Software Product Line Engineering: Concepts and Feature Modelling

Dr. Jaejoon Lee

*School of Computing and Communications*
*Lancaster University*

I am from ...
InfoLab21

A Perfect Place to Focus on your Research!!!
Who am I?

Ph.D. Degree - POSTECH
Thesis: A Feature-Oriented Approach to Developing Dynamically Reconfigurable Products in Product Line Engineering
- Advisor: Prof. Kyo-Chul Kang

- January 2008 to Current: Lecturer, School of Computing and Communications, Lancaster University, Lancaster, UK.
- October 2005 to December 2007: Scientist, Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern, Germany
- March 2001 to August 2006: PhD candidate, POSTECH
- March 2000 to February 2001: Senior Member of Technical Staff, POSTECH
- July 1993 to February 2000: Associate Researcher, LG Electronics

Who am I?

Contribution to International Software Product Line Conference:
- Tutorial chair at SPLC 2014
- Panel chair at SPLC 2013
- Tutorial chair at SPLC 2011
- Program chair SPLC 2010
- Workshop chair SPLC 2009
- Doctoral symposium chair SPLC 2007
- PC member (research track): 2011 - 2015
- PC member (vision track): 2015
Why is software engineering important?

• Airbus 320 accident (Air France Flight 296)
  - Fly-by-wire airbus
  - cause of the accident is disputed
  - 3 passengers died


What is a Software Product Line?

A software product line 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.

Product Line Engineering Processes: Feature-Oriented Reuse Method (FORM)

**Domain Engineering Process**

**Application Engineering Process**

* MPP: Marketing and Product Plan  * PL: Product Line  * Req.: Requirements

**Domain Analysis Technology Evolution**

The Army Movement Control Domain [Cohen et al., 1991]
The Automated Prompt Response System Domain [Krut et al., 1996]
The Telephony Domain [Vici et al., 1998]
The Private Branch Exchange Systems Domain [Kang et al., 1999]
The Car Periphery Supervision Domain [Hein et al., 2000]
The Elevator Control Software Domain [Lee et al., 2000]
The E-Commerce Agents Systems Domain [Griss, 2000]
The Algorithmic Library Domain [Czarnecki et al., 2000]
What is Feature?

Various definitions of “feature”:

- Features are "abstractions" of user or developer visible characteristics of an application domain [FODA90].

- A feature refers to an attribute or characteristics of a system that is meaningful to, or directly affects, the users, developer, or other entity that interacts with a system [NIST94].

- A feature is an essential “property” for its associated concept [ODM98].


Identification of Features through Domain Language Analysis
Overview of Feature and Feature Model

Feature: a prominent or distinctive user-visible aspects, quality, or characteristics of a S/W system or systems.

- **Capabilities**
- **Operating Environment**
- **Domain Technologies**
- **Implementation Techniques**

Overview of Feature and Feature Model

<table>
<thead>
<tr>
<th>Feature Types</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>There are three feature types. They are ‘mandatory,’ ‘optional,’ and ‘alternative’:</td>
<td></td>
</tr>
<tr>
<td>- <strong>Mandatory Feature</strong>: An essential feature for products of a given domain.</td>
<td></td>
</tr>
<tr>
<td>- Representation</td>
<td></td>
</tr>
<tr>
<td>- E.g.: ‘DrivingService’ and ‘RunControl’ features in the ECS product line.</td>
<td></td>
</tr>
<tr>
<td>- <strong>Optional Feature</strong>: A selectable feature for products of a given domain.</td>
<td></td>
</tr>
<tr>
<td>- Representation</td>
<td></td>
</tr>
<tr>
<td>- E.g.: ‘HallCallCancelHandling’ and ‘CarCallCancellation’ features in the ECS product line.</td>
<td></td>
</tr>
</tbody>
</table>
Feature Types

- **Alternative features:** A group of features that cannot exist together for a product of a given domain.
  - Representation

- E.g.: ‘GeneralCallHandling and OneCallOneHandling’ and ‘PassengerElevator’ and ‘FreightElevator’ features in the ECS product line.

