
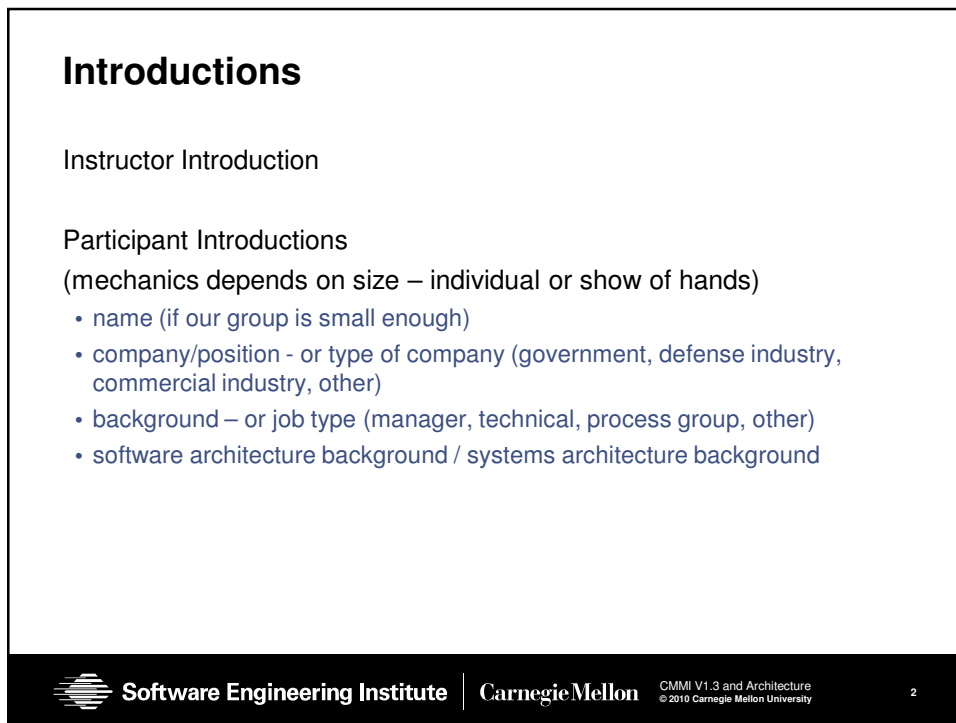


CMMI V1.3 and Architecture

Larry Jones
Mike Konrad

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-2612

 **Software Engineering Institute** | **CarnegieMellon** © 2010 Carnegie Mellon University




Introductions

Instructor Introduction

Participant Introductions
(mechanics depends on size – individual or show of hands)

- name (if our group is small enough)
- company/position - or type of company (government, defense industry, commercial industry, other)
- background – or job type (manager, technical, process group, other)
- software architecture background / systems architecture background

 **Software Engineering Institute** | **CarnegieMellon** CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University 2

Tutorial Learning Outcomes

After completing this half-day tutorial, attendees should

- know the importance of architecture to the achievement of business, product, or mission goals
- know that quality attributes have a dominant influence on a system's architecture
- be familiar with essential architecture-centric engineering activities and some example methods
- know how to specify quality attributes meaningfully through scenarios
- be able to identify where architecture-centric activities and work products are described in CMMI V1.3
- appreciate how to interpret the new architecture-centric material in CMMI V1.3
- know where to find out more about architecture-centric engineering practices



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

3

Conventions & Caveats for the Tutorial

The coverage of architecture-centric practices in CMMI V1.3 are not restricted to software;

- however, the tutorial providers are most conversant with that domain and thus so is this tutorial.

CMMI V1.3 includes updates to CMMI for Acquisition and CMMI for Services. Our focus in the tutorial will be on CMMI for Development but we will often adopt the shorthand "CMMI V1.3."

CMMI uses the term "product" to refer to what is delivered to the customer or end-user. In this tutorial, we will often use the term "system" to refer to the product.

This tutorial cannot completely convey everything you might like to learn about architecture-centric engineering.

- References are provided at the end for you to learn more.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

4



Expected Background of Participants

Participants must have an understanding of the basics of CMMI models.

- This tutorial is not an introduction to CMMI.
- It is not a substitute for upgrade training.

Familiarity with system and software design is useful, but not required.



Topics to be Covered

CMMI V1.3 – Modern Engineering Practices

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers

There are hands-on exercises to give you a grounding in some key concepts.



Presentation Outline

CMMI V1.3 – Context for modern engineering practices changes

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

7

Modern Development Practices in CMMI - The Problem - 1

Much of the engineering content of DEV V1.2 is ten years old.

As DEV was a starting point for the other two constellations, no V1.2 model adequately addresses “modern” engineering approaches.

For example, RD SG 3 and RD SP 3.2 both emphasize functionality and not non-functional requirements (CMMI-SVC SSD SP 1.3 also does too).

Also, Engineering and other PAs rarely mention the following concepts:

- Quality attributes
- Allocation of product capabilities to release increments
- Product lines
- System of systems
- Architecture-centric development practices
- Technology maturation (and obsolescence)
- Agile methods



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

8



Modern Development Practices in CMMI - The Problem - 2

The slides that follow portray where we should be today relative to architecture-centric practices – as opposed to how they were portrayed in CMMI V1.2.

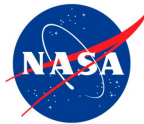
Towards the end of today’s half-day tutorial, we will revisit how CMMI Version 1.3 addresses these and other modern development practices.



Architecture is Important

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

In recent studies by OSD, the National Research Council, NASA, and the NDIA, architectural issues are identified as a systemic cause of software problems in DoD systems.



People are Serious About Architecture

“Software Architect” was identified by CNN Money.com as the #1 “Best Job in America.” (Oct 2010)¹



The US Army has mandated that all Program Executive Offices appoint a Chief Software Architect. (May 2009)²

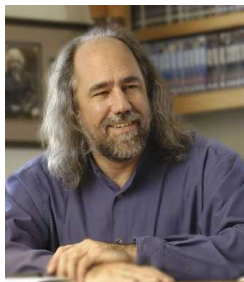


1. <http://money.cnn.com/magazines/moneymag/bestjobs/2010/snapshots/1.html>
2. Memo by LTG N. Ross Thompson, Mil Dept of ASA (ALT) on May 26, 2009.



“Every system has an architecture...

...encompassing the key abstractions and mechanisms that define that system's structure and behavior... In every case - from idioms to mechanisms to architectures - these patterns are either



intentional

or

accidental”

- Grady Booch in the Preface to *Handbook of Software Architecture*



Architecture and Strategy

An *Intentional Architecture* is the embodiment of your business strategy

- Intentional Architecture links technology decisions to business goals



An *Accidental Architecture* limits strategy options

- Accidental Architecture becomes your de facto strategy



Presentation Outline

CMMI V1.3 – Context for modern engineering practices changes

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers



DoD Systems are Increasingly Complex...



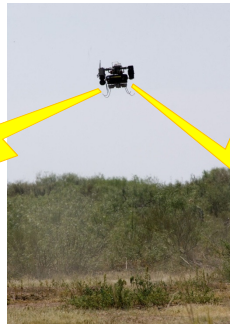
Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

15

...Systems of Systems (SoS) even more so



More and more, software is the integrating element in all manner of systems...



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

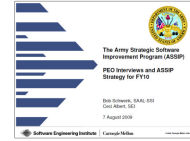
16



Coping with System/Software Complexity is a Must

2008-2009 Interviews with Army PEOs

- Relationship between system engineering and software engineering is driving system complexity
- *Example:* Army Software Blocking/Network Capability Sets - decade-long attempt to horizontally integrate Battle Command software across brigade elements



2009 NASA Study

- Software complexity leads to system and operational complexity (and increases risk)



2009 MIT Study

- Software causes systems to become "interactively complex" (intellectually unmanageable)



Architecture-Centric Practices are Key...

Defense Science Board (1994 & 2000)

- Software architecture techniques can reduce cost and cycle times
- Architecture is "a central theme for software reuse, product lines, and greater exploitation of commercial technology and practices"

Army Workshop on Weapon Software Upgrade Programs (2001)

- Architecture is "a key technical focus for the system"
- Architecture is critical in determining the future ability to upgrade the system
- In 2008, GAO testimony noted similar findings for DoD business systems

NASA (2009)

- "Good software architecture is the most important defense against incidental complexity in software designs, but good architecting skills are not common" →



...But Practices Haven't Kept Up

DoD Tri-Service Assessment Initiative (2003)

- Review of 21 DoD program assessments
 - poor software architecture practices are one of the *systemic causal factors* of software-reliant systems issues



SEI surveys and interviews of Army PMs and PEOs (2004 & 2005)

- PMs/PEOs felt prime contractors' software architecture abilities were only about average
 - Yet, they also felt government program office staffs were not sufficiently skilled to evaluate software architectures



SEI analysis of results from 18 architecture evaluations (2006)

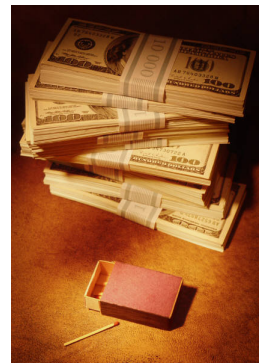
- >50% of the programs had significant *program* risks driven by lack of architecture training/tools and poor architecture planning
- ~2/3 of risks discovered were risks of *omission*
 - e.g., architectural decisions either not made or not captured



Fixing this Sounds Expensive!

Compared to what?

- Over-committing because you don't have a blueprint for the whole system?
- Inefficiency from inability to coordinate work?
- Late rework when defects found in test and integration?
- Delivering late and over budget?
- Developing a failed product that doesn't meet stakeholder's needs?



Architecture is About Structure and Decisions

Structures result from decisions

- Business / mission goals provide a reasoned basis for decisions.
- Each decision is a tradeoff that enables something and precludes other things.
- Tradeoffs are driven by quality attribute requirements.



This is true regardless of the domain
– commercial or defense.



Class Exercise 1



Value Proposition for Architecture-Centric Engineering



Architecture-centric engineering enables the ongoing cost-effective achievement of system-related business and mission goals.

- Early identification and mitigation of design risks result in fewer downstream, costly problems and cost savings in integration and test.
- Sound structure analyses provide objective confidence for achieving system quality.
- Predictable system quality supports the achievement of business and mission goals, which translates into competitive advantage.
- Appropriate flexibility enables cost-effective system evolution.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

23

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

Key to system **evolution**



- manage future uncertainty
- assure cost-effective agility

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



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

24



Software Architecture and Development and Acquisition Risk

Risk mitigation early in the life cycle is key.

