Fall 2007

Systems Engineering Process Applied to Software Product Line Methods

David Pierre
Loyola Marymount University

Follow this and additional works at: http://digitalcommons.lmu.edu/se_etdrps

Part of the Systems Engineering Commons

Recommended Citation
http://digitalcommons.lmu.edu/se_etdrps/11

This Oral Presentation - Campus Accessible Only (with IP restrictions) is brought to you for free and open access by the Systems Engineering at Digital Commons @ Loyola Marymount University and Loyola Law School. It has been accepted for inclusion in Systems Engineering Research Projects and Oral Presentations by an authorized administrator of Digital Commons@Loyola Marymount University and Loyola Law School. For more information, please contact digitalcommons@lmu.edu.
Systems Engineering Process applied to Software Product Line Methods

SELP 695: Systems Engineering Integrative Project

December 10 2007
David Pierre
Loyola Marymount University & Northrop Grumman Corporation
Agenda

- Integrative Project Objectives
- Background / Overview
- Software Product Lines
- BCPL's Architecture
- BCPL's Development Process
- BCPL's Requirement Process
- Risk Management
- Findings / Results
- Summary / Conclusion
Integrative Project Objectives
Integrative Project Objectives

- Research Systems Engineering (SE) efforts on Battle Command Product Line (BCPL) program

- Analytically look at BCPL’s Software Life Cycle as it relates to Software Product Lines (SPL).

- Compare traditional SE methods to SPL’s Product Line Analysis model

- Provide quantitative metrics of BCPL’s cost savings using SPL methods
Background / Overview
Background – FBCB2

• On-the-Move Tactical Command & Control system
  - Situational Awareness – Where am I? Where are my buddies? Where is the enemy? Where are danger zones?
  - C2 Messaging – Operational Orders, Call for Fire, Enemy Reports, Medical Evacuation, Free Text, ...

Recent Recognition

2002: Top 5 Quality Software Project

2003: Network Centric Warfare
    Most Innovative U.S. Government Program

2004: Monticello Award
    The information system having the most direct and meaningful impact on human lives

2005: Battlespace Information Award (UK)
    Best Program in Support of Coalition Operations

• Fielded in three theaters for peacekeeping missions and combat operations
• Combat proven performance in Operations Enduring/Iraqi Freedom (OEF/OIF)
  - "FBCB2 is a winner! The system saved lives...”,
  - "FBCB2-BFT delivered on the move command and control at unprecedented speed and distance..."
**BCPL Evolved From FBCB2**

- In June 2001, GEN Kern commissioned a study of the FBCB2 software architecture to ascertain if FBCB2 was on the "right path" to support the Objective Force
  - The Software Engineering Institute (SEI) studied FBCB2 to assess the short term status and long term viability of the FBCB2 architecture
  - In May 2002, the architecture study findings were presented to ASA/ALT, army staff, and FBCB2 stakeholders

**Recommendation**
- Adopt a **product line** approach for FBCB2
- Adopt an **architecture-driven** approach for FBCB2

**FBCB2 Architecture Study Findings**

1. Rapidly evolved software architecture intended to match rapidly evolved requirements
2. Driven by functionality with no provision for non-functional qualities
3. In the future the non-functional qualities will become more important
4. There is a potential for a larger scope of supported products than currently supported
5. To support future forces and future growth, FBCB2 needs a robust architecture that can readily incorporate new capabilities and support long-term modifiability and sustainability
Software Product Lines
NG produces families of similar systems, such as satellites, that are differentiated by features.
Software Product Lines – What are they?

- A set of software-intensive systems that:
  - Share a common, managed set of features
  - Satisfy the specific needs of a particular market segment or mission
  - Are developed from a common set of core assets in a prescribed way
  - Take economic advantage of commonality and bound variability
Precursors to Software Product Lines

Reuse History: From Ad Hoc to Systematic

- **1960s**: Subroutines
- **1970s**: Modules
- **1980s**: Objects
- **1990s**: Components
- **2000s**: Software Product Lines
How Do Product Lines Help?

