Architecture: Why Your CMMI V1.3 Implementation is Incomplete Without It!

11th CMMI Technology Conference and User's Group

Denver, CO
November 14, 2011

Larry Jones
Mike Konrad

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

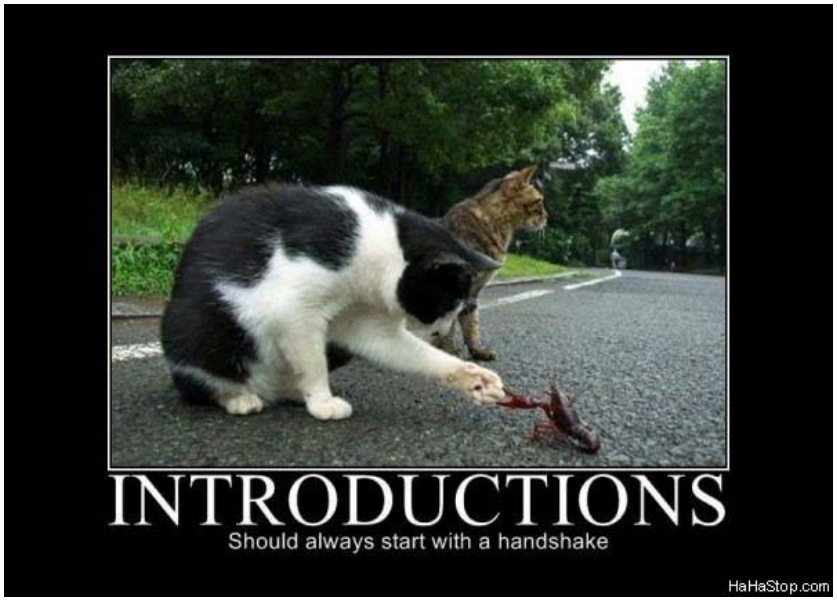
Introductions

Instructor Introduction

Participant Introductions

(mechanics depends on group 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



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

What Have We Learned Over the Past Year? Something was Missing.

A lot of people overlooked the new architecture emphasis.

- Changes in the model that explicitly use the term "architecture" are in the informative material and glossary.
- Those who understand CMMI may overlook the *new expectation* that quality attributes are an equal partner to functionality.
  - Architecture practices thus need to be considered.

Hence we decided to be much less subtle about architecture in the title and we have allowed more time to discuss the "whereabouts" of architecture in the V1.3 model.

Good architecture practices are essential for success with CMMI V1.3!

Conventions & Caveats for the Tutorial

The coverage of architecture-centric practices in CMMI V1.3 is broad, focused on “products” and “solutions” – not just on software.

- But much of the tutorial material came from SEI assets whose focus was software-intensive systems. Please bear this in mind. We believe the principles apply beyond simply software.

Our focus in the tutorial will be on CMMI for Development because that is where the architecture-centric practices are most deeply covered but similar changes were also made to the other two CMMI models.

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.

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 V1.3 Upgrade Training.

Familiarity with product, system, or software design is useful, but not required. (Such familiarity will be of benefit to you in the hands-on exercises, which are intended 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

The Problem: CMMI V1.2 Did Not Adequately Cover Modern Engineering Approaches

Much of the engineering content of CMMI-DEV V1.2 was ten years old.

As DEV was a starting point for the other two constellations, no V1.2 model adequately addressed modern engineering approaches such as architecture-centric engineering.

For example, both RD SG 3 and RD SP 3.2 emphasized functionality and not non-functional requirements.

