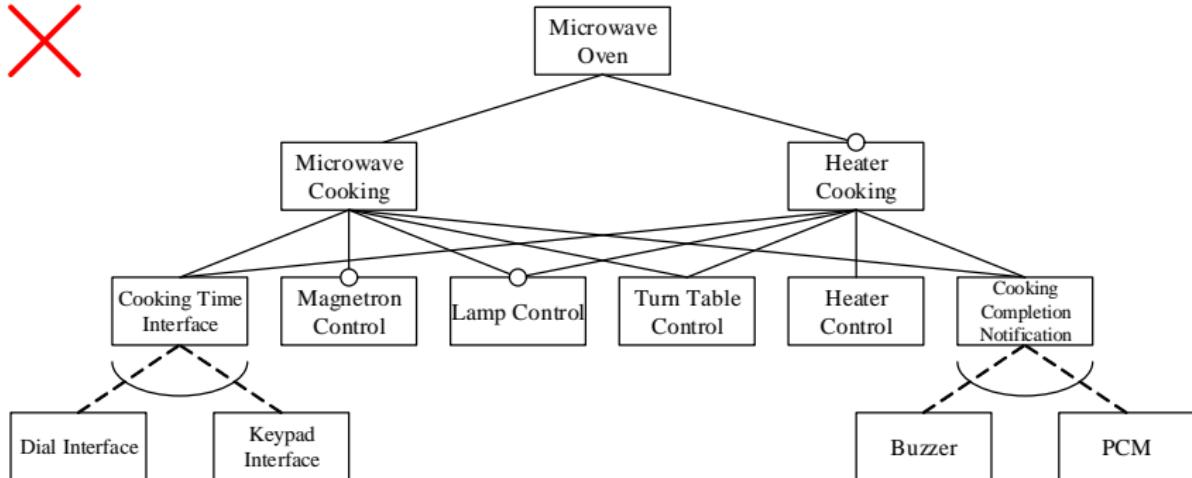
Microwave Oven Example



- ▶ The parent-child relationship between features is either composed-of, generalization, or implemented-by.
- ▶ The feature name may be improper.

Representing Module Invocation Rather than Variability

Example



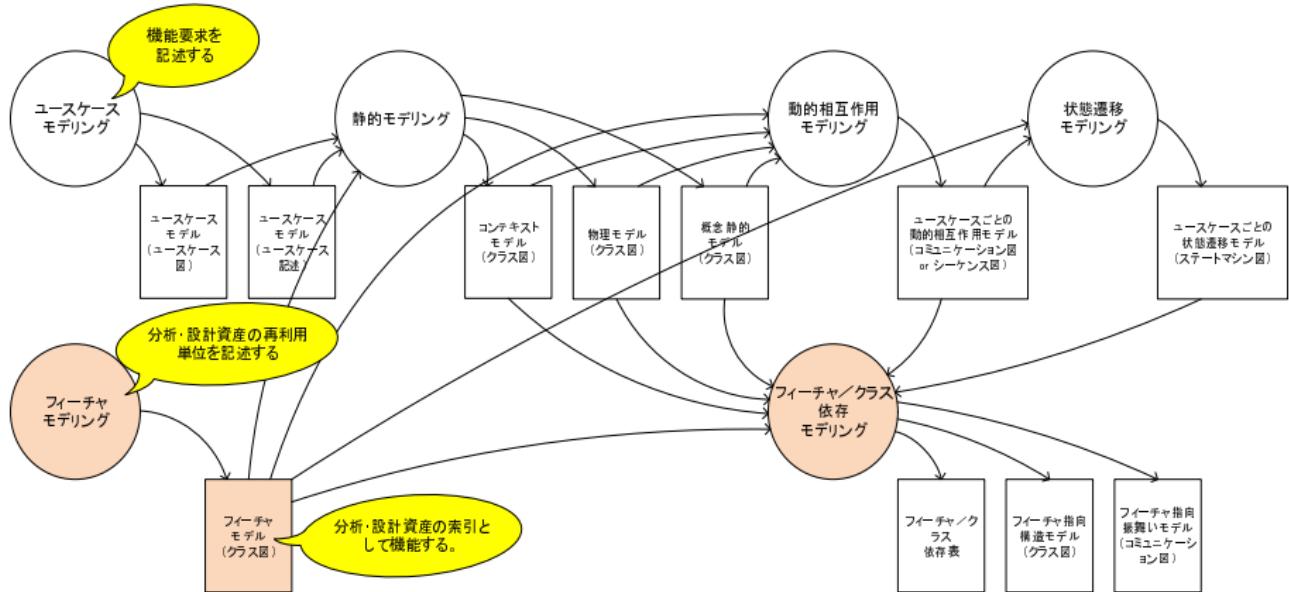
- ▶ Be careful about this anti-pattern when we construct a feature model from existing codes.
- ▶ The analyst may confuse the feature model and the structured chart used in structured design.

Example Development Process

Overview of PLUS

- ▶ A product line development process proposed by Hassan Gomaa (George Mason Univ.)
- ▶ An object-oriented process
- ▶ Describes most of analysis/design artefacts in **UML**
- ▶ Gives a Well-defined process is given

PLUS Domain Engineering Process



Modeling of Variabilities in the Use Case Model

プロダクトライン構築では、プロダクトラインのメンバによって、異なった扱いを受ける可変性を記述する必要がある。

Description of Variability

- ▶ Finer Variability: 単独のユースケースの内部で可変性をモデル化する。
- ▶ Coarser Variability
 - ▶ 単独のユースケース内でモデル化するとわかりにくい。
 - ▶ ユースケース全体の抜き挿しで可変性をモデル化する。
 - ▶ 相違部を別のユースケースとし、ユースケース間の依存関係で可変性をモデル化する。
 - ▶ extend 関係
 - ▶ include 関係

Modeling Coarser Variabilities in the Use Case Model (1)

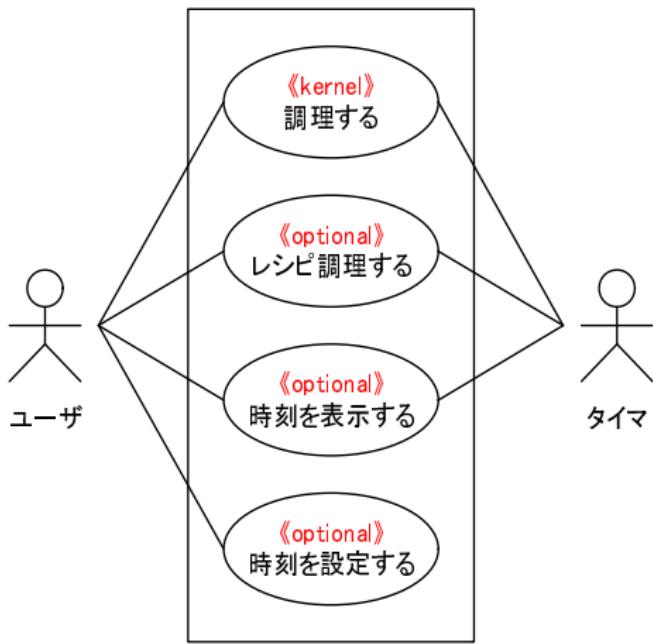
プロダクトラインの各製品は、全てのユースケース（＝システムの外部要件）を実現するわけではない。

Reuse Category of the Use Case: プロダクトラインの各製品でどう再利用されるかによって、プロダクトラインのユースケースを以下のようにカテゴリ化する。

