Product Line Engineering Lecture – PL Approaches and Economy

Dr. Martin Becker
martin.becker@iese.fraunhofer.de
## Schedule - Lectures

<table>
<thead>
<tr>
<th>Date</th>
<th>Content</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-Oct-10</td>
<td>Introduction</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>5-Nov-10</td>
<td>Scoping</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>12-Nov-10</td>
<td>PL Infrastructure I (Variability Modelling)</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>19-Nov-10</td>
<td>PL Infrastructure II (Variability Realization)</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>26-Nov-10</td>
<td>no lecture</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>3-Dec-10</td>
<td>Configuration Management</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>10-Dec-10</td>
<td>PL Economics and Approaches</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>17-Dec-10</td>
<td>Requirements Engineering</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>7-Jan-11</td>
<td>PL-Architectures I</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>14-Jan-11</td>
<td>PL-Architectures II</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>21-Jan-11</td>
<td>Component Engineering</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>28-Jan-11</td>
<td>Quality Assurance</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>4-Feb-11</td>
<td>Organizational Issues / Adoption</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>11-Feb-11</td>
<td>Reengineering / Variant Analysis</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
</tbody>
</table>
### Schedule - Exercises

<table>
<thead>
<tr>
<th>Date</th>
<th>Content</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.11.2010</td>
<td>Scoping, Variability Modeling</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>10.12.2010</td>
<td>VM Realization, Configuration Management</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>14.01.2011</td>
<td>PL Architectures</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>21.01.2011</td>
<td>Component Engineering</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>11.02.2011</td>
<td>Adoption, Variant Analysis</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
</tbody>
</table>
--- Recap

Product Line Infrastructure
Part III: Configuration Management ---

How to realize variability resolution support?
How to manage variants?
Configuration Management to the rescue

- **Software Complexity**: „what is the impact of change X“?
- **Software Evolution**: „which version worked on date X“?
- **Process Complexity**: „who is allowed to perform change X“?

Configuration Management controls
Configuration Management Basic Functions

- **Identification**
  - Identify what elements need to be controlled
  - Identify what elements are already being controlled

- **Control**
  - Control the way a system evolves

- **Status accounting**
  - Obtain information about the way a system evolves

- **Audit**
  - Assess the fulfillment of requirements
Main Concept: Configuration

Configuration

Baseline  Tag  Release

Element

Version

Component

prog.c  lib.c  msg.cat  util.h
CM as an Variability Mechanism

- Concurrent Version System (CVS)
- Configuration selection by name
- Configuration selection by date
- Client
- Different configurations
- Mainly Releases

Image: TortoiseCVS - Update window with list of configurations

Options:
- Get tag/branch/revision
- Get date/time
- Clean copy - WARNING: this will overwrite your local files
- Create missing subfolders
- If no matching revision is found
- Simulate update

Configuration list:
- FIREFOX 2.0.0 8 RC2
- FIREFOX 2.0.0 8 RELEASE
- FIREFOX 2.0.0 9 RC1
- FIREFOX 2.0.0 9 RELEASE
- FIREFOX 2.0 MINIBRANCH
- FIREFOX 2.0 RELEASE
- FIREFOX 2.0a 1 MINIBRANCH
- FIREFOX 2.0a 1 RC1
- FIREFOX 2.0a 1 RELEASE
- FIREFOX 2.0a 2 MINIBRANCH
- FIREFOX 2.0a 2 RELEASE
- FIREFOX 2.0a 3 MINIBRANCH
- FIREFOX 2.0a 3 RELEASE
- FIREFOX 2.0b 1 MINIBRANCH
- FIREFOX 2.0b 1 RC1
- FIREFOX 2.0b 2 MINIBRANCH
- FIREFOX 2.0b 2 RELEASE
- FIREFOX 2.0b 3 MINIBRANCH
- FIREFOX 2.0b 3 RELEASE
Variability Management and Evolution

Single System Development

Changes

Synchronization

Engineering

Variability

Product Line Development

Changes

Synchronization

Engineering
Management of Product Line Variation

Variation in Space x Time x Lifecycle

[source: Bigleveer: The Systems and Software Product Line Lifecycle Framework]
Configuration Identification

