Software Architecture and Product Quality

Linda Northrop
Director, Product Line Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Presentation for the Boston SPIN
May 20, 2003

This work is sponsored by the U.S. Department of Defense.
SEI’s Strategic Functions

- IDENTIFY AND MATURE TECHNOLOGY
- CREATE
- APPLY
- TRANSITION
- AMPLIFY

DoD needs → Technology trends → SEI’s experience → User’s experience

© 2003 by Carnegie Mellon University
Version 052203
Boston SPIN May 2003 - page 2
SEI and the Community

CREATE   APPLY   AMPLIFY   CREATE   APPLY
AMPLIFY   CREATE   APPLY   AMPLIFY   CREATE
APPLY AMPLIFY   CREATE   APPLY   AMPLIFY   CREATE
AMPLIFY   CREATE   APPLY   AMPLIFY   CREATE
DEVELOPERS

ACQUIRERS

RESEARCHERS
Product Line Systems Program

Our Goal: To enable widespread product line practice through architecture-based development
Our Strategy

Software Architecture
   *(Architecture Tradeoff Analysis Initiative)*

Software Product Lines
   *(Product Line Practice Initiative)*

Component Technology
   *(Predictable Assembly from Certifiable Components Initiative)*
Our Customers and Collaborators

ABB
Daimler Chrysler
Caterpillar
Robert Bosch Co.
Raytheon
Foliage
RIM
Unisys
Visteon
LLNL
EPA
FAA
NASA: JSC
NASA: KSC
NASA Goddard
USCG
NRO/CCT
JNIC
DMSO
US Army SOA: TAPO
US Army: FBCB2, CECOM, ATSC, FCS
US Navy: TENA, DDX
US Navy: DDX
US Air Force: F-22, ESC

Philips
Lucent
AT&T
Hewlett Packard
Thomson-CSF
Ericsson
Raytheon
Siemens
Schlumberger
Nokia
Telesoft S.p.A.
Boeing
CelsiusTech
Buzzeo
ALLTEL
Motorola
Cummins, Inc.
General Motors
Lockheed Martin
Salion, Inc.
MarketMaker
Business Success Requires Software Prowess

Software pervades every sector. Software has become the bottom line for many organizations who never envisioned themselves in the software business.
Universal Business Goals

High quality
Quick time to market
Effective use of limited resources
Product alignment
Low cost production
Low cost maintenance
Mass customization
Mind share

improved efficiency and productivity
Software Strategies Are Needed

Business/Mission Goals

System (Software) Strategies

Process Improvement

Improved Architecture Practices

process quality

product quality
If function were all that mattered, any monolithic software would do, *but other things matter*…

The important quality attributes and their characterizations are key.

- Modifiability
- Interoperability
- Availability
- Security
- Predictability
- Portability

Has these qualities

Software Architecture

Quality Attribute Drivers

**Functional Software Requirements**

*analysis, design, development*
Software Architecture: Common Ideas

A software architecture is a “first-cut” at designing the system and solving the problem or fitting the need.

A software architecture is an ad hoc box-and-line drawing of the system that is intended to solve the problems articulated by the specification.

• Boxes define the elements or “parts” of the system.
• Lines define the interactions or between the parts.
Our 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 among them.”

Implications of Our Definition

Software architecture is an abstraction of a system.

Software architecture defines the properties of elements.

Systems can and do have many structures.

Every software-intensive system has an architecture.

Just having an architecture is different from having an architecture that is known to everyone.

If you don’t develop an architecture, you will get one anyway – and you might not like what you get!
Why is Software Architecture Important?

Represents *earliest* design decisions

*First* design artifact addressing

Key to systematic *reuse*

The right architecture paves the way for system success.
The wrong architecture usually spells some form of disaster.

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

- performance
- modifiability
- reliability
- security

- transferable, reusable abstraction
System Qualities and Software Architecture

System Specification

System Quality Attributes *
* Performance
Security
Interoperability
Reliability
Availability
etc.

drive

Software Architecture

drives

System Capabilities and Software Quality

determines level of quality
What Is Architecture-Based Development?