- ▶ **Kernel Use Case:** プロダクトラインの全てのメンバで実現するユースケース。ステレオタイプ《kernel》で表現。
- ▶ **Optional Use Case:** プロダクトラインの一部のメンバでのみ実現するユースケース。ステレオタイプ《optional》で表現。
- ▶ **Alternative Use Case:** プロダクトラインのメンバによって択一的に実現する一群のユースケース。ステレオタイプ《alternative》で表現。

Modeling Coarser Variabilities in the Use Case Model (2)

Use Case Model Example: Microwave Oven Product Line (Gomaa, 2005)



Modeling Finer Variabilities in the Use Case Model (1)

ユースケース上の小さな相違部を、ユースケース内部の「変化点」として扱う。

Variation Point: ユースケースにおいて何かしらの変化が起こる場所。

Description of Variation Points: おおよそ下記のようなことを書けばよい。

- ▶ Name: 変化点の名前。
- ▶ Reuse Category: プロダクトライン中の製品ごとに変化点がどのように有効化されるか。*optional*, *mandatory alternative*, *optional alternative* のいずれか。
- ▶ Line Numbers: 変化点が有効化されるユースケース記述中の行の番号。
- ▶ Function: 変化点で有効化される機能。

Modeling Finer Variabilities in the Use Case Model

(2)

Use Case Description Example: Microwave Oven Product Line
(Gomaa, 2005)

Name: 調理する

Reuse Category: Kernel

Summary: ユーザは電子レンジに食べ物を置き、電子レンジは食べ物を調理する。

Actors: ユーザ (primary), タイマ (secondary)

Pre Conditions: 電子レンジは停止中であること。

Main Sequence

1. ユーザはドアを開け、食べ物を電子レンジに置き、ドアを閉じる。
2. ユーザは調理時間ボタンを押す。
3. システムは調理時間を表示する。
4. ユーザは調理時間をテンキーで入力し、開始ボタンを押す。
5. システムは調理を開始する。
6. システムは残り調理時間を連続的に表示する。
7. タイマは調理時間経過をシステムに通知する。
8. システムは調理を停止し、終了メッセージを表示する。
9. ユーザはドアを開け、食べ物を電子レンジから取り出し、ドアを閉める。
10. システムは表示をクリアする。

Modeling Finer Variabilities in the Use Case Model

(3)

Use Case Description Example: Microwave Oven Product Line (cnt'd)
(Gomaa, 2005)

(前頁より)

Alternative Sequences:

Line 1: ドアが開いているときに、ユーザが開始ボタンを押した。システムは調理を開始しない。

Line 4: ユーザはドアが閉まっていて、かつ電子レンジが空のときに、開始ボタンを押した。システムは調理を開始しない。

Line 4: ユーザはドアが閉まっていて、かつ調理時間が 0 のときに、開始ボタンを押した。システムは調理を開始しない。

Line 6: ユーザは調理中にドアを開けた。システムは調理を停止する。ユーザは食べ物を取り出し、中止ボタンを押す。もしくは、ユーザはドアを閉じ、調理を再開するために開始ボタンを押す。

Line 6: ユーザは調理中に中止ボタンを押す。システムは調理を停止する。ユーザは調理を再開するために開始ボタンを押すかもしれない。または、ユーザは中止ボタンを再び押すかもしれない。このときはシステムはタイマを止め、表示をクリアする。

Post Conditions: 電子レンジは食べ物の調理を完了していること。

Modeling Finer Variabilities in the Use Case Model

(4)

Modeling Optional Functions: Optional 機能を製品の機能として選択した場合、ユースケース中の所定の変化点に当該機能が挿入される。

「調理する」の変化点記述例①

Name: 庫内ランプ

Reuse Category: Optional

Line Numbers: 1, 5, 8, 9

Function: 「庫内ランプ」オプションが選択されると、調理の間、ならびにドアが開いているときにライトが点灯される。ライトはドアが閉じられたとき、ならびに調理が停止したときに消灯される。

「調理する」の変化点記述例②

Name: ターンテーブル

Reuse Category: Optional

Line Numbers: 5, 8

Function: 「ターンテーブル」オプションが選択されると、調理の間、ターンテーブルが回転する。

Features and Use Cases (1)

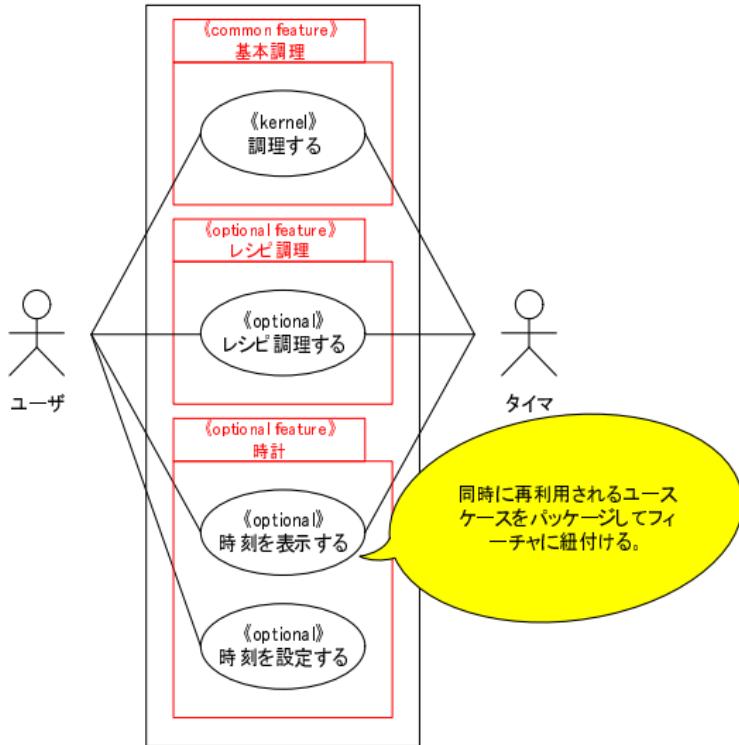
フィーチャは以下のグループとしてモデル化される。

Use Case Model	Feature Model
ユースケース全体	機能フィーチャ
ユースケース中の変化点	
ユースケース中のパラメータ	パラメータ化フィーチャ

ある機能に関係し、同時に再利用されるユースケースや変化点がグループ化され、フィーチャとしてモデル化される。

Features and Use Cases (2)

複数のユースケースとひとつのフィーチャの対応



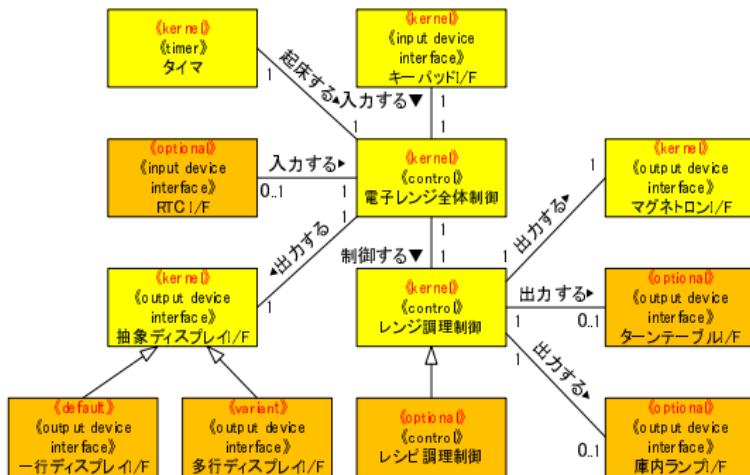
Features and Use Cases (3)