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

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

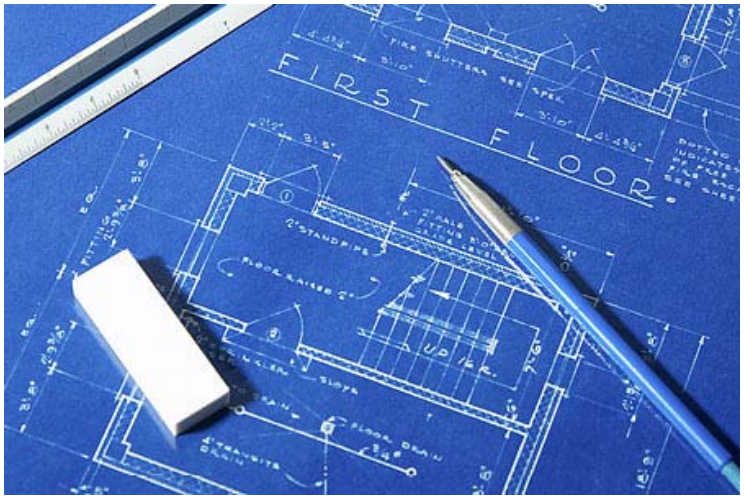
The Solution: Modernize the Engineering Content in CMMI V1.3

The slides that follow portray where the development community 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 approaches.

What Is an Architecture?

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

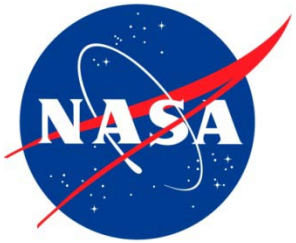


Architecture is Important

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

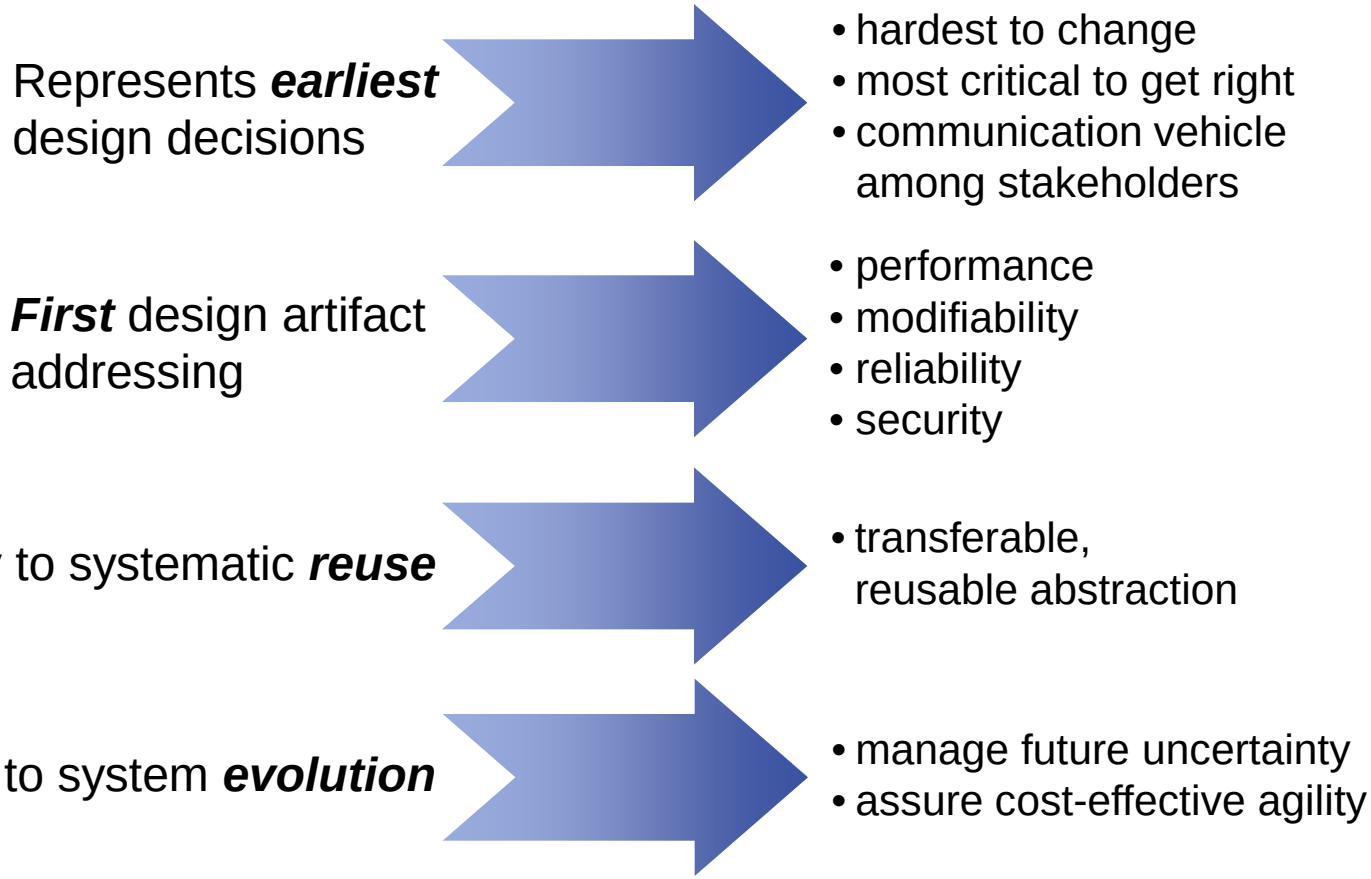
In recent studies by OSD, the National Research Council, NASA, and the NDIA, architectural issues are identified as a systemic cause of software problems in DoD systems.

