Feature-Oriented Approach for Software Product Line

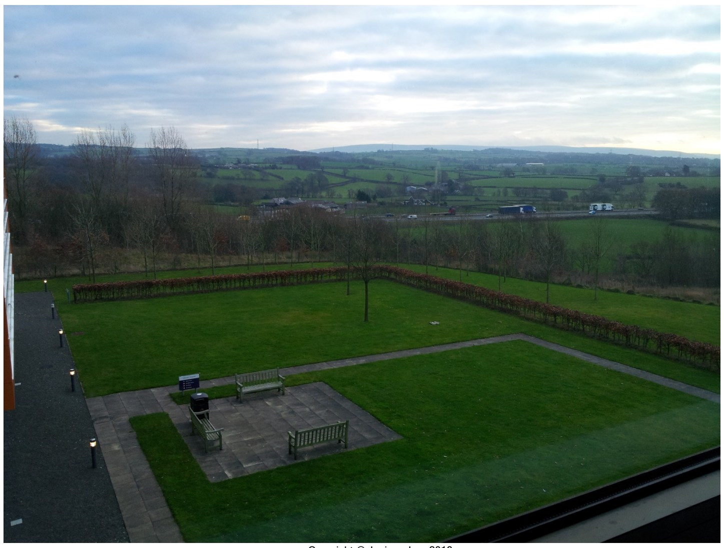
Dr. Jaejoon Lee

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

Feature-Oriented Approach for Software Product Line: *Introduction*

Dr. Jaejoon Lee

School of Computing and Communications
Lancaster University

Some slides are from the book 'Software Engineering' by Ian Sommerville

What is software?

- Software: **computer programs, configuration data and associated documentation** (for end user and maintenance).
- Application areas: commerce, industry, government, medicine, education, entertainment, etc.
- Software products may be developed for a **particular customer** or may be developed for a **general market**.

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

http://en.wikipedia.org/wiki/Air_France_Flight_296

What is software quality?

- Software quality = maintainability + efficiency + usability + dependability
- Maintainability
  - Changing business environment → Software should be written in a way that is easy to change
  - Software, once put into operation, often has a lifetime of 20 - 30 years. Over this time, requirements change and so the software must be easy to change.
- Efficiency
  - Software should not make wasteful use of resources like memory and processor time
- Usability
  - Software must be usable, without undue effort, by the type of user for whom it is designed (Example 3 Mile Island) → appropriate user interface / documentation

What is software quality?

- Dependability = reliability + availability + security + safety
  - Reliability is the probability, over a given time, that the system behave correctly.
  - Availability is the probability that a system will be able to provide services at any given time.
    - E.g. eBay – 57 hours of downtime between December 1998 and June 1999 led to an estimated $5m in refunds.
  - Security is the ability for a system to protect itself from intrusion. e.g. Internet shopping systems need to be secure.
  - Safety is the ability of a system to operate without catastrophic failure. e.g. nuclear power station control, fly-by-wire aircraft.

Software reuse

- In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.
- Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need a design process that is based on systematic software reuse.
- There has been a major switch to reuse-based development over the past 10 years.

Reuse-based software engineering

- Application system reuse
  - The whole of an application system may be reused either by incorporating it without change into other systems (COTS reuse) or by developing application families.
- Component reuse
  - Components of an application from sub-systems to single objects may be reused.
- Object and function reuse
  - Software components that implement a single well-defined object or function may be reused.

Benefits of software reuse

Benefit	Explanation
Increased dependability	Reused software, which has been tried and tested in working systems, should be more dependable than new software. Its design and implementation faults should have been found and fixed.
Reduced process risk	The cost of existing software is already known, whereas the costs of development are always a matter of judgment. This is an important factor for project management because it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as subsystems are reused.
Effective use of specialists	Instead of doing the same work over and over again, application specialists can develop reusable software that encapsulates their knowledge.

Benefits of software reuse

Benefit	Explanation
Standards compliance	Some standards, such as user interface standards, can be implemented as a set of reusable components. For example, if menus in a user interface are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability because users make fewer mistakes when presented with a familiar interface.
Accelerated development	Bringing a system to market as early as possible is often more important than overall development costs. Reusing software can speed up system production because both development and validation time may be reduced.

Problems (Difficulties) with reuse

Problem	Explanation
Increased maintenance costs	If the source code of a reused software system or component is not available then maintenance costs may be higher because the reused elements of the system may become increasingly incompatible with system changes.
Lack of tool support	Some software tools do not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account. This is particularly true for tools that support embedded systems engineering, less so for object-oriented development tools.
Not-invented-here syndrome	Some software engineers prefer to rewrite components because they believe they can improve on them. This is partly to do with trust and partly to do with the fact that writing original software is seen as more challenging than reusing other people's software.

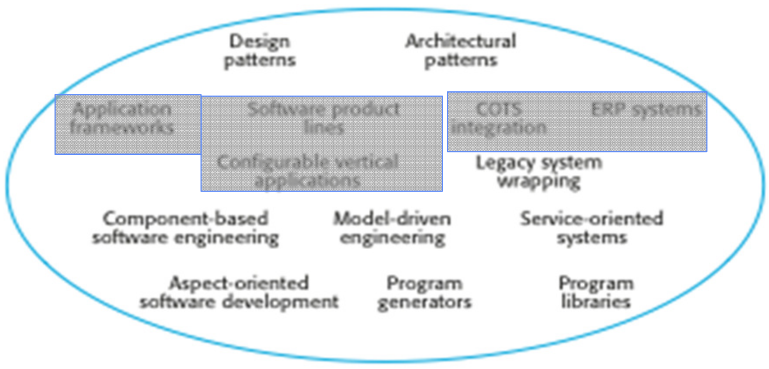
Problems (Difficulties) with reuse

Problem	Explanation
Creating, maintaining, and using a component library	Populating a reusable component library and ensuring the software developers can use this library can be expensive. Development processes have to be adapted to ensure that the library is used.
Finding, understanding, and adapting reusable components	Software components have to be discovered in a library, understood and, sometimes, adapted to work in a new environment. Engineers must be reasonably confident of finding a component in the library before they include a component search as part of their normal development process.

The reuse landscape

- Although reuse is often simply thought of as the reuse of system components, there are many different approaches to reuse that may be used.
- Reuse is possible at a range of levels from simple functions to complete application systems.
- The reuse landscape covers the range of possible reuse techniques.

The reuse landscape

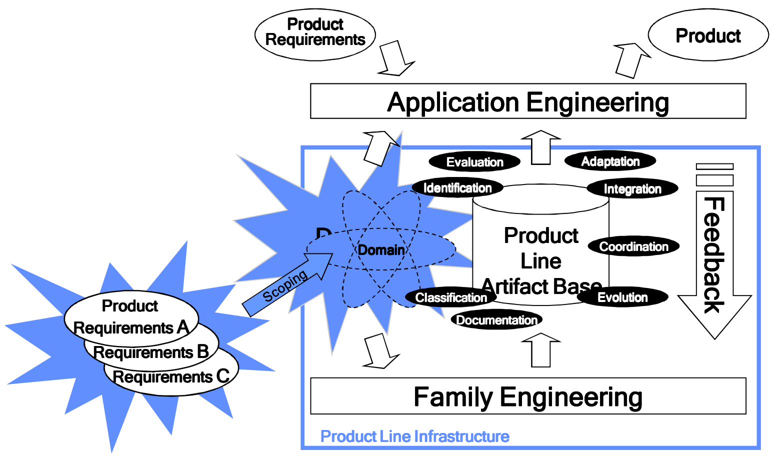


What is a Software Product Line?

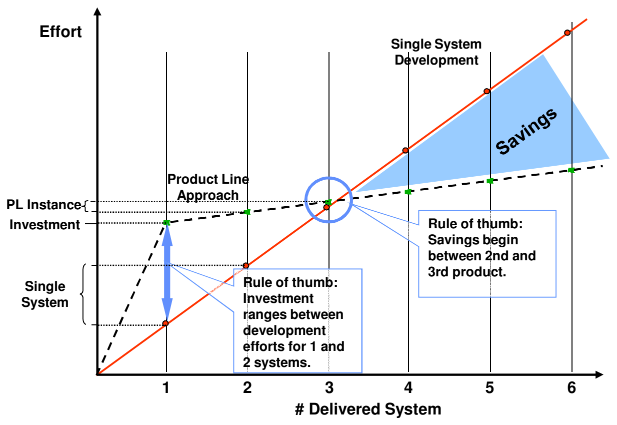
A software product line is a *set* of software-intensive systems sharing a *common, managed set of features* that satisfy the specific needs of a *particular market* segment or mission and that are *developed from a common set of core assets in a prescribed way*.