While a product line approach requires additional investment and rigor up front to develop a reusable library of core assets, the result is a significant reduction in total life-cycle costs.

Benefits include:
- improved productivity
- decreased time-to-market (time-to-field)
- decreased cost
- increased product quality
- increased customer satisfaction
- ability to effect mass customization
- ability to maintain market presence
- ability to sustain unprecedented growth
Software Product Lines

• Are NOT
  • Fortuitous small-grained reuse
  • Single-system development with reuse ("clone and own")
  • Just component-based development
  • Just a reconfigurable architecture
  • Releases and versions of single products
  • Just a set of technical standards

• Are
  • Strategic, large-grained reuse
  • Driven by a technical and a business strategy
  • Systematic use of core assets* to build the multiple products that constitute a software product line

* Core Assets are those reusable artifacts & resources that form the basis for the software product line.
Key Concepts

Use of a common asset base

in production

of a related set of products

Architecture

Production Plan

Scope Definition Business Case
SPL Economics

With Product Line Approach

Current Practice

Cumulative Costs

Number of Products

*Software Product-Line Engineering: A Family-Based Software Development Process* 

© 2006 Carnegie Mellon University
Not just generic software

- Unlike generic software assets—which are designed to accommodate as wide a gamut of products as possible, including those that were never anticipated—core assets, in contrast, are designed precisely and exactly to accommodate the amount of variation needed to support products that are anticipated, because they are in the BCPL scope.

- This is what distinguishes the strategic and systematic reuse of core assets from the ad-hoc and opportunistic reuse of generic software components.

- Additionally, generic software assets are, by necessity, more complex than core assets, and as a result more difficult to use efficiently.
Not just a technical strategy

- The decision to promote a product asset to core must be calculated, and involve both a business, as well as a technical, strategy
  - As a result, requires customer input to the business case for doing so
The BCPL Process Description Reflects all 29 SPL Practice Areas

Determine the Need for a Product Line
- Market Analysis
- Understanding Relevant Domains
- Building a Business Case
- Scoping

Plan the Product Line
- Operations
- Organizational and Technical Planning
- Structuring the Organization
- Funding
- Organizational and Technical Risk Management
- Developing an Acquisition Strategy
- Process Definition
- Tool Support
- Training
- Configuration Management

Develop the Product Line
- Requirements Engineering
- Architecture Definition
- Make/Buy/Mine/Commission Analysis
- COTS Utilization
- Mining Existing Assets
- Architecture Evaluation
- Component Development
- Software System Integration
- Testing

Operate and Maintain the Product Line
- Launching and Institutionalizing
- Data Collection, Metrics, and Tracking
- Customer Interface Management
- Technology Forecasting

Plan and Develop the Product
- Planning
- Requirements Engineering
- Architecture Definition
- Architecture Evaluation
- Component Development
- Software System Integration
- Testing
## Associated Practice Areas

### Establish Product Context
- Market Analysis
- Understanding Relevant Domains
- Technology Forecasting
- Building a Business Case
- Scoping

### Establish Production Capability
- Requirements Engineering
- Architecture Definition
- Architecture Evaluation
- Mining Existing Assets
- Component Development
- COTS Utilization
- Software System Integration
- Testing

### Operate Product Line
- Requirements Engineering
- Architecture Definition
- Architecture Evaluation
- Mining Existing Assets
- Component Development
- COTS Utilization
- Software System Integration
- Testing

### Process
- Process Definition
- Configuration Management
- Tool Support
- Data Collection, Metrics and Tracking
- Technical Planning

### Organization
- Launching and Institutionalizing Funding
- Structuring the Organization
- Operations
- Organizational Planning
- Customer Interface Management
- Organizational Risk Management
- Developing an Acquisition Strategy
- Training
- Launching and Institutionalizing Funding
- Structuring the Organization
- Operations
- Organizational Planning
- Customer Interface Management
- Organizational Risk Management
- Developing an Acquisition Strategy
- Training
- Data Collection, Metrics and Tracking
- Technical Risk Management
Product Line Terminology - 1

- **Core Assets**
  - Those reusable artifacts & resources that form the basis for the software product line.
  - Core Assets are **not** just software components.
  - Core Assets include:
    - architecture
    - reusable software components
    - domain models
    - requirements (SSS, S/SDD, SRS, SDD)
    - documentation & specifications
    - performance models
    - schedules
    - budgets
    - test plans & test cases
    - work plans
    - process descriptions
Variation points

- While the products that the BCPL produces are similar, they are not the same; the extent to which BCPL products may vary are explicitly captured as variation points.