Feature Categories

<table>
<thead>
<tr>
<th>Features fall largely into four categories:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>Capabilities:</strong> A “capability” feature characterizes a distinct service, operation, function, or performance that an application (for the given domain) may possess. They may be divided into “functional” and “non-functional” ones.</td>
</tr>
<tr>
<td>- E.g.: ‘DrivingService,’ ‘RunControl,’ and ‘Capacity’ features in the ECS product line.</td>
</tr>
<tr>
<td>• <strong>Operating Environment:</strong> An “operating environment” feature represents attributes of the environment in which an application is used and operated.</td>
</tr>
<tr>
<td>- E.g.: ‘OperatingSystem,’ ‘PositionSensor,’ and ‘CPU’ features in the ECS product line.</td>
</tr>
<tr>
<td>• <strong>Domain Technology:</strong> A “domain technology” feature represents implementation details that are specific to a given domain and may not be useful in other domains.</td>
</tr>
<tr>
<td>- E.g.: ‘CallHandlingMethod’ and ‘MovingControlMethod’ features in the ECS product line.</td>
</tr>
<tr>
<td>• <strong>Implementation Technique:</strong> An “implementation technique” feature represents implementation details that are generic and may be used in other domains.</td>
</tr>
<tr>
<td>- E.g.: ‘CommunicationMethod’ feature in the ECS product line.</td>
</tr>
</tbody>
</table>
Relationships between Features

There are three types of relationships: ‘composed-of,’ ‘generalization/specialization,’ and ‘implemented-by’:

- **Composed-of:** When a feature consists of several other features, their relationship is “composed-of” and they are represented using solid lines.
  - Representation
    ![Diagram](A \rightarrow B, B \rightarrow C)
  - E.g.: In the ECS product line, ‘DrivingService’ feature is composed of ‘RunControl,’ ‘DoorHandling,’ and ‘CallHandling’ features.

- **Generalization/Specialization:** When a feature is an abstraction of several other features, their relationship is “generalization/specialization” and represented with dotted lines.
  - Representation
    ![Diagram](A \rightarrow B, B \rightarrow C)
  - E.g.: In the ECS product line, ‘CallHandling’ is an abstraction of the ‘HallCallHandling’ and ‘CarCallHandling’ features.

- **Implemented-by:** When a feature is used to implement other features, their relationships are “implemented-by” and are represented using bold-solid lines. This relationship only exits between features in different categories.
  - Representation
    ![Diagram](A \rightarrow B, C \rightarrow D)
  - E.g.: In the ECS product line, the ‘CallHandling’ feature in the Capability layer is implemented using the ‘CallHandlingMethod’ features in the Domain Technology layer.
Composition rules supplement the Feature Model with mutual dependency and mutual exclusion relationships:

- **Require**: One feature requires the existence of another feature, because they are interdependent.
  E.g.: ‘CarCallCancellation’ requires ‘PassengerElevator’ in the ECS domain.

- **Mutually Exclusive (mutex)**: One feature is mutually exclusive with another, because they cannot coexist.
  E.g.: ‘FreightElevator’ mutex ‘CarCallCancellation’ in the ECS domain.

Issues and decisions that record various trade-offs, rationales, and justifications for feature selection.

**Feature Model Example**
Feature Modeling Process

Domain Definition → Feature Identification → Feature Organization → Feature Refinement

Domain Definition

Domain Selection → Domain Scoping → Organizing Domain Analysis Team
**Feature Identification**

- **Examining**
  - Focus on ‘Communication Terminology’ within the same group or with other groups.
  - Find out the rationales behind the feature selection.
  - Find out the composition rules applied to the feature.

- **Gathering**
  - Summarize the results of the examination.
  - Check for unresolved issues.

- **Categorizing**
  - Place each terminology at an appropriate layer of feature model.
    - Capability Layer
    - Operation Environment
    - Domain Technology
    - Implementation Technique

**Feature Identification: A Car Product Line Example**

<table>
<thead>
<tr>
<th>Products Features</th>
<th>A Type</th>
<th>B Type</th>
<th>C Type</th>
<th>D Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Feature Organization

**Grouping & Connecting**
- Connecting the features in the same layer. Their relationships are 'Generalization / Specialization,' 'Composed-of.'
- Connecting the features in other layer. Their relationship is 'Implemented-by.'

**Characterizing**
- Decide the type of each feature. They are 'Mandatory,' 'Optional,' 'Alternative.'

**Constraining**
- Define the constraints among features such as 'require,' 'mutual exclude.'
- Clarify what constrains each feature. (resource value, standards, law, etc.).