Architecture-based development involves
• Creating the business case for the system
• Understanding the requirements
• Creating or selecting the architecture
• Documenting and communicating the architecture
• Analyzing or evaluating the architecture
• Implementing the system based on the architecture
• Ensuring that the implementation conforms to the architecture
• Maintaining the architecture

The architecture must be both prescriptive and descriptive.
Common Impediments to Achieving Architectural Success

Lack of adequate architectural talent and/or experience.
Insufficient time spent on architectural design and analysis.
Failure to identify the quality drivers and design for them.
Failure to properly document and communicate the architecture.
Failure to evaluate the architecture beyond the mandatory government review.
Failure to understand that standards are not a substitute for a software architecture.
Failure to ensure that the architecture directs the implementation.
Failure to evolve the architecture and maintain documentation that is current.
Failure to understand that a software architecture does not come free with COTS or with the C4ISR Framework.
Challenges

What precisely do these quality attributes such as modifiability, security, performance, and reliability mean?

How do you architect to ensure the system will have its desired qualities?

Can a system be analyzed to determine these desired qualities?

How soon can such an analysis occur?

How do you know if software architecture for a system is suitable without having to build the system first?
Architecture Tradeoff Analysis Initiative: Maturing Sound Architecture Practices

Starting Points
- Quality attribute/performance engineering
- Software Architecture Analysis Method (SAAM)
- Security analysis
- Reliability analysis
- Software Architecture Evaluation Best Practices Report
- Software architecture evaluations

Create
- Architecture tradeoff analysis
  - attribute-specific patterns
  - architecture evaluation techniques
- Architecture representation
- Architecture definition
- Architecture reconstruction
Traditional System Development

Operational descriptions
High level functional requirements
Legacy systems
New systems
Specific system architecture
Software architecture

Quality attributes are rarely captured in requirements specifications.
• often vaguely understood
• often weakly articulated

a miracle occurs
Quality Attribute Workshop

The Quality Attribute Workshop (QAW) is a facilitated method that engages system stakeholders early in the lifecycle to discover the driving quality attributes of a software intensive system.

Key points about the QAW are that it is

• system centric
• scenario based
• stakeholder focused
• used before the software architecture has been created
QAW Benefits and Next Steps

**Potential Benefits**
- Increased stakeholder communication
- Clarified quality attribute requirements
- Informed basis for architectural decisions
- Improved architecture documentation

**QAW**
- Quality Attribute Scenarios:
  - raw
  - prioritized
  - refined

**Potential Next Steps**
- Update Architectural Vision
- Refine Requirements
- Create Prototypes
- Exercise Simulations
- Create Architecture
- Architecture Evaluation

Can be used to

© 2003 by Carnegie Mellon University
Version 052203
Boston SPIN May 2003 - page 22
Creating the Software Architecture

There are architecture definition methods and guidelines, many of which focus exclusively on the functional requirements.

It is possible to create an architecture based on the quality architectural drivers.

One way to approach this is to use architectural tactics and patterns and a method that capitalizes on both.
Tactics

The design for a system consists of a collection of design decisions.
- Some decisions are intended to ensure the achievement of the functionality of the system.
- Other decisions are intended to help control the quality attribute responses.
These decisions are called tactics.
- A tactic is a design decision that is influential in the control of a quality attribute response.
- A collection of tactics is an architectural strategy.
Performance Tactics

Summary of performance tactics
Tactics Catalog

Tactics have been defined for the following quality attributes:
- Performance
- Availability
- Maintainability
- Usability
- Testability
- Security

Others are in the works.
Attribute Driven Design

The Attribute Driven Design (ADD) method is an approach to defining a software architecture by basing the design process on the quality attributes the software has to achieve.