- The software architecture is an early life cycle artifact.
- Mid-course correction is possible before great investment.
- Risks don't become problems that have to be addressed during integration and test.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

25

Agile Architecture = Responsiveness

Architecture-centric engineering and an agile development approach are *not* at odds.

Agile development approaches enable you to

- Take on large projects and initiatives
- Break them into smaller chunks (iterations)
- Manage risk
 - Execute-Learn-Feedback-Improve

Agile Architecture provides the blueprint for your iterations

- Enable efficient incremental development
- Minimize technical debt
- Early analysis of qualities like performance and availability
- Efficiently address global qualities like security



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

26



Common Symptoms Stemming From Architectural Deficiencies

Operational

- Communication bottlenecks under various load conditions in a system or throughout a system of systems (SoS)
- Systems that hang up or crash; portions that need rebooting too often
- Difficulty synching up after periods of disconnect and resume operations
- Judgment by users that system is unusable for variety of reasons
- Database access sluggish and unpredictable

Developmental

- Integration schedule blown, difficulty identifying root causes of problems
- Proliferation of patches and workarounds during integration and test
- Integration of new capabilities taking longer than expected, triggering breaking points for various resources
- Significant operational problems ensuing despite passage of integration and test
- Anticipated reuse benefits not being realized



Sample Issues Detectable From Architectural Decisions

Availability:

- Having a single point of failure
- Having no availability mechanisms
- Using an infrastructure that does not support availability mechanisms

Performance:

- Not knowing performance requirements
- Failure to meet performance requirements
 - Not performing any performance modeling or prototyping
 - Unfamiliarity with infrastructure choices
 - Not using known performance mechanisms

Security:

- No support for security
- Not using known mechanisms to support security goals

Modifiability:

- Allocating functionality in a way that jeopardizes portability
- Not supporting the addition and deletion of different devices
- Lack of attention to potential growth paths

Integration:

- Problems with migrating legacy systems
- Lack of uniformity in key areas



This is What Happens



without careful architectural design.
And so it is with software.



Without Effective Software Architecture Practices

.... you get poorly designed software architectures.

Poorly designed software architectures result in

- Greatly inflated integration and test costs
- Inability to sustain systems in a timely and affordable way
- Lack of system robustness
- Undesired, disparate behaviors at the system and at the system-of-systems levels
- In the worst case, product or project cancellation
- In all cases, failure to best support the war fighter

Failure to Meet Business and Mission Goals



A Warning (PERMISSION REQUESTED)

“Architecture” is a very overloaded word.

- All the good words are taken.
- We will explain some common uses of the term and how they differ.



What Is A Software Architecture?

Informally, software architecture is the blueprint describing the software structure of a system.



Formal Definition

“The **software architecture** of a program or computing system is the **structure or structures** of the system, which comprise the **software elements, the externally visible properties** of those elements, and the **relationships among them**.”¹

¹ Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

33

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!



Software Engineering Institute

CarnegieMellon

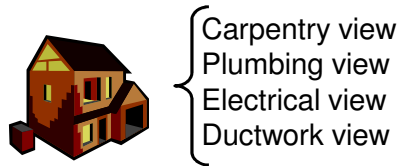
CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

34



Structures and Views - 1

One house, many views



No single view accurately represents the house.

No single view can be used to build the house.

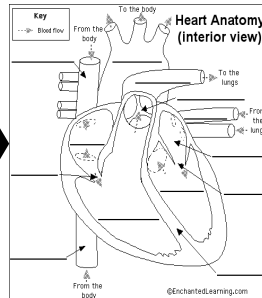
Although these views are pictured differently, and each has different properties, all are related. Together, they describe the architecture of the house.



Structures and Views - 2



A human body comprises multiple *structures*.



a *static view* of one human *structure*



a *dynamic view* of that *structure*

One body has many structures, and those structures have many views. So it is with software.



Enterprise Architecture

Enterprise architecture is a means for describing business structures and the processes that connect them.¹

- Describes the flow of information and activities between various groups within the enterprise that accomplish some overall business activity

Software and its design are not typically addressed explicitly in an enterprise architecture.

¹ Zachman, John A., "A Framework for Information Systems Architecture." *IBM Systems Journal*, 26, 3 (1987): 276-292.



System Architecture

A **system architecture** describes the elements and interactions of a complete system including its hardware elements and its software elements.

System Architecture: "The fundamental and unifying system structure defined in terms of system elements, interfaces, processes, constraints, and behaviors."¹

Systems Engineering is a design and management discipline useful in designing and building large, complex, and interdisciplinary systems.²

¹ Rehtin, E. *Systems Architecting: Creating and Building Complex Systems*. Englewood Cliffs, NJ : Prentice-Hall, 1991.

² International Council On Systems Engineering (INCOSE), Systems Architecture Working Group, 1996.



Where Does Software Architecture Fit?

Enterprise architecture and system architecture provide an environment in which software lives.

- Both provide requirements and constraints to which software architecture must adhere.
- Both are affected by the properties of the software architecture.
- Elements of both are likely to contain software architecture.
- **Neither substitutes for or obviates a software architecture.**

There is a mutual influence and interaction between software, system, and enterprise architectures.

In a large, complex, software-reliant system **both software and system architectures are critical** for ensuring that the system meets its business and mission goals.



What About System of Systems?

Each software-intensive system in a system of systems (SoS) has system and software architectures.

The system of systems has an architecture where the elements are themselves the software architectures of the individual systems.

Software architecture is even more important in an SoS context, not less.



Does DoDAF Address Software Architecture?

Unfortunately, no.

- DoDAF views are required
- software architecture views are not

The Department of Defense Architecture Framework (DoDAF) describes an “architecture” for a large-scale system or system-of-systems.

DoDAF uses the concept of views of a system

- operational view (OV) – participant relationships and information needs
- system (SV) – relates capabilities and characteristics to operational requirements
- technical (TV) – prescribes standards and conventions
- all (AV)

DoDAF views were developed for different purposes and do not address software architecture.



Presentation Outline

CMMI V1.3 – Context for modern engineering practices changes

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers

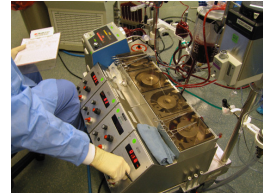


What is Architecture-Centric Engineering?

Architecture-Centric Engineering (ACE) is the discipline of using architecture as the focal point for performing ongoing analyses to gain increasing levels of confidence that systems will support their missions.

Architecture is of enduring importance because it is the right abstraction for performing ongoing analyses throughout a system's lifetime.

The **SEI ACE Initiative** develops principles, methods, foundations, techniques, tools, and materials in support of creating, fostering, and stimulating widespread transition of the ACE discipline.



The Variety of Software-Reliant Systems



Embedded systems
software embedded in hardware devices



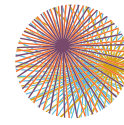
Stand-alone systems
software applications



Software product lines
families of similar systems



Systems of systems
federations of independent systems



Ultra-large-scale systems
webs of software-reliant systems, people, economies, and cultures

There are interactions among these types of systems. The behavior of all these systems is largely determined by their structure.

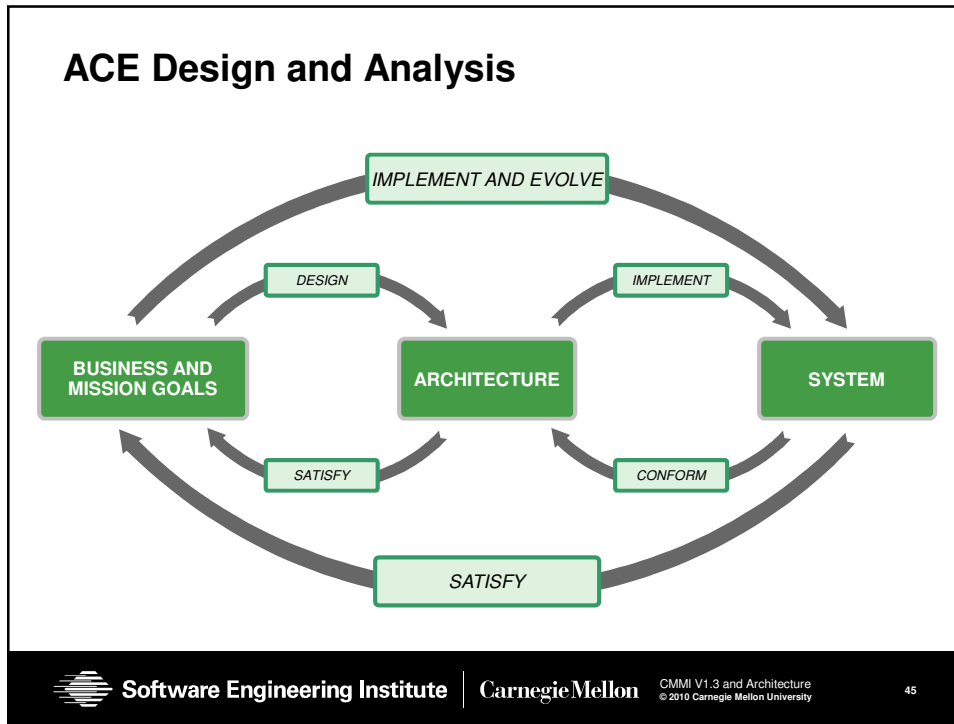
Architecture-centric engineering addresses all types and scales of systems.

Predict and control behavior

Assure and bound behavior

Coupling to organizational structure and practices increases





- ### Principles of ACE
1. Regardless of scale, architecture is the **appropriate abstraction** for reasoning about business/mission goal satisfaction.
 2. **Quality attributes** have a dominant influence on a system's architecture.
 3. Architectural prescriptions must be demonstrably satisfied by the **implementation**.
- Software Engineering Institute | CarnegieMellon CMMI V1.3 and Architecture © 2010 Carnegie Mellon University 46

Architecture – A Bridge to Goal Satisfaction

A good architectural representation should have

- sufficient detail to reason about mission and business goal satisfaction
- sufficient abstraction for a relatively small number of architects to conceptually understand the system
- sufficient detail to appropriately constrain implementation.



All design involves tradeoffs.

Lacking mission and business drivers, the architect has to make assumptions about priorities.

Given well-stated mission and business drivers, the architect has a basis for knowing the priorities among tradeoffs.