- A variation point represents a calculated decision to delay a design decision until product development; this is done to provide sufficient variation to support the breadth of products that are in the BCPL scope.

Attached Process

- Specifies how a core asset is to be used in the development of actual product. 1-to-1 relationship.
BCPL’s Architecture
BCPL Software Architecture – Overview

- Also known as the Product Line Objective Architecture (PLOA)
- A software reference architecture for the BCPL
  - Designed to support every product that is within the BCPL scope
  - Instantiated for each product
- Represents the combined concerns of a large community of stakeholders
  - Both internal and external to NGMS
- Developed in collaboration by Northrop Grumman and CMU SEI
- Architecture is what will permit or preclude a system from achieving its quality goals
BCPL Architecture Objectives

- FBCB2 as a successfully fielded product has been given a golden opportunity to restructure itself
  - Lay a new foundation on which to rebuild and grow
  - Establish a Product Line with a view toward future enhancements and extensions based on proven FBCB2 capabilities
  - Carry forward 10 years of development and lessons learned
  - Consolidate experience into refined modularized adaptable system
  - Incremental development of architecture infrastructure

Overall Goal: Minimize Cost and Effort to Continuously Synchronize FBCB2 Platform
Product Line Architecture

The product line architecture is the foundation of everything.
Why is Software Architecture Important?

Represents *earliest* design decisions

- hardest to change
- most critical to get right
- communication vehicle among stakeholders

First *design* artifact addressing

- performance
- modifiability
- reliability
- security

Key to systematic *reuse*

- transferable, reusable abstraction

The **right architecture** paves the way for system **success**.
The **wrong architecture** usually spells some form of **disaster**.
BCPL Software Architecture – Layered View

- Comm Manager
- Node/System Management Partitions
- Device Agents
- Message Agents
- Application Partitions

- Convenience Routines, System Services, & Generic Utilities
- Node Data Broker
- OS Facade
- COTS / GOTS / NDI
- Operating System Kernel
- Device Drivers

- Hardware Devices

Infrastructure Core
Abstraction Layers & Categories
BCPL’s Domain Analysis / Requirement Process

(Product Line Analysis)
Product Line Analysis Concepts (1 of 2)

- Product line analysis (PLA) is an iterative, incremental process of eliciting, analyzing, specifying, and verifying the early requirements for a product line of software-intensive systems based on an initial business case and market analysis.

- Its goal is to identify opportunities for large-grained reuse across the product line. PLA is the link between the recognition of a business opportunity and the design of a product line architecture.

- It incorporates the views of multiple product line stakeholders in a preliminary requirements model that includes the functional features of products and the software quality attributes (e.g., performance, modifiability) of both the products and their development. The stakeholders providing the necessary input include marketers, managers, customer representatives, and architects.

- The requirements model created by PLA identifies common requirements across the product line and their allowed variants.
Product Line Analysis Concepts (2 of 2)

- Important step in establishing requirements for SW reuse
- It can also support communication, training, tool development and SW spec and design
- Differs from traditional development methods
  - Objective is to represent exploitable commonalities among systems
- Feature Model and Analysis (Key Model)
  - SEI method for domain analysis
  - Features define both common aspects of the domain and differences between related systems in the domain
  - Identify mandatory, optional or alternative characteristics of the related systems
- New practice even though proposed years ago
  - Domain analysis used in mature domains for reuse purposes
  - Little known about the extent of PLA application to real systems
The **use-case model** specifies the product line stakeholders and their key interactions with the product line. Those stakeholders will verify the acceptability of the product line (and of the requirements).

The **feature model** specifies the stakeholders' views of the product line.