Lawrence G. Jones and Linda M. Northrop, Software Product Lines: Capitalizing on Your Process Improvement Investment, European Software Engineering Process Group Conference, Amsterdam, Netherlands, June 2001

Product Line Life Cycle



Product Line Economics – Development Effort



[Böckle, Clements, McGregor, Muthig, Schmid in IEEE Software]

Success Story: Cummins, Inc. (1/2)

- World's largest manufacturer of large diesel engines.
- Product family includes
  - 9 basic engine types
  - 4-18 cylinders
  - 3.9 - 164 liters
  - 12 kinds of electronic control modules
  - 5 kinds of processors
  - 10 kinds of fuel systems
  - diesel fuel or natural gas

Success Story: Cummins, Inc. (2/2)

- Cost
  - Management estimates product line ROI of 10:1
- Time to Market
  - Product cycle time: a year to a few days
- Productivity
  - 20 product groups → 1000 separate applications
  - 75% of all software comes from core assets
  - Productivity improvement of 360%
- Enter new Markets
  - Capability let Cummins enter and dominate industrial diesel engine market
- Quality
  - Software quality is at an all-time high
  - 15 of 15 projects are on track (was 3 of 10)
  - Customer satisfaction is high.

Others

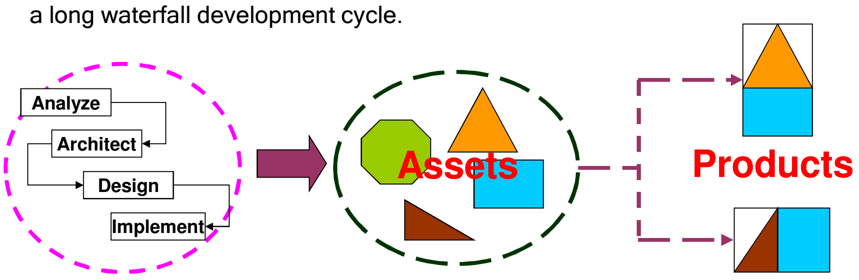
- Nokia – Mobile phones
  - Market/Productivity: went from 4 different phones produced per year to 50 per year
- Hewlett Packard – Printer systems
  - Time to Market: 2-7x cycle time improvement (some 10x)
  - Productivity of Sample Project
    - shipped 5x number of products
    - that were 4x as complex
    - and had 3x the number of features
    - with 4x products shipped/person

http://www.splc.net/fame.html

Product Line Approaches

* Charles Krueger, "Eliminating the Adoption Barrier", IEEE Software, Jul/Aug, 2002, pp. 29-31

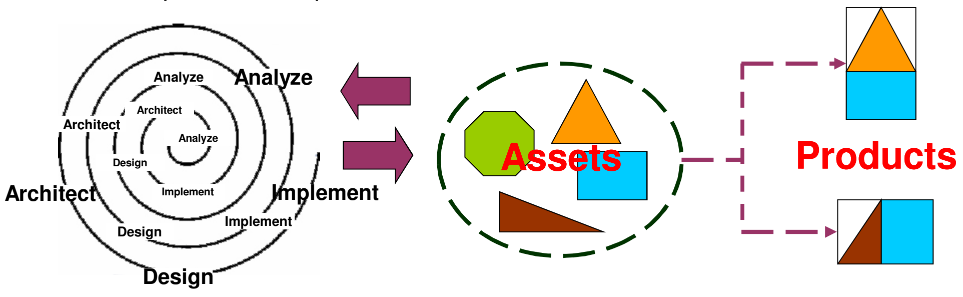
- Proactive approach*
  - The proactive approach to software product lines is like the waterfall approach to conventional software. You analyze, architect, design, and implement all product variations on the foreseeable horizon up front.
  - This approach might suit organizations that can predict their product line requirements well into the future and that have the time and resources for a long waterfall development cycle.



Product Line Approaches

*Charles Krueger, "Eliminating the Adoption Barrier", IEEE Software, Jul/Aug, 2002, pp. 29-31

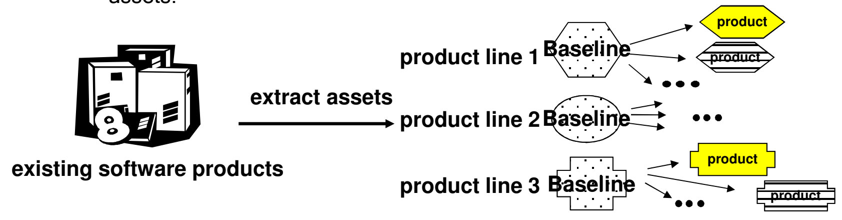
- Reactive approach*
  - The reactive approach is like the spiral or extreme programming approach to conventional. You analyze, architect, design, and implement one or several product variations on each development spiral.
  - This approach works in situations where you cannot predict the requirements for product variations well in advance.



Product Line Approaches

*Charles Krueger, "Eliminating the Adoption Barrier", IEEE Software, Jul/Aug, 2002, pp. 29-31

- Extractive approach*
  - The extractive approach reuses one or more existing software products for the product line's initial baseline.
  - Require lightweight software product line technology and techniques that can reuse existing software without much reengineering.
  - Effective for an organization that wants to quickly transition from conventional to software product line engineering
  - This approach *does not* support the possibility of one organization developing the core assets and a separate organization developing the products based on the core assets.



Feature-Oriented Approach for Software Product Line: *FORM approach and Feature Modeling*

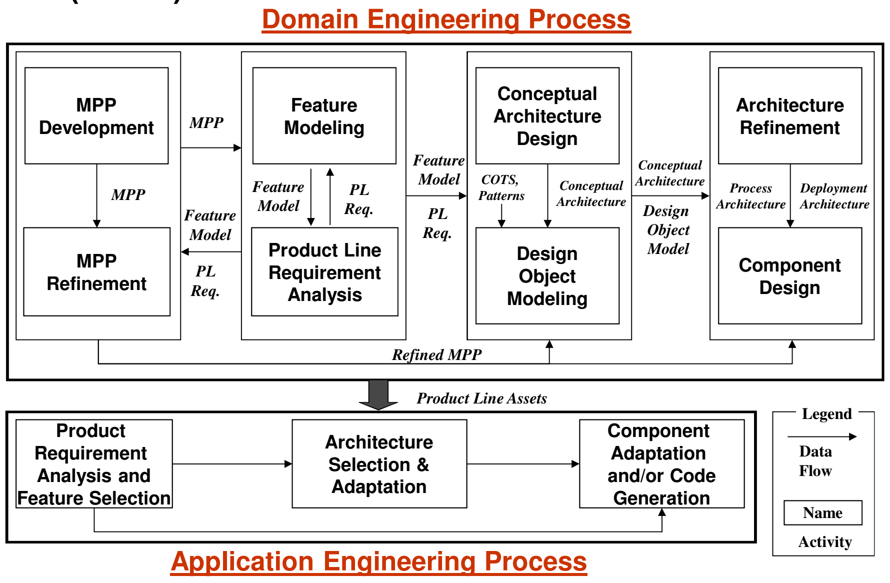
Dr. Jaejoon Lee

School of Computing and Communications
Lancaster University

InfoLab21 | Lancaster University | Lancaster | LA1 4WA | UK
www.infolab21.lancs.ac.uk