Principles of ACE

1. Regardless of scale, architecture is the **appropriate abstraction** for reasoning about business/mission goal satisfaction.
2. **Quality attributes** have a dominant influence on a system's architecture.
3. Architectural prescriptions must be demonstrably satisfied by the **implementation**.



Software System Development



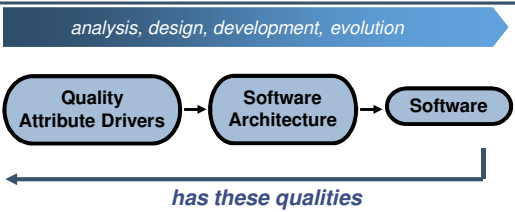


If function were all that mattered, any monolithic software would do, ...**but other things matter...**


The important quality attributes and their characterizations are key.

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

analysis, design, development, evolution



The Non-functional Requirements



Software Engineering Institute | CarnegieMellon


CMMI V1.3 and Architecture
 © 2010 Carnegie Mellon University

49


Quality Attribute Requirements

Quality attributes include

- Performance
- Availability
- Interoperability
- Modifiability
- Usability
- Security
- Etc.



Quality attribute requirements stem from business and mission goals. Key quality attributes need to be characterized in a system-specific way. Otherwise, they are not operational.



Software Engineering Institute | CarnegieMellon

CMMI V1.3 and Architecture
 © 2010 Carnegie Mellon University

50

Users Need Both Functions and Qualities

Required capability
 Low learning threshold
 Ease of use
 Predictable behavior
 Dependable service
 Timely response
 Timely throughput
 Protection from unintended intruders and viruses



.....

Software system/mission goals should address user needs.

User needs often translate to quality attribute requirements.

Scenarios are a powerful way to characterize quality attributes and represent user and other stakeholder views.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
 © 2010 Carnegie Mellon University

51

Specifying Quality Attributes

Quality attributes are rarely captured *effectively* in requirements specifications; they are often vaguely understood and weakly articulated.

Just citing the desired qualities is not enough; it is meaningless to say that the system shall be “modifiable” or “interoperable” or “secure” without details about the context.

The practice of specifying quality attribute scenarios can remove this imprecision and allows desired qualities to be evaluated meaningfully.

A quality attribute scenario is a short description of an interaction between a stakeholder and a system and the response from the system.



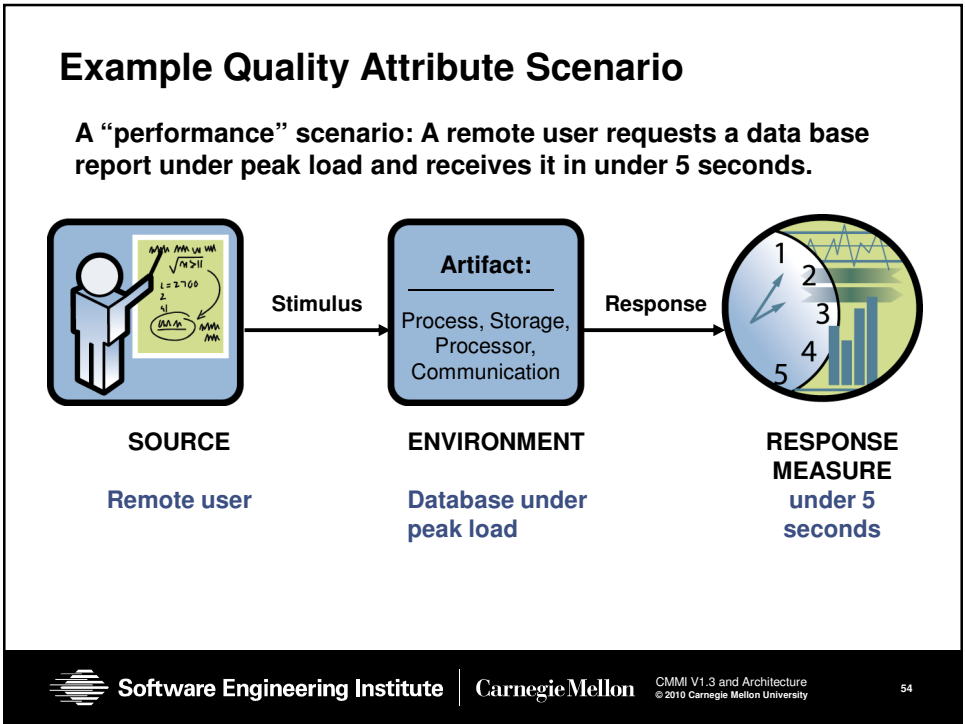
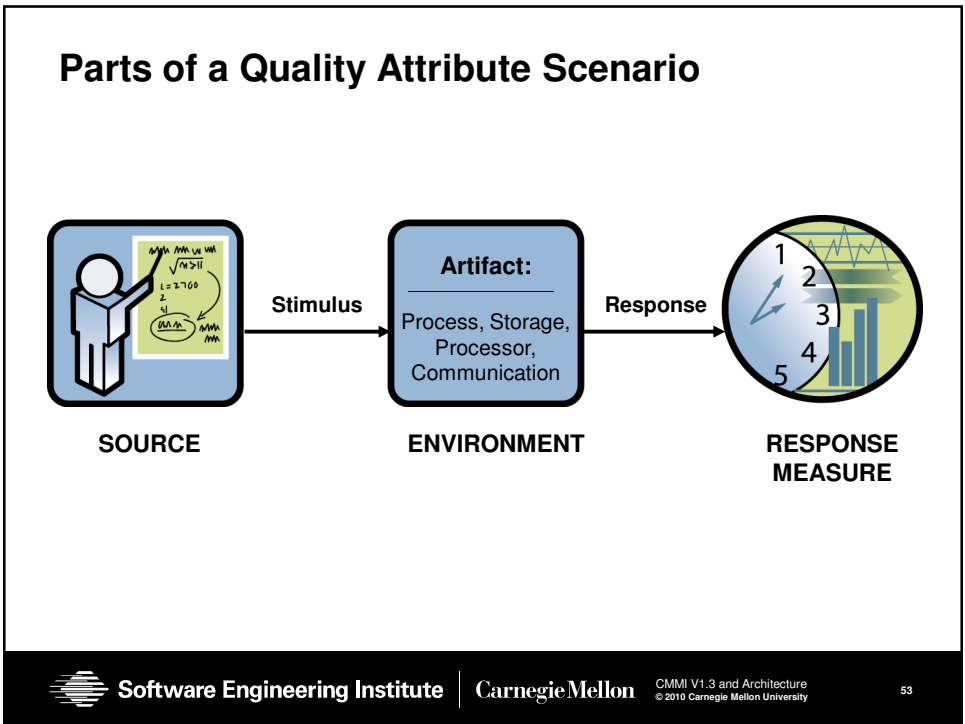
Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
 © 2010 Carnegie Mellon University

52





Class Exercise 2



Principles of ACE

1. Regardless of scale, architecture is the **appropriate abstraction** for reasoning about business/mission goal satisfaction.
2. **Quality attributes** have a dominant influence on a system's architecture.
 - Quality attribute requirements stem from business and mission goals.
 - Key quality attributes need to be characterized in a system-specific way.
 - Scenarios are a powerful way to characterize quality attributes and represent stakeholder views.
3. Architectural prescriptions must be demonstrably satisfied by the **implementation**.

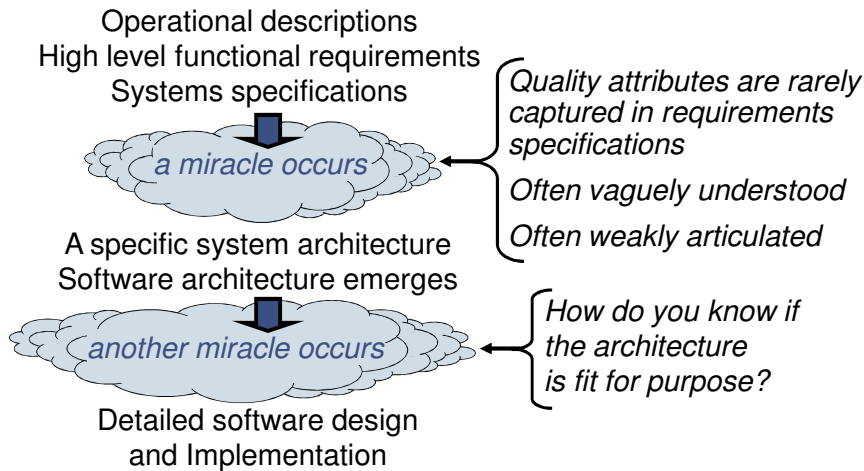


Principles of ACE

1. Regardless of scale, architecture is the **appropriate abstraction** for reasoning about business/mission goal satisfaction.
2. **Quality attributes** have a dominant influence on a system's architecture.
3. Architectural prescriptions must be demonstrably satisfied by the **implementation.**



Typical Software Development Paradigm



Architecture-Centric Activities

Architecture-centric activities include the following:

- creating the **business case** for the system
- understanding the **requirements**
- **creating and/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
- **evolving** the architecture so that it **continues to meet business and mission goals**



Some SEI Techniques, Methods, and Tools

creating the business case for the system	
understanding the requirements	<i>Quality Attribute Workshop (QAW) Mission Thread Workshop (MTW)</i>
creating and/or selecting the architecture	<i>Attribute-Driven Design (ADD) and ArchE</i>
documenting and communicating the architecture	<i>Views and Beyond Approach; AADL</i>
analyzing or evaluating the architecture	<i>Architecture Tradeoff Analysis Method (ATAM); SoS Arch Eval; Cost Benefit Analysis Method (CBAM); AADL</i>
implementing the system based on the architecture	
ensuring that the implementation conforms to the architecture	<i>ARMIN</i>
evolving the architecture so that it continues to meet business and mission goals	<i>Architecture Improvement Workshop (AIW) and ArchE</i>
ensuring use of effective architecture practices	<i>Architecture Competence Assessment</i>



Building the Business Case for the System

How to do this is beyond the scope of this tutorial.

Some common business / mission drivers for systems include

- Reduce total cost of ownership
- Improve capability/quality of system
- Improve market position
- Support improved business processes
- Improve confidence in and perception of system

Results gleaned from

- 25 architecture evaluations
 - 18 government systems, 7 commercial systems
- 190 distinct business goals

Kazman & Bass, *Categorizing Business Goals for Software Architectures*, CMU/SEI-2005-TR-021

<http://www.sei.cmu.edu/reports/05tr021.pdf>



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

61

Understanding the Requirements – The SEI's Quality Attribute Workshop

The purpose of the SEI Quality Attribute Workshop (QAW) is to discover, early in the life cycle, the driving quality attribute requirements of a software-intensive system.

