Achieving Product Qualities Through Software Architecture Practices

Presentation for
CSEE&T 2004
Mar 3, 2004

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.
Achieving Product Qualities Through Software Architecture Practices

Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, 15213

Approved for public release; distribution unlimited

Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion
Software Engineering Institute

Applied R&D laboratory situated as a college-level unit at Carnegie Mellon University, Pittsburgh, PA, USA

Established in 1984

Technical staff of 335

Offices in Pittsburgh, Pennsylvania (USA), Arlington, Virginia (USA) and Frankfurt Germany

Purpose: Help others improve their software engineering practices
SEI’s Strategic Functions

DoD needs
Technology trends

SEI’s experience
User’s experience

DoD needs and mature technology
Technology trends

SEI’s experience
User’s experience

Identify and mature technology

Direct support

Amplify

Transition
SEI and the Community

CREATE  APPLY  AMPLIFY  CREATE  APPLY
AMPLIFY  CREATE  APPLY  AMPLIFY  CREATE
APPLY  AMPLIFY  CREATE  APPLY  AMPLIFY
CREATE  APPLY  AMPLIFY  CREATE  APPLY
AMPLIFY  CREATE  APPLY  AMPLIFY  CREATE
APPLY  AMPLIFY  CREATE  APPLY  AMPLIFY
CREATE  APPLY  AMPLIFY  CREATE  APPLY

DEVELOPERS
ACQUIRERS
RESEARCHERS
Product Line Systems Program

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

Software Architecture
(Software Architecture Technology Initiative)

Software Product Lines
(Product Line Practice Initiative)

Component Technology
(Predictable Assembly from Certifiable Components Initiative)
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.
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
The Ultimate Universal Goal

Substantial
Quick
Sustainable
PROFIT
Software Strategies Are Needed

Business Goals

System (Software) Strategies

Process Improvement

Improved Architecture Practices

process quality

product quality
Presentation Outline

Background

**Software Architecture**

Quality Attributes

Software Architecture 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.
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

Architecture is an abstraction of a system.

Systems can and do have many structures.

Every system *has* an architecture.

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

If you don’t explicitly 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
  - 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.
Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion
Requirements Beget Design

Requirements in various forms

Available knowledge

System

Designer

Architecture
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

Quality Attribute Drivers → Software Architecture → Software

*analysis, design, development*

Has these qualities.
System Qualities and Software Architecture

System Specification
System Quality Attributes *

* Performance
Security
Interoperability
Reliability
Availability
etc.

Software Architecture

System Capabilities and Software Quality

determines level of quality

drives
Architecture and Functionality

Functionality is largely orthogonal to quality attribute requirements.

- Functionality is the ability of a system to do the work it was intended to do.
- Systems are decomposed into elements to achieve a variety of purposes other than function.
  - Architectural choices promote certain qualities as well as implement the desired functionality.
Effects of Architectural Decisions on Quality Attributes

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

- A change in structure improving one quality often affects the other qualities.
- Architecture is critical to the realization of quality attributes.
- These product qualities should be designed into the architecture.
- Architecture can only permit, not guarantee, any quality attribute.
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?
Quality Attribute Scenarios – 1

A solution to the problem of describing quality attributes is to use quality attribute scenarios as a means to better characterize quality attributes.

A quality attribute scenario consists of six parts.
Quality Attribute Scenarios – 2

1. **stimulus** – a condition that affects the system
2. **response** – the activity that results because of the stimulus
3. **source of the stimulus** – the entity that generated the stimulus
4. **environment** – the condition under which the stimulus occurred
5. **artifact stimulated** – the artifact that was stimulated by the stimulus
6. **response measure** – the measure by which the system’s response will be evaluated
Parts of a Quality Attribute Scenario

Source → Stimulus → Artifact: Process, Storage, Processor, Communication → Environment → Response → Response Measure
General and Concrete Scenarios

General scenarios

- are those scenarios that are system independent
- represent quality attribute characterizations
- can be used to create concrete scenarios that are specific to a particular system.

General six-part scenarios exist for

- availability
- modifiability
- performance
- security
- testability
- usability
Modifiability – 1

Definition: Modifiability is about the cost of change and refers to the ease with which a software system can accommodate changes.

