Carnegie Mellon
Software Engineering Institute

Advances in the Art of Software Development

Presentation for

The Naval Industry R&D Partnership Conference 2003

Aug 6, 2003

Linda Northrop
Director, Product Line Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This work is sponsored by the U.S. Department of Defense.

© 2003 by Carnegie Mellon University

Presentation Outline

Background

Software Architecture

Software Architecture Practices

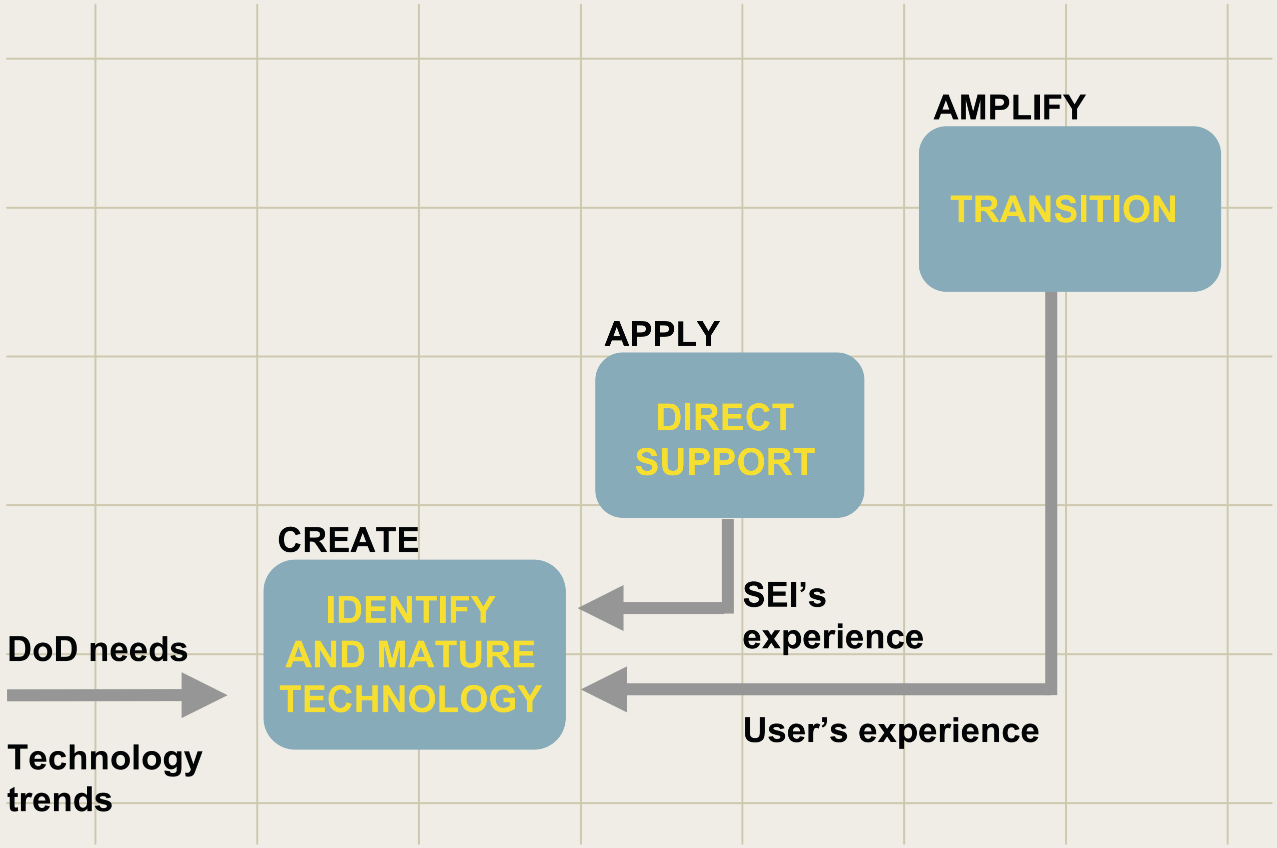
Related Innovative Practices

SEI Software Architecture Support

Conclusion

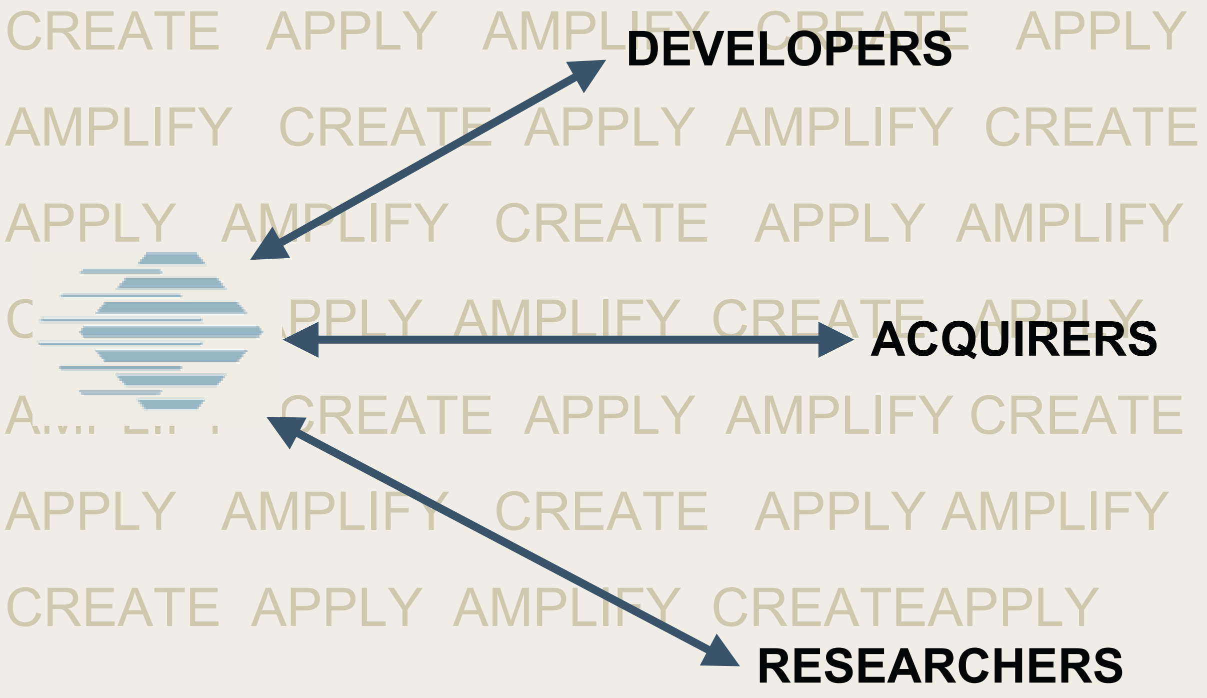
Discussion

SEI's Strategic Functions



Carnegie Mellon
Software Engineering Institute

SEI and the Community



© 2003 by Carnegie Mellon University

Product Line Systems Program

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

Our Strategy

Software Architecture
(Architecture Tradeoff Analysis Initiative)

Software Product Lines
(Product Line Practice Initiative)

Component Technology
(Predictable Assembly from Certifiable Components Initiative)

© 2003 by Carnegie Mellon University

Software Today

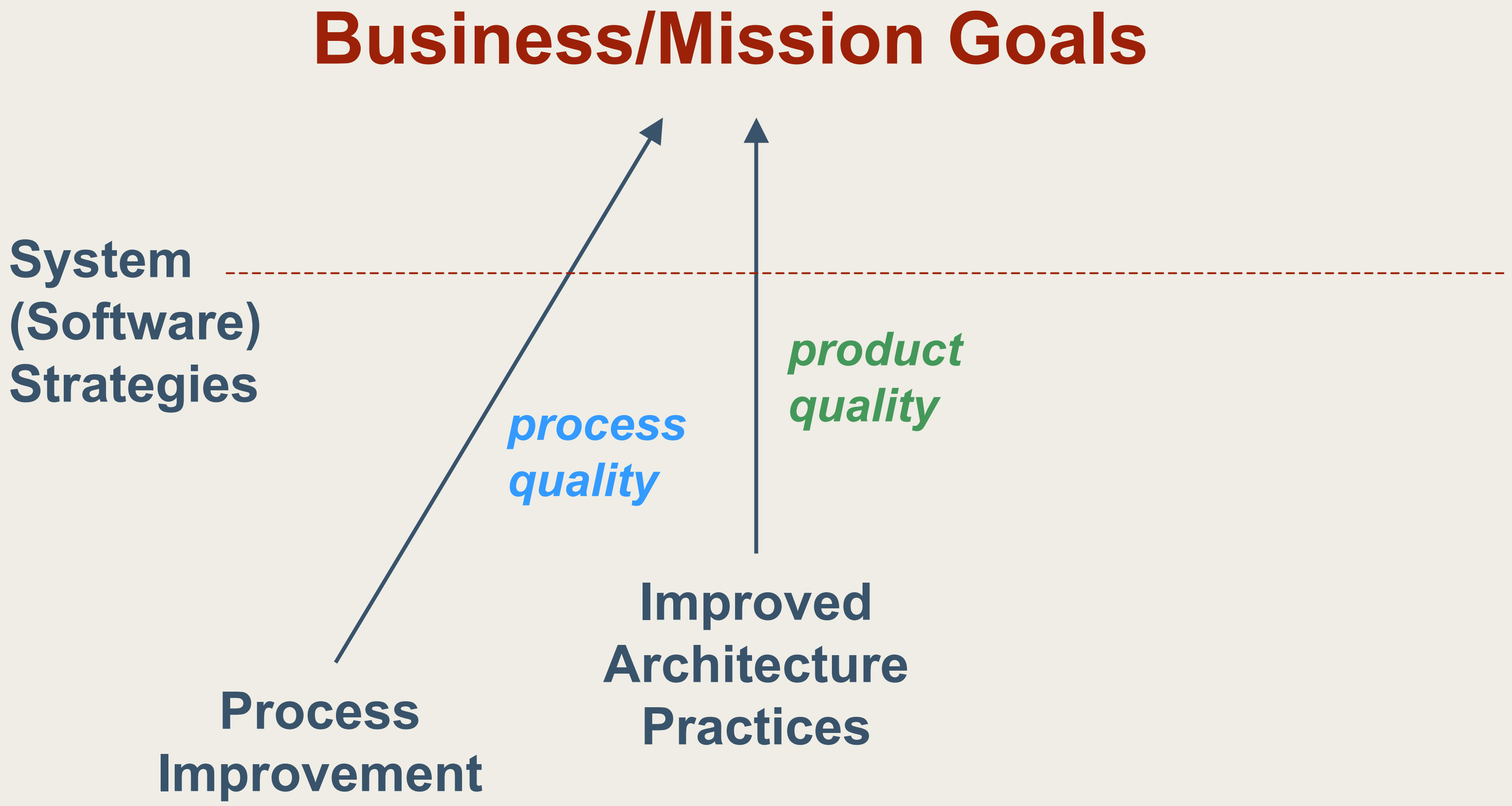
Software is pervasive in today's Navy systems and business operations.