QAW Steps

1. QAW Presentation and Introductions
2. Business/Programmatic Presentation
3. Architectural Plan Presentation
4. Identification of Architectural Drivers
5. Scenario Brainstorming
6. Scenario Consolidation
7. Scenario Prioritization
8. Scenario Refinement

Barbacci, et al., *Quality Attribute Workshops (3rd Ed.)*, CMU/SEI-2003-TR-016

<http://www.sei.cmu.edu/library/abstracts/reports/03tr016.cfm>



Software Engineering Institute

CarnegieMellon

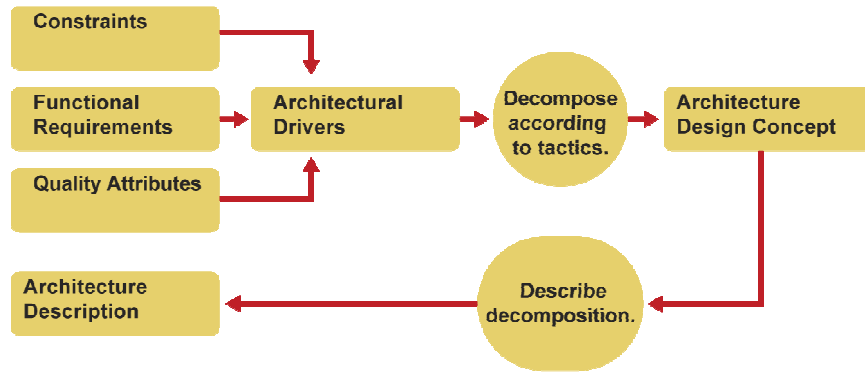
CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

62



An Approach to Architecture Creation

The Attribute-Driven Design (ADD) method is an approach to defining a software architecture by basing the design process on the quality attribute requirements of the system.



Class Exercise 3



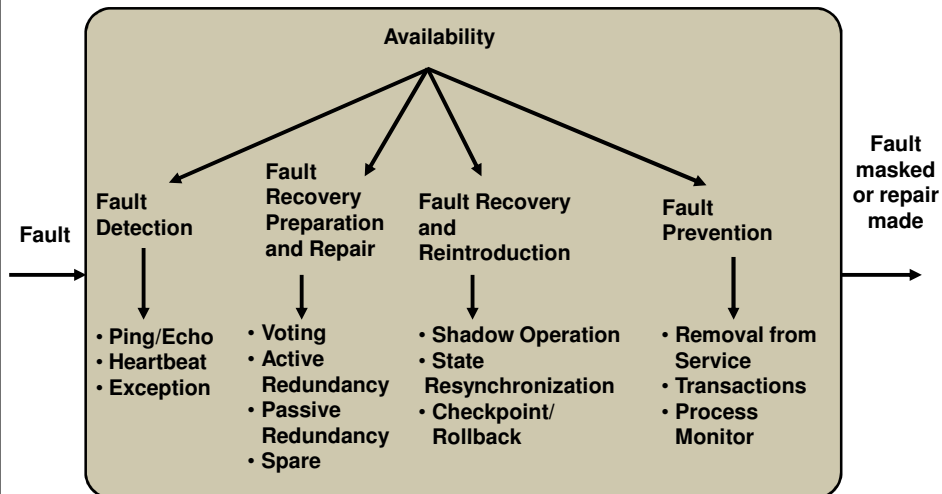
Creating the Architecture

How to do this is beyond the scope of this tutorial.
 Part of the ADD approach is to pick architectural *patterns* and *tactics* that address particular quality attributes.
Patterns represent a packaging of a number of design decisions we refer to as *tactics*.
 Each *tactic* is a design option available to the architect.
 A pattern typically employs several different tactics to promote various quality attributes.
 Example: Tactics to influence *availability* (keep faults from becoming errors) include

- Fault Detection
- Fault Recovery
- Fault Prevention



Summary of Availability Tactics



Other Tactics

There are tactics for

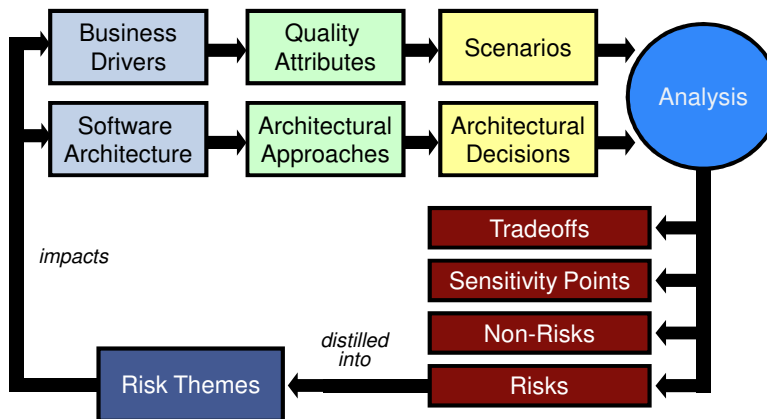
- modifiability
- performance
- security
- testability
- usability

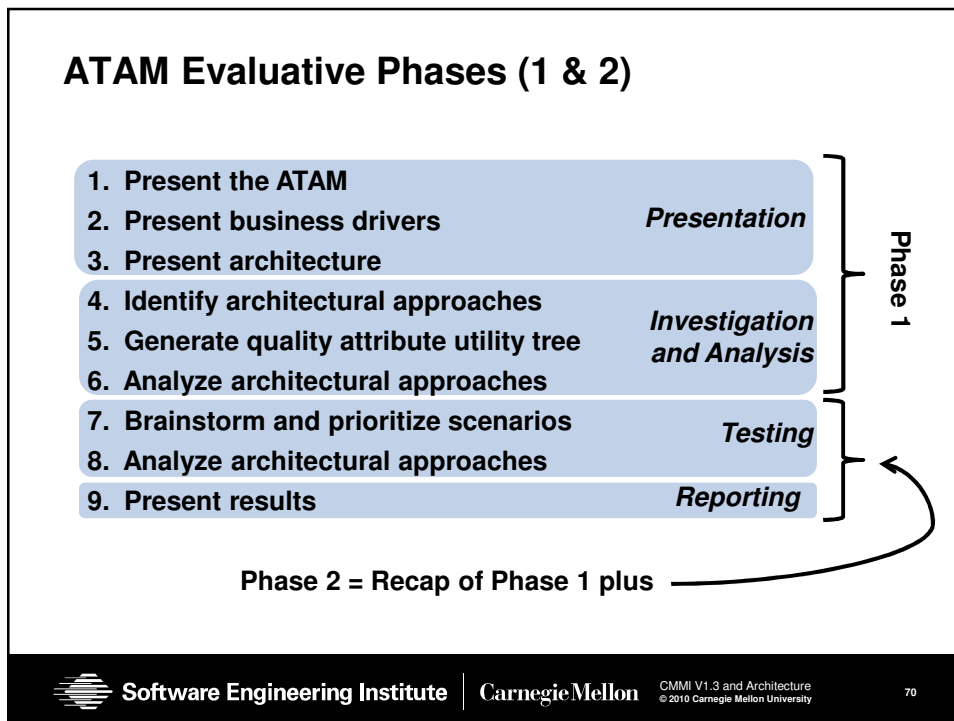
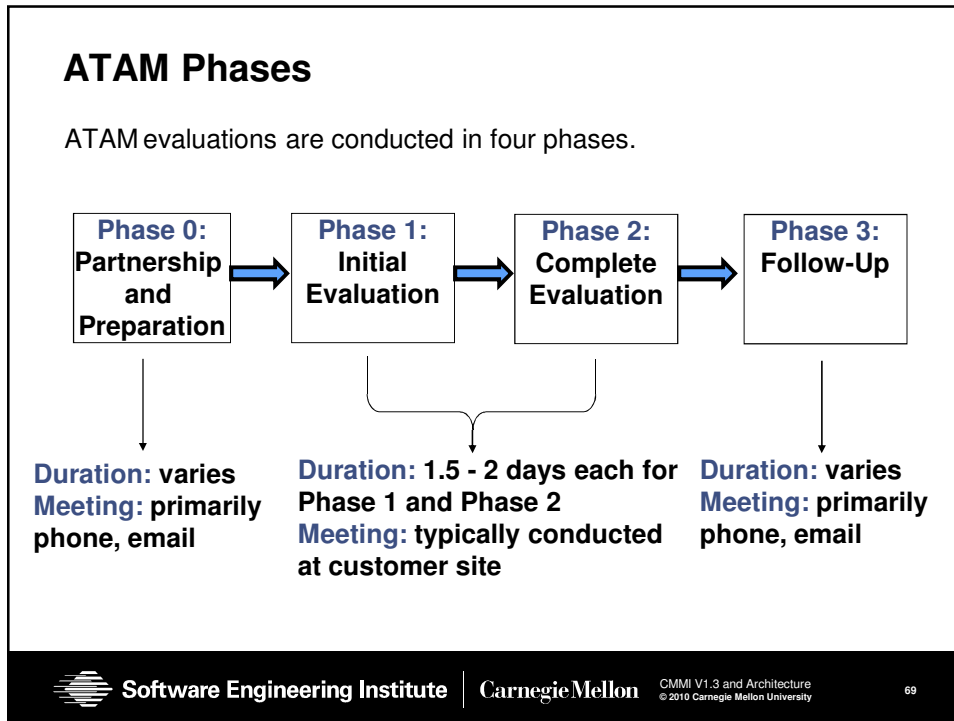
See *Software Architecture in Practice* for a more complete treatment of the subject.



Analyzing the Architecture – SEI’s Architecture Tradeoff Analysis Method® (ATAM®)

The ATAM is an architecture evaluation method that focuses on multiple quality attributes.





Documenting the Software Architecture

Architecture documentation establishes the set of design decisions that must be made along the way to establishing and maintaining the architecture.

An architecture is a multidimensional construct, too involved to be seen all at once.

Recall: systems are composed of many structures.

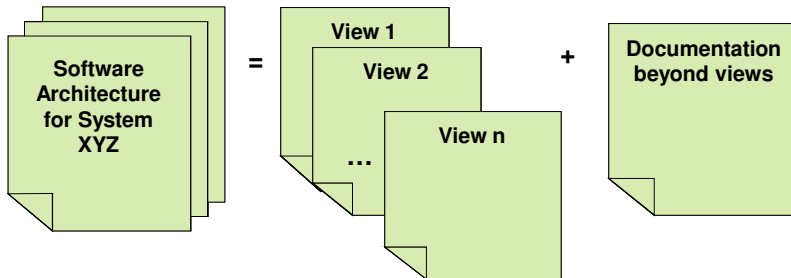
A view is a representation of a structure.