NDIA
National Defense Industrial Association



NRC The National Research Council

Why Is Architecture Important?



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

Architecture is About Structure and Decisions

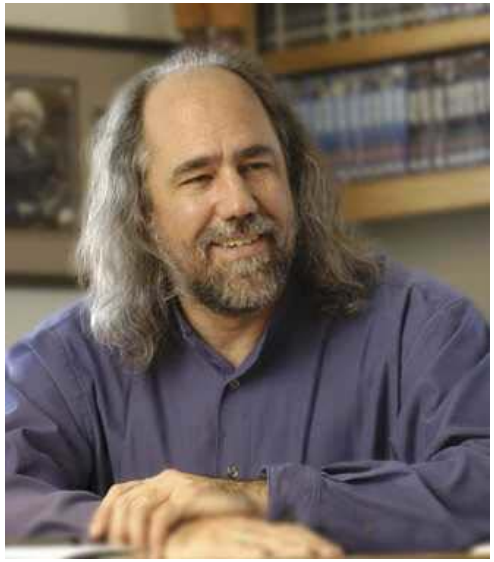
Structures result from decisions

- 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.

“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

Class Exercise 1

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

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.

Formal Definition of Software Architecture

"*The **software architecture** of a computing system is the set of **structures** needed to reason about the system, which comprise **software components, relations** among them and **properties** of both.*"

Clements et al, *Documenting Software Architectures, Second Edition.* Addison-Wesley, 2011

Formal Definition of 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] Rechtin, 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.

Implications

Architecture is an abstraction of a system.

Architecture defines the properties of elements.

Systems can and do have many structures.

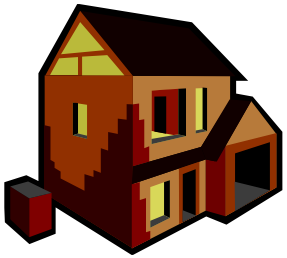
Every software-intensive system *has* an architecture.

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!

Structures and Views - 1

One house, many views



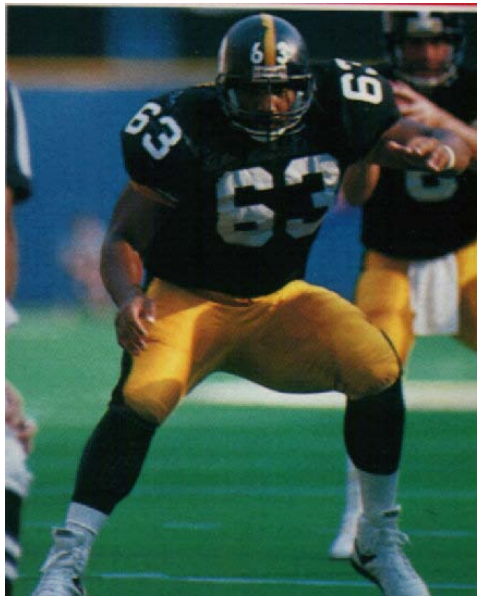
- Carpentry view
- Plumbing view
- Electrical view
- Ductwork view

No single view accurately represents the house.