Poor quality software is the root cause of cost, schedule, and quality deficiencies observed in vast numbers of delivered systems.

High quality software is key to future system and mission success.

© 2003 by Carnegie Mellon University

Software Strategies Are Needed



© 2003 by Carnegie Mellon University

Focus: Software Architecture

The quality and longevity of a software system is largely determined by its architecture.

Too many experiences point to inadequate software architecture education and practices in the DoD and its contractor base and the lack of any real software architecture evaluation early in the life cycle.

Without an explicit course of action focused on software architecture, these experiences are being and will be repeated. The cost of inaction is too great to the DoD and to the war fighter.

© 2003 by Carnegie Mellon University

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

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.

© 2003 by Carnegie Mellon University

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."

Bass L.; Clements P.; Kazman R. *Software Architecture in Practice 2nd Edition* Reading, MA: Addison-Wesley, 2003.

© 2003 by Carnegie Mellon University

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!*

© 2003 by Carnegie Mellon University

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

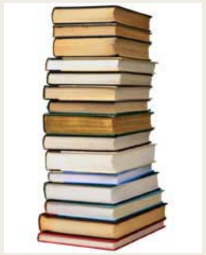
Key to systematic *reuse*

- transferable, reusable abstraction

The right architecture paves the way for system success.

The wrong architecture usually spells some form of disaster.

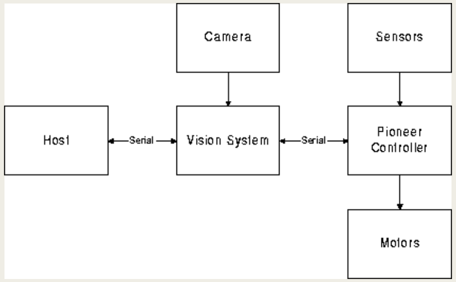
Requirements Beget Design



Requirements in various forms

Available knowledge

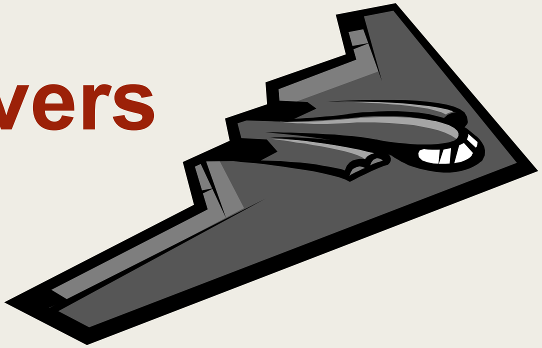
System



Designer

Architecture

Business/Mission Drivers

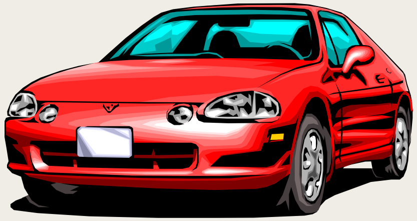


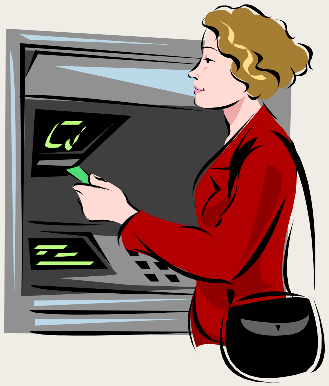
Mission
- capability
- flexibility

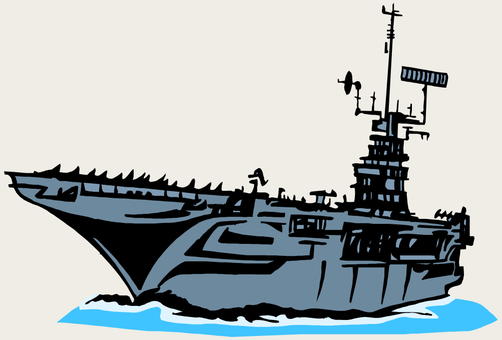
Business
- cost
- schedule

Technology
- evolution obsolesce
- standards, COTS

Constraints
- legacy systems
- mandated HW/SW/OS Languages







© 2003 by Carnegie Mellon University

Software System Development

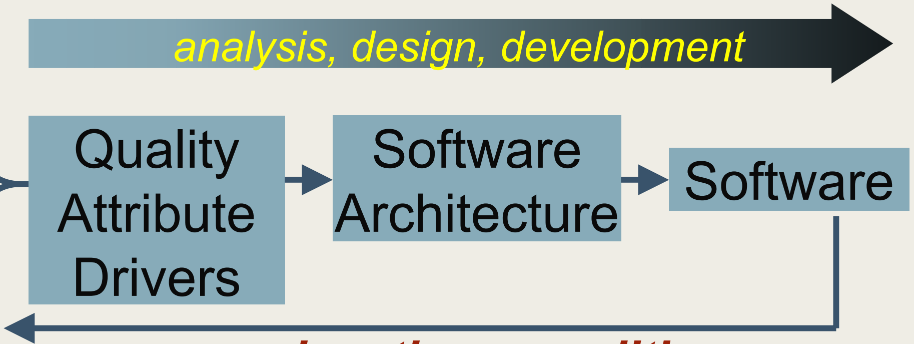
Functional Software Requirements

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

The Reality About Software Architecture.

Quality attribute requirements are the primary drivers for architectural design.

The degree to which a system meets its quality attribute requirements is dependent on architectural decisions.

Software development needs to be driven by architectural decisions.

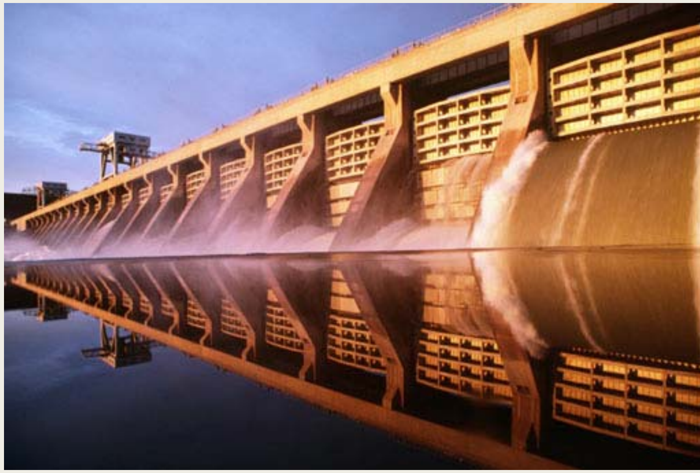
Architecture-centric development is key.

What Is Architecture-centric Development?