Areas of concern include

• identifying what can change
  - functions, platforms, hardware, operating systems, middleware, systems it must operate with, protocols, and so forth
  - quality attributes: performance, reliability, future modifiability, and so forth

• When will the change be made and who will make it?
Modifiability – 2

General scenario considerations:

<table>
<thead>
<tr>
<th>Source</th>
<th>End user, developer, system administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Add/delete/modify functionality or quality attribute</td>
</tr>
<tr>
<td>Environment</td>
<td>Runtime, compile time, build time, design time</td>
</tr>
<tr>
<td>Artifacts</td>
<td>System: user interface, platform, environment, system that interoperates with target system</td>
</tr>
</tbody>
</table>
## General scenario considerations (continued):

| Response | • Locate places in the architecture to be modified.  
|          | • Make modifications without affecting other functionality.  
|          | • Test the modification with minimal effort.  
|          | • Deploy the modification with minimal effort.  
| Response | • Cost in terms of the number of affected components, effort, and money  
| Measure  | • Extent to which this modification affects other functions and/or quality attributes  

Sample Modifiability Scenario

A developer wishes to change the user interface (UI) code at design time. The modification is made with no side effects, in three hours.

<table>
<thead>
<tr>
<th>Source</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Wishes to change the UI</td>
</tr>
<tr>
<td>Artifact</td>
<td>Code</td>
</tr>
<tr>
<td>Environment</td>
<td>At design time</td>
</tr>
<tr>
<td>Response</td>
<td>Modification is made with no side effects</td>
</tr>
<tr>
<td>Response Measure</td>
<td>In three hours</td>
</tr>
</tbody>
</table>
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.
Influence of System Stakeholders - 1

Stakeholders have an interest in the construction of a software system. Stakeholders might include
• customers
• users
• developers
• project managers
• marketers
• maintainers

Stakeholders have different concerns that they wish to guarantee and/or optimize.
Influence of System Stakeholders – 2

Development organization's management stakeholder
- Low cost, keeping people employed, leveraging existing corporate assets!

Marketing stakeholder
- Neat features, short time to market, low cost, parity with competing products!

End user stakeholder
- Behavior, performance, security, reliability, usability!

Maintenance organization stakeholder
- Modifiability!

Customer stakeholder
- Low cost, timely delivery, not changed very often!

How can I make sure the system has all that?
Stakeholder Involvement

The organizational goals and the system properties required by the business are rarely understood, let alone fully articulated.

Customer quality attribute requirements are seldom documented, which results in
- goals not being achieved
- inevitable conflict between different stakeholders

Architects must identify and actively engage stakeholders in order to
- understand real constraints of the system
- manage the stakeholders’ expectations
- negotiate the system’s priorities
- make tradeoffs
Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion
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 Technology

Attribute-specific patterns
Architecture expert

Life Cycle Practices

• Architectural requirements elicitation
• Architecture definition
• Architecture representation
• Architecture evaluation
• Architecture reconstruction
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
Specific system architecture
  Software architecture
Detailed design
Implementation

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

a miracle occurs
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
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
QAW Benefits and Next Steps

Potential Benefits
- Increased stakeholder communication
- Clarified quality attribute requirements
- Informed basis for architectural decisions

Potential Next Steps
- Update Architectural Vision
- Refine Requirements
- Create Prototypes
- Exercise Simulations
- Create Architecture
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.
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.
Tactics Catalog

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

Others are in the works.
Performance Tactics

Summary of performance tactics
Attribute-Driven Design

The Attribute-Driven Design (ADD) method, developed at the SEI, is an approach to defining a software architecture that bases the decomposition process on the quality attributes the software must fill.

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.
Evolutionary Delivery Life Cycle

ADD is positioned after requirements analysis and can begin when architectural drivers are known with some confidence.
ADD Method's Inputs and Outputs

Inputs

• constraints
• functional requirements
• quality attribute requirements

Outputs

• first several levels of module decomposition
• various other views of the system as appropriate
• set of elements for functionality and the interactions among them
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.
Views

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

Not all system elements, some of them.

A view binds an element type and relation type of interest, and illustrates them.
View-Based Documentation

Views give us our basic principle of architecture documentation:

*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 are

1. education - introducing people to the project
2. communication - among stakeholders
3. analysis - assuring quality attributes
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

Specific system architecture
Software architecture

Detailed design
Implementation

A miracle occurs

A Critical leap!
How do you know if the architecture is fit for purpose?

another miracle occurs
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.
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
The **ATAM**

The SEI has developed the Architecture Tradeoff Analysis Method\textsuperscript{SM} (ATAM\textsuperscript{SM}).

The purpose of ATAM is: \textit{to assess the consequences of architectural decisions in light of quality attribute requirements and business goals.}
Purpose of ATAM – 1

The ATAM is a method that helps stakeholders ask the right questions to discover potentially problematic architectural decisions.

Discovered risks can then be made the focus of mitigation activities: e.g. further design, further analysis, prototyping.

Surfaced tradeoffs can be explicitly identified and documented.
Purpose of ATAM – 2

The purpose of the ATAM is NOT to provide precise analyses . . . the purpose IS to discover risks created by architectural decisions.