The **object model** specifies the product line responsibilities that support those features.
The context model is a high level pictorial representation of the scope of the analysis.

The intent of the diagram is to identify the focus of study.
Next is the Features Model which is a brainstorming exercise by domain experts to hash out features.

Here is where variations, commonalities are identified.
### BCPL Domain Analysis: Refinement Table

<table>
<thead>
<tr>
<th></th>
<th>Configurability</th>
<th>Interoperability</th>
<th>Security</th>
<th>Useability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Data Types</td>
<td>N/A</td>
<td>▪ Standards: Translation of XML to VMF or DCAT to XML</td>
<td>▪ Data Identification: Tagged with classification in DCAT message</td>
<td>N/A</td>
</tr>
<tr>
<td>Data Exchange</td>
<td>▪ Interface Standardization: using DCI</td>
<td>▪ Shared Behavioral Expectations: Sends compatible advertisements prior to publishing reports</td>
<td>▪ Data Integrity: Maintain classification of messages</td>
<td>▪ Operability: Operator sees a familiar interface</td>
</tr>
<tr>
<td>Configuration</td>
<td>▪ Data Standardization: utilizing metadata</td>
<td>▪ Version Compatibility: Using PASS naming convention</td>
<td>▪ Access Control: Operator needs to Login with username/password</td>
<td>▪ Operability: Operator sees a familiar interface</td>
</tr>
</tbody>
</table>

Next the quality attributes and their relationship to the functional features are outlined in a refinement table.

This analysis excludes the following quality attributes: Maintainability, reliability, testability, performance, integrability, affordability, and predictability because their not applicable to these listed feature.
### BCPL Domain Analysis: Sub-Feature

<table>
<thead>
<tr>
<th>Level</th>
<th>Feature</th>
<th>Description</th>
<th>Products</th>
<th>Variability</th>
<th>Constraints</th>
<th>Alternatives</th>
<th>Optional?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Advertise ment</td>
<td>Proclamation to DDS what data will be published.</td>
<td>NOC, TSG</td>
<td>The metadata used for each advertisement can vary across products.</td>
<td>Dependent on DCI features</td>
<td>None</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

- The Sub-Feature Attributes table provides the variability analysis derived from the feature model.
- The variation content contained within this table is focused at the component level rather than the system level.
## BCPL Domain Analysis: Use Case Table

<table>
<thead>
<tr>
<th>User</th>
<th>Function</th>
<th>Description</th>
<th>Pre-Condition</th>
<th>Responsibility</th>
<th>Post-Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator/Operator</td>
<td>Configuration</td>
<td>Enter settings to connect to DDS.</td>
<td>-S6 must have network settings defined.</td>
<td>-Set IP address -Set Port -Set Protocol -Set Username -Set Password</td>
<td>-Configuration settings are saved at the NOC.</td>
</tr>
<tr>
<td>Administrator/Operator</td>
<td>Normal Operation</td>
<td>Create Advertisement</td>
<td>-DAPs must be followed</td>
<td>-Enter metadata -Create Advertisement -Send advertisement</td>
<td>-Advertisement is sent to DDS.</td>
</tr>
<tr>
<td>Administrator/Operator</td>
<td>Normal Operation</td>
<td>Create Subscription</td>
<td>-BFA DAPs need to be available -Query for advertisements from DDS</td>
<td>-Enter metadata -Create subscription -Send subscription</td>
<td>-Reports are received from DDS</td>
</tr>
</tbody>
</table>

- The Use case table starts to flush out responsibilities that end up as software requirements.
BCPL Requirements Engineering

- Aspects peculiar to product lines:
  - **Requirements elicitation** for a product line must capture anticipated variations explicitly over the foreseeable lifetime of the product line.
  - **Requirements analysis** for a product line involves finding commonalities and identifying variations.
  - **Requirements specification** now includes preparation of a product-line-wide set of requirements and product-specific requirements.
  - **Requirements verification** now includes a broader reviewer pool and occurs in stages.
  - **Requirements management** must now make allowances for the dual nature of the requirements engineering process and the staged (common, specific) nature of the activity.
Requirements Engineering : Documentation

Initializing Documents
- Stage Setting: High Level Internal and External Documents
  - Customer Supplied (SOW, ORD, CDD, ICD, etc)
  - Internally Generated (BCPL Scope, TISDD, Studies, SWADD, etc)

RE Practice Area Documents
- System / Subsystem Specification (SSS)
  - Captures High Level Requirements of Customer Requirements Within Scope of BCPL
  - Contains highest level “shall" statements derived from our customer documents.
    - Product (e.g., NOC, TSG, Vehicle, MTS, etc) and Core Asset requirements are traceable to this document.