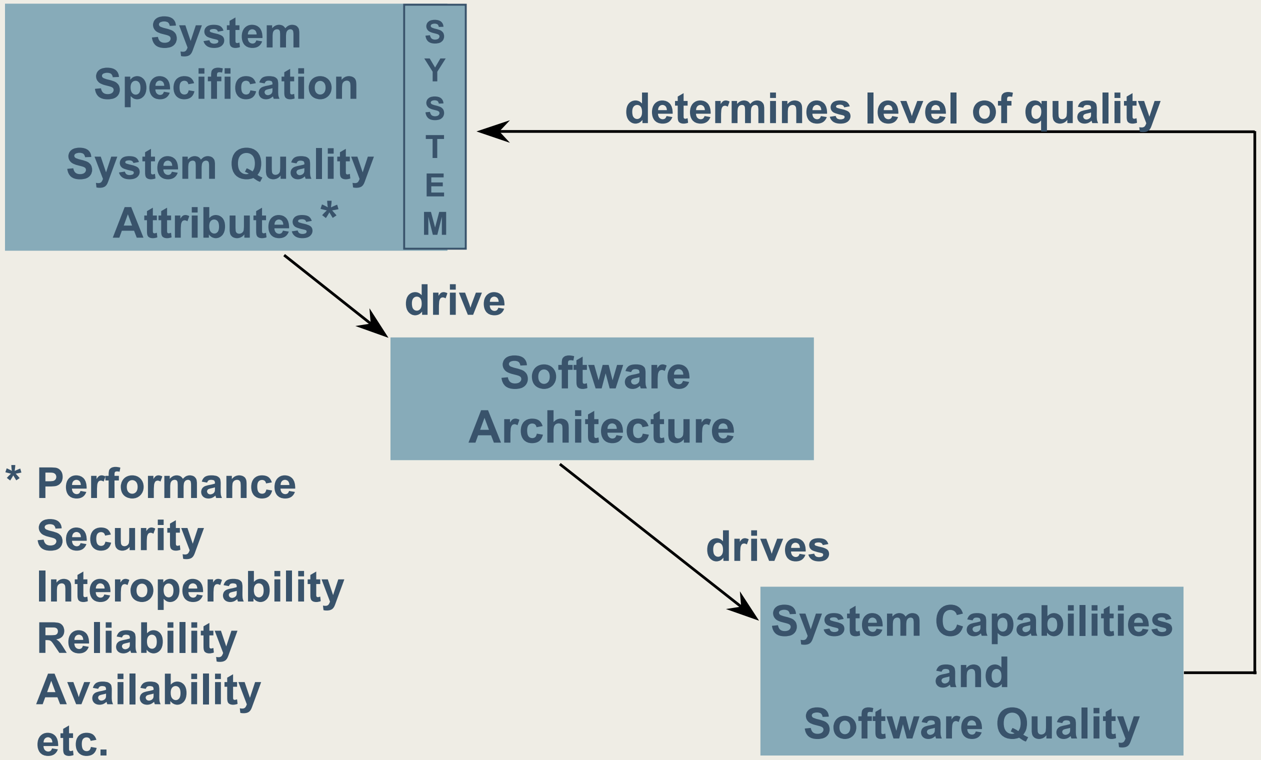
Architecture-centric development involves

- 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.



System Qualities and Software Architecture



* Performance
Security
Interoperability
Reliability
Availability
etc.

© 2003 by Carnegie Mellon University

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 DoD Framework.

Challenges

What are the driving quality attributes for your system?

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?

How do you document a software architecture?

How do you know if software architecture for a system is suitable without having to build the system first?

Can you recover an architecture from an existing system?

SEI Work in Software Architecture: 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

© 2003 by Carnegie Mellon University

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

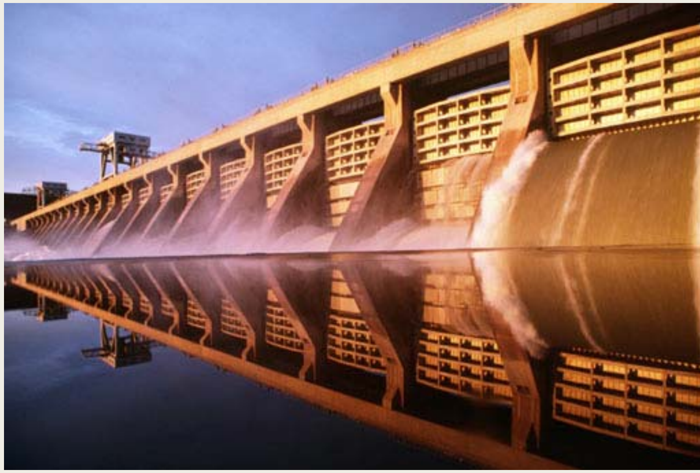
© 2003 by Carnegie Mellon University

What Is Architecture-centric Development?

Architecture-centric development involves

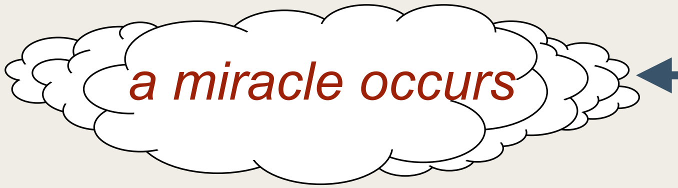
- Creating the business case for the system
- Understanding the requirements
- Creating or selecting the architecture
- Documenting and communicating the architecture
- Analyzing or evaluating the architecture
- Implementing the system based on the architecture
- Ensuring that the implementation conforms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

The architecture must be both prescriptive and descriptive.



Traditional System Development

Operational descriptions
High level functional requirements
Legacy systems
New systems



Quality attributes are rarely captured in requirements specifications.

- *often vaguely understood*
- *often weakly articulated*

Specific system architecture
Software architecture

Detailed design
Implementation

© 2003 by Carnegie Mellon University

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

© 2003 by Carnegie Mellon University

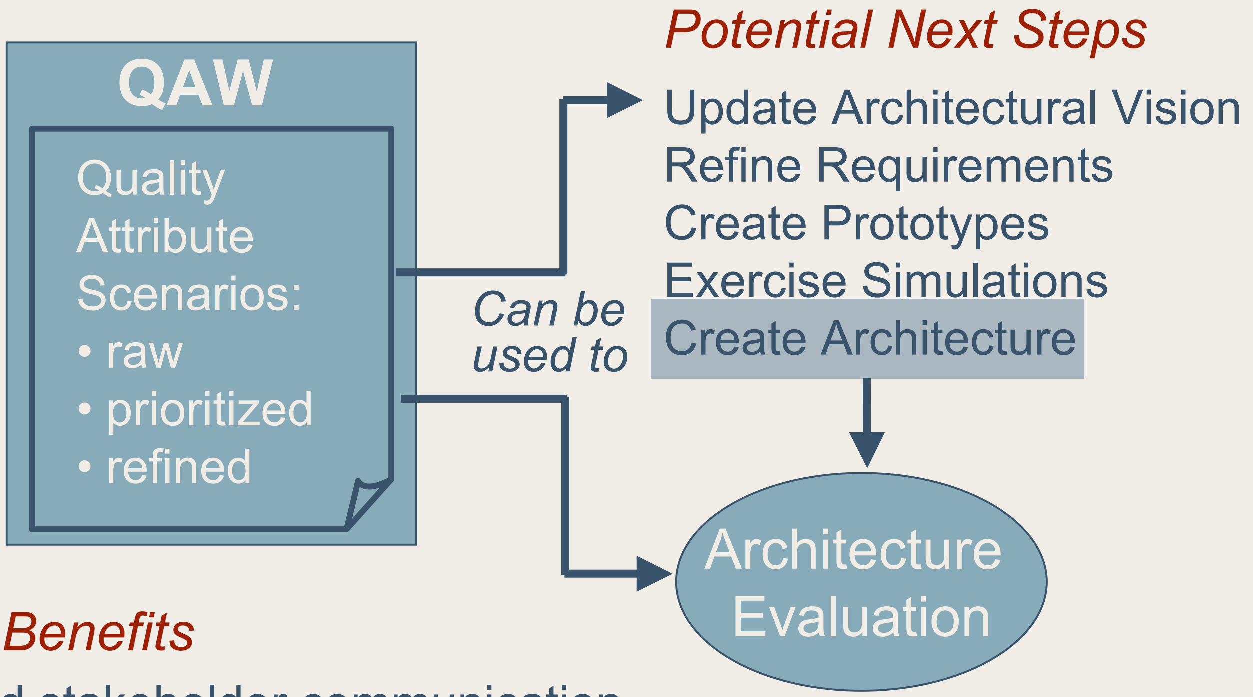
Quality Attribute Workshop Steps

1. Introductions and QAW Presentation
2. Business/Mission Presentation
3. Architecture Plan Presentation
4. Identify Architectural Drivers
5. Scenario Brainstorming
6. Scenario Consolidation
7. Scenario Prioritization
8. Scenario Refinement

Iterate as necessary with broader stakeholder community

© 2003 by Carnegie Mellon University

QAW Benefits and Next Steps



Potential Benefits

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

Example Scenario Refinement

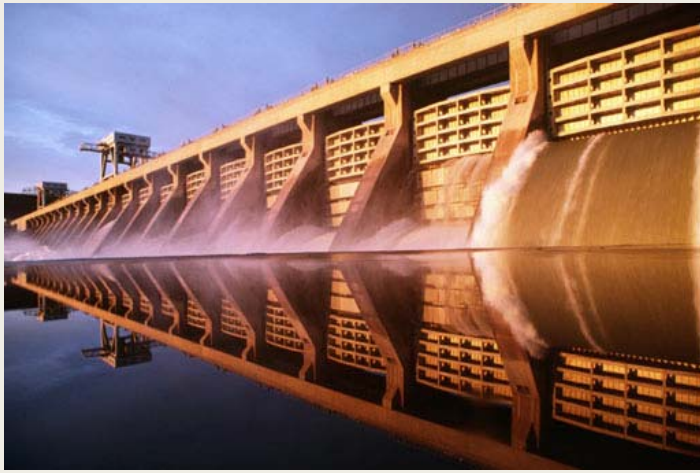
Scenario:	When garage door senses an obstacle, the system will stop the door in 1 millisecond
Business Goals:	reduced liability, competitive features
Actors: -Organizations -Systems -People	Homeowner
Relevant Quality Attributes:	Safety, Performance.
Questions:	How large do objects in the way of the closing door have to be before they are detected? Who will perform installation of the system? Will we be liable if the system is installed improperly?
Issues:	May have to train installers to prevent malfunctions and associated legal issues.

What Is Architecture-centric Development?

Architecture-centric 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.