We use views to manage complexity by separating concerns.



View-Based Documentation

Views give us our basic principle of architecture documentation



Documenting an architecture is a matter of documenting the relevant views, and then adding documentation that applies to more than one view.

The choice of views used depends on the nature of the system and the stakeholder needs.



Software Architecture Documentation Needs

Runtime views to show how software will handle:

- hazards, faults, and errors
- fault tolerance/reconfigurations
- performance
- data (e.g., quality, timeliness, ownership, access privileges)
- interface boundaries

Non-runtime views of software (vital to project planning, allocating work assignments, designing for modifiability, reusability, portability, extensibility, etc., facilitating incremental development, and a host of other critical purposes)

Architectural decisions and the rationale/implications/impact of those decisions on key system qualities



So How Well Does This Work? Study: Impact of Army Architecture Evaluations

Twelve Army programs that had conducted ATAM or QAW exercises in a study to elicit the perceived impact the ATAM evaluations and QAWs had on system quality and the practices of the acquisition organization.

Results showed

- 6/12: cost less than or equal to traditional techniques
- 10/12: quality of results greater than or equal to traditional techniques
- 10/12: helped understand and control cost and schedule
- 12/12: increased understanding of system's quality attribute requirements, design decisions, and risks
- 12/12: good mechanism for communication among stakeholders
- 8/12: improved the architecture

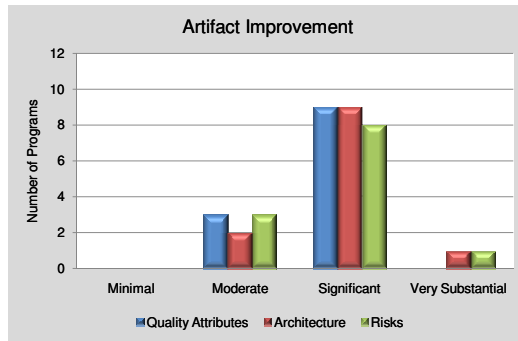


The context of use had a significant impact on the results enjoyed.
Architecture-centric acquisition is key to reaping maximal benefit.



Architecture Practices are Having an Impact 1 of 2

Results of 2008 survey of 12 Army projects that employed ATAM/QAW²



- Most reported *significant* improvement in their architecturally-significant artifacts
- Architecture teams were able to achieve understanding of stakeholder expectations and the implications of architectural decisions on user needs

² Source: Impact of Army Architecture Evaluations, CMU/SEI-2009-SR-007



Software Engineering Institute

CarnegieMellon

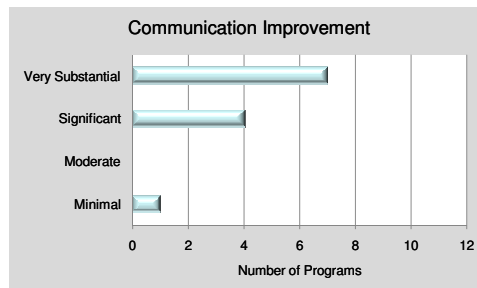
CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

75

Architecture Practices are Having an Impact 2 of 2

Results of 2008 survey of 12 Army projects that employed ATAM/QAW

- Majority reported *very substantial* or *significant* improvement in stakeholder communication
- Stakeholders, collectively, are able to achieve a common understanding of the system under development
 - Increases likelihood that product will address expectations/user needs
 - Improves chances for program success



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

76



Themes From the Army Presentations - 1

“The ATAM architecture evaluations resulted in improved documentation, improved communication, reduced risk in schedule and cost, and a higher quality product to the warfighter.”

“Independent, 3rd party architecture evaluation is quite beneficial for programs that are considered high risk, and/or for which the PM has no visibility into architecture/design.”

“The ATAM is an effective mechanism for getting the stakeholders to work together and identify architectural risks early in the acquisition/development life cycle when they can still be mitigated in a cost effective manner.”

- “It is important that programs (and their supporting contractors) have good risk management procedures so that risks uncovered by an ATAM evaluation are properly tracked and mitigated.”



Themes From the Army Presentations - 2

“QAW should be part of the operational architecture community to ensure quality attributes, and not just functionality, are appropriately addressed.”

- “QAW results were very beneficial to conducting follow-on ATAM evaluations because the QAW scenarios and architectural drivers can carry forward.”
- “QAWs at the system and system of system (SoS) requirements levels are a good thing and should especially be applied on US Joint Forces Command (JFCOM) programs so all stakeholder requirements can be suitably addressed.”

“QAWs and the ATAM are making a very good impact on Army programs, perhaps more than the SEI is aware of. The SEI needs to codify this and send the message to Army management.”

“The importance of having had the backing of Army senior leadership and ASSIP funding is that the beneficiaries— the Army programs—went from “Nay-Sayers” to “Yea-Sayers.””



Implementing and checking conformance

Press on to implementing the system in accordance with the architecture.

Have processes and supporting tools to check for conformance with the architecture.

Unfortunately, a lot of this work today is not automated.



Principles of ACE

1. Regardless of scale, architecture is the **appropriate abstraction** for reasoning about business/mission goal satisfaction.
2. **Quality attributes** have a dominant influence on a system's architecture.
3. Architectural prescriptions must be demonstrably satisfied by the **implementation**.
 - Software architecture must be central to software development activities.
 - These activities must have an explicit focus on quality attributes.
 - These activities must directly involve stakeholders – not just the architecture team.
 - The architecture must be descriptive and prescriptive.



Extending these ideas to Systems and Systems of Systems

The previous discussion was based largely on software engineering practices.

The ideas and techniques have been extended into the realm of systems and systems-of-systems.

Initial results are positive.

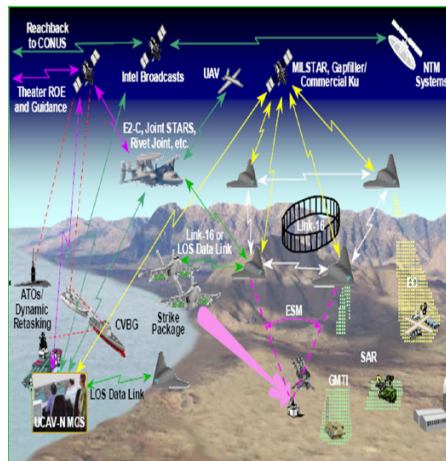


System / SoS Architecture Problems

Severe integration and runtime problems arise due to inconsistencies in how quality attributes are addressed in system and software architectures.

This is further exacerbated in an SoS context where major system and software elements are developed concurrently and oftentimes independently.

A uniform approach for **specifying** quality attribute requirements and **evaluating SoS and system architectures** against such requirements is needed.



The Need for Augmented Mission Threads in DoD SoS Architecture Definition

DoDAF is the SoS architecture framework for the DoD.

- It provides a good set of architectural views for an SoS architecture.
- It inadequately addresses cross-cutting quality attribute considerations.

System use cases focus on a functional slice of the system.

More than DoDAF and system use cases are needed to ensure that the SoS architecture satisfies its end-to-end functional requirements *and* quality attribute needs.

SoS end-to-end mission (operational or user) threads augmented with quality attribute considerations are needed to help develop, and later evaluate, the SoS architecture.



One Approach

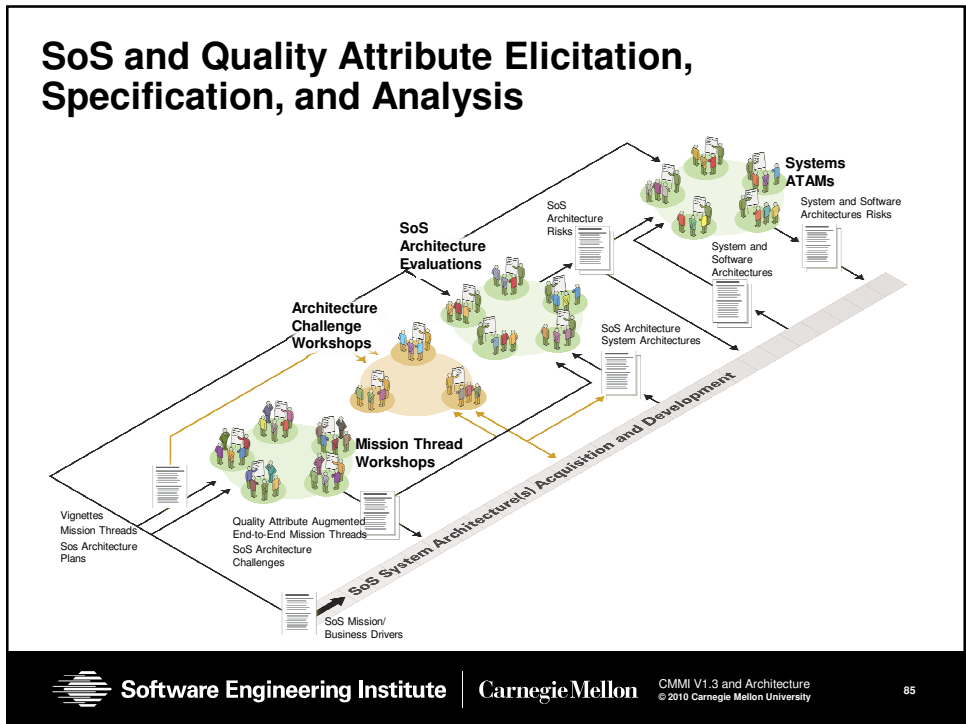
SEI developed and applied a two-pronged approach to address the early identification of quality attribute inconsistencies, ambiguities, and omissions within system and SoS architectures (in Directed and Acknowledged SoS contexts).

1. Perform a "first pass" identification of inconsistencies, ambiguities, and omissions across the constituent systems, at the SoS level, using end-to-end mission threads that are augmented with quality attribute concerns from SoS stakeholders.

The approach involves a series of workshop and evaluations.

- Mission Thread Workshop
 - Architecture Challenge Workshop
 - SoS Architecture Evaluation
2. Constituent systems that are "problematic" are further evaluated using the system and software architecture evaluation method (based on the ATAM), using the augmented mission threads from the Mission Thread Workshops.
 - System and Software ATAM





Architectural Reuse

An architecture represents a significant investment.
 Why use it for only one system?