Feature Representation by the Table

Feature Name	Feature Reuse Category	Use Case	Use Case Reuse Category	VP Name
基本調理	Common	調理する	Kernel	
庫内ランプ	Optional	調理する	(変化点)	庫内ランプ
ターンテーブル	Optional	調理する	(変化点)	ターンテーブル
レシピ調理	Optional	レシピ調理する	Optional	
1行ディスプレイ	Default	調理する	(変化点)	表示器
多行ディスプレイ	Alternative	調理する	(変化点)	表示器
時計	Optional	時刻を表示する	Optional	
		時刻を設定する	Optional	
時間制	Parameterized	時刻を表示する	(変化点)	12／24 時間制
		時刻を設定する	(変化点)	12／24 時間制

Static, Dynamic Interaction, and State Dependency Modeling

- ▶ プロダクトラインの構造と振舞いをクラス図、コミュニケーション図、シーケンス図、ステートマシン図で記述する。
- ▶ 製品間の共通要素、可変要素はステレオタイプで明示される。
- ▶ 製品間の可変要素については当該可変要素が有効となるフィーチャ選択条件がガード条件として記述される。



Feature-Class Dependency Modeling

各フィーチャの実現に寄与するクラスを表に整理する。

Feature Name	Feature Reuse Cat.	Class Name	Class Reuse Cat.	Class Parameter
電子レンジ基本	Common	電子レンジ全体制御	Kernel	
		タイマ	Kernel	
		キーパッド I/F	Kernel	
		抽象ディスプレイ I/F	Kernel	
基本調理	Common	レンジ調理制御	Kernel	
		マグネットロン I/F	Kernel	
レシピ調理	Optional	レシピ調理制御	Optional	
庫内ランプ	Optional	庫内ランプ I/F	Kernel	
		レンジ調理制御	Kernel	庫内ランプ
ターンテーブル	Optional	ターンテーブル I/F	Optional	
		レンジ調理制御	Kernel	ターンテーブル
一行ディスプレイ	Default	一行ディスプレイ I/F	Variant	
多行ディスプレイ	Variant	多行ディスプレイ I/F	Variant	
時計	Optional	RTC I/F	Optional	

Case Studies

Bosch (1)

- ▶ ガソリン／ディーゼルエンジン制御ソフトウェア（開発者数1,000人）
- ▶ ポイント
 - ▶ 市場分析に基づくプロダクトラインの確定
 - ▶ 非機能要件にウェイトを置くアーキテクチャ設計
- ▶ 出典
 - ▶ Mirjam Steger, Christian Tischer, Birgit Boss, Andreas Müller, Oliver Pertler, Wolfgang Stolz, and Stefan Ferber, "Introducing PLA at Bosch Gasoline Systems: Experiences and Practices," *Proc. Software Product Line Conf. 2004*, pp.34–50, Aug. 2004.
 - ▶ Frank J. van der Linden, Klaus Schmid, and Eelco Rommes, *Software Product Lines in Action*, Springer-Verlag, 2007.

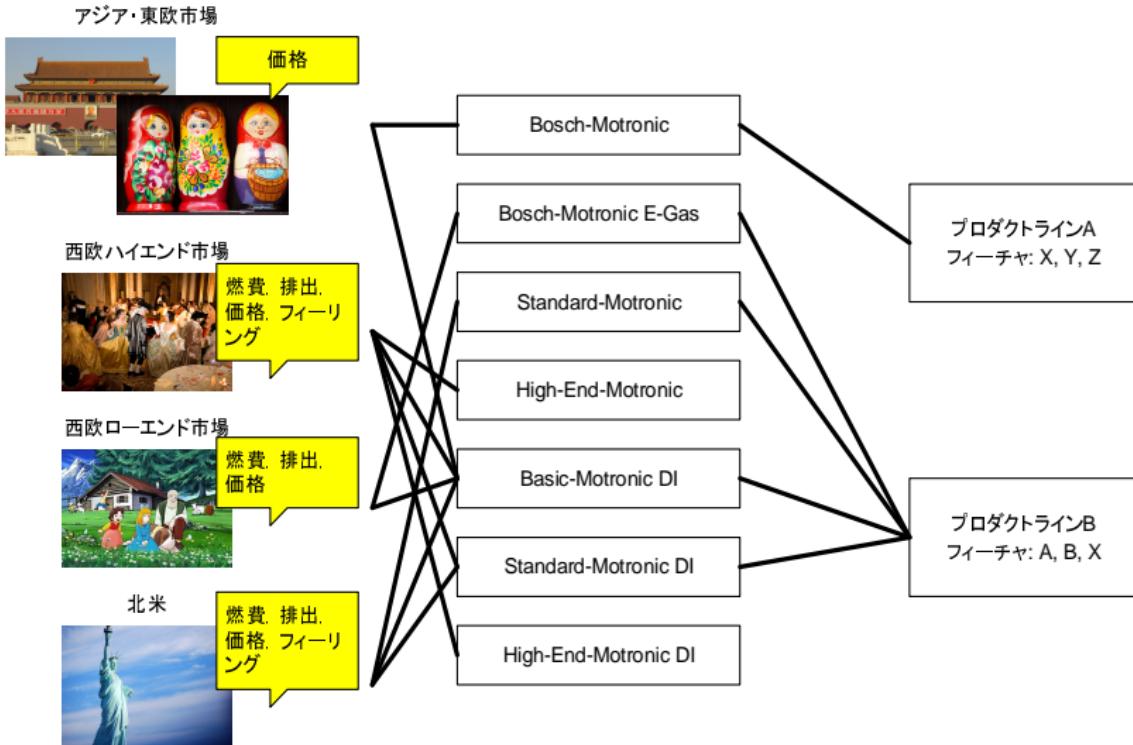
Bosch (2)

市場分析に基づくプロダクトラインの画定

- ▶ プロダクトライン導入前にプラットフォームベースの開発を行い、ひとつのプラットフォームから様々な製品モデルを作り出していたが、以下のようなリスクが顕著に現れるようになってきた。
 - ▶ 必要リソース量の増大
 - ▶ ソフトウェア結合とキャリブレーション手順の複雑化
- ▶ 製品のポートフォリオ分析を行い、製品展開を行う市場領域を整理した。
- ▶ これまでの1 プラットフォーム体制を改め、エンジン制御ソフトウェア開発体制を以下の3つに分けた。
 - ▶ エンジン制御基本のプロダクトライン開発
 - ▶ 標準システムのプロダクトライン開発
 - ▶ 高機能システムの個別開発
- ▶ 各プロダクトラインで標準オプションと顧客専用オプションを定義した。
- ▶ 以上の決定はソフトウェアアーキテクチャやプロセスの定義に影響した。

Bosch (3)