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.

© 2003 by Carnegie Mellon University

Tactics - 1

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.*

© 2003 by Carnegie Mellon University

Tactics - 2

Tactics bridge quality attribute model and architectural design

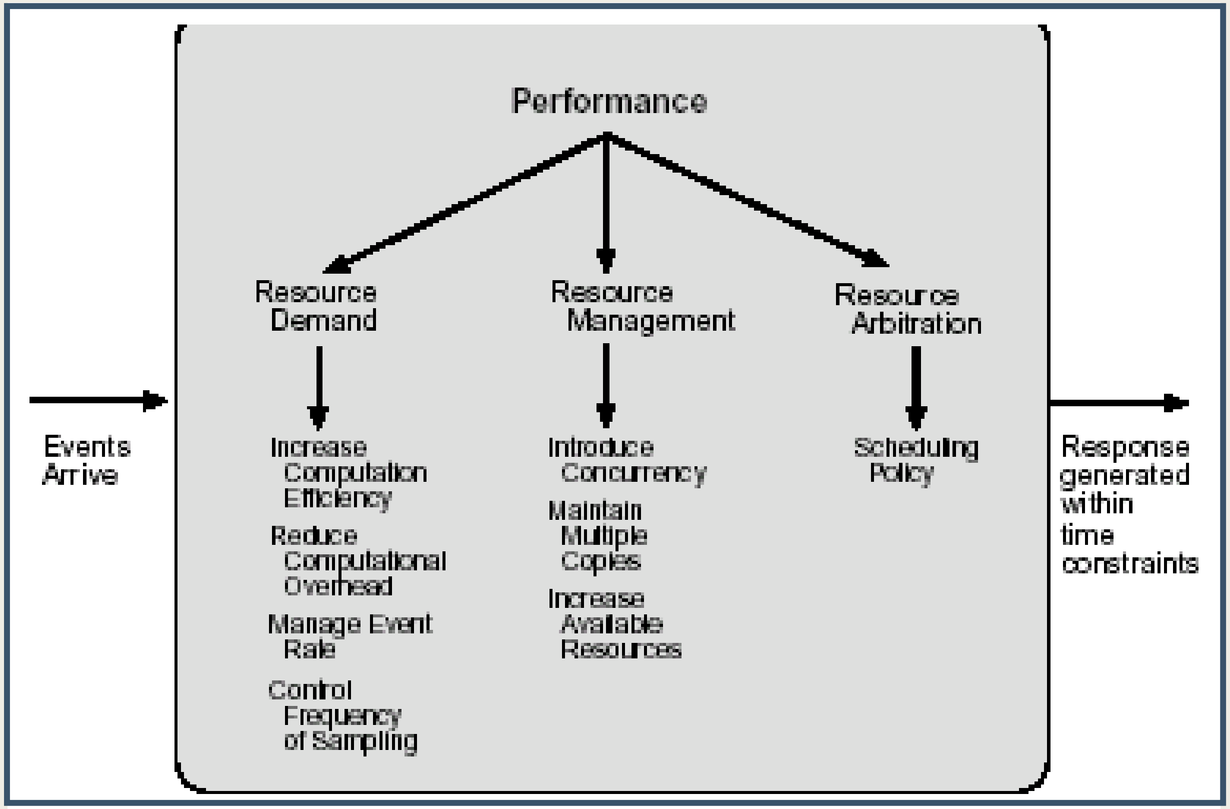
- Modifiability model has concepts such as “dependency”,
- Tactic translates that into “introduce intermediary” to break dependency

Quality attribute models may not yet have been articulated to explain tactics

- Tactics created from bottom up by attribute experts
- Experts have implicit models in their heads
- Suggests models that should be documented and further explored

Performance Tactics

Summary of performance tactics


Performance
Resource Demand
Resource Management
Resource Arbitration
Events Arrive
Increase Computation Efficiency
Reduce Computational Overhead
Manage Event Rate
Control Frequency of Sampling
Introduce Concurrency
Maintain Multiple Copies
Increase Available Resources
Scheduling Policy
Response generated within time constraints

Tactics Catalog

Tactics have been defined for the following quality attributes:

- Performance
- Availability
- Maintainability
- Usability
- Testability
- Security

Others are in the works.

© 2003 by Carnegie Mellon University

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.

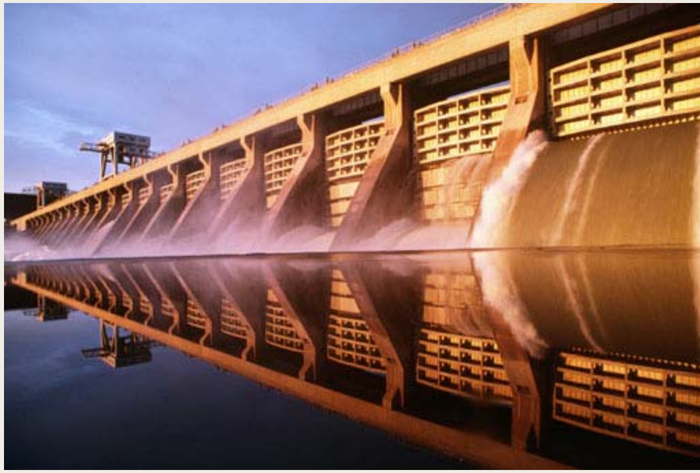
© 2003 by Carnegie Mellon University

What Is Architecture-centric Development?

Architecture-centric 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.



Importance of Architecture Documentation

Architecture documentation is important if and only if *communication* of the architecture is important.

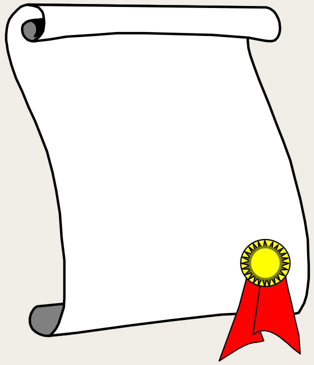
- How can an architecture be used if it cannot be understood?
- How can it be understood if it cannot be communicated?

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.

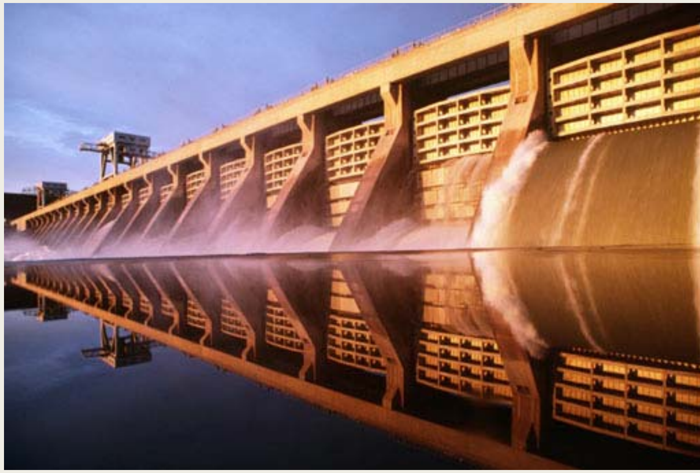
© 2003 by Carnegie Mellon University

What Is Architecture-centric Development?

Architecture-centric 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.



Traditional System Development

Operational descriptions
High level functional requirements
Legacy systems
New systems

a miracle occurs

Specific system architecture
Software architecture

another miracle occurs

Detailed design
Implementation

A Critical leap!

How do you know if the architecture is fit for purpose?

© 2003 by Carnegie Mellon University

Why Evaluate Architectures?

All design involves tradeoffs.

A software architecture is the earliest life-cycle artifact that embodies significant design decisions and tradeoffs.

- The earlier that risks are identified, the earlier that mitigation strategies can be developed potentially avoid the risks altogether.
- The earlier that defects are found, the less it costs to remove them.