It follows a recursive decomposition process where, at each stage in the decomposition, tactics and architectural patterns are chosen to satisfy a set of quality scenarios.
Importance of Architecture Documentation

Architecture documentation is important if and only if communication of the architecture is important.
  • How can an architecture be used if it cannot be understood?
  • How can it be understood if it cannot be communicated?

Documenting the architecture is the crowning step to creating it. Documentation speaks for the architect, today and 20 years from today.
Seven Principles of Sound Documentation

Certain principles apply to all documentation, not just documentation for software architectures.

1. Write from the point of view of the reader.
2. Avoid unnecessary repetition.
3. Avoid ambiguity.
4. Use a standard organization.
5. Record rationale.
6. Keep documentation current but not too current.
7. Review documentation for fitness of purpose.
View-based Documentation

An architecture is a very complicated construct and its almost always too complicated to be seen all at once.

Software systems have many structures or views.
   • No single representation structure or artifact can be the architecture.
   • The set of candidate structures is not fixed or prescribed: architects need to select what is useful for analysis or communication.

A view is a representation of a set of system elements and the relations associated with them.

Documenting a software architecture is a matter of documenting the relevant views, and then adding information that applies to more than one view.
Which Views are Relevant?

Which views are relevant? It depends on

• who the stakeholders are
• how they will use the documentation.

Three primary uses for architecture documentation

• Education - introducing people to the project.
• Communication - among stakeholders.
• Analysis - assuring quality attributes.
Why Analyze an Architecture?

All design involves tradeoffs.

A software architecture is the earliest life-cycle artifact that embodies significant design decisions.
SEI’s Architecture Tradeoff Analysis Method$^{SM}$ (ATAM)$^{SM}$

ATAM is an architecture evaluation method that
- focuses on multiple quality attributes
- illuminates points in the architecture where quality attribute *tradeoffs* occur
- generates a context for ongoing quantitative analysis
- utilizes an architecture’s vested stakeholders as authorities on the quality attribute goals
Conceptual Flow of ATAM

- Business Drivers
- Quality Attributes
- Scenarios
- Architectural Approaches
- Architectural Decisions
- Risks
- Non-Risks
- Sensitivity Points
- Tradeoffs

Risk Themes

impacts
distilled into
Analysis
ATAM Steps

1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results
ATAM Benefits

There are a number of benefits from performing ATAM analyses:

- Clarified quality attribute requirements
- Improved architecture documentation
- Documented basis for architectural decisions
- Identified risks early in the life-cycle
- Increased communication among stakeholders

The results are improved architectures.
Where Can the ATAM Be Used?

Architecture decisions, little or no code

Alternative candidate architectures

Existing System
ATAM Experience

By an SEI Team

• Internal
  - user-interface tool
  - avionics system
  - furnace control system
• Commercial
  - engine control systems
  - automotive systems
  - healthcare information management system
  - financial information system
• Non-defense Government
  - physics models
  - water quality models
• Academic
  - required part of masters-level Carnegie Mellon architecture course
  - on software engineering projects (MSE-Carnegie Mellon)

By a Non-SEI Team

• Automotive systems
• Consumer electronics systems
Defense-Related ATAM Experience

Completed

Army (Picatinny Arsenal) - Mortar Fire Control Systems  
Air Force (SND C2 SPO) - Space Battle Management Core System  
Air Force - NATO-Midterm AWACS  
NRO/NASA - Space Object Technology Group (SOTG) Reference Architecture  
NASA Goddard - Earth Observing System  
JNTF - Wargame 2000  
NASA Houston - Space Shuttle Software  
Army TAPO – Common Avionics Architecture System

Under way

Army – Future Combat System  
Army – FBCB2  
Army – Army Training Support System  
Navy – DDX
Architecture Tradeoff Analysis Initiative: Enabling Sound Architecture Practices

Starting Points
- Quality attribute/performance engineering
- Software Architecture Analysis Method (SAAM)
- Security analysis
- Reliability analysis
- Software Architecture Evaluation Best Practices Report
- Software architecture evaluations