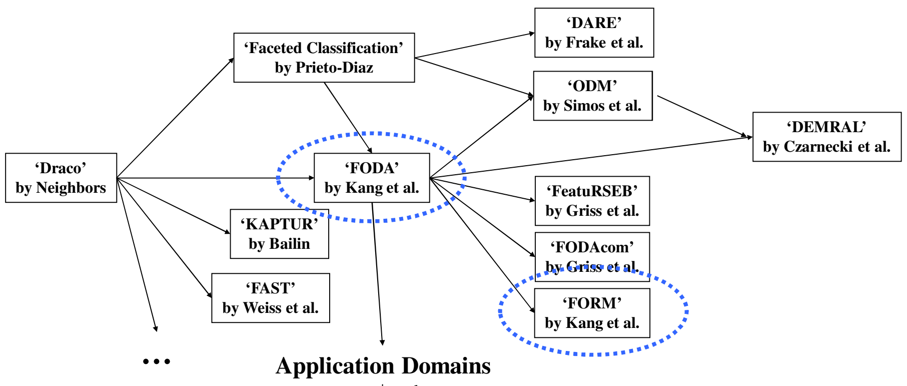
InfoLab21
Computing Department

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



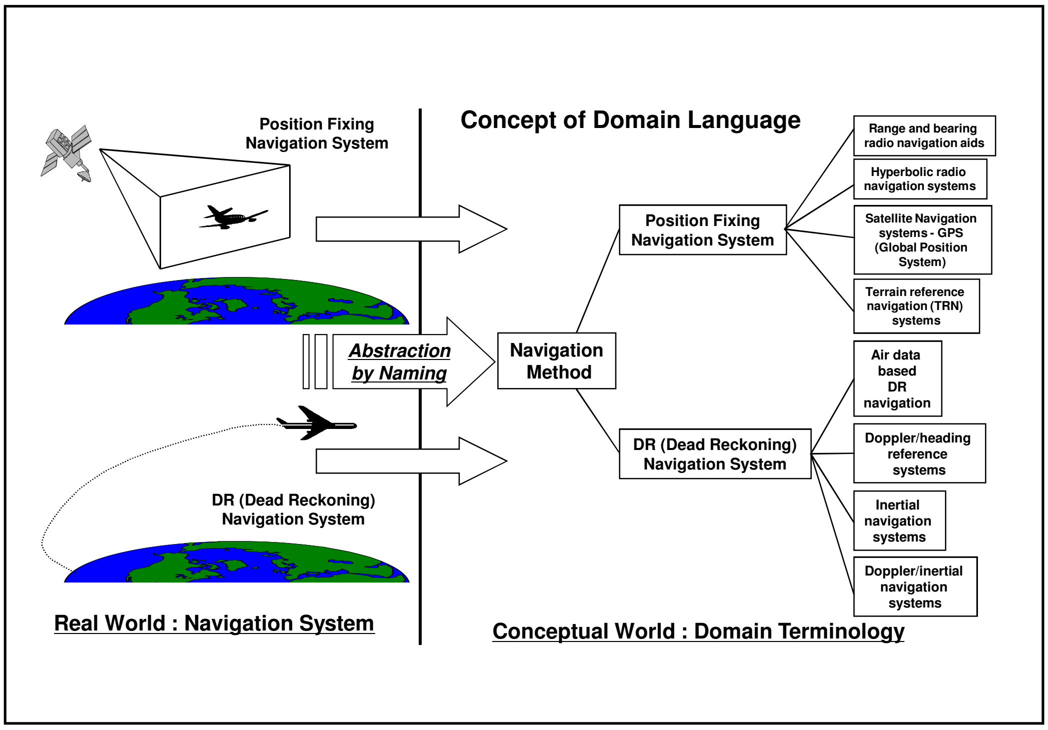
* MPP: Marketing and Product Plan * PL: Product Line * Req.: Requirements

Domain Analysis Technology Evolution

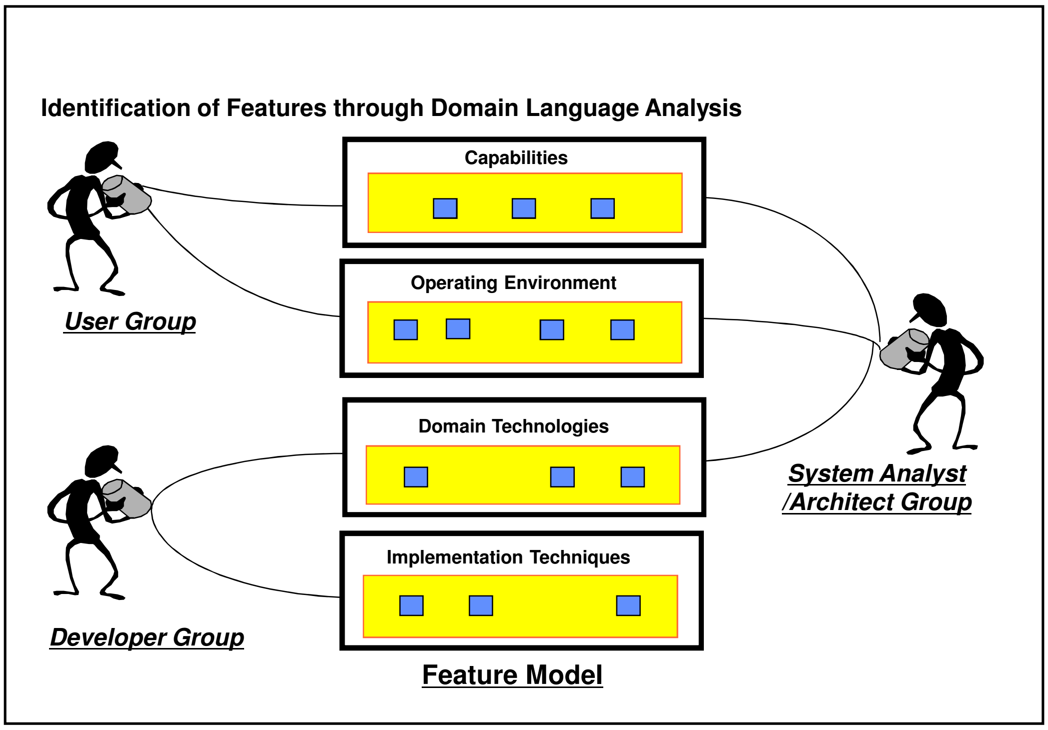


The Army Movement Control Domain [Cohen et al., 1991]
The Automated Prompt Response System Domain [Krut et al., 1996]
The Telephony Domain [Vici et al., 1998]
The Private Brach Exchange Systems Domain [Kang et al., 1999]
The Car Periphery Supervision Domain [Hein et al., 2000]
The Elevator Control Software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

Concept of Domain Language


Position Fixing Navigation System
Abstraction by Naming
DR (Dead Reckoning) Navigation System
Real World : Navigation System
Navigation Method
Position Fixing Navigation System
DR (Dead Reckoning) Navigation System
Range and bearing radio navigation aids
Hyperbolic radio navigation systems
Satellite Navigation systems - GPS (Global Position System)
Terrain reference navigation (TRN) systems
Air data based DR navigation
Doppler/heading reference systems
Inertial navigation systems
Doppler/inertial navigation systems
Conceptual World : Domain Terminology

Identification of Features through Domain Language Analysis



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].**

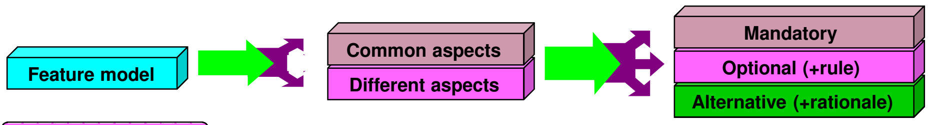
[FODA90] K. Kang, S. Cohen, J, Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report, CMU/SEI-90-TR-21, Software engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1990.

[NIST94] National Institute of Standard and Technology Special Publication 500-222, MD 20899-001, Gaithersburg, December 1994.

[ODM98] M. Simos and J. Anthony. "Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering," Proceedings of the Fifth International Conference on Software Reuse, IEEE Computer Society Press, 1998.