© 2003 by Carnegie Mellon University

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

© 2003 by Carnegie Mellon University

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

© 2003 by Carnegie Mellon University

ATAM^{SM} Phase 1 Steps

1. Present the ATAM^{SM}
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

Phase 1

© 2003 by Carnegie Mellon University

ATAM^{SM} Phase 2 Steps

1. Present the ATAM^{SM}
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

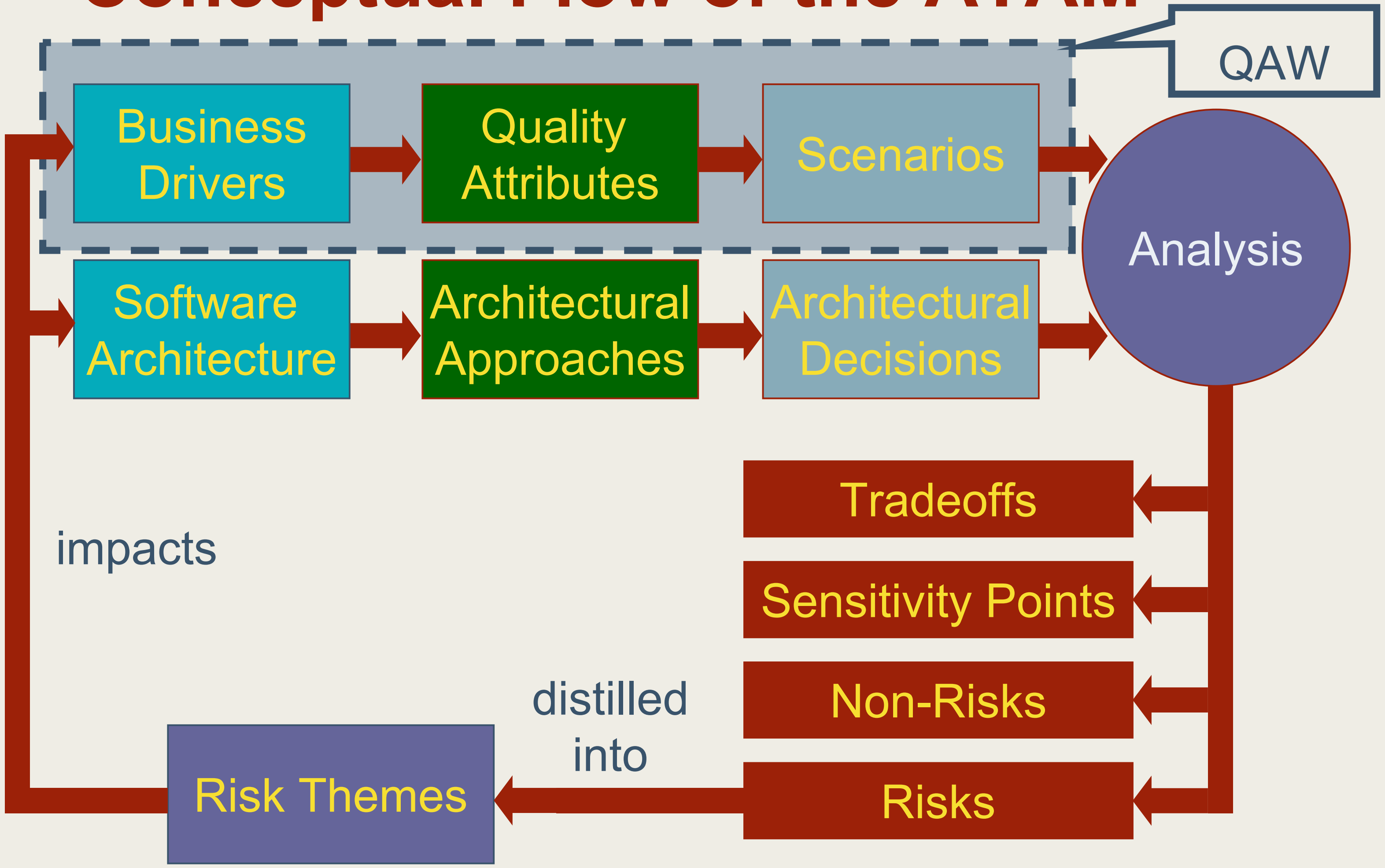
Recap Phase 1

Phase 2

Do this

© 2003 by Carnegie Mellon University

Conceptual Flow of the ATAM^{SM}

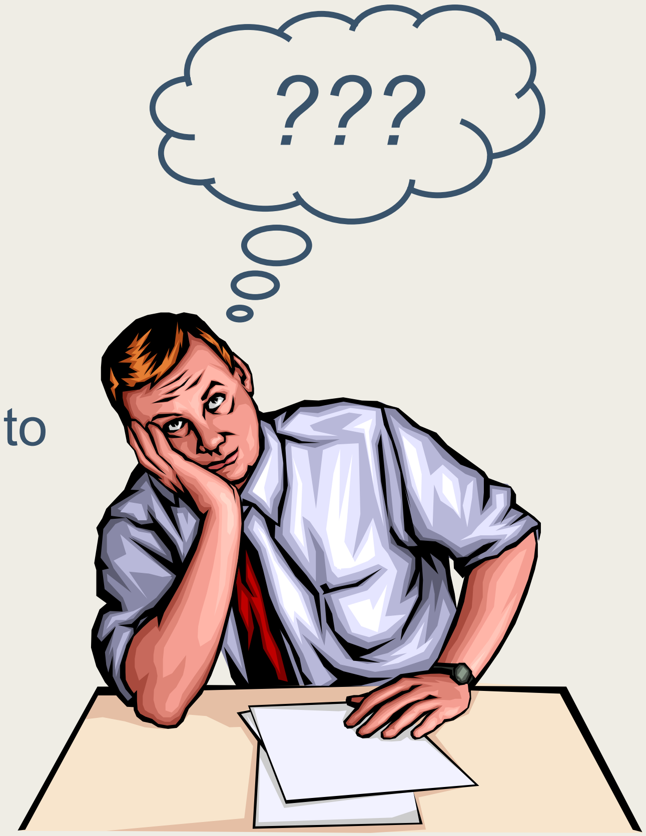


When Can the ATAM Be Used?

Early where there is an architecture, but there is little or no code.

To evaluate alternative candidate architectures.

To evaluate an existing system prior to major commitments to upgrade or replace the system.



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.

© 2003 by Carnegie Mellon University

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*
JNIC – *MD War*

Architecture Evaluation Experience

Benefits of early architecture evaluations

- Evaluations using the Architecture Tradeoff Analysis Method^{SM} (ATAM^{SM}) uncover an average 20 risks per two-day evaluation. Experience over a wide range of domains attributes these risks to
  - unknowns (requirements, hardware, COTS)
  - side effects of architectural decisions
  - improper architectural decisions
  - interactions with other organizations that provide system components
- Evaluations performed by AT&T have resulted in 10% productivity increase per project

© 2003 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2003 by Carnegie Mellon University

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.

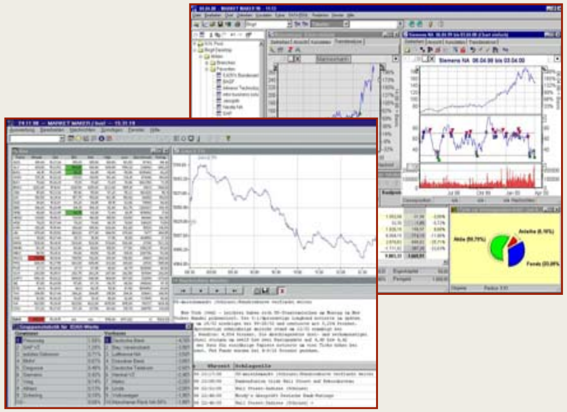
And oh by the way, the systems are more complex.

© 2003 by Carnegie Mellon University

The Truth is …Few Systems Are Unique

Most organizations produce families of similar systems, differentiated by features.

A Proven Solution



Software Product Lines

What is a Software Product Line?

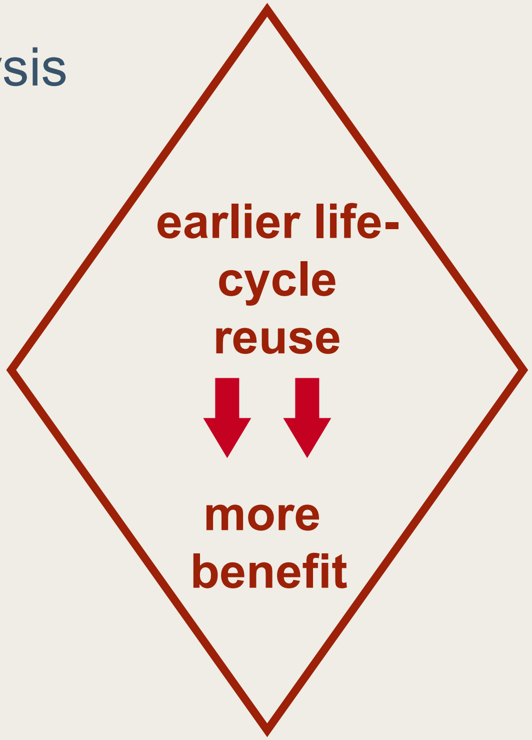
A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

© 2003 by Carnegie Mellon University

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



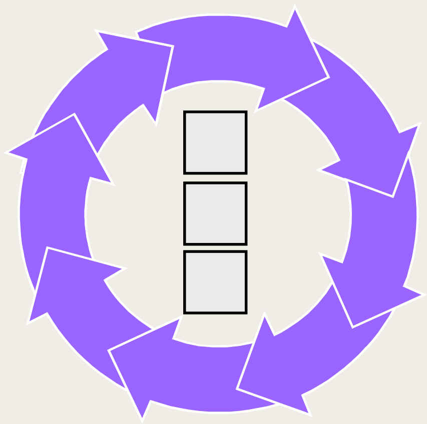
Software product lines epitomize strategic reuse.