Create
- Architecture tradeoff analysis
  - attribute-specific patterns
  - architecture evaluation techniques
- Architecture representation
- Architecture definition
- Architecture reconstruction

Apply/Amplify
- Architecture Evaluations
- Architecture Coaching
- Architecture Reconstructions
- Books
- Courses
- Certificate Programs
- Acquisition Guidelines
- Technical Reports
- Web site
Widespread Transition: SEI Software Architecture Curriculum

Six courses
• Software Architecture Familiarization
• Documenting Software Architectures
• Software Architecture Design and Analysis
• Software Product Lines
• ATAM Evaluator Training
• ATAM Facilitator Training

Three certificate programs
• Software Architecture Professional
• ATAM Evaluator
• ATAM Lead Evaluator

In addition
• Architecture Analysis Guidelines for Acquisition Managers
## Certificate Program Course Matrix

<table>
<thead>
<tr>
<th>ATAM Lead Evaluator: 5 Courses &amp; Coaching</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Architecture Professional:</strong> 4 Courses</td>
</tr>
<tr>
<td><strong>ATAM Evaluator Training</strong></td>
</tr>
<tr>
<td><strong>ATAM Evaluator 2 courses</strong></td>
</tr>
<tr>
<td><strong>Software Architecture Familiarization</strong></td>
</tr>
<tr>
<td><strong>Documenting Software Architectures</strong></td>
</tr>
<tr>
<td><strong>Software Architecture Design and Analysis</strong></td>
</tr>
<tr>
<td><strong>Software Product Lines</strong></td>
</tr>
<tr>
<td><strong>ATAM Evaluator 2 courses</strong></td>
</tr>
<tr>
<td><strong>ATAM Facilitator Training</strong></td>
</tr>
<tr>
<td><strong>ATAM Coaching</strong></td>
</tr>
</tbody>
</table>
Associated Texts

**Software Architecture in Practice, 2nd Edition**

**Documenting Software Architectures: Views and Beyond**

**Evaluating Software Architectures: Methods and Case Studies**

**Software Product Lines: Practices and Patterns**
Another Challenge

Over the next $n$ years you have $m$ similar systems under development and mildly (wildly) different development approaches.

At the same time you have less money to spend, fewer people to work with, and less time to get the job done.

And oh by the way, the systems are more complex.
The Truth is … Few Systems Are Unique

Most organizations produce families of similar systems, differentiated by features.
CelsiusTech: 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 35:65 to 80:20
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
National Reconnaissance Office / Raytheon: Control Channel Toolkit

Ground-based spacecraft command and control systems

increased quality by 10X
incremental build time reduced from months to weeks
software productivity increased by 7X
development time and costs decreased by 50%
decreased product risk
Market Maker GmbH: MERGER

Internet-based stock market software

each product “uniquely” configured
three days to put up a customized system
Nokia Mobile Phones

Product lines with 25-30 new products per year

Across products there are
• varying number of keys
• varying display sizes
• varying sets of features
• 58 languages supported
• 130 countries served
• multiple protocols
• needs for backwards compatibility
• configurable features
• needs for product behavior change after release
A Proven Solution

Software Product Lines
How Did They Do It?

strategic reuse

business strategy and technical strategy

employed to achieve explicit business goals
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.
How Do Product Lines Help?

Product lines amortize the investment in these and other *core assets*:

- requirements and requirements analysis
- domain model
- software architecture and design
- performance engineering
- documentation
- test plans, test cases, and data
- people: their knowledge and skills
- processes, methods, and tools
- budgets, schedules, and work plans
- Software components

*Software product lines epitomize strategic reuse.*
The Key Concepts

Use of a common asset base in production of a related set of products
The Key Concepts

Use of a common asset base

in production

of a related set of products

Architecture

Production Plan

Scope Definition Business Case
Organizational Benefits

Improved productivity
by as much as 10x

Decreased time to market (to field, to launch...)
by as much as 10x

Decreased cost
by as much as 60%

Decreased labor needs
by as much as 10X fewer software developers

Increased quality
by as much as 10X fewer defects
Necessary Changes

The architecture is the foundation of everything.
Product Line Practice

Contexts for product lines vary widely
- nature of products
- nature of market or mission
- business goals
- organizational infrastructure
- workforce distribution
- process discipline
- artifact maturity

But there are universal essential activities and practices.
A Framework for Software Product Line Practice

The three essential activities and the descriptions of the product line practice areas form a conceptual framework for software product line practice.