- System / Subsystem Design Description (SSDD)
  - Derived from the SSS
  - Captures Battle Command Product Line System Level Design Details
    - This document includes the allocation of high level (SSS) requirements to products, and describes the larger system architecture.
    - This document includes the Concept of Execution
      - Documents the various system level threads requiring coordination with subsystems

- Battle Command Product Line Specification (BCPL Spec)
  - Derived from the SSS
    - Captures SSS requirements that are allocated to software Core Assets (CA)
  - Derived from the SSDD
    - Captures the variation of a software requirement (variation point) to support the range of products identified in the SSDD
Low level requirements satisfy higher level requirements

Product Requirement Trace

- Product SDD
  - Unit Level Requirement

- Product SRS
  - Testable Requirement

BCPL SSDD
- Product Allocated High Level Requirement Allocations

Core Asset (CA) Requirement Trace

- Core Asset SDD
  - Unit Level Requirement

- Core Asset SRS
  - Testable Requirement

BCPL Spec
- SSS requirements allocated to CA and SSDD variations to requirements (variation points)

BCPL SSS
- Requirement reflective of customer specified high level requirements

Customer Supplied Documents

A requirement in the product SRS is satisfied by a requirement maintained as a Core Asset
CA Requirement vs. Product Requirement

Assignment of a requirement to CA vs. a program is derived from the definition of program developed requirements:

If the requirement’s solution is:
- (1) needed by only the one project and there is no strategic reason to have it developed by CA,
- or
- (2) the requirement has wide appeal but there are development time constraints within CA,

then the requirement is developed by the project.

Conversely stated:
Core Assets will develop software to satisfy a requirement if one of the following driving forces exist:

1. If there some high level overarching strategic reasons to solve the requirement as a Core Asset, then it belongs to Core Assets.
2. If the solution to the requirement represents “Future Think”, then it belongs to Core Assets.
3. If the requirement has broad appeal but there is a time constraint that cannot be met by CA, then it is to be developed by the project and later taken over by CA.
4. If there is a strategic reason to develop the requirement as a Core Asset, then it belongs to Core Assets.
BCPL’s Development Process
• The three essential activities are highly iterative and inextricably linked

• “New core assets might, and most often do, evolve out of product development”
The building blocks in the product line that are specifically designed to vary and be reused in multiple products are called core assets.
“Building a product that has previously unrecognized commonality with another product already in the product line will create pressure to update the core assets and provide a basis for exploiting that commonality for future products”
Product Development-2

- Products are comprised of components that are:
  - 1. Used directly from the core asset base
  - 2. Used directly after exercising built-in variation points
  - 3. Used after adaptation
  - 4. Used after modification (i.e. "clone-and-own")
  - 5. Product specific

- "Whether a product component is adapted or built from scratch, it should be reviewed ultimately for "promotion" to the core asset base"
Risk Management
BCPL Risk Management Process

Risk Documentation

RM Planning
- Risk Mgmt Planning

Continuous RM Execution
- Identify Risks
- BCPL RMB
- Analyze Risks
- Monitor Risks
- Mitigate Risks

Update Plans
BCPL Risk Management Process - cont

Diagram showing the process:
- OCD/Risk Plan defines Risk Identification
- Potential Risks: Accept/Reject Assign Leader
- Risk Tracking
- Risk Mitigation Plan
- Risk Database
- Selected Risks
- Risk Management
- Earned Value Management System (EVMS)
- BCPL Goals
- KPPs
- Metrics: CPI SPI TPM status updates
- New Metrics
- Process Improvement Initiatives
- New Ideas Complaints
- TPMs
- May generate