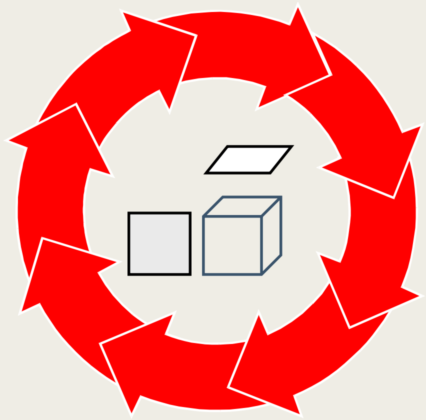
The Key Concepts

Use of a common asset base

in production

of a related set of products





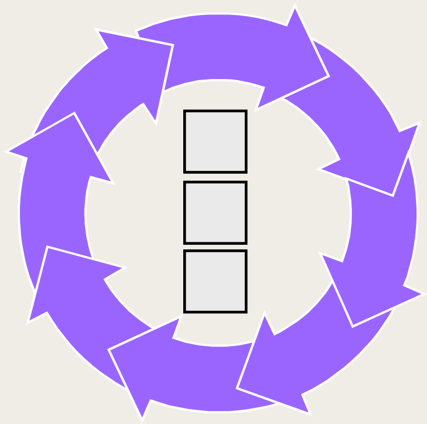
© 2003 by Carnegie Mellon University

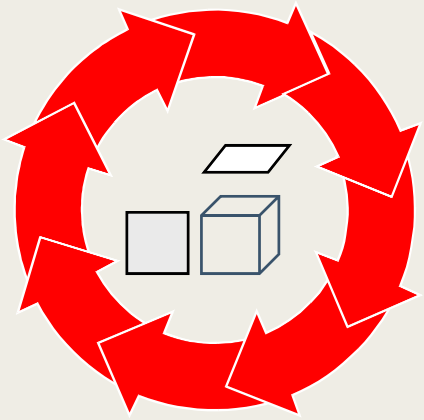
The Key Concepts

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

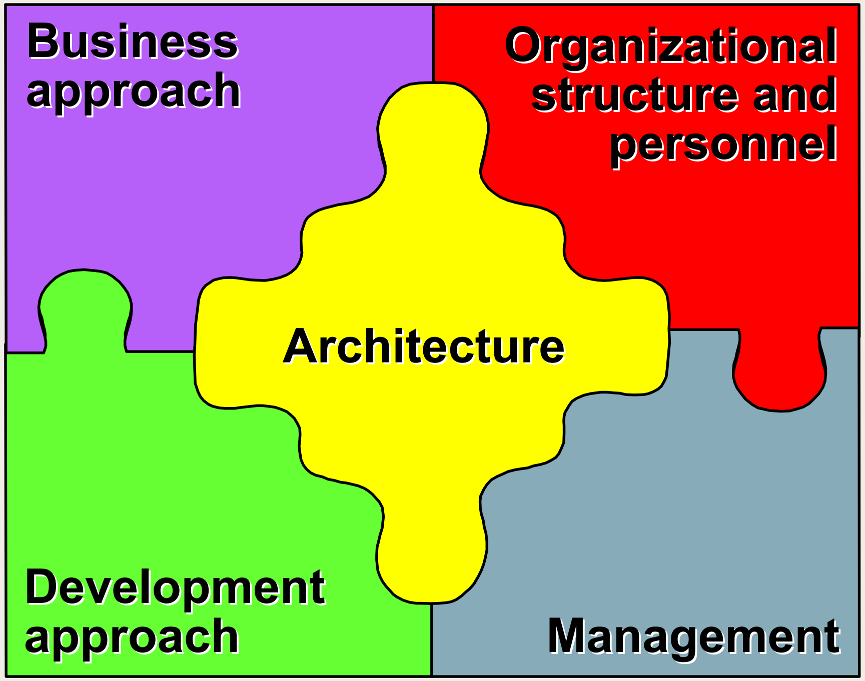
Decreased cost
by as much as 60%

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

Increased quality
by as much as 10X fewer defects

© 2003 by Carnegie Mellon University

Necessary Changes



The architecture is the foundation of everything.

© 2003 by Carnegie Mellon University

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.

© 2003 by Carnegie Mellon University

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.

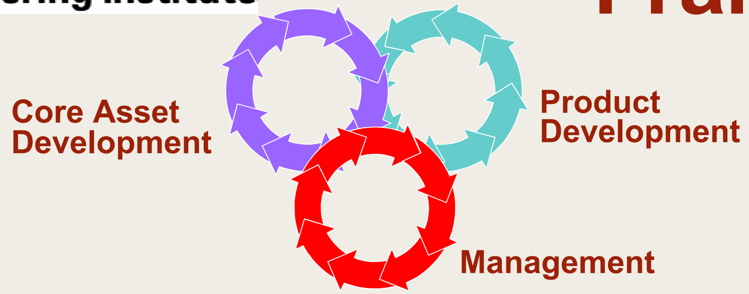
This framework is evolving based on the experience and information provided by the community.

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

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

© 2003 by Carnegie Mellon University

Framework



Essential Activities

Architecture Definition Architecture Evaluation Component Development COTS Utilization Mining Existing Assets Requirements Engineering Software System Integration Testing Understanding Relevant Domains	Configuration Management Data Collection, Metrics, and Tracking Make/Buy/Mine/Commission Analysis Process Definition Scoping Technical Planning Technical Risk Management Tool Support	Building a Business Case Customer Interface Management Implementing an Acquisition Strategy Funding Launching and Institutionalizing Market Analysis Operations Organizational Planning Organizational Risk Management Structuring the Organization Technology Forecasting Training
Software Engineering	**Technical Management**	**Organizational Management**

Practice Areas

© 2003 by Carnegie Mellon University

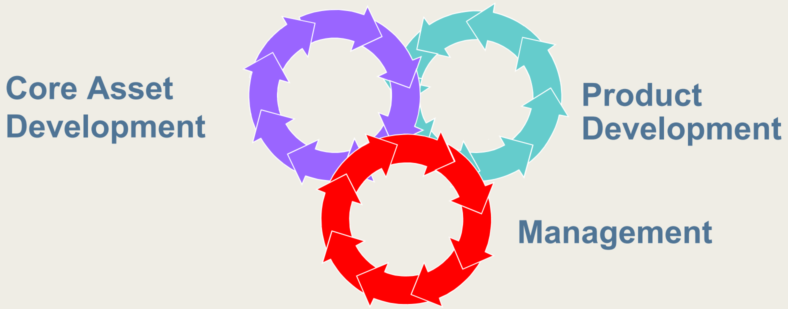
Dilemma: How Do You Apply the 29 Practice Areas?

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

29 is a “big” number.

How to Make It Happen

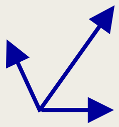
Essential Activities

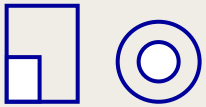


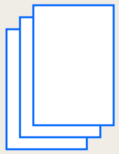
Practice Areas

Software Engineering	Technical Management	Organizational Management

Guidance


Probe


Patterns


Case Studies

© 2003 by Carnegie Mellon University

What’s Different About Reuse with Software Product Lines?