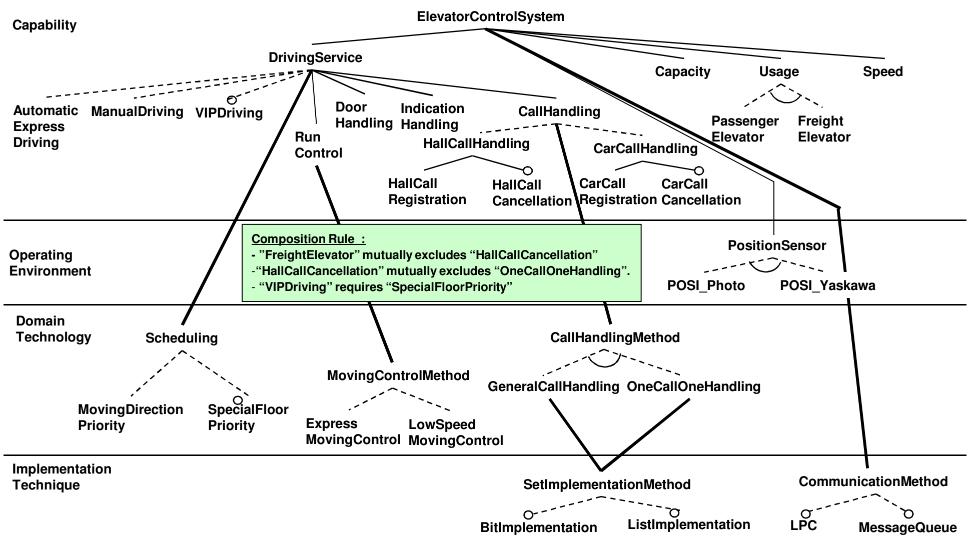
Overview of Feature and Feature Model

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



Capabilities	**of applications from the end-user's perspective.**
Operating Environment	**in which applications are used and operated. (H/W, S/W platform, O/S, interfaces with different types of devices)**
Domain Technologies	**that are commonly used in a domain (e.g., navigation methods in the avionics domain).**
Implementation Techniques	**designer's decision on algorithms and data structures.**

Feature Model Example



Composition Rule :
- "FreightElevator" mutually excludes "HallCallCancellation"
-"HallCallCancellation" mutually excludes "OneCallOneHandling".
- "VIPDriving" requires "SpecialFloorPriority"

References

- [FODA90] K. Kang, S. Cohen, J, Hess, W. Nowak, and S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Technical Report, CMU/SEI-90-TR-21, Software engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1990.
- [NIST94] National Institute of Standard and Technology Special Publication 500-222, MD 20899-001, Gaithersburg, December 1994.
- [ODM98] M. Simos and J. Anthony. "Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering," Proceedings of the Fifth International Conference on Software Reuse, IEEE Computer Society Press, 1998.
- K. Lee, K. Kang, and J. Lee, "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering," Cristina Gacek, editor, Software Reuse: Methods, Techniques, and Tools: Proceedings of the Seventh Reuse Conference (ICSR7), Austin, U.S.A., Apr.15-19, 2002, Heidelberg, Germany: Springer Lecture Notes in Computer Science Vol. 2319, 2002, pp. 62-77.
- K. Kang, J. Lee, and P. Donohoe, "Feature-Oriented Product Line Engineering," IEEE Software, Vol. 9, No. 4, July/August 2002, pp. 58-65.

Questions,
Comments, …

?

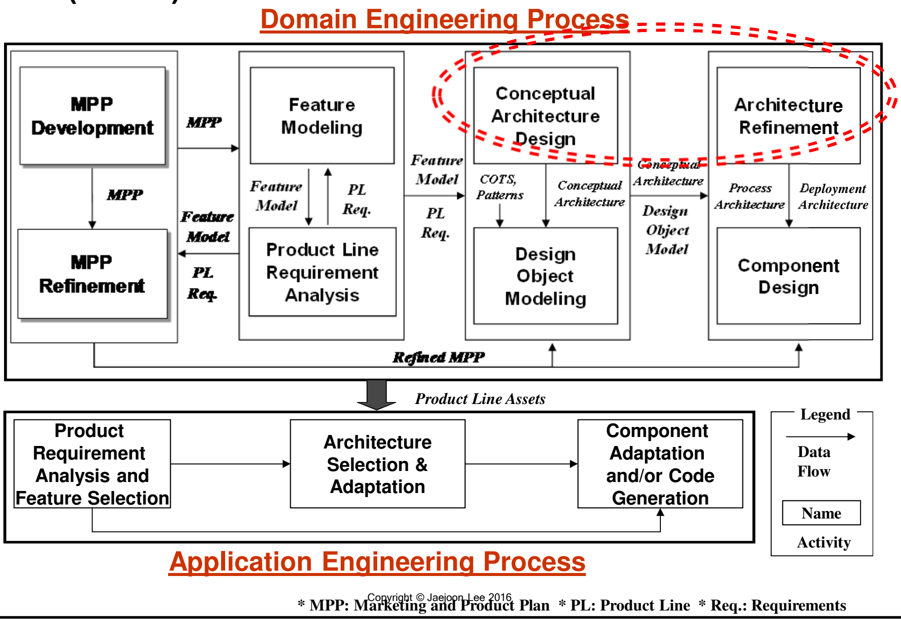
Contact
Dr Jaejoon Lee: j.lee3@lancaster.ac.uk

Feature-Oriented Approach for Software Product Line: *Software Architecture*

Dr. Jaejoon Lee

School of Computing and Communications
Lancaster University

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



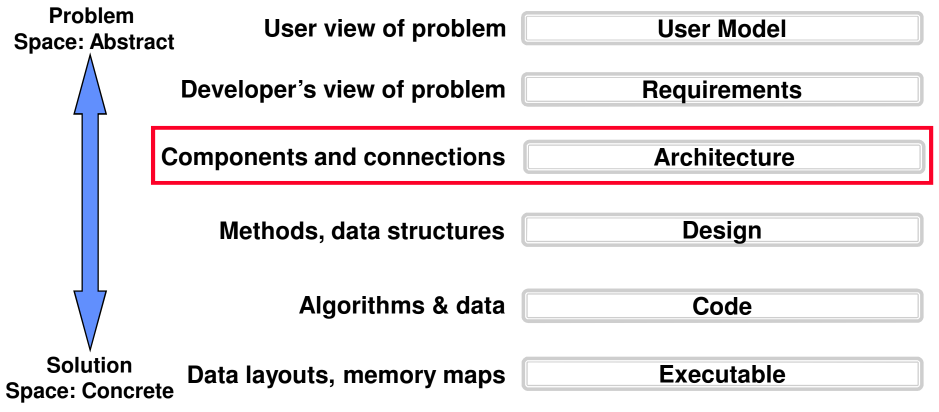
Definition of Software Architecture

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships between them
- The term also refers to documentation of a system's software architecture
- Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects

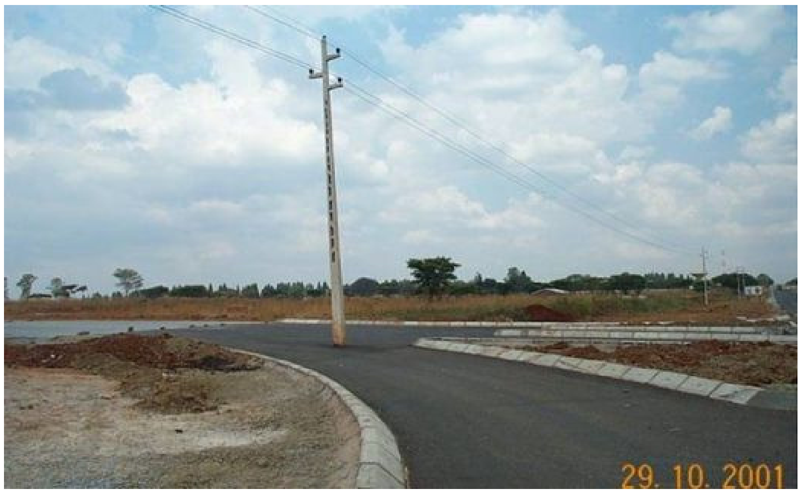
Architecture vs. Design

- What’s the difference between them?
  - Architecture is design, but not all design is architecture
  - Many design decisions are left unbound and left to the discretion and good judgment of downstream designers and implementers
  - The architecture establishes constraints on downstream activities, which produces artifacts that are compliant with the architecture
  - But, architecture does not define an implementation