スコーピング例



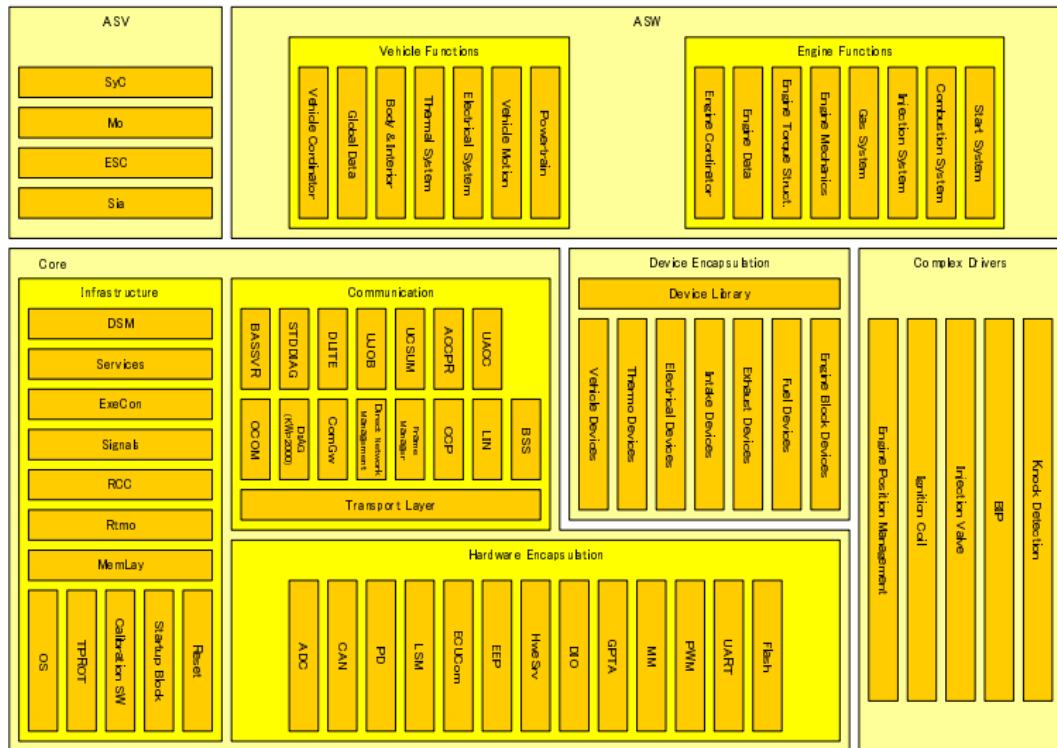
Bosch (4)

非機能要求にウェイトを置くアーキテクチャ設計

- ▶ 非機能要件を重視し、非機能要件を分析、優先度付け、検討結果に基づいて新しいアーキテクチャを定義した。
- ▶ 新しいアーキテクチャは ATAM により評価された。
- ▶ RAM／フラッシュメモリ、実行時メモリ使用量などの必要リソース量はフィーチャ分析に基づくソフトウェア再設計によって削減された。
- ▶ 分散された拠点での開発を支えるべく、インターフェースの定義は重視された。
- ▶ ソフトウェアアーキテクチャへのレイヤ構造の導入。

Bosch (5)

アーキテクチャ例 (Steger, 2007)



Bosch (6)

プロダクトライン開発方法論の現場への展開計画

1. PLA の研究とカスタマイズ

2. プロセスおよび開発方法論の設計と試行

- ▶ 開発タスクと成果物をプラットフォームベースのものと顧客依存のものとに明確に分離。
- ▶ 追加された開発工程： フィーチャ分析， ソフトウェアアーキテクチャ設計， インターフェース管理， ソフトウェアコンポーネント設計， パッケージング， 参照コンフィギュレーション
- ▶ 追加されたマネジメント作業： スコーピング， PLA ゴールに対する初期測定

3. 標準開発プロセスとしての展開と定着化

- ▶ 既存プラットフォームの再設計
- ▶ 中間マネジメント層対象のプロダクトラインプロセスワークショップの開催
- ▶ 新プロセス工程の標準開発プロセスへの組込み
- ▶ プロダクトラインエンジニアリングとアーキテクチャに関するトレーニング

market maker Software AG (1)

- ▶ 株式市場データ管理ソフトウェアプロダクトライン（開発者数 25 人）
- ▶ ポイント
 - ▶ プロダクトライン開発体制への短期間での移行
 - ▶ プロダクトラインのビジョン共有のための密なコミュニケーション
 - ▶ アーキテクチャの崩壊を防ぐアーキテクチャの保守と評価
 - ▶ きめの細かい変更管理
- ▶ 出典
 - ▶ Frank J. van der Linden, Klaus Schmid, and Eelco Rommes, *Software Product Lines in Action*, Springer-Verlag, 2007.

market maker Software AG (2)

プロダクトライン開発体制への短期間での移行

- ▶ プロダクトラインの最初の製品をプロジェクト開始後 12 ヶ月以内に出すこととした。
 - ▶ ドメインエンジニアリングをどこまでも続けることによる工期の長期化を招かないよう終了条件を定めた。
- ▶ プロダクトライン開発チームには新しく雇用した社員をあてた。
 - ▶ 他の仕事に手を煩わせることなくプロダクトライン開発に専念させる。
 - ▶ 但し、会社には密な形で組み入れ、ドメイン専門家のレビューを受けられるようにする。
 - ▶ 新チームは既存システム中の暗黙的な条件を見つけ出すことにも寄与した。
- ▶ リファレンスアーキテクチャができたらすぐに第一の製品の導出を行った。
 - ▶ リファレンスアーキテクチャ上でプロダクトラインに関する種々のアイデアをテストし、設計上の瑕疵を洗い出すことができた。
- ▶ 既存システムをラッピングしたコンポーネントを積極的に使用した。
 - ▶ コンポーネント開発時に深いドメイン知識を必要としなかった。

market maker Software AG (3)

プロダクトラインのビジョン共有のための密なコミュニケーション

- ▶ アプリケーション導出時には、意思決定者が開発チームと直接議論し、開発チームは実現可能性を判断した。
- ▶ 市場要求の変化や新技術の出現にあわせて、プロダクトラインの新しいビジョンを定めた。
 - ▶ 開発中の決定はすべてこのビジョンに則しているかを評価した。
 - ▶ なぜその決定がされたのかを関係者全員で共有した。
- ▶ ドメインエンジニアリングとアプリケーションエンジニアリングを完全には分離しなかった。
 - ▶ ドメインエンジニアリングへのフィードバックに要する時間を短縮した。
 - ▶ コンポーネントの実際にどのように使用されるか把握しやすくなった。
 - ▶ 特に最初の製品導出では分けない方がよい。
- ▶ ソフトウェア開発、マネジメント、営業部門からの人員で構成され、熟練者が指揮を執るスコーピングチームを組織した。
 - ▶ 重要な決定を個人に任せず、プロダクトラインのビジョンと開発目標を共有する。

market maker Software AG (4)

アーキテクチャの崩壊を防ぐアーキテクチャの保守と評価

- ▶ 四半期ごとにリファレンスアーキテクチャの評価を実施し、アーキテクチャの修正計画とそれらの尺度による品質目標を立てた。
 - ▶ フラウンホーファ IESE による「M-System」を尺度として採用した。
 - ▶ クラス、コンポーネントの結合度などデータ収集の容易な尺度を使用した。
 - ▶ コスト面から厳格な評価は行わず、人間による解釈の余地を残す評価を行った。
 - ▶ 他と大きくかけ離れた評価値を示すクラスは問題があるものとして熱かった。
- ▶ リファレンスアーキテクチャ評価の結果によってリファクタリングを実施した。
 - ▶ リファクタリングはシステムの状態をつかむのに有用だった。
 - ▶ リファクタリングは納期プレッシャで生じたソフトウェアブランチを統合するときにも実施した。
- ▶ リファレンスアーキテクチャの保守と評価のためにアーキテクチャマネージャをおいた。

market maker Software AG (5)