Business dimension

Iteration

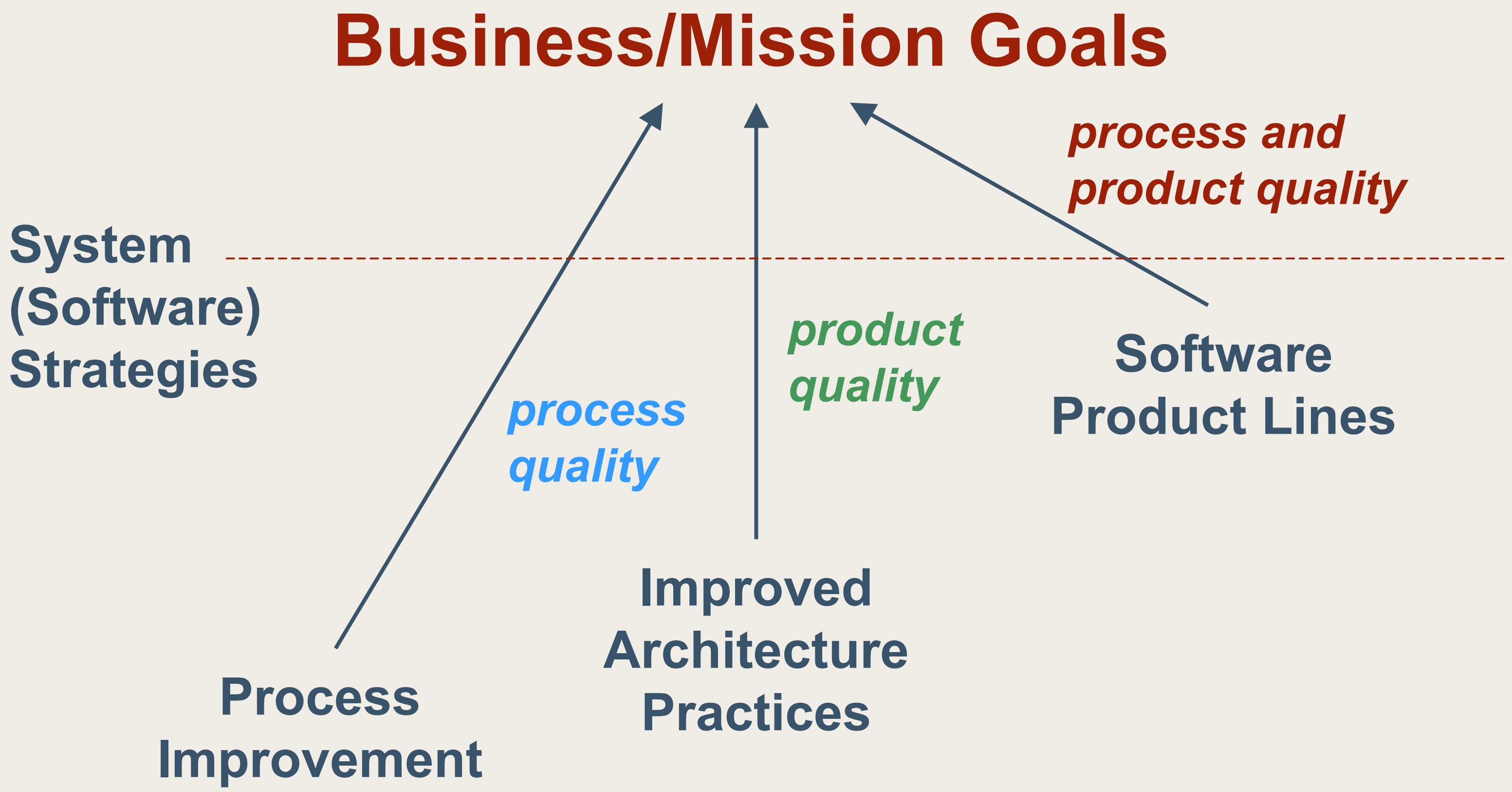
Architecture focus

Pre-planning

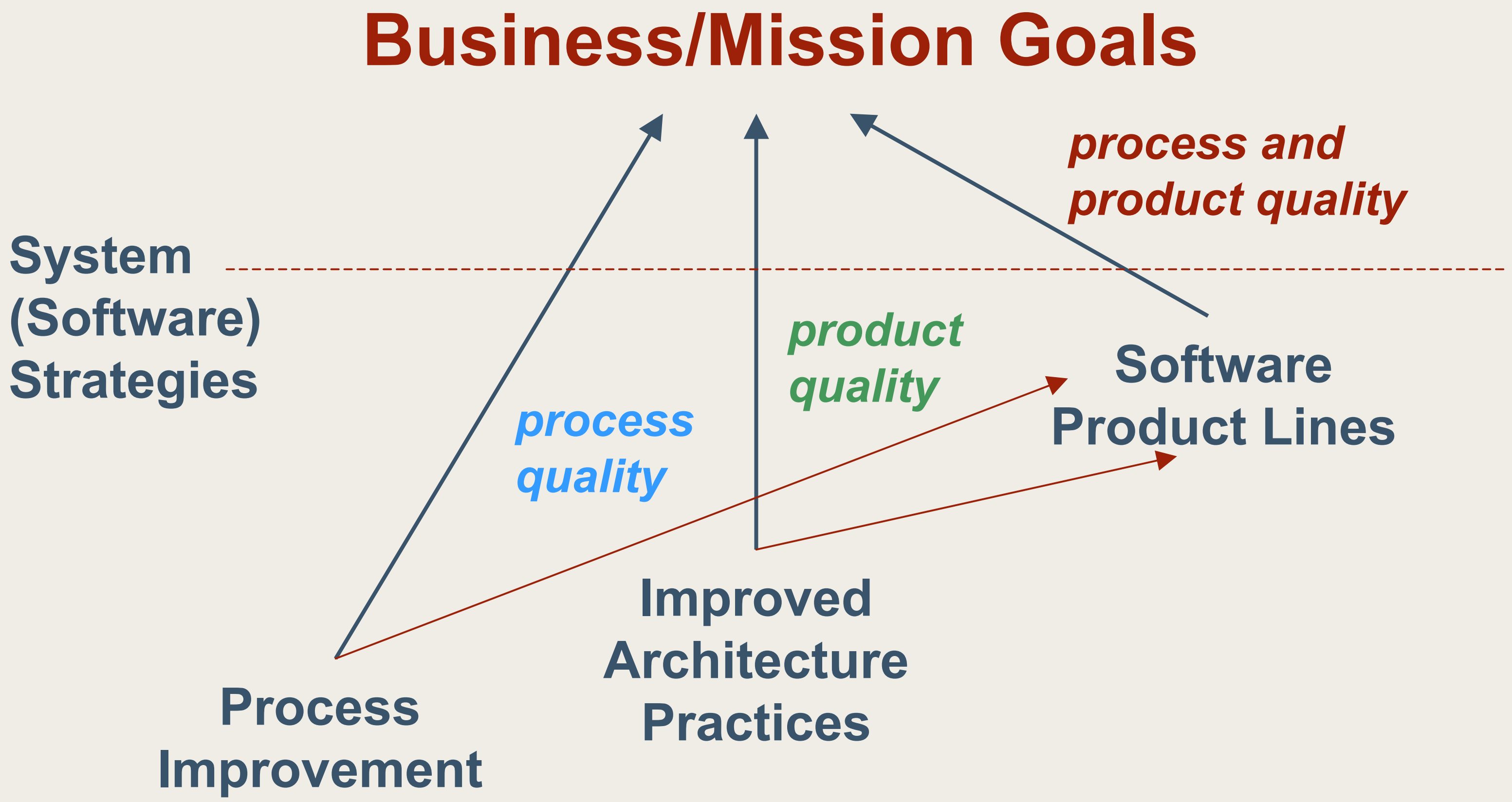
Process and product connection

© 2003 by Carnegie Mellon University

Software Product Line Strategy in Context



Software Product Line Strategy in Context



© 2003 by Carnegie Mellon University

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.

© 2003 by Carnegie Mellon University

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)

© 2003 by Carnegie Mellon University

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

© 2003 by Carnegie Mellon University

Status

PACC premises were validated on an internal system and through an ABB Feasibility Study.

PACC became an SEI 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

We are looking for organizations to collaborate with in the application of this research.

© 2003 by Carnegie Mellon University

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

© 2003 by Carnegie Mellon University

SEI Work in Software Architecture : 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

SEI Software Architecture Curriculum

NEW

Six courses

- Software Architecture: Principles and Practices
- 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 (short tutorial not part of the curriculum)

© 2003 by Carnegie Mellon University

About the Curriculum

Software professionals can take individual courses based on specific needs or interests
or complete one or more of the following three specially designed **certificate programs**:

- Software Architecture Professional
- ATAM^{SM} Evaluator
- ATAM^{SM} Lead Evaluator

The ATAM certificate programs qualify individuals to perform or lead SEI-authorized ATAM evaluations.

© 2003 by Carnegie Mellon University

Certificate Program Course Matrix

ATAM Lead Evaluator: 5 Courses & Coaching

Software Architecture Professional: 4 Courses	*Software Architecture: Principles and Practices*	*Documenting Software Architectures*	*Software Architecture Design and Analysis*	*Software Product Lines*
	ATAM Evaluator Training	*ATAM Facilitator Training*	*ATAM Coaching*	
	ATAM Evaluator 2 courses			

About all the Courses

All of the courses are two-day learning experiences that involve lectures and exercises.

The materials provided include books and class lecture slides.

Prerequisites are enforced.

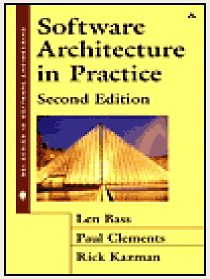
Delivery of the SEI software architecture courses is scheduled in 2003 at both the SEI Pittsburgh, PA and Frankfurt, Germany offices.

Any of the courses can also be scheduled for on site delivery.

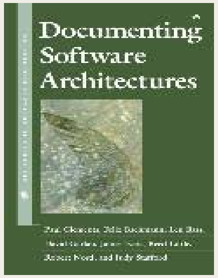
© 2003 by Carnegie Mellon University

Associated Texts

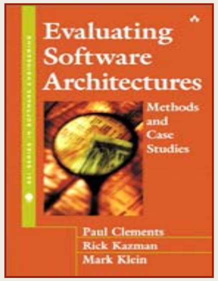
Software Architecture in Practice, 2nd Edition



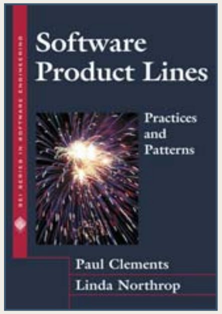
Documenting Software Architectures: Views and Beyond



Evaluating Software Architectures: Methods and Case Studies



Software Product Lines: Practices and Patterns



© 2003 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

2003 Schedule

2003 Courses	APR	MAY	JUN	JUL	AUG	SEPT	OCT	NOV	DEC
Software Architecture: Principles and Practices	16-17 PGH		23-24 PGH			4-5 EUR 22-23 PGH			2–3 PGH
Documenting Software Architectures			25-26 PGH			8 - 9 EUR			
Software Architecture Design and Analysis						24-25 PGH			3-4 EUR
Software Product Lines							15-16 EUR		9-10 PGH
ATAM Evaluator Training		20-21 PGH		16-17 PGH		10-11 EUR	15-16 PGH		9-10 PGH
ATAM Facilitator Training								18-19 PGH	

© 2003 by Carnegie Mellon

Carnegie Mellon
Software Engineering Institute

SEI Software Product Line Contributions

Practice Integration:

- A Framework for Software Product Line Practice^{SM}, Version 4.1, http://www.sei.cmu.edu/plp/framework.html
- Acquisition Companion to the Framework

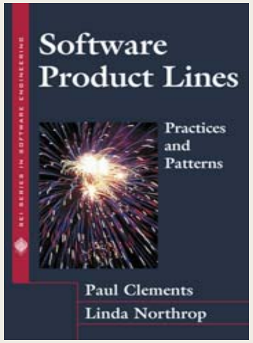
Techniques and Methods

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

Book

Software Product Lines: Practices and Patterns

- Practices (Framework, Version 4.0)
- patterns
- case studies



Conferences

SPLC 2004 – Sept 2004

© 2003 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

Spreading the Software Product Line Word

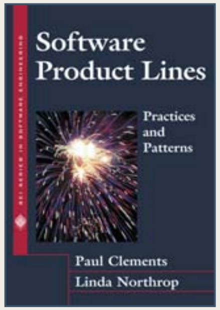
Courses

Book

Software product line concepts, practices, and patterns

Essentials of Software Product Lines

Software Product Lines



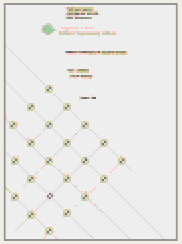
Architecture design

Attribute-Driven Design

Mining assets

Options Analysis for Reengineering^{SM}

Reports



Product line analysis

Product Line Analysis Tutorial

Acquisition Guidelines

Acquisition Executive Tutorial

Web

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion

Architecture Principles

Software architecture is important because it

- provides a communication vehicle among stakeholders
- is the result of the earliest design decisions
- is a transferable, reusable abstraction of a system