Role of Software Architecture: Bridging the Gap

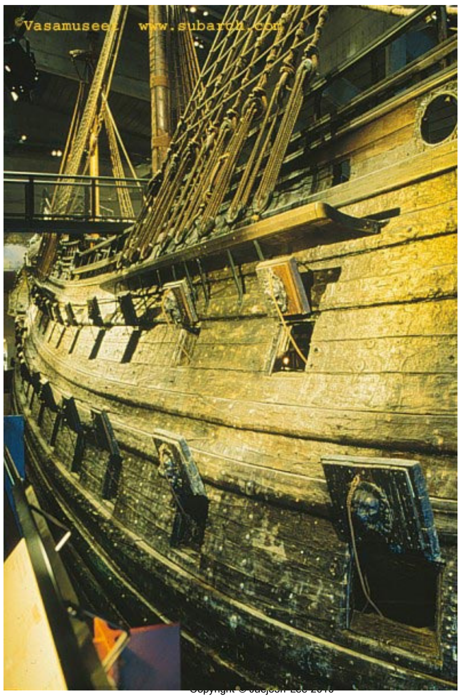


Bad design can be dangerous



Swedish Ship *Vasa*

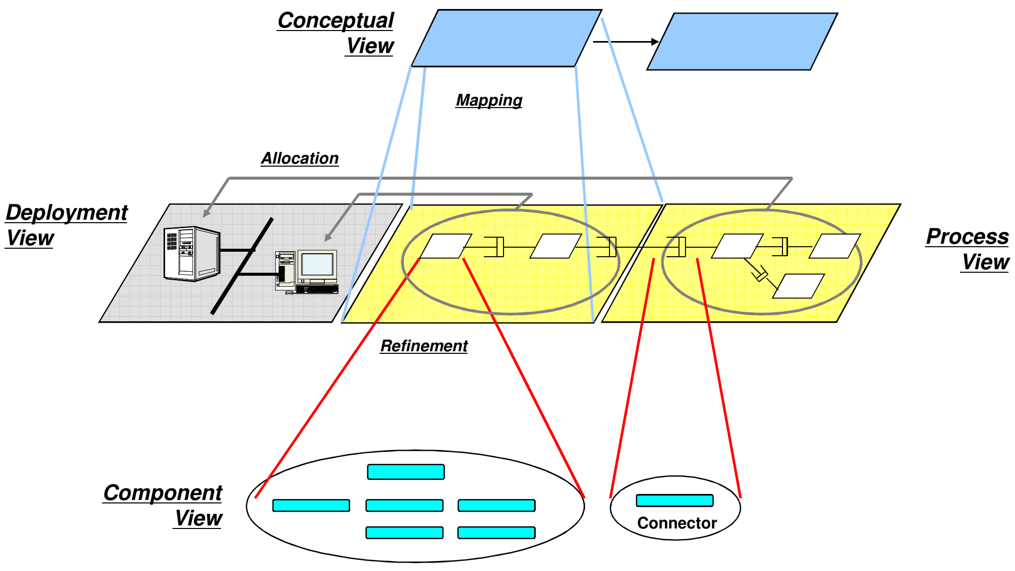
Swedish Ship *Vasa*



Product Line Architecture

- A product line architecture (PLA, also referred to as Reference Architecture) is a generic software architecture for a product line that embodies the architectures for all product line members
- A product line architecture differs from a single system architecture in that it has to cover additional concerns:
  - what are the common parts of the architecture?
  - what are the variable parts?
  - how does a particular instance architecture look like?
  - how can it be ensured that all intended product line members are indeed supported by the architecture?

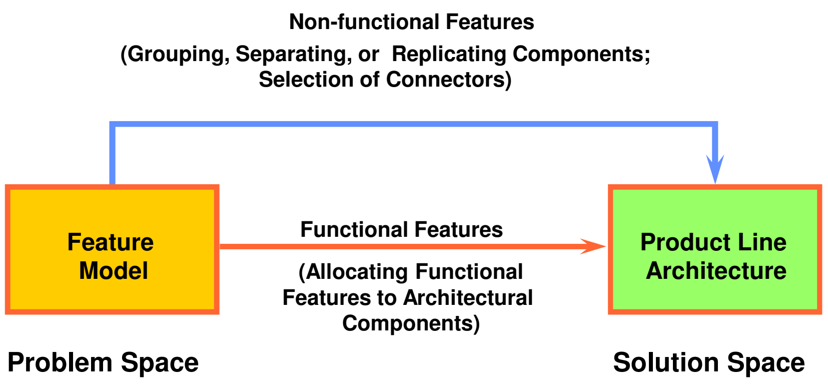
FORM Product Line Architecture Views



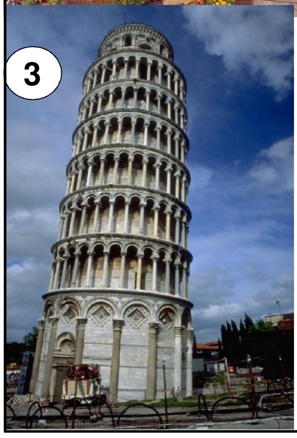
FORM Architecture Views

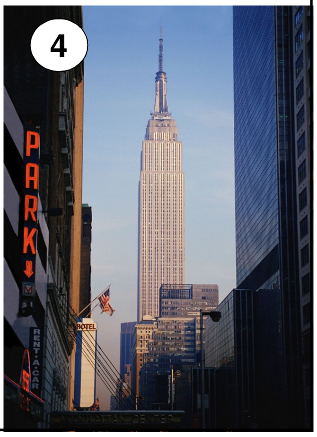
- Conceptual view: it shows major functional components of software, each of which may be decomposed into sub-components.
- Process view: a set of concurrent processes and interactions between these processes are identified.
- Deployment view: it shows an allocation of processes to hardware resources
- Component view: focuses on the organization of the software components in the software development environment

Mapping Feature Model to Product Line Architecture


Non-functional Features
(Grouping, Separating, or Replicating Components; Selection of Connectors)
Feature Model
Functional Features
(Allocating Functional Features to Architectural Components)
Product Line Architecture
Problem Space
Solution Space

ant to liv





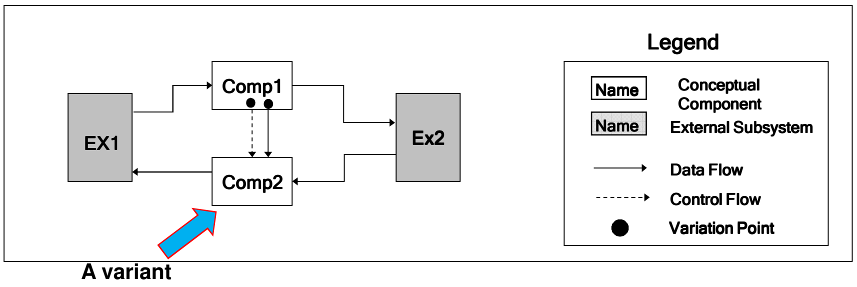
Variation Points

- What is a variation point?

  A variation point represents a place where different variants can be bound for different product configurations.

Conceptual View

- A conceptual view shows major functional components of software, each of which may be decomposed into sub-components.
- Communications between conceptual components are modeled as data or control flow. At this level, the flow directions and types of messages are the main concerns.

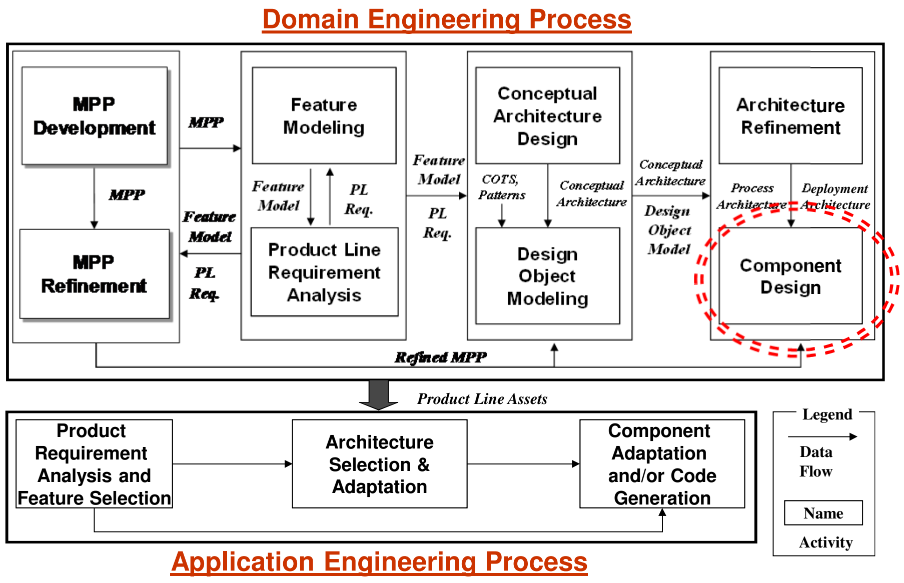


Feature-Oriented Approach for Software Product Line: *Product Line Component Design and Implementation*

