Variability Management in Software Product Line Engineering

Kyo Chul Kang, POSTECH Fellow

International Advanced School on Automotive Software Engineering

March 7-8, 2011
Nagoya, Japan

Pohang University of Science and Technology (POSTECH)

About me

- At the University of Michigan (ISDOS Project)
  - Requirements engineering (PSL/PSA, Meta System)
  - Meta modeling (Meta System, System Encyclopedia Manager)
- At Bellcore and Bell Labs
  - Experienced software reuse issues in industry
- At SEI
  - Developed FODA, a commonality/variability analysis method
- AT POSTECH
  - Extended FODA and created a successor (FORM)
  - Developing a CASE environment (ASADAL)
Variety of automobile engines

- Gasoline Engine
- Diesel Engine
- LPG Engine
- LNG Engine
- Hybrid Engine
- CRDI (Common Rail Direct Injection)
- HTI (High Tech Injection)
- VGT (Variable Geometry Turbocharger)

Images from Cars in India, ATZ Online, Indian Cars Bikes, LPG Auto network, TradeKey, Indimoto, and Zawolzhsky Motorny Zawod

Prologue

- Soft software lives forever; Hard software has no life
- We are developing software as if it is hardware
- Product line software engineering is about making software soft
- Product line engineering shows what software engineering ought to be
Agenda

- Traditional Software Engineering
- Product Line Engineering and Variability Management
- Feature-Oriented Product Line Engineering
- Product Line Adoption Issues

Traditional Software Engineering

- Typical software engineering process
- For single application!

- Requirements analysis
- Design
- Coding
- Validation testing
- Integration testing
- Unit testing
Traditional Software Engineering

- Development of single application system
- Requirements analyst
  - Elicit and specify requirements for a single target system
- Designer
  - Design for the specified requirements

But what happens

- Software maintenance requires
  - Adding new functions
  - Removing existing functions
  - Changing functions
  - Correcting errors and improving performance
  - But
    - Source code modification without refactoring
    - Copy-and-modify reuse
As the results

- Proliferation of versions
- Design and code decay
  - Bad structure
  - Spaghetti code
  - Unused code
  - Brittle code
- Software maintenance can cost as much as 80% of the entire lifecycle cost

The problem

- Software is developed like hardware
- There is little effort to build "Softness" into software
- The traditional approach can no longer support development of products with
  - Diverse market needs
  - Time-to-market pressure
  - Fierce feature competition
Why important
- Functionality of products implemented as software (e.g., more than 10M lines of code for TV products)
- Diverse market needs
- Time-to-market pressure (e.g., new TV models every 3-6 months)
- Software as valuable asset of an organization
  • Accumulated systems development knowledge is packaged as reusable asset

Softness
- Maintainability
- Adaptability
- Portability
- Interoperability
- Reusability
### Related engineering principles and techniques

- **Modularity**
- **Abstraction**
- **“Layering”**
- **David Parnas’**
  - Information hiding
  - Program families
- **Meta programming and application generators**
- **There are many mechanisms to use; How do we know what to “hide”**

### Evolution of Reuse Concepts

<table>
<thead>
<tr>
<th>70’s</th>
<th>80’s</th>
<th>90’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>76 Daffner and Koellmer</td>
<td>84 Davis &amp; Feilser</td>
<td>94 Doug Meloys</td>
</tr>
<tr>
<td>77 Yoshihiro Matsumoto Software Factory</td>
<td>94 Domain Analysis</td>
<td>96 CBSE</td>
</tr>
<tr>
<td>76 David Parnas Program Families</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Code Reuse**
  - Reusable Code Components
  - Object-Oriented Library (e.g., The Boost C++ Component, Hg++, etc.)

- **Design Reuse**
  - Design Patterns
  - Architecture Style
  - Architecture Description Language
  - Domain Analysis
  - CBSE

- **Architecture**
  - Design Patterns
  - Architecture Style
  - Architecture Description Language

- **Software Architecture**
  - DVD Programs

- **Development**
  - Anti Patterns

- **Evolution**
  - Object-Oriented Library
  - Reusable Code Components
  - Design Patterns
  - Architecture Style
  - Architecture Description Language
  - Domain Analysis
  - CBSE
**Agenda**

- Maintainability and Reusability
- **Product Line Engineering and Variability Management**
- Feature-Oriented Product Line Engineering
- Product Line Adoption Issues

---

**Product Line Engineering**

Systematic Reuse in the Context of a Product Line: **“Building softness into software”**

- Product line: “a family of systems sharing a common set of features”
Software Product Line Engineering (SPLE) is an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch.