This framework is evolving based on the experience and information provided by the community.

Version 4.0 – in *Software Product Lines: Practices and Patterns*

Version 4.1 –
http://www.sei.cmu.edu/plp/framework.html
### Framework

#### Essential Activities

<table>
<thead>
<tr>
<th>Software Engineering</th>
<th>Technical Management</th>
<th>Organizational Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture Definition</td>
<td>Configuration Management</td>
<td>Building a Business Case</td>
</tr>
<tr>
<td>Architecture Evaluation</td>
<td>Data Collection, Metrics, and Tracking</td>
<td>Customer Interface Management</td>
</tr>
<tr>
<td>Component Development</td>
<td>Make/Buy/Mine/Commission Analysis</td>
<td>Implementing an Acquisition Strategy</td>
</tr>
<tr>
<td>COTS Utilization</td>
<td>Process Definition</td>
<td>Funding</td>
</tr>
<tr>
<td>Mining Existing Assets</td>
<td>Scoping</td>
<td>Launching and Institutionalizing</td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>Technical Planning</td>
<td>Market Analysis</td>
</tr>
<tr>
<td>Software System Integration</td>
<td>Technical Risk Management</td>
<td>Operations</td>
</tr>
<tr>
<td>Testing</td>
<td>Tool Support</td>
<td>Organizational Planning</td>
</tr>
<tr>
<td>Understanding Relevant Domains</td>
<td></td>
<td>Organizational Risk Management</td>
</tr>
</tbody>
</table>

### Practice Areas

© 2003 by Carnegie Mellon University
Version 052203
Boston SPIN May 2003 - page 61
SE Practice Area Relationships

Domain Understanding feeds Requirements

Requirements drive Architecture

Architecture specifies components

Components
SE Practice Area Relationships

- Domain Understanding feeds Requirements
- Requirements drive Architecture
  - specifies components
  - Make
  - Buy
  - Mine
  - Commission
- Components
SE Practice Area Relationships

- Domain Understanding
  - Understanding Relevant Domains
  - Requirements Engineering
  - Architecture Definition
  - Architecture Evaluation

  - Makes
    - Component Development
    - Make
  - Buys
    - COTS Utilization
    - Buy
  - Mines
    - Mining Existing Assets
    - Mine
  - Commissions
    - [Developing an Acquisition Strategy]
    - Commission

  - Components
  - Software System Integration
  - Testing
SE Practice Area Relationships

Domain Understanding

- Understanding Relevant Domains

Requirements

- Requirements Engineering

Architecture

- Architecture Definition
- Architecture Evaluation

Make/Buy/Mine/Commission Analysis

Make

- Component Development
- existing talent

Buy

- COTS Utilization
- market availability

Mine

- Mining Existing Assets
- legacy base

Commission

- [Developing an Acquisition Strategy]
- organizational policy

Components

Software System Integration

Testing
Dilemma: How Do You Apply the 29 Practice Areas?

Organizations still have to figure out how to put the practice areas into play.

29 is a “big” number.
# How to Make It Happen

## Essential Activities

- Core Asset Development
- Product Development
- Management

## Practice Areas

<table>
<thead>
<tr>
<th>Software Engineering</th>
<th>Technical Management</th>
<th>Organizational Management</th>
</tr>
</thead>
</table>

## Guidance

- **Probe**
- **Patterns**
- **Case Studies**

---

© 2003 by Carnegie Mellon University  
Version 052203  
Boston SPIN May 2003 - page 67
SEI Product Line Case Studies

CelsiusTech: Ship System 2000

National Reconnaissance Office / Raytheon: Control Channel Toolkit (CCT)

Cummins, Inc.: Diesel Engine Control Systems

Market Maker GmbH: MERGER

US Naval Underwater Warfare Center: Rangeware

Salion, Inc.: Revenue Acquisition Management solutions
How to Make It Happen

Essential Activities

Practice Areas

Software Engineering
Technical Management
Organizational Management

Guidance

Probe
Patterns
Case Studies
Patterns Can Help

Patterns are a way of expressing common context and problem-solution pairs.