Dr. Jaejoon Lee

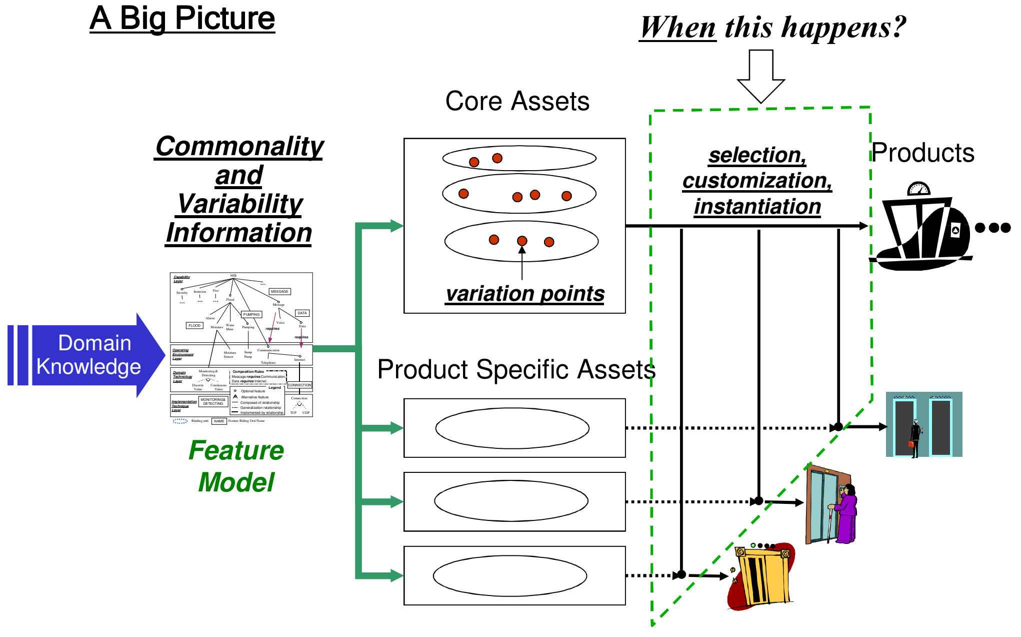
School of Computing and Communications
Lancaster University

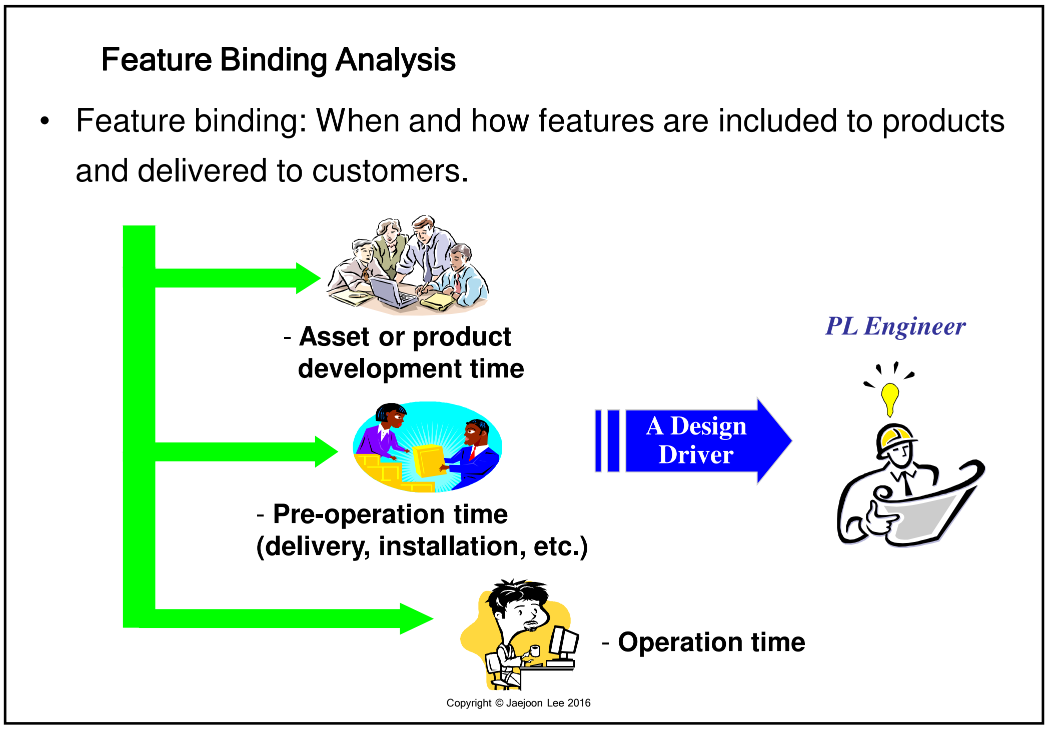
Product Line Engineering Processes



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

A Big Picture

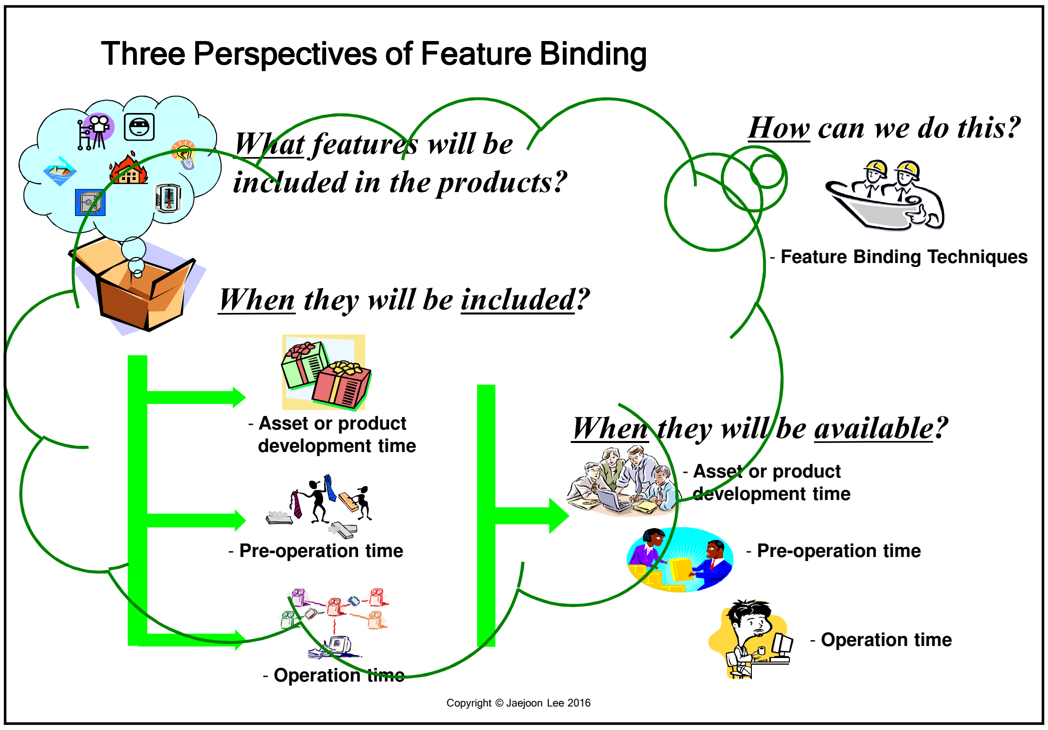




Feature Binding Analysis

- Feature binding: When and how features are included to products and delivered to customers.