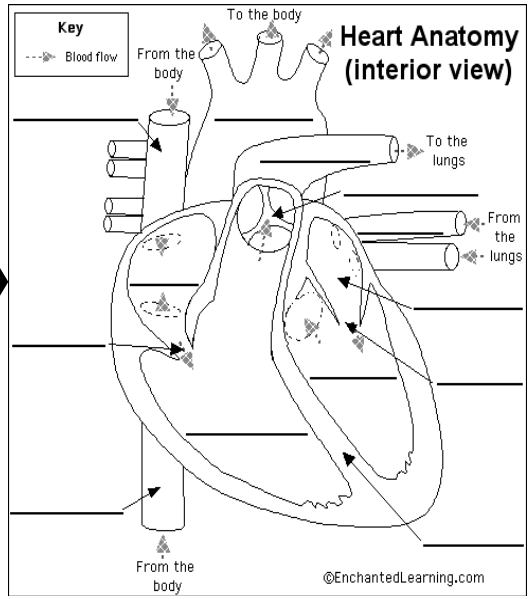
No single view can be used to build the house.

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 and systems.

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?

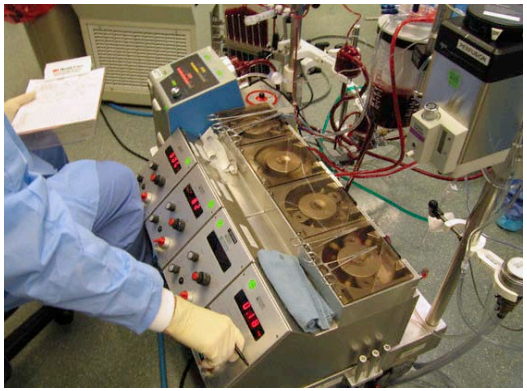
Summary

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.

System Development

Functional Requirements

If function were all that mattered, any monolithic implementation 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

Quality Attribute Drivers → **Software & System Architectures** → **Software & System**

has these qualities

The Non-functional Requirements

Specifying Quality Attributes

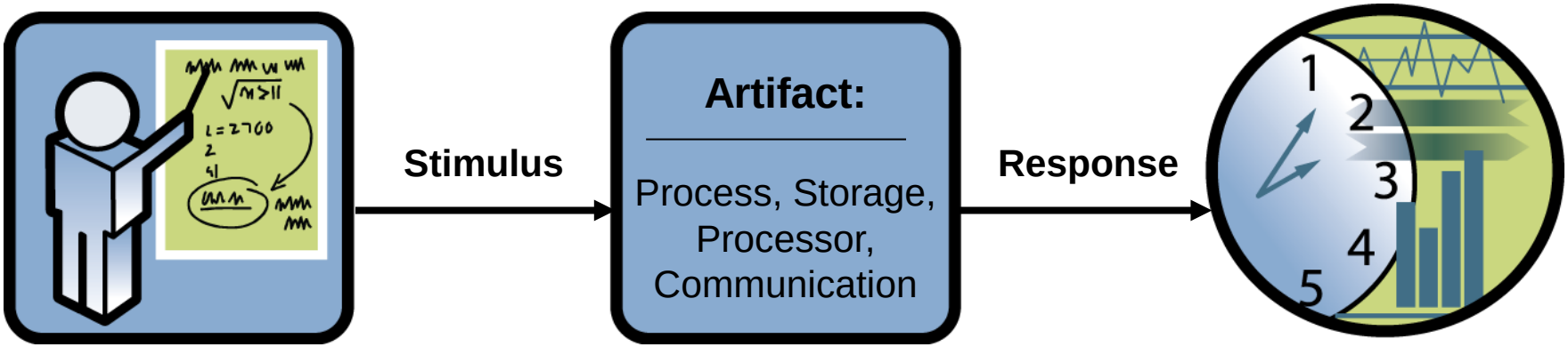
Quality attributes are rarely captured *effectively* in requirements specifications; they are often vaguely understood and weakly articulated.