Finer Change Management

- ▶ ソフトウェアブランチを生じた変更を主ブランチに統合する計画は変更管理で行った。
 - ▶ ソフトウェアブランチを生じる変更は納期プレッシャによるものか、顧客ニーズへの対応によるものかを区別しなければならない。
 - ▶ 製品のデッドラインが迫っているときは、一旦ソフトウェアのブランチを作り、あとで変更を主ブランチに統合するほうがよい。
- ▶ 変更管理のために以下の役割を置いた。
 - ▶ **変更マネージャ**： 要求された変更に関する影響分析の実施、営業部門や経営部門からの変更要求の評価、リファクタリングの指示、リリース計画の策定を行う。最初の製品を出してからは最も重要な役割。
 - ▶ **要求ディスパッチャ**： 修正を要する問題のうち、些細なものは開発者へ、その他は変更マネージャに転送する。
 - ▶ **問題トラッカ**

Philips Consumer Electronics (1)

- ▶ TV セットのソフトウェアプロダクトライン（開発者数 250 人）
- ▶ ポイント
 - ▶ プロジェクト指向の組織を資産 & 製品指向の組織に変更
 - ▶ サブシステム単位で資産管理
 - ▶ 組込み向けコンポーネントモデル「Koala」の開発と導入
- ▶ 出典
 - ▶ Frank J. van der Linden, Klaus Schmid, and Eelco Rommes, *Software Product Lines in Action*, Springer-Verlag, 2007.

Philips Consumer Electronics (2)

Migration from Project Oriented Organization

- ▶ 以前は製品ごとに大きなチームを作り、それぞれウォータフォールの開発プロセスを回していた。
- ▶ プロダクトライン開発の導入にあたり、資産チームと製品チームからなる構造に組織を変更した。
 - ▶ **資産チーム**： 中規模のチーム。イタレーティブな開発プロセスで継続的に資産を開発、進化させる長期のプロジェクトを回す。
 - ▶ **製品チーム**： 小規模のチーム。資産を利用して製品を構築する。
- ▶ 資産チームは製品チームからの出資を集め、一人の開発マネージャの指揮の下に回される。
- ▶ 資産チームと製品チームのバランスが肝要。
 - ▶ 資産チームはしばしば製品チームから特定製品のために働くことを要求される。
 - ▶ 資産チームから製品チームへの短期間の出向を認め、特定製品用の資産のブランチができるることを許容した。
 - ▶ 資産チームには十分なリソースを割り当てておく。

Philips Consumer Electronics (3)

Asset Management in Sub-System by Sub-System

- ▶ 資産をサブシステム単位で管理する。
 - ▶ 各サブシステムの資産は m 個の製品で使用される。
 - ▶ 各製品は n 個のサブシステム資産から構成される。
 - ▶ 製品とサブシステムの関係はアーキテクチャで規定される。
- ▶ 統合テストのために、実際には市場に出荷されることのない「仮の製品」を早い段階で作る。
 - ▶ サブシステムの欠陥は他のシステムとつないだときにわかることが多いため、サブシステムごとに独立してテストしたのでは足りない。
- ▶ サブシステムはひとつの事業所のひとつの資産チームで開発するようにする。
 - ▶ 異なる事業所でひとつのサブシステムを開発するとコミュニケーションやシステム結合の問題が生じやすい。
- ▶ サブシステムの改変に関する厳しいルール、サブシステムを統合形を事前に用意しておく。
 - ▶ サブシステム間の依存関係の複雑化はプロダクトライン開発のリスクになる。

Hitachi & Hitachi High-Technologies (1)

- ▶ 自動分析装置ソフトウェアプロダクトライン（開発者数不明）
- ▶ ポイント
 - ▶ ドメインエンジニアリング専属チームを構築できず、製品開発チームしか置けない場合の開発組織とプロセス
 - ▶ 製品開発チーム間の合議の場としての「チャンピオンチーム」
- ▶ 出典
 - ▶ Yasuaki Takebe, Naohiko Fukaya, Masaki Chikahisa, Toshihide Hanawa, and Osamu Shirai, "Experiences with Software Product Line Engineering in Product Development Oriented Organization," *Proc. Software Product Line Conf. 2009*, pp.275–283, Sep. 2009.

Hitachi & Hitachi High-Technologies (2)

Consensus meeting by the champion team

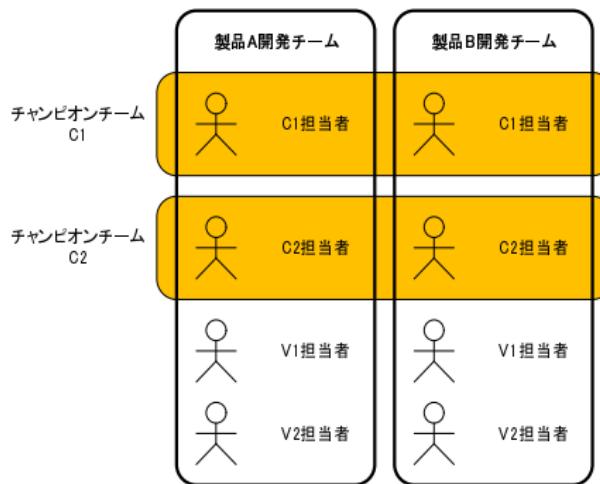
- ▶ プロダクトラインの基準製品を定め、基準製品からリファレンスアーキテクチャを構築し、そのリファレンスアーキテクチャ上で共通部と相違部を分ける。
 - ▶ リファレンスアーキテクチャは後の開発を縛るため基準製品の開発は重要である。
 - ▶ 基準製品の開発メンバはチャンピオン候補生で組織する。
 - ▶ チャンピオン候補生は後発製品の開発チームに移っていく。
 - ▶ 月例スコーピング会議を実施し、プロダクトラインの大局観を見失わないように努める。
- ▶ プロダクトラインの製品開発チームにおいて共通部を担当する開発者を集めて、共通部ごとに以下のタスクを担う「チャンピオンチーム」を構築する。
 - ▶ コア資産の機能の策定
 - ▶ コア資産開発作業の製品開発チームへの割当てとスケジューリング
 - ▶ 開発されたコア資産の設計と実装のレビューと承認
 - ▶ コア資産はチャンピオンチームではなく、各製品開発チームで作られる。
- ▶ 製品開発のニーズに応える使い勝手の良いコア資産が開発される。

Hitachi & Hitachi High-Technologies (3)