- Asset or product development time
- Pre-operation time (delivery, installation, etc.)
- Operation time
A Design Driver
PL Engineer



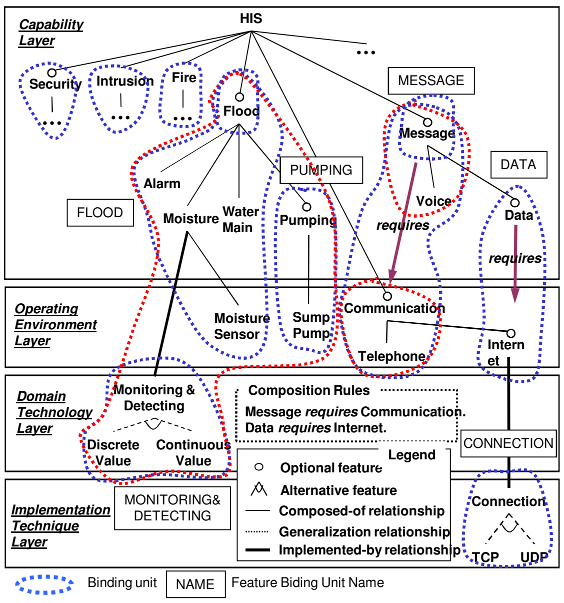
Three Perspectives of Feature Binding

What features will be included in the products?
How can we do this?
- Feature Binding Techniques
When they will be included?
- Asset or product development time
- Pre-operation time
- Operation time
When they will be available?
- Asset or product development time
- Pre-operation time
- Operation time

What: Feature Binding Units

- What is a feature binding unit?
  - We define a feature binding unit as a set of features that are related to each other via compose-of, generalization/specialization, and implemented-by relationships and composition rules (i.e., require and mutually exclude).
- Feature binding unit identification starts with identification of independently configurable service features.
  - A service feature represents a major functionality of a system and may be added to and removed from as a unit.
  - A service feature uses other features (e.g., operational, environmental, and implementation features) to function properly.
  - The constituents of a binding unit can be found by traversing the feature model along the feature relationships and composition rules.

What: Feature Binding Units



When: Feature Binding Time

- Feature Binding Time
  - Generally, feature binding time has been looked at from the software development lifecycle viewpoint. However, there exits another dimension that is based on the binding state of a feature binding unit.
  - That is, some feature binding units may be developed and included in product line assets at asset development time, but their availability can be determined at installation time by enabling or disabling the feature binding units.
  - Furthermore, activation of the available features may have to be controlled to avoid a feature interaction problem.
- Feature binding time analysis with additional view on *feature binding state* provides more precise framework for feature binding analysis.
  - It includes *inclusion* and *availability* states and *activation rules*.

When: Feature Binding Time

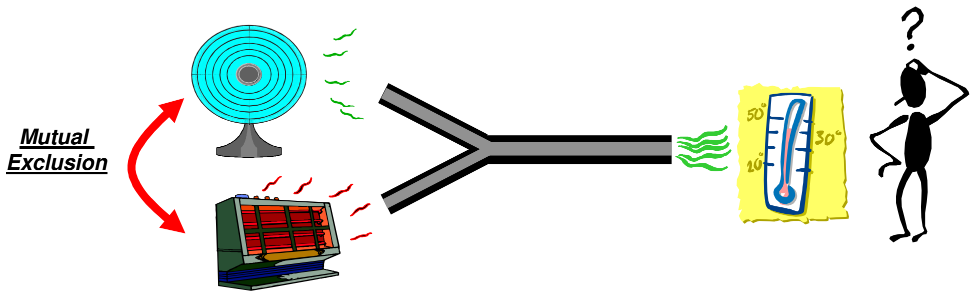
Product Lifecycle View

	Inclusion	***Availability***	***Activation Rule***
Operation	**PUMPING**	**PUMPING**	
Pre-Operation (Installation)		**FLOOD, MESSAGE, DATA, CONNECTION, MONITORING& DETECTING**	• **MESSAGE** ***requires*** **INTRUSION, FIRE, or FLOOD activated.** • **FIRE** ***has higher priority than*** **FLOOD.** • **FIRE** ***has higher priority than*** **INTRUSION.**
Product Development	**FLOOD, MESSAGE, DATA, SECURITY, CONNECTION**	**SECURITY**	
Asset Development	**FIRE, INTRUSION, MONITORING& DETECTING**	**FIRE, INTRUSION**	

Feature Binding State View

When: Feature Binding Time

- Feature Activation Rules
  - The activation rules provide information on concurrency of feature binding unit activation and they are defined in terms of mutual exclusion, dependency, and priority schemes.



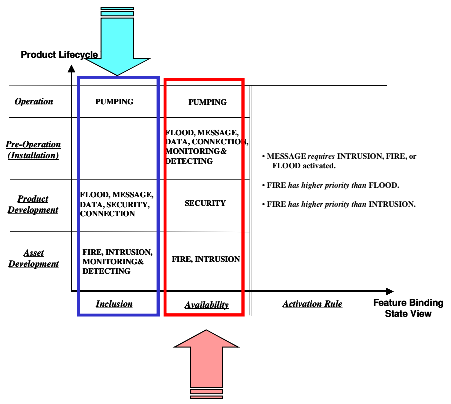
How: Feature Binding Techniques

- Selection of binding techniques depends both on binding time and quality attributes (e.g., flexibility) required for products.
  - Delaying binding time to a later phase of the lifecycle may provide more flexibility. But, applicable implementation techniques are limited and they usually require more performance overheads.
- We propose a classification of feature binding techniques based on the feature binding states.
  - Binding techniques for the feature 'inclusion' should be able to control feature inclusion by including or excluding code segments or components from products.
  - Binding techniques for the feature 'availability' should provide mechanisms for enabling or disabling access to features.
  - In addition, we should also explore techniques for dynamic or static binding of features.

How: Feature Binding Techniques

Binding Techniques for feature 'inclusion': Code generation, pre-processing, macro processing, Internet Component Download (ICD)*, etc.

Binding techniques for static/dynamic feature binding: Dynamic binding of objects, menus, plug-ins, etc.


Product Lifecycle
Operation
Pre-Operation (Installation)
Product Development
Asset Development
PUMPING
PUMPING
FLOOD, MESSAGE, DATA, CONNECTION, MONITORING& DETECTING
FLOOD, MESSAGE, DATA, SECURITY, CONNECTION
SECURITY
FIRE, INTRUSION, MONITORING& DETECTING
FIRE, INTRUSION
• MESSAGE requires INTRUSION, FIRE, or FLOOD activated.
• FIRE has higher priority than FLOOD.
• FIRE has higher priority than INTRUSION.
Inclusion
Availability
Activation Rule
Feature Binding State View

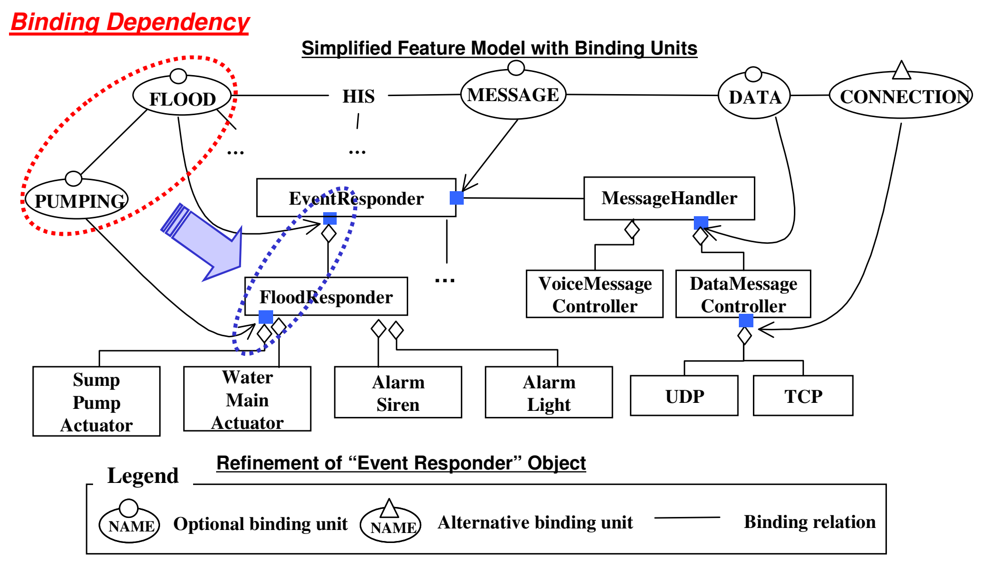
Binding techniques for feature 'availability': Load tables, authentication based access control, etc.