- Define the elements that need to be controlled
  - The controlled elements are called **Configuration Items (CIs)**

- Aspects of identification
  - Selection (what is a CI, what is not a CI)
  - Structure (CI hierarchies)
  - Naming (unique identification)
  - Characterization (further meta data)
  - Access (CI server location, access rules)
**Software Architecture as a structuring mechanism**

Nesting of architectural elements can guide the creation of directories and files in CM

Configuration item candidates

Dependencies → Bottleneck in conventional config. management
Configuration Control

- Main component of configuration management

- Configuration Control areas
  - Version management
  - Change management
  - Build management
  - Release management
Version Management

Main line

Branch

Revision

Merge
Problems with Parallelism (1/2)

Unclear Dependencies
Solution: Layer on top of configuration management

- Product Line Engineer
- Customization Layer
  - PLE evolution control scenarios
  - Storage and controlled evolution of versioned artifacts
- Configuration Management
--- PL Approaches ---

What approaches can be followed for PLE?
Achieving Systematic Reuse

Tacit

Explicit

Public (shared)

Private (unique)

Systematic Reuse
Product Line Strategy

Overall Strategy

- **Degree of Commonality and Variability Support (How much?):**
  - Platform, Product Line, Configurable Product Base
- **Lifecycles (How coordinated?):**
  - Product Line: Independent Lifecycles of FE and AE\(<N>\)
  - Production Line: Single Lifecycle of FE and AE\(<N>\)
Degree of Commonality and Variability Support

- independent products
- standardized infrastructure
  - operating system
  - commercial components, such as DBMS, GUI
- platform
  - develop and maintain a platform based on which the products or applications are build
  - Infrastructure + all functionality that is common to all products or applications
  - increase the amount of functionality in the platform to the level where functionality common to several but not all products becomes part of the shared artefacts
- software product line
- configurable product base
- program of product lines
  - very large systems:
    - software architecture is defined
    - several or all of the components are software product lines
  - many products in relatively stable domains:
    - mature the support for product derivation
    - one configurable product base that, either at the organization

Product Line

- FE and AE artefacts are managed separately
- High degree of independence of FE and AE
  - ➔ different organizational units
  - ➔ different locations
- Increase of overall artefact base
- Manage the interdependencies between the different artefacts
- N+1 Configuration Management Problems
Complexity of Product-centric Thinking Impedes Portfolio Production

Order $N^2$ Complexity

“Vertical” Product Perspective
Production Line

- almost no AE activities
- AE activities are wrapped up in the FE activities
  - evolve the omni potent reuse infrastructure
  - All applications can be derived completely and automatically
  - vast majority of work products that have to be managed are FE work products
  - Application models are the only application specific work product
- reduces the n+1 dimensional configuration management problem to the well known 1 dimensional problem
Shift in Perspective: Efficient Means of Production

(source: Biglever. Provided courtesy of BigLever Software)
Shift in Perspective: Efficient Means of Production

Feature-based Abstraction

Simplicity of a Single Automated Production Line

Requirements Engineers

Architects

Developers

Quality Assurance

Reusable SPL Assets

Profile A

Feature Profiles

Product Line Management

Design Models

Source Code

Test Cases

Gears Product Configurator

Key

Variation Points

[source: Biglever. Provided courtesy of BigLever Software]
Benefits of a Software Production Line

Economy of Scale from Automated Production
- Increase in the scope of product diversity
- Increase in the scale of different products effectively delivered and maintained

Cost Savings from Efficiency and Productivity
- Increase in productivity and efficiency
- Reduction in per-product development cost and overhead
- Higher profit margins

Faster Profits from Faster Time to Market
- Reduction in time-to-market for new and updated products
- Increased agility to react to new opportunities and changing market conditions

Better Products from Better Quality
- Increase in customer-perceived product quality
- Reduction in defect density
- Improved risk management

[source: Biglewer. Provided courtesy of BigLever Software]
Product Line Strategy: Per Domain