SEI: Software Product Line Practice

http://www.sei.cmu.edu/plp/essentials/
Celsius Tech

Celsius Tech: Ship System 2000

A family of 55 ship systems

Integration test of 1-1.5 million SLOC requires 1-2 people
Rehosting to a new platform/OS takes 3 months
Cost and schedule targets are predictably met
Performance/distribution behavior known in advance
Customer satisfaction is high
Hardware-to-software cost ratio changed from 35:1 to 80:20

Cummins Inc.

Cummins Inc.: Diesel Engine Control Systems

Over 20 product groups with
over 1000 separate engine applications

Product cycle time was slashed from 250 person-months to a few person-months
Build and integration time was reduced from one year to one week
Quality goals are exceeded
Customer satisfaction is high
Product schedules are met

http://www.sei.cmu.edu/plp/essentials/
Important technical elements of product line engineering

- Commonality and Variability
- Architecture (structure, “bone”)
  - Stability based on common properties
- Variation points and Variants
  - Flexibility based on expected variations
- Encapsulation of design decisions that may change
  - Information hiding, abstraction, etc.

Technical Advances

- Paradigm change
  - From single systems to product line/family
  - “Good software engineering” focusing on maintainability and reusability
- Commonality and variability analysis
  - Feature analysis
- Domain-oriented Architectures and Components (from objects and collaborations)
  - Variation points and variants
  - “high option potentials”
- Domain specific languages and generators
**Basic Principles**

How do we develop product line software?

- **Variability analysis**: Think about what may change!
  - Looking across a family of applications in the product line
    - In different markets
  - Looking ahead for anticipated changes
    - Expected requirements from a market analysis
    - Emerging markets
    - Study of emerging technologies
  - Exploring product usage contexts
    - Both static and dynamic dimensions

- **Building variability into software**
  - Architecture: Variation points and variants
  - Component: Hide (encapsulate) design decisions that may change!
    - Abstract and expose unchanging functional properties as interface!

**Commonality and Variability**

- **Functional Variability**
- **Contextual Variability**
- **Binding Time Variability**
- **Evolution of Functional and Contextual Variability**

![Spatio-Temporal Space Diagram](image-url)
### Feature Model Example: HIS Product Line

#### Functional Variability

**Capability Layer**
- Security
- Intrusion
- Fire
- Flood
- HMI
- Standard
- Advanced

**Operating Environment Layer**
- Motion
- Alarm
- Door
- Smoke
- Moisture
- Water

**Domain Technology Layer**
- Monitoring & Detecting
- Discrete Value
- Continuous Value

**Implementation Technique Layer**
- Connection
- TCP
- UDP

**Composite Rules**
- Flood requires Moisture Sensor.
- Pumping requires Sump Pump.
- Message requires Communication.
- Data requires Internet.

**Composition Rules**
- Optional feature
- Composed-of relationship
- Generalization relationship
- Implemented-by relationship

### Contextual Variability

**Product contexts**
- Different/evolving operating environments (e.g., technologies)
- Different legal and cultural constraints
- Different marketing strategies
- Different evolving market needs
- Dynamically changing usage contexts
Usage Context Example: Elevators

Contextual Variability

Safety & Comfortability

Capacity

Safety & Capacity

Passenger Elevator

Hospital/Bed Elevator

Freight Elevator

Three Perspectives of Feature Binding

Binding Time Variability

What features will be included in the products?