* **Microsoft Developers Network (MSDN), Introduction to Internet Component Download (ICD), http://msdn.microsoft.com/workshop/delivery/download/overview/entry.asp**

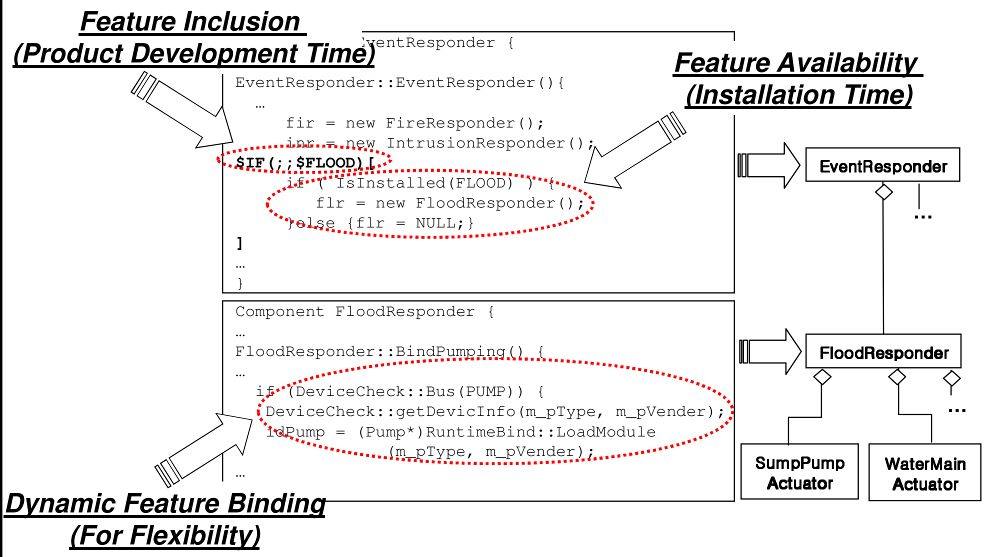
Variation Point Identification

- For feature binding to be feasible, variation points for optional and alternative binding units should be identified in the design object model.
  - We need to be sure that all objects that implement the features of a binding unit are bound together with appropriate implementation techniques.
- To manage variation points of a binding unit consistently, mapping between binding units and variation points should be established.
  - If there is difficulty establishing this relationship, the related objects should be examined for further decomposition, refinement, or restructuring.
  - The binding dependency should also be preserved among variation points.

Variation Point Identification



Component Specification

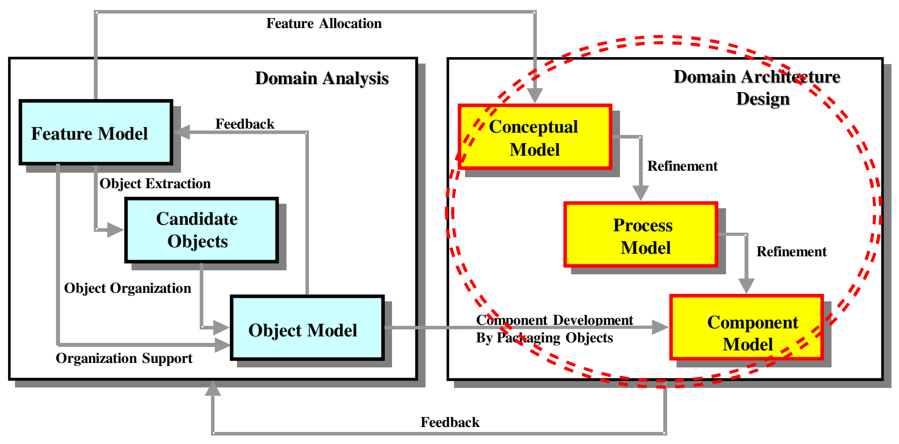


```
ventResponder {

EventResponder::EventResponder(){
  …
     fir = new FireResponder();
     inr = new IntrusionResponder();
$IF(;;$FLOOD)[
     if ( IsInstalled(FLOOD) ) {
        flr = new FloodResponder();
     }else {flr = NULL;}
]
…
}
```

```
Component FloodResponder {
…
FloodResponder::BindPumping() {
…
  if (DeviceCheck::Bus(PUMP)) {
   DeviceCheck::getDevicInfo(m_pType, m_pVender);
   idPump = (Pump*)RuntimeBind::LoadModule
                   (m_pType, m_pVender);
…
```

Component Integration

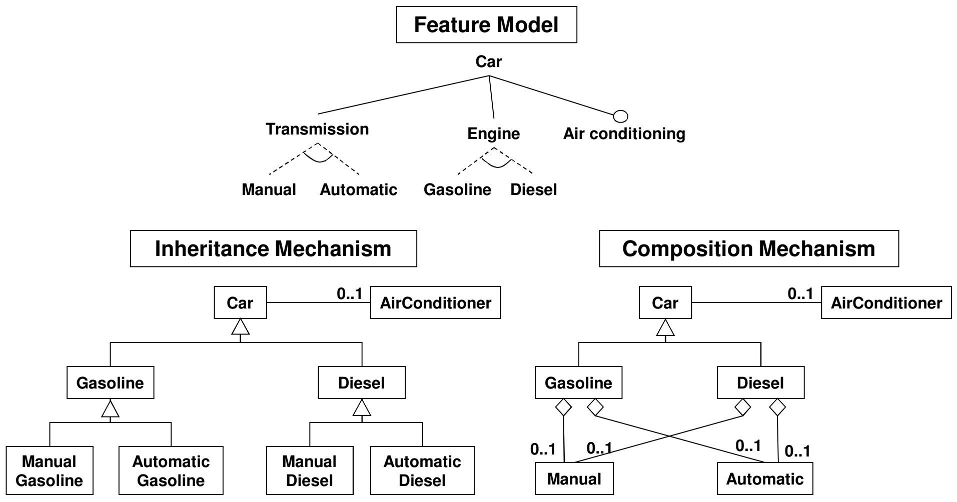


The relationships between the feature model and the architecture model

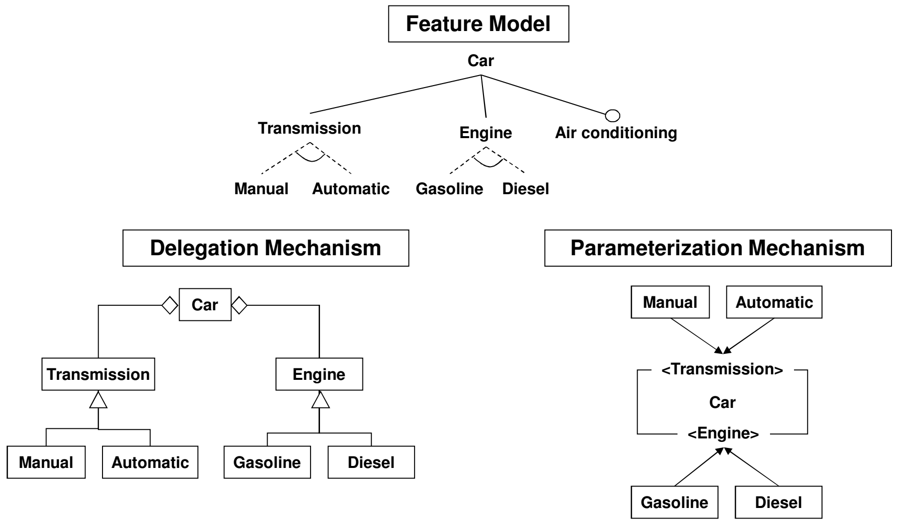
Implementation Techniques (1/3)

- Components encapsulate variations. These components must be refined incorporating product specific features into the components.
- Mechanisms used to implement the variability are:
  - Inheritance
  - Composition
  - Delegation
  - Parameterization
  - Macro processing

Implementation Techniques (2/3)



Implementation Techniques (3/3)



Summary

- Feature binding units are identified and feature binding time is determined with consideration of feature activation rules and market needs.
- Explicit identification of feature binding units, and binding decisions with views on product line lifecycle and feature binding states could clarify requirements for feature binding.
- Feature binding time may change, and it should be explored and be incorporated into product line development.

What's next?

- **Issues:** **management of**
  - **Dependability**
  - **Scalability**
  - **Software composition / decomposition**
  - **…**

- ***Dynamic Software Product Line***
- ***Internet of Things (IoT)***
- ***Cyber Physical System***

Questions,
Comments, …

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