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.

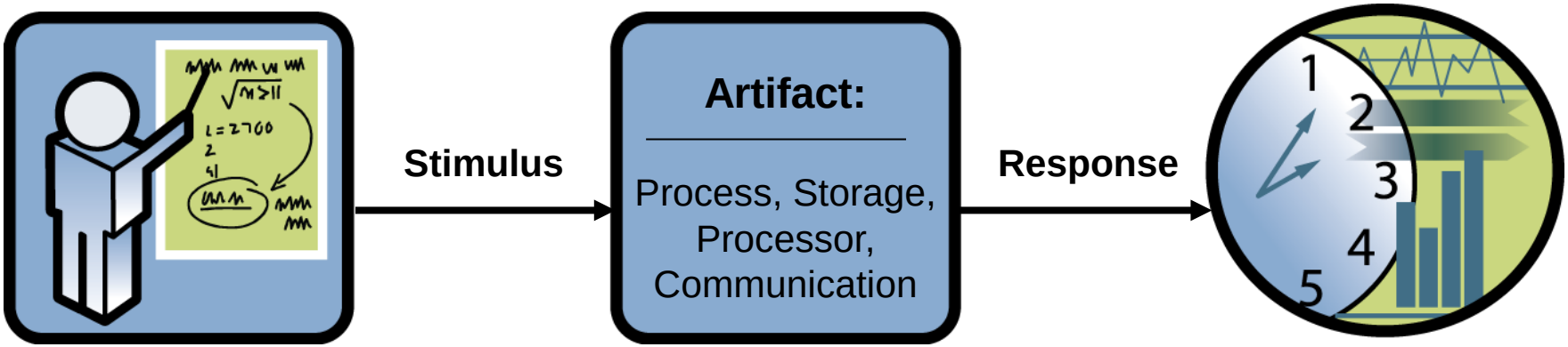
A quality attribute scenario is a short description of an interaction between a stakeholder and a system and the response from the system.

Parts of a Quality Attribute Scenario



Example Quality Attribute Scenario

A "performance" scenario: A remote user requests a data base report under peak load and receives it in under 5 seconds.



SOURCE

Remote user

ENVIRONMENT

Database under peak load

RESPONSE MEASURE

under 5 seconds

Additional Example Scenarios

Security: “A correctly identified individual modifies system data from an external site incorrectly. The system maintains an audit trail and the correct data is restored within one day.”

Modifiability: “A user requests a change to the user interface; The modification is made by a developer with no side effects, in three hours.”

Availability: “An unanticipated external message is received by a process during normal operation. The process informs the operator of the message’s receipt, and the system continues to operate with no downtime.”

Class Exercise 2

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

*** = indicates a software engineering method that has been extended to systems engineering**

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

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.