Decision made per domain assessed
- Relative ranking

Strategies
- Order of supported domains (When?)
  - Start with most promising areas

Note: early successes are crucial while migrating to product line engineering
Product Line Strategy: Per Domain II

- Variability Adoption (How introduced?)
  - Proactive (aka. Revolution; predictive)
    - Look ahead, define the product line’s scope proactively
    - Learn all you can from domain analysis
  - Reactive (aka. Evolution; agile)
    - Start with 1-2 products
    - React to new customers as they arrive
  - Extractive (reengineering-driven)
    - Extract commonality from existing products
    - Form common asset base from what you already have

PL Strategy is determined by domain strategies and must match business objectives
### Strategy Decision: Revolution vs. Evolution

<table>
<thead>
<tr>
<th>Evolution</th>
<th>Revolution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
<td><strong>Con</strong></td>
</tr>
<tr>
<td>- smaller delay for first product</td>
<td>- influence of first product:</td>
</tr>
<tr>
<td>- smaller initial investment</td>
<td>- Lack of genericity</td>
</tr>
<tr>
<td>- smaller risk of investments</td>
<td>- inadequate architectural decisions</td>
</tr>
<tr>
<td></td>
<td>- long term effort is higher</td>
</tr>
<tr>
<td></td>
<td>- stronger architectural erosion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revolution</th>
<th>Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pro</strong></td>
<td><strong>Con</strong></td>
</tr>
<tr>
<td>- possibility to implement company strategy</td>
<td>- delay for being on the market</td>
</tr>
<tr>
<td>- Balancing of multiple products</td>
<td>- higher risk (if strategy is wrong)</td>
</tr>
<tr>
<td>- Potential of establishing a high-throughput product line</td>
<td>- higher upfront investment is necessary</td>
</tr>
</tbody>
</table>
Guidelines for Choosing Your Strategy – Pro Evolution

- Limited resources or buy-in
- Some deadlines fixed in the near-time
- Unclear product plans
- Insufficient knowledge in sub-domain
- Changing technologies in sub-domain
Guidelines for Choosing Your Strategy – Pro Revolution

- Too many products close up
- Need to enter a new market segment
- Need to build on reengineered assets
- A large number of variations is foreseeable in a sub-domain
- Sub-domain is well known and understood
Combination of Domain Strategies

Theory (R/E extremes at PL level)

- - revolution
- - - evolution

PL Scope supported by infrastructure

Time (in delivered single systems)
Company X is about to enter a new market
Once into the market it is clear that:
- customer request may not be turned down
- strong competitors may appear if the market turns out to be profitable
- limited man power
Case Study A – Revolution Dominance (2/2)

Decisions made
- Invest in a transition and infrastructure build up phase
- Explicit analysis of technology, competitors, and future products (scoping)
- Define a generic architecture!
- Develop generic assets

Results
- Infrastructure set up: <2 times the time of building single product
- Now: products can be developed in 1/10th of the time of competitors; sometimes delivery within a week
Case Study B – Evolution Dominance (1/2)

Company B is already positioned in their market with several variants. Once into the market it is clear that:

- customer request always new and unexpected variations (i.e., expected variation saturation will not occur)
- customization and maintenance gets more and more complex
- limited man power but project pressure
Case Study B – Evolution Dominance (2/2)

Decisions made
- introduce technology for managing variations explicitly and systematically
- refactor components while introducing new variations or during maintenance
- coordinate refactoring with normal development and maintenance activities

Results
- no special PLE phases
- now: more time spent on the development of new features
--- PL Economics ---

How much will the PL cost?
What is the ROI?
Systematic Reuse Promises …

- up to 90% of reuse of existing solutions (Basset)
- reduction of TTM
  - from 1 year to 1 week (Cummins)
  - by factor of seven (Hewlett-Packard)
- productivity improvement
  - 400% (Motorola)
  - factor six (Hewlett-Packard)

but there are also costs and risks to be considered
Starting a product line strategy requires an upfront investment higher than using single system approach.

The higher the commonality the sooner the payback.

Rule of thumb: Savings begin between 2nd and 3rd product.