Reference Architecture

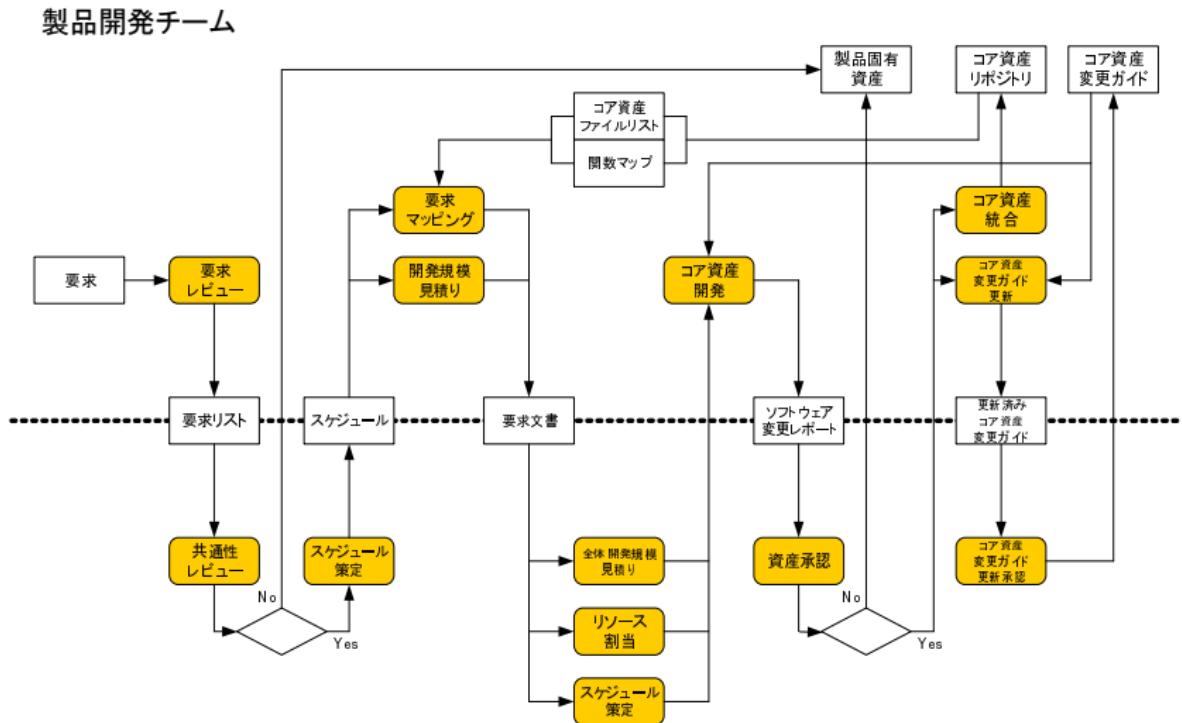
C1 (共通部)	C2 (共通部)
V1 (可変部)	V2 (可変部)

Development Organization



Hitachi & Hitachi High-Technologies (4)

Development Process



Fujitsu Kyushu Network Technologies (1)

- ▶ 社内の複数の開発プロジェクトでプロダクトライン開発を実践
- ▶ ポイント
 - ▶ 会社全体の活動としてプロダクトライン開発方法論を導入
 - ▶ 製品仕様決定権のない設計会社にも関わらずプロダクトライン開発方法論の導入に成功
- ▶ 出典
 - ▶ Takashi Iwasaki, Makoto Uchiba, Jun Otsuka, Koji Hachiya, Tsuneo Nakanishi, Kenji Hisazumi and Akira Fukuda, "An Experience Report of Introducing Product Line Engineering across the Board," *Proc. 14th Int. Software Product Line Conf. (SPLC) 2010*, Vol.2, pp.255–258, Sep. 2010.
 - ▶ J. Otsuka, K. Kawarabata, T. Iwasaki, M. Uchiba, T. Nakanishi, K. Hisazumi, and A. Fukuda, "Small Inexpensive Core Asset Construction for Large Gainful Product Line Development: Developing a Communication System Firmware Product Line," *Proc. 15th Int. Software Product Line Conf. (SPLC) 2011*, 5 pages, Aug. 2011.

Fujitsu Kyushu Network Technologies (2)

- ▶ 会社概要
 - ▶ 富士通（株）の設計専門子会社
 - ▶ 約 800 名のソフト、ハード技術者
 - ▶ CMMI レベル 3 (2007 年)
- ▶ 6 つの開発分野
 - ▶ アプリケーションソフトウェア
 - ▶ 通信サーバソフトウェア
 - ▶ システムファームウェア
 - ▶ ドライバファームウェア
 - ▶ FPGA/LSI
 - ▶ PCB/装置

Fujitsu Kyushu Network Technologies (3)

- ▶ もとより再利用への取組みはソフト、ハードを問わず続けてきていた。
 - ▶ .NET フレームワークや IP などサードパーティ製再利用資産の活用
 - ▶ 自社開発資産のリポジトリへの蓄積
 - ▶ 苦労の割には期待ほどの成果をあげていない現実
- ▶ 2 年間の導入計画を立ててプロダクトライン開発方法論を導入
 - ▶ 先行調査段階（2007～2008 年）
 - ▶ 専任チームの立ち上げ： 先行調査を実施。プロダクトラインのパラダイムを学習。
 - ▶ パイロットプロジェクトの実施： ファイル共有ソフトを対象に実験的実施。フィーチャモデリングガイドライン
 - ▶ 導入段階（2008 年）
 - ▶ 経営陣のトップダウンで導入 WG を結成： 専任チーム + 6 開発分野からの技術者
 - ▶ プロダクトライン開発方法論導入ガイドラインの規定
 - ▶ 実践段階
 - ▶ システム系ファームウェア（移動体ネットワーク機器）
 - ▶ ドライバ系ファームウェア（パケット伝送装置）
 - ▶ PCB ／ 装置（パケット伝送装置）

Fujitsu Kyushu Network Technologies (4)

▶ 導入成果

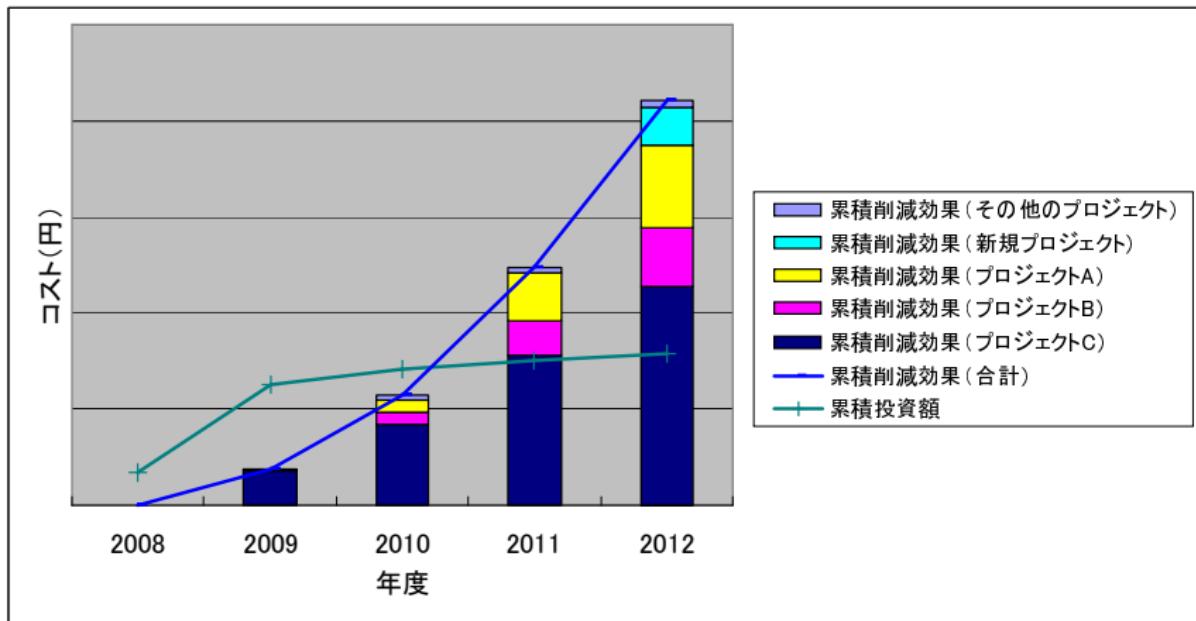
- ▶ システム系ファームウェア（移動体ネットワーク機器）：部分的導入にもかかわらず成功。2製品目で黒字転換（＝既存の開発プロセスによる見積もり値を大きく下回る実開発費用）
- ▶ ドライバ系ファームウェア（パケット伝送装置）：設計フェーズまでの成果物の（製品間をまたがる）一本化に成功。ソースコードの一本化は不可能。性能要求が厳しく、すり合わせ開発部分が大きいのが理由。
- ▶ PCB／装置（パケット伝送装置）：導入効果なし。製品間相違性が部署の開発担当部分には現れず、むしろ担当外の FPGA の内部などに現れる。また、担当部分の製品間相違性はそもそもよく理解されていた。