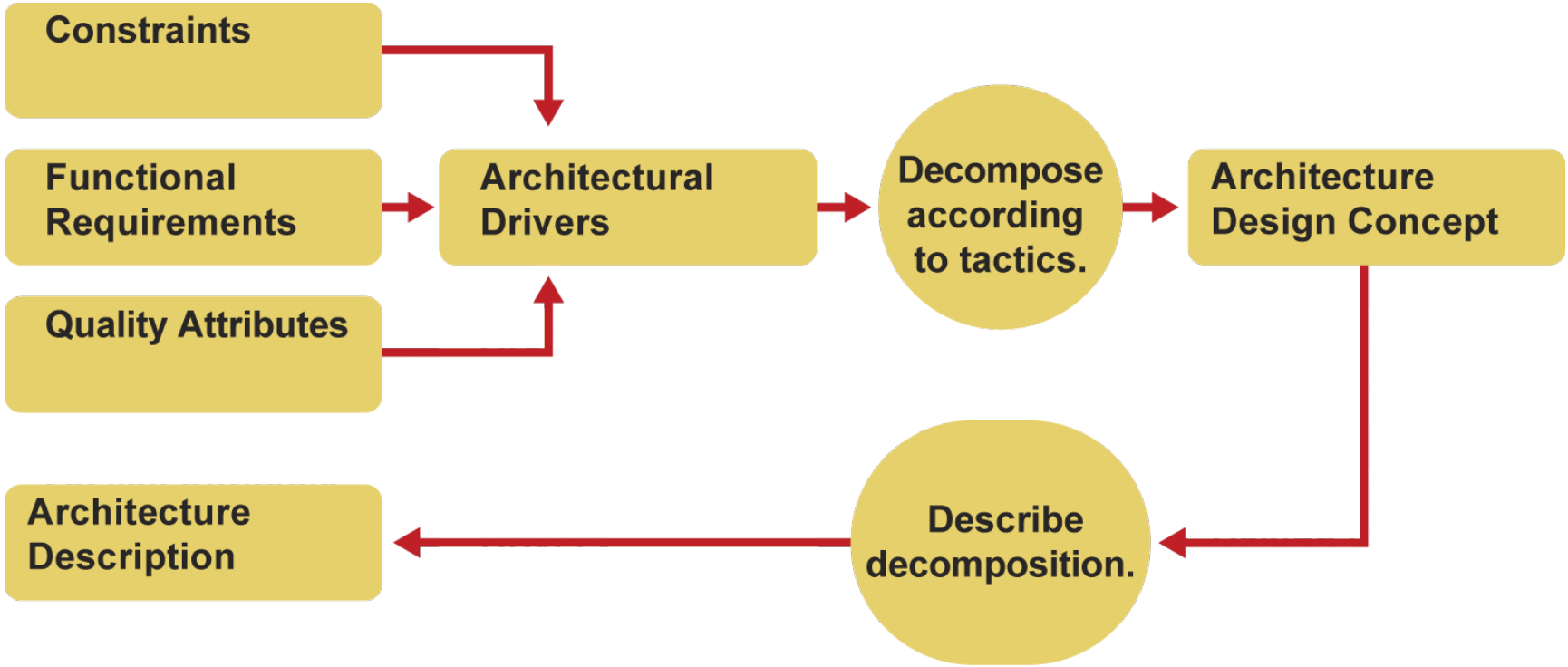
QAW Steps

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

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*.