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

Title: Variability Management in Software Product Line Engineering
© 2011 by Pohang University of Science and Technology
Department of Computer Science and Engineering
Software Engineering Laboratory, R.O. Korea
Agenda

- Maintainability and Reusability
- Product Line Engineering and Variability Management
- Feature-Oriented Product Line Engineering
- Product Line Adoption Issues

Feature-Oriented Domain Analysis (FODA)
Feature-Oriented Domain Analysis (FODA)

Number of Citations

1990–2010
Total: 2050
Unknown/Overlapping: 804

2050 citation counts!
(February 1, 2011)
Feature-Oriented Domain Analysis (FODA)

The core of FORM lies in the analysis of domain features and the use of these features to develop reusable and adaptable domain artifacts.
Feature-Oriented Product Line Engineering

FORM (Feature-Oriented Reuse Method)

Product Line Asset Development Process

Product Development Process

Marketing and Product Plan (MPP)

Different cultural traits and legal constraints

Different computing knowledge backgrounds

Different computing environments

Title: Variability Management in Software Product Line Engineering
© 2011 by Pohang University of Science and Technology
Department of Computer Science and Engineering
Software Engineering Laboratory, R.O. Korea
Marketing Plan

A marketing plan includes a market analysis with an assessment of the market, and a marketing strategy with a plan for realizing the business opportunities with products that meet the needs.

The market analysis includes:
- need assessment
- customer profiles
  - end-user skill levels
  - cultural and legal constraints
- business opportunities
  - price range
  - time to market

The marketing strategy may initially include:
- an outline of product delivery methods: how the products will be delivered to customers

---

Elements of MPP

Product Line Initiation

**Marketing Plan (Business Concerns)**

*Market analysis*
- Market segment
  - Needs assessment
  - User profile
  - Cultural and legal constraints
- Business opportunities
  - Time to market,
  - Price range, etc.

*Marketing strategy*
- Product delivery methods
- Other business considerations

**Product Plan (Engineering Concerns)**

*Product features*
- Product functional features
  - Feature lists
  - Feature description
- Non-functional features
  - Usability, scalability, etc.

*Product features delivery methods*
- Feature coverage
- Feature binding time
- Feature binding techniques
Product Plan

Once the marketing plan has been defined, it is important to spend some effort on identifying the characteristics of products in a product line in terms of features and developing a plan for incorporating features.

The product plan includes

- Product features:
- Product feature-delivery methods:
- Feature release plan for future:
### Feature Model Example: HIS Product Line

#### Product Line Initiation

**Marketing and Product Plan for HIS PL**

<table>
<thead>
<tr>
<th>Market Segments</th>
<th>Office building: High-End (HE) product</th>
<th>Household: Low-End (LE) product</th>
</tr>
</thead>
<tbody>
<tr>
<td>User/Maintainer Profile</td>
<td>Dedicated engineers with computer science backgrounds.</td>
<td>No computer knowledge is assumed.</td>
</tr>
<tr>
<td>Legal Constraints</td>
<td>Emergency control services must conform to codes of each country.</td>
<td>Emergency control services must conform to codes of each country.</td>
</tr>
<tr>
<td>Feature Delivery Method</td>
<td>Feature selection from a predefined set of features (Feature Selection Method)</td>
<td>Prepackaged Method</td>
</tr>
<tr>
<td>Product Features</td>
<td>Fire, Intrusion, Flood, Security, and other customer specific features</td>
<td>Fire, Intrusion, Flood</td>
</tr>
<tr>
<td>Quality Attributes</td>
<td>Safety, Reliability, Scalability</td>
<td>Safety, Reliability, Scalability, Usability</td>
</tr>
</tbody>
</table>

#### Product Feature Binding Time

- Product Delivery Time
- Product Build Time

#### Feature Model Example: HIS Product Line

**Product Line Analysis**

```
Capability Layer
  - Security
  - Intrusion
  - Fire
  - Flood
  - Administration
  - HM
  - Standard
  - Advanced
  - Motion
  - Alarm
  - Smoke
  - Water
  - Moisture
  - Monitoring & Detecting
    - Discrete
    - Continuous

Domain Technology Layer
  - Operating Environment
    - Monitoring & Detecting
    - Sensing Devices
    - Actuating Devices

Implementation Technology Layer
  - Optional feature
  - Alternative feature

Composition Rules
- Flood requires Moisture Sensor
- Pumping requires Sump Pump
- Message requires Communication Data

Redundancy Control
- Active
- Standby
```

**Quality Attributes**
- Safety
- Scalability
- Usability
- Fault Tolerance

**Composition Rules**
- Flood requires Moisture Sensor
- Pumping requires Sump Pump
- Message requires Communication Data

**Redundancy Control**
- Active
- Standby
Object Identification Example from the Feature Model

Feature Model

Identified Candidate Objects

Design Object Model Example
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

Feature Binding Units

Legend:
- Optional feature
- Alternative feature
- Composed-of relationship
- Generalization relationship
- Implemented-by relationship
Variation Point Identification

Binding Dependency

Simplified Feature Model with Binding Units

Legend

Refinement of “Event Responder” Object

Architecture Design

Conceptual Architecture
Product Development

Product development is a process of developing a specific product making use of the product line asset developed during the product line asset development process.