Feature Model: A Car Product Line Example

```
Car
   +-------------------------------+-----------------+-------------------+
   |                              | Multi-Language  |
   +-----------------------------+-----------------+-------------------+
   | English                     | French          |
   +-----------------------------+-----------------+-------------------+
   | English                     | Spanish         |
   +-----------------------------+----------------+-------------------+
```

<table>
<thead>
<tr>
<th>Products Features</th>
<th>A Type</th>
<th>B Type</th>
<th>C Type</th>
<th>D Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse Power</td>
<td>Yes (50 – 200)</td>
<td>Yes (50 – 100)</td>
<td>Yes (100 – 200)</td>
<td>Yes (150 – 200)</td>
</tr>
<tr>
<td>Air Conditioning</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-Language</td>
<td>English and Spanish</td>
<td>English</td>
<td>English</td>
<td>English, Spanish, and French</td>
</tr>
</tbody>
</table>
Feature Refinement

- Review the feature model with domain experts who did not participate in the feature identification.
- Review the feature model with the domain products such as manuals, source codes, design documents, etc.

If necessary,
- Reorganize the features.
- Add, delete, substitute the features.
- Correct the descriptions.

Product Line Requirements Engineering

- Based on “Product Line Analysis: A Practical Introduction” by G. Chastek, et al. (CMU/SEI-2001-TR-001)
Product Line Requirements

• The requirements for a product line
  - Currently needed capabilities
  - Anticipated future requirements
  - Likely future product variations

Common requirements

The set of requirements for the product line (larger than the combined requirements for actual products)

Sources of product line requirements

- Identified Opportunity (e.g., market analysis)
- Product Line Development
- Asset Development
- Products
- Executive
- End User
- Product Line Developer
Product Line Requirements Model

• The four interrelated work products
  - Use-case model: specifies the product line (PL) *stakeholders* and their key *interactions* with the PL. (*A use-case exposes features to stakeholders.*)
  - Feature model: specifies the *stakeholders’ needs in terms of features.*
  - Object model: specifies the PL *responsibilities* that support the features. (*An object is an embodiment of features.*)
  - Dictionary
    - defines the *terminology* utilized in the work products.
    - supports a *consistent* view of the PL requirements.

Modeling Strategies

• Two strategies for PL requirements modeling
  - A feature-driven strategy
    - Focuses on features
    - Use-cases validate the feature model and identify system responsibilities
  - A use-case driven strategy
    - Use-case modeling elicits and discovers requirements
    - Feature model organizes and represents the commonalities and variabilities of these requirements
Modeling Strategies

• The feature-driven strategy is appropriate for organizations with experience
  - Exploits domain experts’ knowledge to rapidly develop the requirements model
  - Enabling designers to explore architectures early in the product line development cycle
  - Feature modeling focuses on the commonalities of functional features, and then introduces variations as refinements
  - Experts apply use cases to verify the identified features and their variants
    - Check that the features of each product can be instantiated from the model

Modeling Strategies

• The use-case-driven strategy can be applied without experts’ direct and constant participation
  - The strategy can be applied when a clear product vision has not been established
Feature Model

- Variability relation with other models
  - Inclusion or exclusion of variable features causes variations in the use case model and/or object model
  - Variation Type
    - Coarse-grained variation
      - Inclusion/exclusion of a use case
      - Inclusion/exclusion of an object
    - Fine-grained variation (internal changes of a unit)
      - Use case scenario reordering
      - Inclusion/exclusion of a use case scenario statement
      - Inclusion/exclusion of an attribute or a method
      - Inclusion/exclusion of a block statement in a method

Legend
- Optional feature
- Alternative feature
- Composed-of relationship
- Generalization relationship
- Implemented-by relationship
Use Case Model

- **Use-Case Modeling**
  - Generate use cases
    - Features may be used as a starting point for use-case modeling
  - Identify system responsibilities for each use case
  - Check consistency between the feature model and the use cases
    - Check if each use-case can be mapped to features; if not, there may be features that are missing from the feature model
    - Check if all the features are addressed by the use cases; if not, there may be use-cases that are missing
    - Check if variations of use-cases and features are consistent
    - Check if the use-cases are consistent with the semantics of the features defined in the dictionary
Use Case Model: Coarse-grained Variation

- Consistency between FM and UC
  - Feature selection affects use case selection

**Feature Model**

**Use Case Model**

Only use cases AGM001 and AGM006 are shown for illustration.