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



The DoD acquires families of similar systems.

The Real Truth About Reuse

Reuse means using an item more than once.

“The XYZ System is built with 80% reuse.”

A statement like this is vacuous.

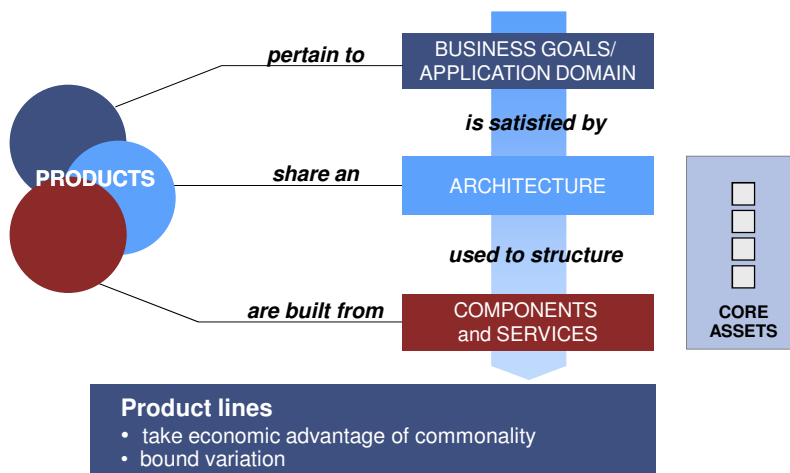
- It is not clear what is being reused.
- It is not clear that the “reuse” has any benefit.

Reusing code or components without an architecture focus and without pre-planning results in

- Short-term perceived win
- Long-term costs and problems
- Failure to meet business goals



Reuse That Pays Off: Software Product Lines



Software Product Lines

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 test data
- people: their knowledge and skills
- processes, methods, and tools
- budgets, schedules, and work plans
- components and services



PRODUCT LINES = STRATEGIC REUSE



Successful Software Product Lines

Improvements in cost, time to market, and productivity that come with successful product lines abound.

- **Cummins** reduced the time it takes to produce software for a diesel engine from one year to one week.
- **Motorola** realized a 400% productivity improvement in a family of one-way pagers.
- **Hewlett-Packard** reduced time to market by a factor of seven and increased productivity by a factor of four in a family of printers.
- The **NRO** built a ground control system with 10% of the expected number of developers and reduced defects by 90%.
- **Nokia** reports producing 25 to 30 different phone models per year by using a product line approach.



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

91

Widespread Application - 1



Feed control and farm management software



Asea Brown Boveri

Gas turbines, train control, semantic graphics framework



Computer printer servers, storage servers, network camera and scanner servers



Bold Stroke Avionics



Dialect

Internet payment gateway infrastructure products



Customized solutions for transportation industries

E-COM Technology Ltd.

Medical imaging workstations

ERICSSON

AXE family of telecommunications switches



Firmware for computer peripherals



Elevator control systems



Software for engines, transmissions and controllers



Lucent Technologies
Bell Labs Innovations

5ESS telecommunications switch

NOKIA

Mobile phones, mobile browsers, telecom products for public, private and cellular networks



RAID controller firmware for disk storage units



Interferometer product line



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

92



Widespread Application - 2

PHILIPS

High-end televisions,
PKI telecommunications switching system,
diagnostic imaging equipment

Rockwell Collins

Commercial flight control system avionics,
Common Army Avionics System (CAAS),
U.S. Army helicopters

symbian

EPOC operating system



Test range facilities

RICOH

Office appliances

SALiON™

TARGET. WIN. DELIVER.™
Revenue acquisition
management systems

TELVENT

Industrial supervisory control
and business process
management systems



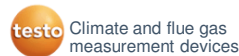
Command and control
simulator for Army fire
support

BOSCH

Automotive gasoline systems

SIEMENS

Software for viewing and quantifying
radiological images



Climate and flue gas
measurement devices



Support software



Pagers product line



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

93

Software Product Lines in the DoD

Organizations having or adopting a software product line approach include

- US Army C-E LCMC: Advanced Multiplex Test System (AMTS)
- Army Training Information Systems Directorate: Army Training Information Architecture (ATIA)
- Overwatch Textron Systems: Overwatch Intelligence Center (OIC) Software Product Line
- OneSAF: OneSAF Product Line Architecture
- Joint Battle Command – Platform product line
- Rockwell Collins: Common Avionics Architecture System (CAAS)
- PEO Simulation, Training & Instrumentation (PEO STRI): Live Training Transformation Components plus Common Training Instrumentation Architecture (LT2/CTIA)
- PEO Simulation, Training & Instrumentation (PEO STRI): SE Core - Synthetic Environment Core (SE Core) is the Army's Common Virtual Environment (CVE)
- US Army Joint Fires Product Line
- Common Driver Training Product Line
- Northrop Grumman Common Link Integration Processing product line
- USMC Live Training Transformation product line



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

94



Presentation Outline

CMMI V1.3 – Context for modern engineering practices changes

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

95

Modern Development Practices in CMMI - 1

For Version 1.3, CMMI provides better guidance in support of architecture-centric practices

- creating the **business case** for the system (partially in RD)
- understanding the **requirements** (RD)
- **creating and/or selecting** the architecture (TS)
- **documenting and communicating** the architecture (RD, TS)
- **analyzing or evaluating** the architecture (RD, TS, VAL, VER)
- **implementing** the system based on the architecture (TS; A/PL notes)
- ensuring that the implementation **conforms** to the architecture (VER)
- **evolving** the architecture so that it **continues to meet business and mission goals** (implicit in the phrase “establish and maintain”)

The above repeats the “Architecture-Centric Activities” slide seen earlier.
(Elaborations indicate where the practice is addressed in CMMI V1.3.)



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

96



Modern Development Practices in CMMI - 2

CMMI V1.3 provides improved terminology to support architecture-centric practices

- Updated the glossary to include new terms (and modified some old terms)
- Updated the informative material (especially ARD and ATM in ACQ; RD, TS, and VER in DEV; and SSD in SVC) to:
 - make use of the new terms
 - bring more emphasis to quality attributes and thus strike a better balance between functional and non-functional requirements
- Replaced selected uses of overloaded terms such as “performance” with an appropriate qualifying phrase.



CMMI Support for: creating the business case for the system

CMMI V1.3 touches on the “why” for the business in many places, including OPF, OPM, OPP, QPM, RD. Focusing here only on RD:

RD SP 1.1 Elicit Needs

Elicit stakeholder needs, expectations, constraints, and interfaces for all phases of the product lifecycle.

RD SP 1.2 Transform Stakeholder Needs into Customer Requirements

Transform stakeholder needs, expectations, constraints, and interfaces into prioritized customer requirements.

[snip] Relevant stakeholders representing all phases of the product’s lifecycle should include *business* as well as technical functions. In this way, *concepts* for all product related lifecycle processes are considered concurrently with the *concepts* for the products. *Customer requirements result from informed decisions on the business as well as technical effects of their requirements. [Emphasis added]*



CMMI Support for: understanding requirements - 1

CMMI support for understanding requirements is mostly found in the RD PA (and secondarily in a few other places, especially VAL).

- SG 1 Develop Customer Requirements
 - SP 1.1 Elicit Needs
 - SP 1.2 ~~Develop the~~ Transform **Stakeholder Needs into** Customer Requirements
- SG 2 Develop Product Requirements
 - SP 2.1 Establish Product and Product Component Requirements
 - SP 2.2 Allocate Product Component Requirements
 - SP 2.3 Identify Interface Requirements
- SG 3 Analyze and Validate Requirements
 - SP 3.1 Establish Operational Concepts and Scenarios
 - SP 3.2 Establish a Definition of Required Functionality **and Quality Attributes**
 - SP 3.3 Analyze Requirements
 - SP 3.4 Analyze Requirements to Achieve Balance
 - SP 3.5 Validate Requirements



CMMI Support for: understanding requirements - 2

Specific Goal and Practice Changes (most of them in RD)

Changed RD SG 3 so it no longer appears to focus on functionality.

SG 3 Analyze and Validate Requirements
The requirements are analyzed and validated, ~~and a definition of required functionality is developed.~~

Changed SP 1.2 to make stakeholder/customer priorities more explicit.

SP 1.2 ~~Transform Stakeholder Needs into~~ Develop the Customer Requirements
*Transform stakeholder needs, expectations, constraints, and interfaces into **prioritized** customer requirements.*

Changed RD SP 3.2 to add emphasis to non-functional requirements.

SP 3.2 Establish a Definition of Required Functionality **and Quality Attributes**
*Establish and maintain a definition of required functionality **and quality attributes.***



CMMI Support for: understanding requirements - 3

RD (especially) and other PAs: Informative Material Changes

Added and revised the informative material throughout these PAs to appropriately mention the following engineering concepts:

- quality attributes (i.e., non-functional requirements or “ilities”)
- product lines, system of systems
- architecture-centric practices
- allocation of product capabilities to release increments
- technology maturation (and obsolescence)

These concepts are mentioned in example boxes, in examples provided in the notes, and in discussion that mentions various approaches that can be used.

When functional requirements are discussed, mention of quality attributes is added to balance the view of requirements.



CMMI Support for: understanding requirements - 4

In RD SP 1.1 Elicit Needs

- Added the following examples of techniques to elicit needs:
 - [snip] Questionnaires, interviews, and scenarios (operational ~~scenarios~~, sustainment, and development) obtained from end users
 - Operational, sustainment, and development walkthroughs and end-user task analysis
 - Quality attribute elicitation workshops with stakeholders

- Added Example Work Product:

Results of requirements elicitation activities

In RD SP 1.2 Transform Stakeholder Needs into Customer Requirements

- Added the following new subpractice:
 - 2. Establish and maintain a prioritization of customer functional and quality attribute requirements.**



CMMI Support for: understanding requirements - 5

In RD SP 2.1 Establish Product and Product Component Requirements

- Added a note to Subpractice 2 (deriving requirements that result from design decisions):

Architectural decisions, such as selection of architecture patterns, introduce additional derived requirements for product components. For example, the Layers Pattern will constrain dependencies between certain product components.

- Added the following new subpractice:

3. Develop architectural requirements capturing critical quality attributes and quality attribute measures necessary for establishing the product architecture and design.