Process:
1. OCD/Risk Plan defines Risk Identification
2. Potential Risks: Accept/Reject Assign Leader
3. Risk Tracking
4. Risk Mitigation Plan
5. Risk Database
6. Selected Risks
7. Risk Management
8. Earned Value Management System (EVMS)
9. BCPL Goals
10. KPPs
11. Metrics: CPI SPI TPM status updates
12. New Metrics
13. Process Improvement Initiatives
14. New Ideas Complaints
15. TPMs
Findings / Results
Some Product Line Strategy Questions

1. Is it better to design a new product for PL replication or should one replicate an existing product?

2. What proportion of a system should be unique and what proportion should be based on a common PL?

3. How complete should the PL package be --- what should be the PL reuse replication costs?

4. Should the PL portion of replicated systems be maintained in common or should each replicated system be maintained separately?
Example Cost Comparison: PL Reuse Replication
Cost Factor=0.2, PL Portion=0.8 (High End PL)

Acquisition and Maintenance Cost Comparison

- Total Savings = 67%
- Little difference between common and separate PL maintenance
- Positive benefit after two systems
- Sizeable PL investment
Analysis: This is the program's first Product Line metric. It shows the percent of code used from core assets plus the percent of code used from product builders to make a deliverable product.
ESLOC per Product

SLOCS are the total Source Lines of Code collected by the project.

ESLOCs are Equivalent Source Lines Of Code -- just the deliverable code generated by the project staff (machine-generated code is counted as a fraction which represents the human labor to generate it).
ESLOC Progress and Plan

**Analysis:** This table looks at the difference in code developed versus code to be developed. This is indication for upper management to see what effort is left to implement the final product.

<table>
<thead>
<tr>
<th>Product</th>
<th>May 31, 2007 ESLOC Total</th>
<th>New ESLOC to Do Estimate</th>
<th>Total ESLOC Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSG*</td>
<td>41,278</td>
<td>8,000</td>
<td>49,278</td>
</tr>
<tr>
<td>Vehicle</td>
<td>252,333</td>
<td>2,000</td>
<td>254,333</td>
</tr>
<tr>
<td>CORE</td>
<td>1,370,972</td>
<td>111,490</td>
<td>1,482,462</td>
</tr>
<tr>
<td>NOC</td>
<td>385,141</td>
<td>25,000</td>
<td>385,141</td>
</tr>
<tr>
<td>NOC_Temp</td>
<td>31,866</td>
<td>200</td>
<td>31,866</td>
</tr>
</tbody>
</table>
Quality Goals Addressed (Lean)

- **Extensibility and subset-ability**
  - New functionality can be easily added, or removed
  - Greater range of platforms supported
    - Reduced footprint -- Systems only use needed parts of BCPL

- **Reliability, maintainability, and constructability**
  - Same core components used on all systems
  - Applications added without modifications to core
  - Less coupling, greater modularity
Summary...

- Product lines introduce requirements for new and sustained organizational investments
- For product lines to be successful, stakeholders must know the commitments and benefits up front
- Simple cost modeling can:
  - Answer major product line questions
  - Provide guidance
  - Identify trends
  - Avoid pitfalls
  - Increase the likelihood of product line success
- Benefits of the BCPL approach
  - Reduced cost to field software products
  - Reduced time to field software products
  - Improved product quality (performance, reliability, and modifiability)
Questions ?????
Backup
Simple Product Line Cost Model*

- **Assumptions**
  - Product line (PL) development and repeated reuse in system development of N systems of constant size (1 unit)
  - Constant PL reuse portion for each system, R
  - Design for reuse cost penalty, a
  - PL reuse replication cost, b (learning curve, technology insertion, fees, modifications, system-specific documentation and verification activities)
  - Constant maintenance portion for each development period, m

- **Product line development cost with design for reuse:** (1+a)*R
- **System acquisition cost with PL use:** b*R + (1-R)
- **Maintenance cost:** m * N \text{delivered systems} (independent maintenance) or
  m *[ (1+a)*R + (1-R) * N \text{delivered systems} ] (common PL maintenance)
Cost Model Parameter Values