<table>
<thead>
<tr>
<th>AGM001</th>
<th>Play Selected Game</th>
<th>&lt; Configuration : Save/load game : Save score : Speed control&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGM002</td>
<td>Exit Game</td>
<td></td>
</tr>
<tr>
<td>AGM003</td>
<td>Save Game</td>
<td></td>
</tr>
<tr>
<td>AGM004</td>
<td>Save Score</td>
<td></td>
</tr>
<tr>
<td>AGM005</td>
<td>Check Previous Best Score</td>
<td></td>
</tr>
<tr>
<td>AGM006</td>
<td>Play Brickles &lt; Collision : Configuration&gt;</td>
<td>&lt; Macro processing</td>
</tr>
<tr>
<td>AGM007</td>
<td>Play Pong</td>
<td></td>
</tr>
<tr>
<td>AGM008</td>
<td>Play Bowling</td>
<td></td>
</tr>
<tr>
<td>AGM009</td>
<td>Initialization</td>
<td></td>
</tr>
<tr>
<td>AGM010</td>
<td>Animation Loop</td>
<td></td>
</tr>
<tr>
<td>AGM011</td>
<td>Install Game</td>
<td></td>
</tr>
<tr>
<td>AGM012</td>
<td>Uninstall Game</td>
<td></td>
</tr>
<tr>
<td>AGM013</td>
<td>Set Speed of Play</td>
<td></td>
</tr>
</tbody>
</table>
**Use Case Model: Fine-grained Variation**

- **AGM001: Play Selected Game**
  
  `<;Configuration ;Save/load game ;Save score ;Speed control>`
  
  - Use Case Level: Abstract
  - Actor: GamePlayer or GameInstaller
  - Preconditions: AGM011: Install game has completed successfully.
  - Postconditions: Actor has won/lost/tied and the game is ready to play again.

  **Parameterized with features**

  ```
  #Macro processing
  $IF(;Configuration == "Mouse driver")
  {;Mouse driver = true;}
  $ELSEIf( ;Configuration == "Keyboard driver")
  {;Keyboard driver = true;}
  ```

  **Use Case Model: Fine-grained Variation**

<table>
<thead>
<tr>
<th>Actor</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selects PLAY from the menu</td>
<td>Initializes the game and displays the gameboard</td>
</tr>
<tr>
<td>$IF(;Mouse driver) (Left-clicks) $ELSEIf( ;Keyboard driver) (Pressed any key)</td>
<td></td>
</tr>
<tr>
<td>to begin play</td>
<td>Start game action</td>
</tr>
<tr>
<td>$IF(;Mouse driver) (Left-clicks) $ELSEIf( ;Keyboard driver) (Uses the keyboard)</td>
<td></td>
</tr>
<tr>
<td>to enter commands</td>
<td>Responds to the command in the expected manner</td>
</tr>
<tr>
<td>Responds to Won/Lost/Tied box with</td>
<td>Returns the gameboard to its initialized, ready-to-play state</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  **Alternative Courses of Action**

<table>
<thead>
<tr>
<th>Actor</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>At any time, may select EXIT from the menu</td>
<td>See AGM002: Exit game</td>
</tr>
</tbody>
</table>
  | $IF( ;Save/load game)
  [At any time, may select SAVE from the menu] |
  | $IF( ;Save/load game) [See AGM003: Save game] |
  | $IF( ;Save score)
  [At any time, may select SAVE SCORE from the menu] |
  | $IF( ;Save score) [See AGM004: Save score] |
  | $IF( ;Speed control)
  [At any time, may select SPEED CONTROL from the menu] |
  | $IF( ;Speed control) [See AGM013: Set speed of play] |
Feature Interaction Problem (A Home Integration System (HIS) Product Line Example)

The problem of unexpected side-effects when a feature is added to a set of features is generally known as the feature interaction problem. It should be carefully analyzed and resolved to meet the safety requirement.

Feature interaction between the fire and flood features

State Diagram for Door, Intrusion, Fire Event Handling Features of HIS PL
State Diagram for Door, Intrusion, Fire, Flood Event Handling Features of HIS PL

Orthogonal Region

Questions, Comments, ...

Contact
Dr Jaejoon Lee: j.lee3@lancaster.ac.uk