CMMI Support for: understanding requirements - 6

In RD SP 2.2 Allocate Product Component Requirements

- Added a note:

The product architecture provides the basis for allocating product requirements to product components. [snip] In cases where a higher level requirement specifies ~~performance a quality attribute~~ that will be the responsibility of more than one product component, the ~~performance~~ must quality attribute can sometimes be partitioned for unique allocation to each product component as a derived requirement, however, other times the shared requirement should instead be allocated directly to the architecture. [snip]

- Revised first four subpractices:

1. Allocate requirements to functions.
2. Allocate requirements to product components and the architecture.
3. Allocate design constraints to product components and the architecture.
4. Allocate requirements to delivery increments.



CMMI Support for: understanding requirements - 7

In RD SG 3 Analyze and Validate Requirements

- Added a note:

Architecturally significant quality attributes are identified based on mission and business drivers.

In RD SP 3.1 Establish Operational Concepts and Scenarios

- Changed Subpractice 1 to read:

1. Develop operational concepts and scenarios that include ~~functionality, performance~~ operations, installation, development, maintenance, support, and disposal as appropriate.

Identify and develop scenarios, consistent with the level of detail in the stakeholder needs, expectations, and constraints in which the proposed product or product component is expected to operate.

Augment scenarios with quality attribute considerations for the functions (or other logical entities) described in the scenario.



CMMI Support for: understanding requirements - 8

In RD SP 3.2 Establish a Definition of Required Functionality and Quality Attributes

- Added a note (split here for readability):

Such approaches have evolved in recent years through the introduction of architecture description languages, methods, and tools to more fully address and characterize the quality attributes, allowing a richer (e.g., multi-dimensional) specification of constraints on how the defined functionality will be realized in the product, and facilitating additional analyses of the requirements and technical solutions.

Some quality attributes will emerge as architecturally significant and thus drive the development of the product architecture. These quality attributes often reflect cross-cutting concerns that may not be allocatable to lower level elements of a solution. A clear understanding of the quality attributes and their importance based on mission or business needs is an essential input to the design process.

- Revised the subpractices in line with the above note.



CMMI Support for: understanding requirements - 9

In RD SP 3.4 Analyze Requirements to Achieve Balance

- Added the following new subpractice:

4. Assess the impact of the architecturally significant quality attribute requirements on the product and product development costs and risks.

When the impact of requirements on costs and risks seems to outweigh the perceived benefit, relevant stakeholders should be consulted to determine what changes may be needed.



CMMI Support for: understanding requirements - 10

In TS Introductory Notes

- Added **technology maturation and obsolescence** as additional drivers of requirements changes in maintenance and sustainment projects.

In VAL Introductory Notes

Reinforced when validation occurs in the product lifecycle.

“[snip] validation is performed early (**concept/exploration phases**) and incrementally throughout the product lifecycle (**including transition to operations and sustainment**).”

In VAL SP 1.1 Select Products for Validation

Added additional examples of products and product components that can be validated:

access protocols and **data interchange reporting formats**

Added example of validation method:

incremental delivery of working and potentially acceptable product



CMMI Support for: the architecture - 1

CMMI support for:

- creating/selecting
- documenting/communicating
- analyzing/evaluating

the architecture

Is mostly found in the first two goals of TS:

- SG 1 Select Product Component Solutions
 - SP 1.1 Develop Alternative Solutions and Selection Criteria
 - SP 1.2 Select Product Component Solutions
- SG 2 Develop the Design
 - SP 2.1 Design the Product or Product Component
 - SP 2.2 Establish a Technical Data Package
 - SP 2.3 Design Interfaces Using Criteria
 - SP 2.4 Perform Make, Buy, or Reuse Analyses



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

109

CMMI Support for: the architecture - 2

TS Informative Material Changes

“Quality attribute models, simulations, prototypes or pilots can be used to provide additional information about the properties of the potential design solutions to aid in the selection of solutions. Simulations can be particularly useful for projects developing systems-of-systems.” *[TS Intro Notes]*

“Architectural ~~features~~ choices and patterns that ~~provide a foundation for product improvement and evolution~~ support achievement of quality attribute requirements are considered.

[snip] COTS alternatives [snip] can require modifications to aspects such as interfaces or a customization of some of the features to ~~better achieve product~~ correct a mismatch with functional or quality attribute requirements, or with architectural designs. *[TS SG 1 note]*



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

110



CMMI Support for: the architecture - 3

TS Informative Material Changes (continued)

In TS SP 1.1 Develop Alternative Solutions and Selection Criteria

- Added an additional consideration for selection criteria:
Achievement of key quality attribute requirements, such as product timeliness, safety, reliability, and maintainability
- Added new subpractice 4.
4. Identify re-usable solution components or applicable architecture patterns.

In TS SP 2.1 Design the Product or Product Component

- Added additional examples of architecture definition tasks.
 - Selecting architectural patterns that support the functional and quality attribute requirements, and instantiating or composing those patterns to create the product architecture
 - Formally defining component behavior and interaction using an architecture description language



CMMI Support for: the architecture - 4

TS Informative Material Changes (continued)

In TS SP 2.2 Establish a Technical Data Package

- Added new subpractice 2.
2. Determine the views to be used to document the architecture.
Views are selected to document the structures inherent in the product and to address particular stakeholder concerns.

In TS SP 2.3 Design Interfaces Using Criteria

- Added to what “interface designs include:”
 - stimulus and data characteristics for software, including sequencing constraints or protocols
 - resources consumed processing a particular stimulus
 - Exception or error handling behavior for stimuli that are erroneous or out of specified limits.



CMMI Support for: implementing the system based on the architecture - 1

CMMI V1.3 support for implementing the system is mostly found in the third goal of the TS PA.

SG 3 Implement the Product Design

SP 3.1 Implement the Design

SP 3.2 Develop Product Support Documentation

TS Informative Material Changes

In TS SP 3.1 Implement the Design

- In Subpractice 1, added **aspect oriented programming** as a software coding methods example.



CMMI Support for: implementing the system based on the architecture - 2

Other Informative Material Changes

Special notes for Agile and for Product Lines have been inserted in the Intro Notes of various PAs in V1.3.

Changes Supporting Use of Agile Methods

Because CMMI practices are written for use in a broad variety of contexts, business situations, and application domains, it is not possible (even if it were appropriate) to advocate any specific implementation approach.

However, **Agile methods and approaches** are now in wider use, and so for V1.3, it seemed appropriate to acknowledge this, identify how Agile approaches can address CMMI practices and conversely, identify the value that CMMI can bring to Agile implementations.

The next set of slides describe how CMMI V1.3 addresses Agile methods.



Addressing Agile - 1

The Problem

Developers that use Agile methods sometimes resist using CMMI because they can't see how CMMI practices can complement or improve the effectiveness of Agile methods.

Overview of Solution

Added guidance to the appropriate PAs to do the following:

- Help users interpret the practices in a context where Agile methods are used
- Reinforce the applicability of the practices in an Agile environment
- Send the message that CMMI is a robust best practice framework meant to be used in Agile environments as well as other development environments



Addressing Agile - 2

Solution

Added a new section to DEV Chapter 5 entitled "Interpreting CMMI When Using Agile Approaches"

- This section describes how CMMI practices can apply in a variety of development environments. It also describes the interpretive guidance that has been added to selected PAs for use in Agile environments.

Added interpretive guidance to the following PAs:

- In DEV: CM, REQM, PP, RD, TS, PI, VER, PPQA, and RSKM
- In ACQ: AM, ATM, PMC, and PP
- In SVC: SSD

Added in DEV and SVC (SSD only) Agile-related examples as bullets in example boxes (informative material).



Addressing Agile - 3

A note added in the RD Intro Notes:

In Agile environments, requirements are communicated and tracked through mechanisms such as **product backlogs, story cards, and screen mock-ups**. [snip] Traceability and consistency across requirements and work products is addressed through the mechanisms already mentioned as well as during start-of-iteration or end-of-iteration activities such as “**retrospectives**” and “**demo days**.” *[Emphasis added]*

A note added in the TS Intro Notes:

In Agile environments, the focus is on early solution exploration. **By making the selection and tradeoff decisions more explicit, the Technical Solution process area helps** improve the quality of those decisions, both individually and over time. [snip] **When someone other than the team will be working on the product in the future**, release information, maintenance logs, and other data are typically included with the installed product. **To support future product updates**, rationale (for trade-offs, interfaces, and purchased parts) is captured **so that why the product exists can be better understood**. [snip] *[Emphasis added]*



Addressing Agile - 4

For more information about using Agile in development and acquisition, and the relationship to CMMI, see:

- Glazer, Hillel; Dalton, Jeff; Anderson, David; Konrad, Mike; & Shrum, Sandy. *CMMI or Agile: Why Not Embrace Both!* (CMU/SEI-2008-TN-003). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, November 2008.
<http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm>
- Lapham, Mary Ann; Williams, Ray C.; Hammons, Charles; Burton, Daniel; and Schenker, Fred. *Considerations for Using Agile in DoD Acquisition* (CMU/SEI-2010-TR-022). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon® University, April 2010.
<http://www.sei.cmu.edu/library/abstracts/reports/10tn002.cfm>
- McMahon, Paul E., “Integrating CMMI into Agile Development: Case Studies and Proven Techniques for Faster Performance Improvement.” Addison-Wesley, 2011.



CMMI Support for: implementing the system based on the architecture - 3

Likewise, notes have been added to the Intro Notes of selected PAs to explain how the PA can be effectively applied in a **product line** environment.



Addressing Product Lines

An example of a note added in the RD Intro Notes:

For product lines, engineering processes (including requirements development) may be **applied to at least two levels in the organization**. At an organizational or product line level, a “commonality and variation analysis” is performed to help elicit, analyze, and establish **core assets** for use by projects within the product line. At the project level, these core assets are then used as per the **product line production plan** as part of the project’s engineering activities. *[Emphasis added]*

An example of a note added in the TS Intro Notes:

For product lines, these practices apply to both **core asset development** (i.e., building for reuse) and **product development** (i.e., building with reuse). Core asset development additionally requires **product line variation management** (the selection and implementation of product line variation mechanisms) and **product line production planning** (the development of processes and other work products that define how products will be built to make best use of these core assets). *[Emphasis added]*



CMMI Support for: ensuring implementation conforms to the architecture - 1

CMMI support for ensuring the implementation conforms to the architecture is mostly found in the VER PA. (And also in notes and subpractices of PI SP 3.3 and TS SP 3.1 and 3.2.)