Every software-intensive system *has* a software architecture

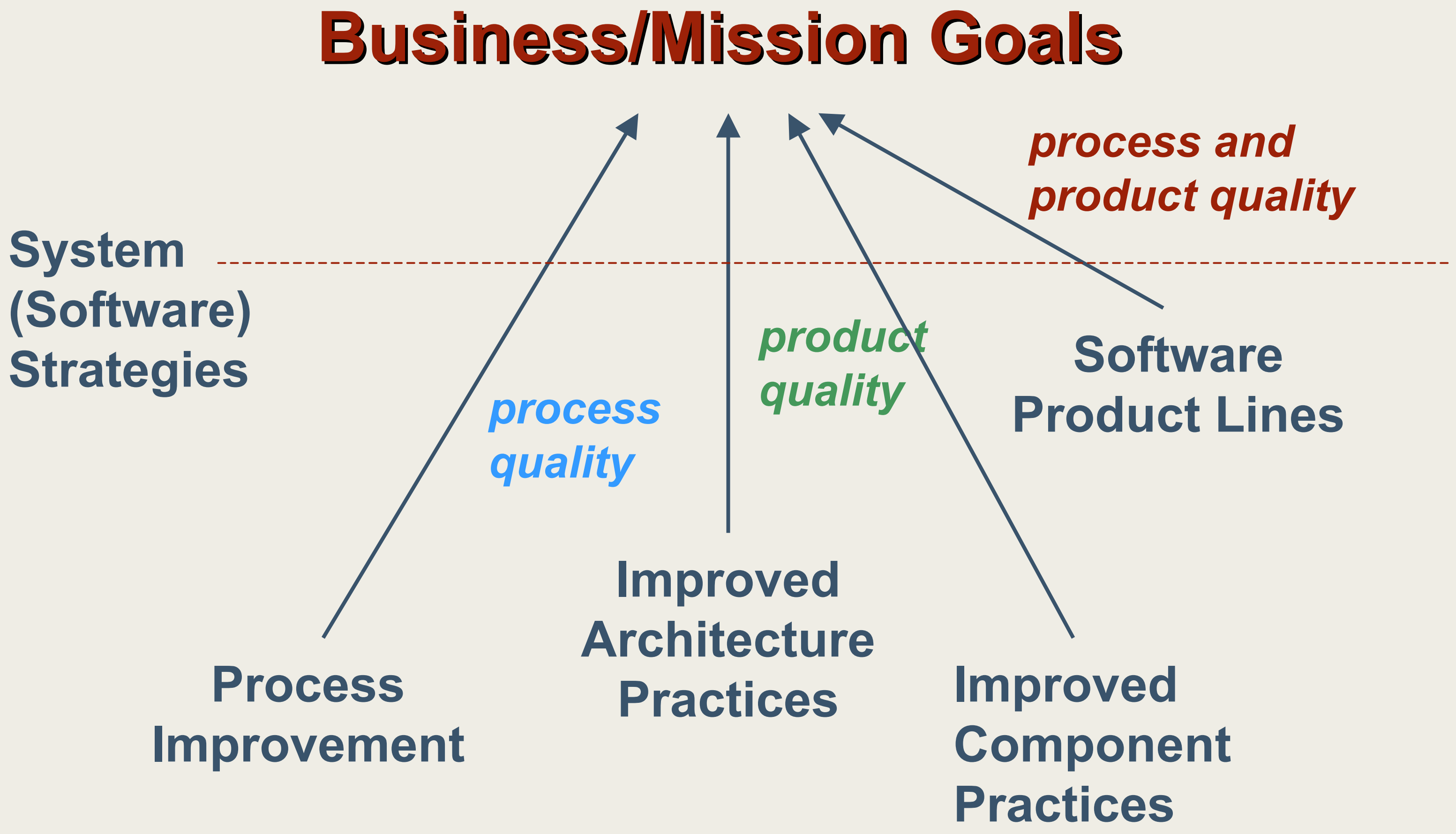
Just having an architecture is different from having an architecture that is known to everyone, much less one that is fit for the system's intended purpose.

An architecture-centric approach to development is essential for high product quality.

A software product line approach is a proven way to build high quality families of similar systems.

© 2003 by Carnegie Mellon University

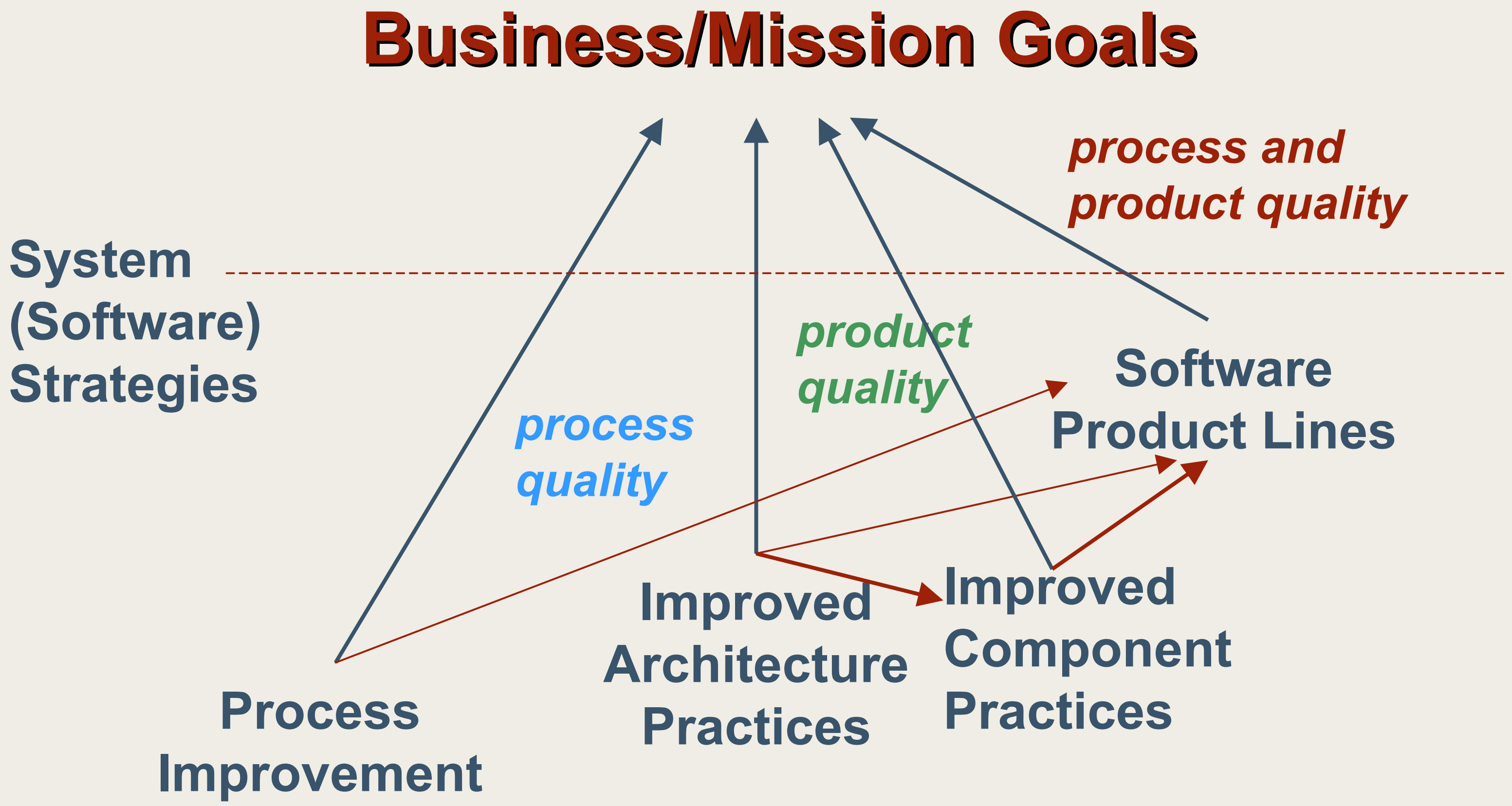
The Total Picture


Business/Mission Goals
process and product quality
System (Software) Strategies
product quality
Software Product Lines
process quality
Improved Architecture Practices
Process Improvement
Improved Component Practices

© 2003 by Carnegie Mellon University

Carnegie Mellon
Software Engineering Institute

The Total Picture


Business/Mission Goals
process and product quality
System (Software) Strategies
product quality
Software Product Lines
process quality
Improved Architecture Practices
Improved Component Practices
Process Improvement

© 2003 by Carnegie Mellon University

Conclusion

Software architecture is critical to product quality.

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

Contact Information

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

Terry Dailey
Program Integration Directorate
Navy Lead
Telephone: 703-908-8213
Email: etd@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

© 2003 by Carnegie Mellon University

Presentation Outline

Background

Software Architecture

Software Architecture Practices

Related Innovative Practices

SEI Software Architecture Support

Conclusion

Discussion