We want to find trends: correlation between architectural decisions and predictions of system properties.
ATAM Phases

ATAM evaluations are conducted in four phases.

Phase 0: Partnership and Preparation
- Duration: varies
- Meeting: primarily phone, email

Phase 1: Initial Evaluation
- Duration: 1.5 - 2 days each for Phase 1 and Phase 2
- Meeting: typically conducted at customer site

Phase 2: Complete Evaluation
- Duration: varies
- Meeting: primarily phone, email

Phase 3: Follow-up
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
Example Utility Tree

Utility

Performance
- Data Latency
- Transaction Throughput

Modifiability
- New product categories
- Change COTS

Availability
- H/W failure
- COTS S/W failures

Security
- Data confidentiality
- Data integrity

- Reduce storage latency on customer DB to < 200 ms.
- Deliver video in real time
- Add CORBA middleware in < 20 person-months
- Change web user interface in < 4 person-weeks
- Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
- Restart after disk failure in < 5 minutes
- Network failure detected and recovered in < 1.5 minutes
- Credit card transactions are secure 99.999% of the time
- Customer DB authorization works 99.999% of the time
Conceptual Flow of the ATAM

1. Business Drivers
2. Quality Attributes
3. Scenarios
4. Architectural Approaches
5. Architectural Decisions
6. Risk Themes
7. Sensitivity Points
8. Non-Risks
9. Risks

Analysis

Tradeoffs

QAW
When to Use the ATAM

The ATAM can be used throughout the life cycle *when there is a software architecture to evaluate.*

The ATAM can be used

- after an architecture has been specified but there is little or no code
- to evaluate architectural alternatives
- to evaluate the architecture of an existing system

To perform an ATAM evaluation, *there must be a software architecture to evaluate.*

- An ATAM evaluation is inappropriate if the software architecture of the system has not been created yet.
ATAM Benefits

The benefits of performing ATAM evaluations include

• clarified quality attribute requirements
• improved architecture documentation
• documented basis for architectural decisions
• identification of risks early in the life cycle
• increased communication among stakeholders

The result is improved architectures.
Architecture Evaluation Experience

Benefits of early architecture evaluations
• Evaluations using the Architecture Tradeoff Analysis Method℠ (ATAM℠) 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
Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

**SEI Software Architecture Support**

Conclusion

Discussion
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 Technology
- Attribute-specific patterns
- Architecture expert

Life Cycle Practices
- Architectural requirements elicitation
- Architecture definition
- Architecture representation
- Architecture evaluation
- Architecture reconstruction

Apply/Amplify
- Architecture Evaluations
- Architecture Coaching
- Architecture Reconstructions
- Books
- Courses
- Certificate Programs
- Acquisition Guidelines
- Technical Reports
- Web site
- Workshops

© 2004 by Carnegie Mellon University
SEI Software Architecture Curriculum

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

Coming in 2005: SEI Software Product Line Curriculum
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\textsuperscript{SM} Evaluator
- ATAM\textsuperscript{SM} Lead Evaluator

The ATAM certificate programs qualify individuals to perform or lead SEI-authorized ATAM evaluations.
# Certificate Program Course Matrix

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

Any of the courses can also be scheduled for on site delivery.
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
The Software Architecture Workshop for Educators is a three-day forum for sharing SEI software architecture technology with educators and for jointly determining ways to incorporate these concepts and methods into academic courses.

Schedule: Aug 16-17: Software Architecture: Principles and Practices Course
Aug 18: Facilitated Discussion
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

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

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 is critical to achieving and implementing an appropriate architecture.
SEI Unique Contribution

The SEI work in software architecture technology and its associated methods are notably unique in their

- explicit focus on quality attributes
- direct linkage to business and mission goals
- explicit involvement of system stakeholders
- high-quality published materials for practitioner consumption
- grounding in state-of-the-art quality attribute models and reasoning frameworks
The Total Picture

Business/Mission Goals

System (Software) Strategies

Process Improvement

Improved Architecture Practices

Improved Component Practices

Software Product Lines

process and product quality

process quality

process quality
The Total Picture

Business/Mission Goals

System (Software) Strategies

Process Improvement

Improved Architecture Practices

Improved Component Practices

Software Product Lines

process and product quality

process quality

product quality

process quality
Conclusion

Software architecture is critical to achieving key product qualities.

Software architecture, product line practices, and predictable component practices hold great potential for achieving business and mission goals in the development of software-intensive systems.
Contact 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
Presentation Outline

Background

Software Architecture

Quality Attributes

Software Architecture Practices

SEI Software Architecture Support

Conclusion

Discussion