- SG 1 Prepare for Verification
 - SP 1.1 Select Work Products for Verification
 - SP 1.2 Establish the Verification Environment
 - SP 1.3 Establish Verification Procedures and Criteria
- SG 2 Perform Peer Reviews
 - SP 2.1 Prepare for Peer Reviews
 - SP 2.2 Conduct Peer Reviews
 - SP 2.3 Analyze Peer Review Data
- SG 3 Verify Selected Work Products
 - SP 3.1 Perform Verification
 - SP 3.2 Analyze Verification Results



CMMI Support for: ensuring implementation conforms to the architecture - 2

In VER SG 1 Prepare for Verification

- Changed a note to read:

Methods of verification include, but are not limited to, inspections, peer reviews, audits, walkthroughs, analyses, architecture evaluations, simulations, testing, and demonstrations.

In VER SP 1.1 Select Work Products for Verification

- Added additional examples of verification methods:

software architecture conformance evaluation and continuous integration (i.e., Agile approach).

In VER SP 1.3 Establish Verification Procedures and Criteria

- Added new example of sources of verification criteria:

customers reviewing work products collaboratively with developers



CMMI Support for: ensuring implementation conforms to the architecture - 3

In VER SP 2.1 Prepare for Peer Reviews

- In Subpractice 1, added additional example of types of peer review:
architecture implementation conformance evaluation

In VER SP 2.3 Analyze Peer Review Data

- In Subpractice 4, added additional examples of peer review data that can be analyzed:
user stories or case studies associated with a defect and the end-users and customers who are associated with defect



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

123

CMMI Support for: evolving the architecture so that it continues to meet business and mission goals - 1

The need for evolution arises from both inside and outside:

“As the organization improves its process performance or as business strategies change, new business objectives are identified and associated quality and process performance objectives are derived.” [OPM SG 1 Notes]

These objectives then drive the activities we read about in the project management and engineering PAs such as RD.

The phrase “establish and maintain” appears in the CMMI practices. It implies that key artifacts may need to change to remain useful (see next slide). If higher-level objectives change, the artifact may need to too.

As an example from RD:

“The modification of requirements due to approved requirement changes is covered by the “maintain” aspect of this specific practice; [snip].” [SP 2.1 note]



Software Engineering Institute

CarnegieMellon

CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

124



CMMI Support for: evolving the architecture so that it continues to meet business and mission goals - 2

The definition for “establish and maintain” was changed in V1.3 to support the evolution described on the previous slide.

Establish and maintain

DEFINITION

Create, document, use, and revise . . . as necessary to ensure it remains they remain useful.

The phrase “establish and maintain” ~~means more than a combination of its component terms;~~ . . . plays a special role in communicating a deeper principle in CMMI: work products that have a central or key role in work group, project, and organizational performance should be given attention to ensure they are used and useful in that role.

This phrase has particular significance in CMMI because it often appears in goal and practice statements . . . and should be taken as shorthand for applying the principle to whatever work product is the object of the phrase.



Changes in CMMI Terminology - 1

Allocated requirement

Improved the definition and provided additional examples of what things requirements can be allocated to.

The improvements to the definition make the substance of the solution space and allocation of requirements to it more explicit, allowing for superior architectures and more insightful analyses (including verification) of requirements and technical solutions.

DEFINITION

Requirement that ~~leverages~~ results from levying all or part of ~~the performance and functionality of~~ a higher level requirement on a lower level architectural element or design component.

More generally, requirements can be allocated to other logical or physical components including people, consumables, delivery increments, or the architecture as a whole, depending on what best enables the product or service to achieve the requirements.



Changes in CMMI Terminology - 2

Architecture

This term is included in the Glossary for the first time. (V1.2 used the phrase “product architecture” throughout but never defined it.)

This term and its use throughout the rest of the model is intended to encourage use of proven, architecture-centric practices and the recognition of “architecture” as a principal engineering artifact.

DEFINITION

The set of structures needed to reason about a product. These structures are comprised of elements, relations among them, and properties of both.

In a service context, the architecture is often applied to the service system.

Note that functionality is only one aspect of the product. Quality attributes, such as responsiveness, reliability, and security, are also important to reason about. Structures provide the means for highlighting different portions of the architecture. (See also “functional architecture.”)



Changes in CMMI Terminology - 3

Definition of required functionality and quality attributes

The “definition of required functionality” term has been removed from CMMI because of the implicit suggestion that functionality be addressed first or has highest priority. The term has been replaced with one that is intended to help ensure a sufficiently balanced focus (functional *and* non-functional) in requirements analysis.

DEFINITION

A characterization of required functionality and quality attributes obtained through “chunking,” organizing, annotating, structuring, or formalizing the requirements (functional and non-functional) to facilitate further refinement and reasoning about the requirements as well as (possibly, initial) solution exploration, definition, and evaluation.

As technical solution processes progress, this characterization can be further evolved into a description of the architecture versus simply helping scope and guide its development, depending on the engineering processes used; requirements specification and architectural languages used; and the tools and the environment used [snip].



Changes in CMMI Terminology - 4

“Functional analysis” and “functional architecture”

These terms are now “cul de sacs” in the model.

The only place these terms now appear in CMMI-DEV V1.3 outside the Glossary is in the first note of RD SP 3.2 and as an example work product.

The note contrasts the approaches implied by these terms with “modern engineering approaches” that encourage a more balanced treatment of requirements, functional and non-functional.



Changes in CMMI Terminology - 5

Product line

DEFINITION

A group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission- and that are developed from a common set of core assets in a prescribed way.

The development or acquisition of products for the product line is based on exploiting commonality and bounding variation (i.e., restricting unnecessary product variation) across the group of products. The managed set of core assets (e.g., requirements, architectures, components, tools, testing artifacts, operating procedures, software) includes prescriptive guidance for their use in product development. Product line operations involve interlocking execution of the broad activities of core asset development, product development, and management.

Many people use “product line” just to mean the set of products produced by a particular business unit, whether they are built with shared assets or not. We call that collection a “portfolio,” and reserve “product line” to have the technical meaning given here.



Changes in CMMI Terminology - 6

Quality attribute

This term is now included in the Glossary for the first time. The term is intended to supplant others – especially those focusing on only a few dimensions (e.g., “performance”) – to encourage a broader view of non-functional requirements. The term was refined through much effort, as neither ISO 25030 (SQuaRE) nor the original SEI definitions were quite satisfactory.

DEFINITION

A property of a product or service by which its quality will be judged by relevant stakeholders. Quality attributes are characterizable by some appropriate measure.

Quality attributes are non-functional, such as timeliness, throughput, responsiveness, security, modifiability, reliability, and usability. They have a significant influence on the architecture.



Changes in CMMI Terminology - 7

Performance (not a term appearing by itself in Glossary)

One of our purposes for V1.3 was to achieve greater clarity in the engineering practices of CMMI. This purpose is aided when the term “performance,” which has many meanings, is used unambiguously and correctly throughout. Thus, uses of the term “performance” were reviewed for clarity, and where appropriate, qualified, e.g.:

- supplier’s performance
- project performance
- product performance
- technical performance
- organization’s performance
- cost, schedule, performance
- performed process (CL1)
- process performance
- period of performance
- service delivery performance
- project progress and performance
- fit, form, function, performance



Related Changes

Product Integration

We revised PI SP 1.1 and the terminology used from an emphasis on “integration sequence” to an emphasis on “**integration strategy**” to reflect the complexity of product integration.

The product integration strategy describes the approach for receiving, assembling, and evaluating the product components that comprise the product.

SP 1.1 Establish an ~~Determine~~ Integration Strategy Sequence
 Establish and maintain a ~~Determine the product component~~
 integration strategy ~~sequence~~.

Related changes were made elsewhere in the PI PA.



Presentation Outline

CMMI V1.3 – Context for modern engineering practices changes

Introduction to Architecture

Essential Architecture Practices

Where Are the Architecture-Centric Practices in CMMI V1.3?

Summary

Questions and Answers



Summary & Conclusions

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

Early identification of architectural risks saves money and time.

There are proven practices to help ensure that suppliers and acquirers can develop and acquire systems that have appropriate architectures.

CMMI V1.3 has a new emphasis on architecture.

The efficacy of the architecture has a direct impact on program or mission success, and customer satisfaction.



References - 1

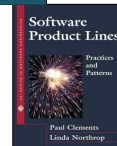
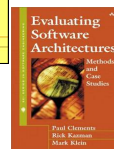
Software Architecture in Practice, Second Edition
Bass, L.; Clements, P.; & Kazman, R. Reading, MA:
Addison-Wesley, 2003.

Evaluating Software Architectures: Methods and Case Studies

Clements, P.; Kazman, R.; & Klein, M. Reading, MA:
Addison- Wesley, 2002.

Documenting Software Architectures: Views and Beyond
Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.;
Little, R.; Nord, R.; & Stafford, J. Reading, MA:
Addison-Wesley, 2002.

Software Product Lines: Practices and Patterns
Clements, P.; Northrop, L. Reading, MA: Addison-Wesley,
2001.



References - 2

You can find a moderated list of references on the “Software Architecture Essential Bookshelf”

<http://www.sei.cmu.edu/architecture/start/publications/bookshelf.cfm>

Grady Booch: Handbook of Software Architecture (currently only an on-line reference):

<http://www.handbookofsoftwarearchitecture.com/index.jsp?page=Main>

CMMI for Development, Version 1.3

<http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm>

(also available as a book from the SEI Series on Software Engineering:)

Chrissis, Mary Beth; Konrad, Mike; & Shrum, Sandy. CMMI: Guidelines for Process Integration and Product Improvement, 3rd Edition. Boston: Addison-Wesley, 2011.



The SEI Software Architecture Curriculum

Six Courses	Three Certificate Programs		
	Software Architecture Professional	ATAM Evaluator	ATAM Leader
Software Architecture Principles and Practices*	✓	✓	✓
Documenting Software Architectures	✓		✓
Software Architecture Design and Analysis	✓		✓
Software Product Lines	✓		✓
ATAM Evaluator Training		✓	✓
ATAM Leader Training			✓
ATAM Observation			✓

✓ : required to receive certificate
* : available through e-learning



Contact Information

Larry Jones

Research, Technology, and Systems
Solutions Program
Telephone: 719-481-8672
Email: lgj@sei.cmu.edu

Mike Konrad

SEPM
Telephone: 412-268-5813
Email: mdk@sei.cmu.edu

U.S. Mail:

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

World Wide Web:

<http://www.sei.cmu.edu/productlines>
SEI Fax: 412-268-5758



Questions



NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

**Software Engineering Institute****Carnegie Mellon**CMMI V1.3 and Architecture
© 2010 Carnegie Mellon University

141