Product development proceeds by:
- analyzing user's requirements,
- selecting appropriate and valid product line features from the feature model,
- identifying the corresponding architecture models,
- completing the product development by reusing software components, and
- adapting components as needed.

Feature Selection for LE-HIS

with the fault tolerant and without the flood feature

<table>
<thead>
<tr>
<th>Capability Layer</th>
<th>Feature Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>Fire</td>
</tr>
<tr>
<td>Intrusion</td>
<td>Flood</td>
</tr>
<tr>
<td>Detection</td>
<td>Action</td>
</tr>
<tr>
<td>Door</td>
<td>Motion</td>
</tr>
<tr>
<td>Smoke</td>
<td>Smoke Sensor</td>
</tr>
<tr>
<td>Sump</td>
<td>Pumping</td>
</tr>
<tr>
<td>Water Main</td>
<td>Water Sensor</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operating Environment Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
</tr>
<tr>
<td>Direct</td>
</tr>
<tr>
<td>Event-based</td>
</tr>
<tr>
<td>Scheduled</td>
</tr>
<tr>
<td>Periodic</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Domain Technology Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion Detection</td>
</tr>
<tr>
<td>Continuous</td>
</tr>
<tr>
<td>Discrete</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implementation Technique Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
</tr>
<tr>
<td>UDP</td>
</tr>
</tbody>
</table>

Composition Rules
- Flood requires Smoke Sensor
- Pumping requires Sump Pump
- Message requires Communication
- Data requires Internet

Quality Attributes
- Usability
- Safety
- Fault tolerant
- Scalability

Selected Feature
- Moisture Sensor
- Pumping
- Communication
- Internet
- TCP
- UDP
- Active
- Standby
- Optional feature
- Alternative feature

Composition of relationship
- Generalization relationship
- Implemented-by relationship

Title: Variability Management in Software Product Line Engineering
© 2011 by Pohang University of Science and Technology
Department of Computer Science and Engineering
Software Engineering Laboratory, R.O. Korea
### Feature Selection for LE-HIS

**without** the fault tolerant and **with** the flood feature

#### Conceptual Architecture Selection

**With the Fault Tolerant and Communication features, without the Flood feature**

**With the Flood feature, without the Fault Tolerant and Communication features**

---

**Legend**

- Optional Feature
- Black-box Reuse
- White-box Reuse
- Alternative Features
- Shared Features
Agenda

- Maintainability and Reusability
- Product Line Engineering and Variability Management
- Feature-Oriented Product Line Engineering
- Product Line Adoption Issues

Adoption Issues

- How to change to PL-based organization
  - How to evolve: staged process model for reuse adoption
  - Key process areas
    - Best practices
  - Metrics
    - Key indicators: cost of production, time to market, project completion time, etc.
    - Relationship between reuse, quality, and productivity
    - Relationship between reuse and ROI for sustainability of a reuse program

Process models

- Proactive vs. reactive vs. extractive models
  - Best practices
- PL process vs. agile methods
Adoption Issues

Variability Management

Discovery and Modeling
- Functions
- Non-functional attributes
- Usage and operating contexts

Configuration
- Decision model
- Rationales
- Goals and Issues

Management
- Centralized
- Distributed

Adoption Issues

ROI analysis
- Estimating ROI from a reuse program
- Estimating benefits from strategic market position

Asset management (How to make PL-based development happen in an organization)
- Who should develop assets (with variation points)
- Who should maintain assets (variation management)
- Who will be responsible for quality assurance
- Who should enforce the use of assets
- Models (best practices)
  - Centralized vs. distributed
For good software engineering, maintainability and reusability must be built into software while designing!

For softness, design must base on commonalities and variations of a product line

- Think about what may change!
  - Look across similar applications
  - Look ahead for anticipated changes
  - Look at product contexts as well as functionality

Thank you!