▶ 成功要因

- ▶ フィーチャモデリングによる「見える化」
- ▶ プロダクトライン開発方法論の限定的適用
- ▶ プロダクトライン導入チームの横串活動
- ▶ 大学や地域コミュニティとの連携
- ▶ 経営層の理解

Fujitsu Kyushu Network Technologies (5)

Economical Impact



Material from Fujitsu Kyushu Network Technologies

Aisin Seiki (1)

- ▶ 自動車のボディ系製品に対してプロダクトライン開発を実践
- ▶ ポイント
 - ▶ フィーチャモデリングによりソフトウェア仕様書の記述の抽象度と関心事を分離
 - ▶ 連続する派生品開発の過程でソフトウェア仕様書から失われていた「仕様の意図」を回復
 - ▶ アーキテクチャを再定義、コードはスクラッチから再開発
 - ▶ フィーチャモデルとアーキテクチャ定義による「見える化」とコミュニケーションの改善、それによるQCDの改善
- ▶ 出典
 - ▶ Yoichi Nishiura, Masaki Asano, and Tsuneo Nakanishi, "Migration to Software Product Line Development of Automotive Body Parts by Architectural Refinement with Feature Analysis," Proc. 25th Asia-Pacific Software Engineering Conf., Dec. 2018.

Aisin Seiki (2)

Background

- ▶ 自動車ボディ系製品の派生品開発を車種展開のたびにclone-and-ownで続けてきていたが、新しい派生製品を開発する際のコストが抑えられないことが問題になっていた。
 - ▶ 既存ソフトウェアの内部の構造と振舞いが極めて見通しの悪い状態になっていた。
 - ▶ 車種展開の際にその車種に向けた過剰に局所的な最適化が行われるようなことがあった。
- ▶ ソフトウェア仕様書からは仕様の意図（要求）がいつしか記述されないようになっていた。
 - ▶ 当初は少人数のステークホルダ間で設計思想がよく共有されていた。
 - ▶ 派生品が増え、開発に携わる技術者が増えていく過程で、システム仕様書で多くを縛るようになっていった。
 - ▶ ソフトウェア仕様書には（抽象度の低い）設計の結果のみが書かれるようになり、ソフトウェア技術者は仕様書通りにソフトウェアを開発することが是とされるようになった。

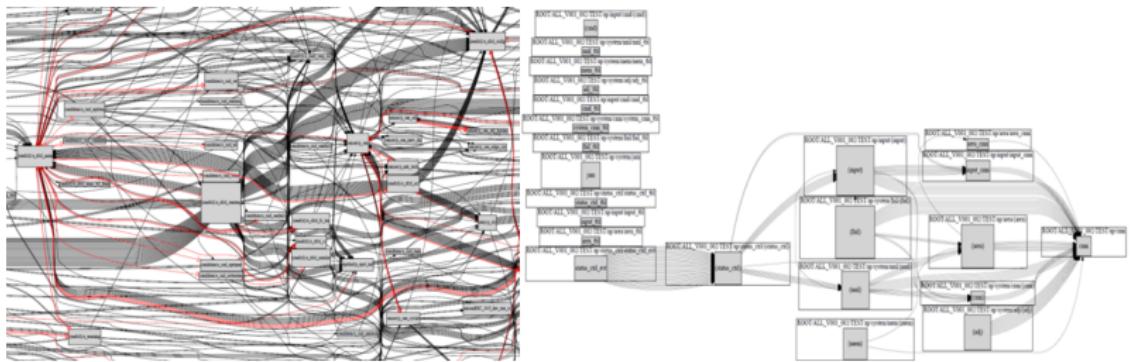
Aisin Seiki (3)

Tactics for Migration

- ▶ ソフトウェア仕様書を読み解き、要求を回復するとともに、フィーチャモデリングによって要求と仕様の抽象度と関心事を分離していった。
- ▶ 並行して構造化分析を実施し、プロダクトラインのアーキテクチャを再定義した。
- ▶ アーキテクチャの再定義にあたり、フィーチャモデルで規定される抽象度と関心事の分離構造に従って、モジュール間のインターフェースを揃えた。

Aisin Seiki (4)

Improvement on Architecture: An architecture with loosely coupled modules is defined. (Left: Before, Right: After)

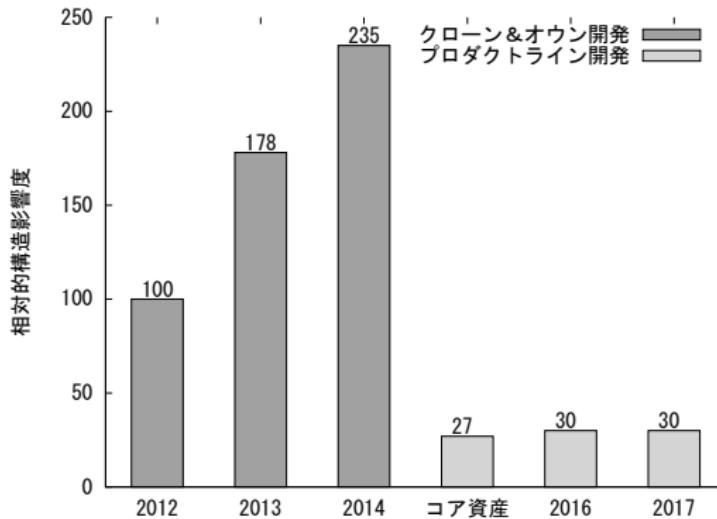


Material from Aisin Seiki

Aisin Seiki (5)

Improvement on Architecture

- アーキテクチャの複雑さを示す構造影響度は大幅に低減
- その後の車種展開でも構造影響度はほとんど増加せず

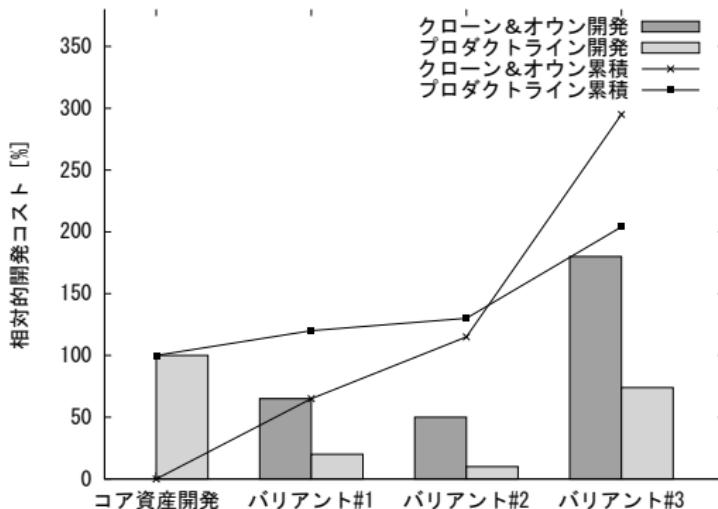


Material from Aisin Seiki

Aisin Seiki (6)

Reduction of Development Cost

- ▶ 従来の clone-and-own を続ける場合の 1/3 に開発コストを低減
- ▶ コア資産開発に要した費用は 3 派生製品の開発コスト削減分で回収



(Material from Aisin Seiki)

Aisin Seiki (7)