Patterns have been found to be useful in building architecture, economics, software architecture, software design, software implementation, process improvement, and others.

Patterns assist in effecting a divide and conquer approach.
Software Product Line Practice Pattern

**Pattern**

- **Context** – organizational situation
- **Problem** – what part of a product line effort needs to be accomplished
- **Solution** – grouping of practice areas
  - relations among these practice areas (and/or groups if there is more than one)
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Variants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly Line</td>
<td></td>
</tr>
<tr>
<td>Cold Start</td>
<td>Warm Start</td>
</tr>
<tr>
<td>Curriculum</td>
<td></td>
</tr>
<tr>
<td>Each Asset</td>
<td>Each Asset Apprentice</td>
</tr>
<tr>
<td></td>
<td>Evolve Each Asset</td>
</tr>
<tr>
<td>Essentials Coverage</td>
<td></td>
</tr>
<tr>
<td>Factory</td>
<td></td>
</tr>
<tr>
<td>In Motion</td>
<td></td>
</tr>
<tr>
<td>Monitor</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Process Improvement</td>
</tr>
<tr>
<td>Product Builder</td>
<td>Product Gen</td>
</tr>
<tr>
<td>Product Parts</td>
<td>Green Field</td>
</tr>
<tr>
<td></td>
<td>Barren Field</td>
</tr>
<tr>
<td></td>
<td>Plowed Field</td>
</tr>
<tr>
<td>What to Build</td>
<td>Analysis</td>
</tr>
<tr>
<td></td>
<td>Forced March</td>
</tr>
</tbody>
</table>
How to Make It Happen

Essential Activities

Core Asset Development
Product Development
Management

Practice Areas

Software Engineering  Technical Management  Organizational Management

Guidance

Probe  Patterns  Case Studies
What is a Product Line Technical Probe?

A method for examining an organization’s readiness to adopt or ability to succeed with a software product line approach

• diagnostic tool based on the Framework for Software Product Line Practice™

• practice areas are used in the data collection and analysis
Summary of SEI Contributions

Practice Integration:
• Acquisition Companion to the Framework

Techniques and Methods
• product line analysis
• architecture definition – Attribute-Driven Design (ADD)
• architecture evaluation – Architecture Tradeoff Analysis Method\textsuperscript{SM} (ATAM\textsuperscript{SM})
• mining assets – Options Analysis for Reengineering\textsuperscript{SM} (OAR\textsuperscript{SM})
• Product Line Technical Probe\textsuperscript{SM}

Book
Software Product Lines: Practices and Patterns
• Practices (Framework, Version 4.0)
• patterns
• case studies

Conferences
SPLC 2004 – Sept 2004
What’s Different About Reuse with Software Product Lines?

Business dimension

Iteration

Architecture focus

Pre-planning

Process and product connection
Software Product Line Strategy in Context

Business/Mission Goals

System (Software) Strategies

Process Improvement

Improved Architecture Practices

Software Product Lines

process and product quality

process quality

process quality
Challenge

Software components are critical to today’s systems and product lines

BUT the behavior of component assemblies is unpredictable.