- **Design-for-reuse cost penalty: a**
  - COCOMO II: $a = 0.24$ across PLs, $a = 0.15$ within a PL
  - For the cost model used here: $a = 0.24$

- **Product line reuse cost: b**
  - Potential contributors: PL evaluation ($0 - 0.08$), learning curve ($0 - 0.15$), product modifications ($0 - 0.30$), test ($0 - 0.25$), documentation ($0 - 0.10$)
  - For the cost model used here: parameterize with $b = 0.2, 0.4, 0.6$

- **PL proportion: R**
  - Based on PL domain and business objectives
  - For the cost model used here: parameterize with $R = 0.2, 0.4, 0.6, 0.8$

- **Maintenance: m**
  - Combination of defect corrections and minor enhancements
  - Typical annual maintenance costs are $0.05 - 0.20$
  - For the cost model used here: $0.15$ per delivered system per delivery period
Comparison Strategies

1. Acquisition cost with no PL reuse
2. Acquisition cost with PL reuse
3. Acquisition and maintenance cost with no PL reuse
4. Acquisition and maintenance cost with PL acquisition reuse and common PL maintenance
5. Acquisition and maintenance cost with PL acquisition reuse but program-separate PL maintenance
6. Product line development and maintenance
Comparison Measures for PL Strategies

- Total effort in developing N systems (N=1-5)
  - Units of effort to develop one system without PL reuse
  - Effort could also be measured in staff months or equivalent source lines of code (ESLOC)

- Payback period – crossover point when PL reuse begins to realize savings
Example Cost Comparison: PL Reuse Replication

Cost Factor = 0.2, PL Portion = 0.8 (High End PL)

Acquisition and Maintenance Cost Comparison

- Cum Acq w/ Reuse
- Cum Acq w/o Reuse
- Cum Acq & Maintenance w/ Full Reuse
- Cum Acq & Maintenance w/ Acq Reuse Only
- Cum Acq & Maintenance w/o Reuse
- Product Line Cum Cost

Total Savings = 67%

Little difference between common and separate PL maintenance

Positive benefit after two systems

Sizeable PL investment
Example Cost Comparison: PL Reuse Replication

Cost Factor = 0.6, PL Portion = 0.2 (Low End PL)

Acquisition and Maintenance Cost Comparison

- Cum Acq w/ Reuse
- Cum Acq w/o Reuse
- Cum Acq & Maintenance w/ Full Reuse
- Cum Acq & Maintenance w/ Acq Reuse Only
- Cum Acq & Maintenance w/o Reuse
- Product Line Cum Cost

Cumulative Costs (Unit System Cost)

Total Savings = 5%

Little difference between common and separate PL maintenance

Positive benefit after four systems

Small PL investment
Is it better to design a new product for PL replication or should one replicate an existing product?

Answer: It is better to replicate an existing system since there are no initial investment costs and the savings are better than a newly developed PL until development of four systems.

---

**New PL vs. Existing Product Reuse**

- **No Reuse**
- **Existing Product Reuse**
- **New PL Development**

---

- **No reuse**
- **Existing product reuse with 0.8 reuse content and 0.6 reuse replication factor**
- **New PL with 0.8 reuse content and 0.2 reuse replication factor**
Answers to PL Strategy Questions (2 of 3)

What proportion of a system should be unique and what proportion should be based on a common PL? How complete should the PL package be --- how low should the PL reuse replication costs be?

Answer: In general, significant savings (~20%) requires \( (R-b) > 0.25 \).
Answers to PL Strategy Questions (3 of 3)

Should the PL portion of replicated systems be maintained in common or should each replicated system be maintained separately?

Answer: There is little difference in overall savings but there may be differences in customer/organizational costs and complexities.

-acquisition and maintenance cost comparison

Little difference between common and separate PL maintenance

Cumulative Costs (Unit System Cost)

Systems Developed
Overall Size in ESLOCs by Month

The counts represented are the Effective Logical Source Lines of Code (ESLOC).