Rule of thumb: Investment ranges between development efforts for 1 and 2 systems.

[Weiss/Lai]
Cost Components of Product Lines

- $C_{org}$ – Education and training
- $C_{cab}$ – Investment into PL infrastructure (core asset base)
- $C_{unique}$ – Development of product-specifics
- $C_{reuse}$ – Reuse of PL artifacts

$Cost = C_{org} + C_{cab} + \sum_{i=1}^{n} (C_{unique}(p_i) + C_{reuse}(p_i))$

- Varies with the number of products
- Upfront investment, fixed costs

- $n =$ Number of products of a product line
Education and training

- People have to be educated about PL concepts
- PL approach changes some sub-processes of the software development process
  - How to instantiate a component?
  - How to request changes in a core asset?
- Understand Development FOR reuse (Domain Engineering) x Development WITH reuse (Application Engineering)

\[ Cost = C_{org} + C_{cab} + \sum_{i=1}^{n} (C_{unique}(p_i) + C_{reuse}(p_i)) \]
Creation of PL Infrastructure

- Before start reuse, it is necessary to create the core assets

\[
\text{Cost} = C_{\text{org}} + C_{\text{cab}} + \sum_{i=1}^{n} (C_{\text{unique}}(p_i) + C_{\text{reuse}}(p_i))
\]
Reuse costs

- In order to be reused, a core asset needs to be:
  - Adapted
  - Configured
- New features may require changes in the core asset
  - Impacts in the whole product line should be analyzed before any changes

\[
Cost = C_{org} + C_{cab} + \sum_{i=1}^{n} (C_{unique}(p_i) + C_{reuse}(p_i))
\]
Product Specific costs

- Development of product specific components
- Tests

\[ Cost = C_{org} + C_{cab} + \sum_{i=1}^{n} (C_{unique}(p_i) + C_{reuse}(p_i)) \]
The „Classical“ Scenario

Cprod ~ 12 PY
Ccab 150% x 70% x Cprod + Cscoping ~ 17 PY
Fcab 10%
Creuse 70% x Fcab x Cprod ~ 0.8 PY
Cunique 30% x Cprod ~ 3.5 PY
Corg 8 weeks per person * 30 people
= 240 person weeks ~ 6 PY

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

- How sensitive are cost prediction of the economic model wrt. input parameters?
- Is product line still attractive option, if, for example:
  - Commonality level is “only” 60% instead predicted 70%
  - If the additional effort for developing software reusable is 175% instead of 150%
  - If the # of future products is 10 instead of 15

[Ganesan, Muthig, Yoshimura @ SPLC 2006]
ROI after 3rd Product?*

There is 80% certainty for the ROI in the range of 100% to 150% after 3 products

*Based on Monte-Carlo Simulation
Number of Product Line Members?

- Implicit assumption: ROI proportional to # of products derived from a product line
  - Inherently build into economic model!

- Practical experience, however, tells us that cost for PL members grows over time

- Sensitivity analysis identifies most critical variables of ROI prediction
  - # of products (!)
  - Additional effort for making software reusable
Product Line Generations

- Due to infrastructure degeneration, reuse becomes more expensive
  - More adaptation effort
  - More product-specifics

- Need for reinvestment too keep product line viable (switching to next PL generation)
  - Continuous investment corresponds to single product generations
Incremental product line development enables better risk control.

Dimensions of incrementality:
- Products
- Functionality

Scoping as an Economics-Driven Process
Scoping

**Which benefits can be expected from planning:**
- Early analysis (estimation) of benefits associated with product line development
  - Appropriate go-/no-go decision (business case)
  - Detailed break-down of what earns you which benefit
- How should we phase incremental product line development to gain optimal benefits?

**Early risk detection**
- What risks are involved when performing product line development?
- Which sub-domains are most problematic?
- How can difficulties and risks be addressed?
Scoping Result

The key result of planning is:

- **What should we make reusable?**
- What should be regarded part of a product family? (which products / which features)
- What parts of the product line (sub-PL) is most appropriate for product line development?
- Which functionality / components should be developed for reuse? (→ architectural trade-offs)
Further Reading