• “interface” abstractions are not sufficiently descriptive
• behavior of components is, in part, an a priori unknown
• behavior of component assemblies must be discovered

The result is costly development and decreased assurance.
A Solution

Predictable Assembly from Certifiable Components (PACC)
The Vision

Our vision is to provide the engineering methods and technologies that will enable

• properties of assemblies of components to be reliably predicted, by construction
• properties of components used in predictions to be objectively trusted

We refer to the end-state as having achieved predictable assembly from certifiable components (PACC)
Prediction Enabled Component Technology (PECT)

PECT is the approach we propose to achieve PACC goals.
At the grossest level, PECT is the integration of component technology with analysis technology.
Industrial Demonstration

Customer: ABB Corporate Research Center

Customer Information
- Transforming from heavy industry in power plant equipment to IT products and services in process automation

Purpose
- First year of collaboration to demonstrate the feasibility of PACC in substation automation
- Second year of collaboration to demonstrate the feasibility of PACC in industrial robotics

Problem Being Solved
- Predictable assembly from certifiable components in substation automation domain
  - operator level latency (PECT)
  - controller level latency (PECT)
  - combined operator-controller latency (PECT^2)
- and in robotics domain
- Reliability and safety scenarios are under investigation

Status
- Feasibility study for substation automation completed
- Robotics work underway
Status

PACC premises were validated on an internal system and through an ABB Feasibility Study.
PACC became an initiative as of October 2002.
The emphasis of work in 2002-03 is to ready PECT for practitioner use

- practical automation for building and using PECTs
  - conceptual framework of PECT was generalized in and was more rigorously defined
  - specification language (CCL) was defined and tools are currently being developed
- model checking was introduced for reliability verification
- technical advances in timing and reliability analysis paves the way to real industry trial, real payoff potential
The Total Picture

**Business/Mission Goals**

- Improved Architecture Practices
- Improved Component Practices
- Software Product Lines
- Process Improvement
- System (Software) Strategies

- **process quality**
- **product quality**
- **process and product quality**
The Total Picture

Business/Mission Goals

System (Software) Strategies

Process Improvement

Software Product Lines

Improved Component Practices

Improved Architecture Practices

Process and product quality

product quality

process quality

System (Software) Strategies
Conclusion

Software architecture, product line practices, and predictable component practices hold great potential for achieving business and mission goals in software-intensive systems.

Software architecture is critical to product quality.
For More Information

Linda Northrop
Director
Product Line Systems Program
Telephone: 412-268-7638
Email: lmn@sei.cmu.edu

U.S. mail:
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890

World Wide Web:
http://www.sei.cmu.edu/ata
http://www.sei.cmu.edu/plp

SEI Fax: 412-268-5758
How Does Process Improvement Relate to Software Product Lines?

Process discipline is required to succeed with software product lines.

An organization’s process improvement efforts poise it to succeed with software product lines.

Questions to ask

• What CMM maturity level do I have to have to be successful with product lines?
• Does my process improvement prowess guarantee my success with software product lines?
Process Discipline Provides a Foundation for Product Line Practice

Product line practice involves strategic reuse.

A strategic effort requires more coordination, discipline, and commonality of approach than a more independent effort.

An organization with a culture of process discipline is better poised for product line success.

The question again is, “How much process discipline?”
Answers -1

Process discipline provides an important foundation for software product line practice.

It would be *very useful* to be CMMI Level 2 (project focus) in this minimum set of Process Areas

- Requirements Management
- Project Planning
- Configuration Management
- Requirements Development

It would be *even more useful* to be able to standardize these processes across organizational units (Level 3).
Product line practice is supported by both CMMI model representations.

- continuous (focus on the “minimum” set of Process Areas)
- staged (establish a more solid foundation with a more comprehensive set of Process Areas).

Process maturity is a very helpful foundation. However, success in software product lines requires mastery of many other essential practice areas.

- important technical and technical management practices *plus* product line extensions to CMMI Process Areas
- cross-project strategic business processes not address by CMMI models
For More Details

*Software Process Improvement and Product Line Practice: CMMI and the Framework for Software Product Line Practice*

CMU/SEI-2002-TN-012

Available on the SEI web site at

[www.sei.cmu.edu/publications/documents/02.reports/02tn012.html](http://www.sei.cmu.edu/publications/documents/02.reports/02tn012.html)