Secondary Effects

- ▶ コンポーネント単位でのシミュレーション環境を用いた評価、さらにはその自動化が容易になった。
- ▶ 海外を含めた分散拠点での開発が容易になった。
- ▶ MCU の 32 ビット化や Autosar への対応が容易になった。
- ▶ システム仕様書の記述内容の後工程（＝ソフトウェア開発工程）への影響に目が向けられるようになった。
- ▶ システム開発工程とソフトウェア開発工程を通してのプロセスの見直しにつながった。

Reasons to Fail in Introducing SPL

- ▶ Do not understand and disseminate the paradigm of SPL.
- ▶ Adopt SPL in a top-down manner without consulting engineers.
- ▶ Do not understand present products.
- ▶ Do not envision future products.
- ▶ Expect to make a profit in a hurry.
- ▶ Adopt SPL without the approval of the company (will success easily but will not continue due to personnel relocation)
- ▶ Do not budget for migration to SPL enough.
- ▶ Throw survey of existing artefacts to younger staffs or software consultants
- ▶ Do not establish a system to ask for domain expert's cooperation.
- ▶ Make a hasty attempt to migrate to SPL development.
- ▶ Persist in full automatic derivation of products
- ▶ Persist in full application of SPL (to products and to processes)
- ▶ Dogmatism, excessive optimism, excessive pessimism

What We Should Think before Introducing SPL

- ▶ Is your development problem really the SPL matter?
- ▶ How is the scale of your product line?
- ▶ Can the SPL paradigm work well in your domain?

SPL vs. XDDP

Product Line Development vs. XDDP

Product Line Development	XDDP
Paradigm	Methodology (Process)
Entire comprehension is oriented.	Partial comprehension is allowed.
Modular development is oriented. (variation point concept)	Integral development is allowed.
Plan driven	Change driven
Core assets concept	No core assets
Architecture conscious	Architecture less-conscious
Higher adoption barrier	Lower adoption barrier (?)
Lazy implementation of variation points	Lazy modification of codes
Traceability from features	Traceability from addition and modification requirements

- ▶ The both never conflicts. (Their abstraction levels are different.)
- ▶ Global optimization by lazy realization is a common point of the both.

XDDP in Product Line Development

Employing XDDP in Product Line Development

- ▶ Evolution of the architecture
- ▶ Evolution of the core assets
- ▶ Migration from clone-and-own to product line development
- ▶ Feedback from application engineering to domain engineering

Summary

Summary (1)

- ▶ SPL is a technology for **planned reuse** of software assets.
- ▶ SPL is a **paradigm** with the following key ideas:
 - ▶ Separation of commonality and variability
 - ▶ Architecture centric development
 - ▶ Separation of domain engineering (core asset construction) and application engineering (core asset reuse)
 - ▶ Separation of the problem space and the solution space
 - ▶ Traceability from variabilities
 - ▶ Controlled modification of variation points
- ▶ It is important to understand the paradigm and consider how embody the key concepts in your engineering process for SPL practice.
- ▶ Adoption of SPL needs cost and has risks.

Summary (2)

- ▶ Variability analysis is the first and essential work for SPL practice.
- ▶ Feature modeling has the following benefits:
 - ▶ Comprehending variability
 - ▶ Indexing core assets
 - ▶ Abstraction and separation of concerns (pre-design of the architecture)

Bibliographic Info (1)

1. Paul Clements and Linda Northrop, *Software Product Lines: Practice and Patterns*, Addison-Wesley, 2001.
ソフトウェアプロダクトラインの基本書。
2. Paul Clements, Linda Northrop (著), 前田 卓雄 (翻訳), 『ソフトウェアプロダクトライン: ユビキタスネットワーク時代のソフトウェアビジネス戦略と実践』, 日刊工業新聞社, 2003.
上の本の訳書。
3. Klaus Pohl, Günter Böckle, and Frank van der Linden, *Software Product Line Engineering: Foundation, Principles, and Techniques*, Springer, 2005.
ソフトウェアプロダクトラインのもうひとつの基本書。最初に読むとよい。
4. Klaus Pohl, Günter Böckle, Frank van der Linden (著), 林 好一, 吉村 健太郎, 今関 剛 (訳), 『ソフトウェアプロダクトラインエンジニアリング: ソフトウェア製品系列開発の基礎と概念から技法まで』, SiB アクセス, 2009.
上の本の訳書。
5. Hassan Gomaa, *Designing Software Product Lines with UML*, Addison-Wesley, 2004.
オブジェクト指向プロダクトライン開発方法論 PLUS の教科書。プロダクトラインのパラダイムをいかに開発プロセスに取り入れるかのよいお手本。
6. Hassan Gomaa, *Designing Concurrent, Distributed, and Real-Time Application with UML*, Addison-Wesley, 2001.
オブジェクト指向分散リアルタイムアプリケーション開発方法論 COMET の教科書。上の本と並行して読むとよい。

Bibliographic Info(2)

1. Krzysztof Czarnecki and Ulrich W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.
フィーチャモデリングに関するまとまった解説。
2. Krzysztof Czarnecki and Ulrich Eisenecker (著) , 津田 義史, 今関 剛, 朝比奈 勲 (翻訳) ,
『ジェネレーティブプログラミング』, 翔泳社, 2008.
上の本の訳書。
3. Frank J. van der Linden, Klaus Schmid, and Eelco Rommes, *Software Product Lines in Action*, Springer-Verlag, 2007.
FEFに関する解説あり。プロダクトライン開発事例が豊富。

Reference (1)

1. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Technical Report*, CMU/SEI-90-TR-21, SEI/CMU, Nov. 1990.
2. K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," *Annals of Software Engineering*, 5, pp.143–168, 1998.
3. Krzysztof Czarnecki and Ulrich W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.
4. Paul Clements and Linda Northrop, *Software Product Lines: Practice and Patterns*, Addison-Wesley, 2001.
5. Michalis Anastasopoulos and Cristina Gacek, "Implementing Product Line Variabilities," *Proc. of the Symp. on Software Reusability (SSR) '01*, pp.109–117, 2001.
6. John D. McGregor, Linda M. Northrop, Salah Jarrad, and Klaus Pohl, "Initiating Software Product Lines," *IEEE Software*, Vol.9, No.4, pp. 24–27, July/August 2002.
7. Paul Clements, "Being Proactive Pays Off," *IEEE Software*, Vol.9, No.4, pp.28–31, July/August 2002.
8. Charles Krueger, "Eliminating the Adoption Barrier," *IEEE Software*, Vol.9, No.4, pp.28–31, July/August 2002.
9. Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe, "Feature-Oriented Product Line Engineering," *IEEE Software*, Vol.9, No.4, pp.58–65, July/August 2002.
10. Kwanwoo Lee, Kyo C. Kang, and Jaejoon Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," *Proc. of the 7th Int. Conf. on Software Reuse (ICSR)*, pp.62–77, 2002.

Reference (2)

1. Hassan Gomaa, *Designing Software Product Lines with UML*, Addison-Wesley, 2004.
2. Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker, "Staged Configuration Using Feature Models," *Proc. of the 3rd Int. Conf. on Software Product Line Conf. (SPLC2004)*, pp.266–283, 2004.
3. Thomas von der Maßen and Horst Lichter, "Deficiencies in Feature Models," *Proc. the SPLC Workshop on Software Variability Management for Product Derivation*, 2004.
4. Klaus Pohl, Günter Böckle, and Frank van der Linden, *Software Product Line Engineering: Foundation, Principles, and Techniques*, Springer, 2005.