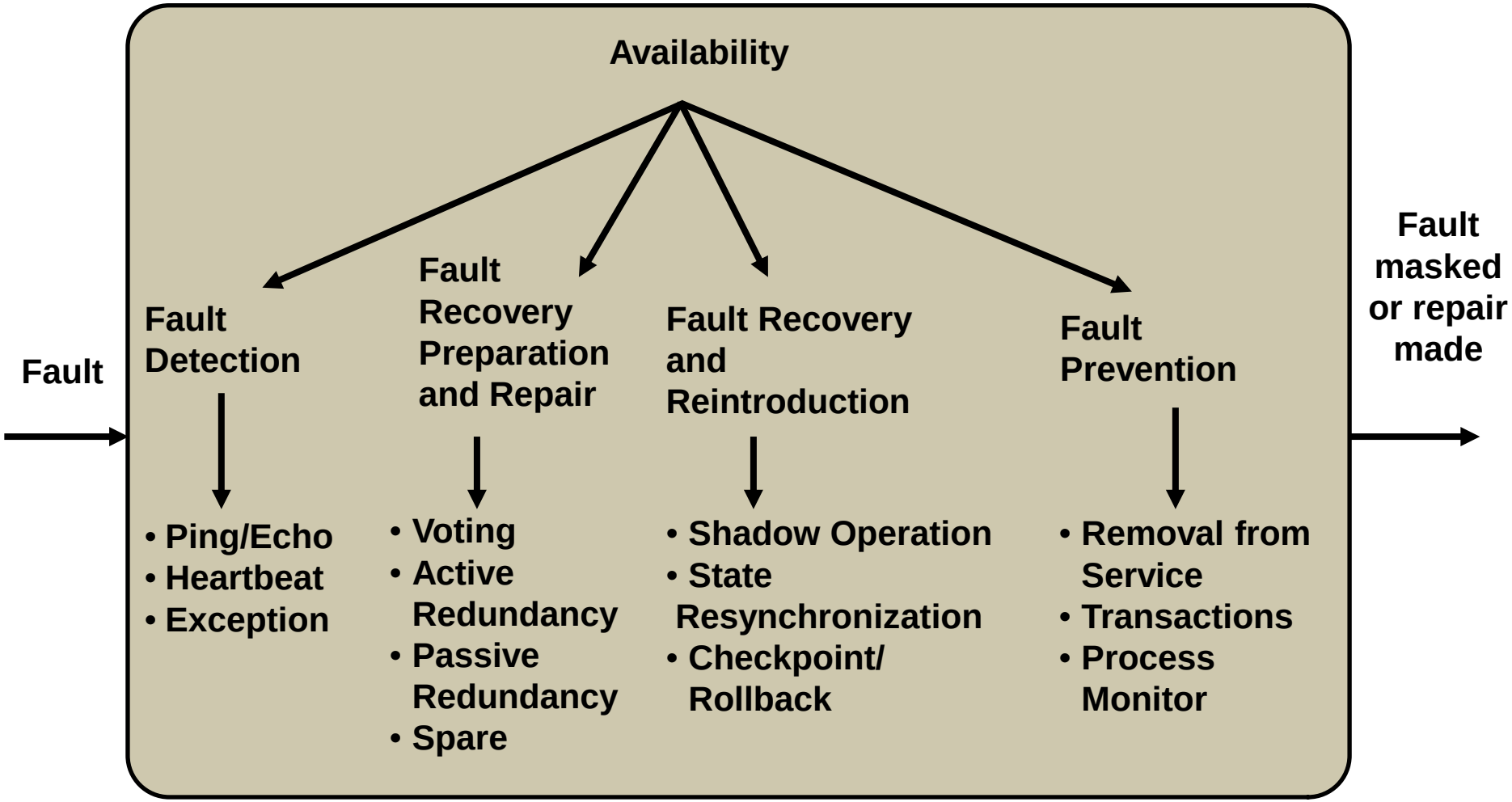
Each *tactic* is a design option available to the architect.

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

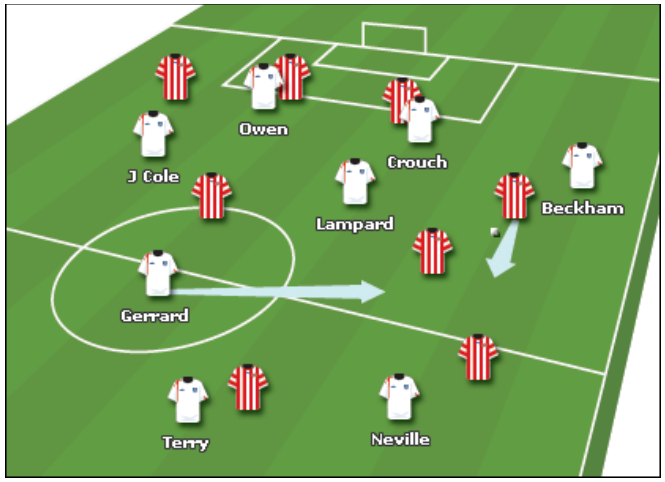
Summary of Availability Tactics



Other Tactics

There are tactics for

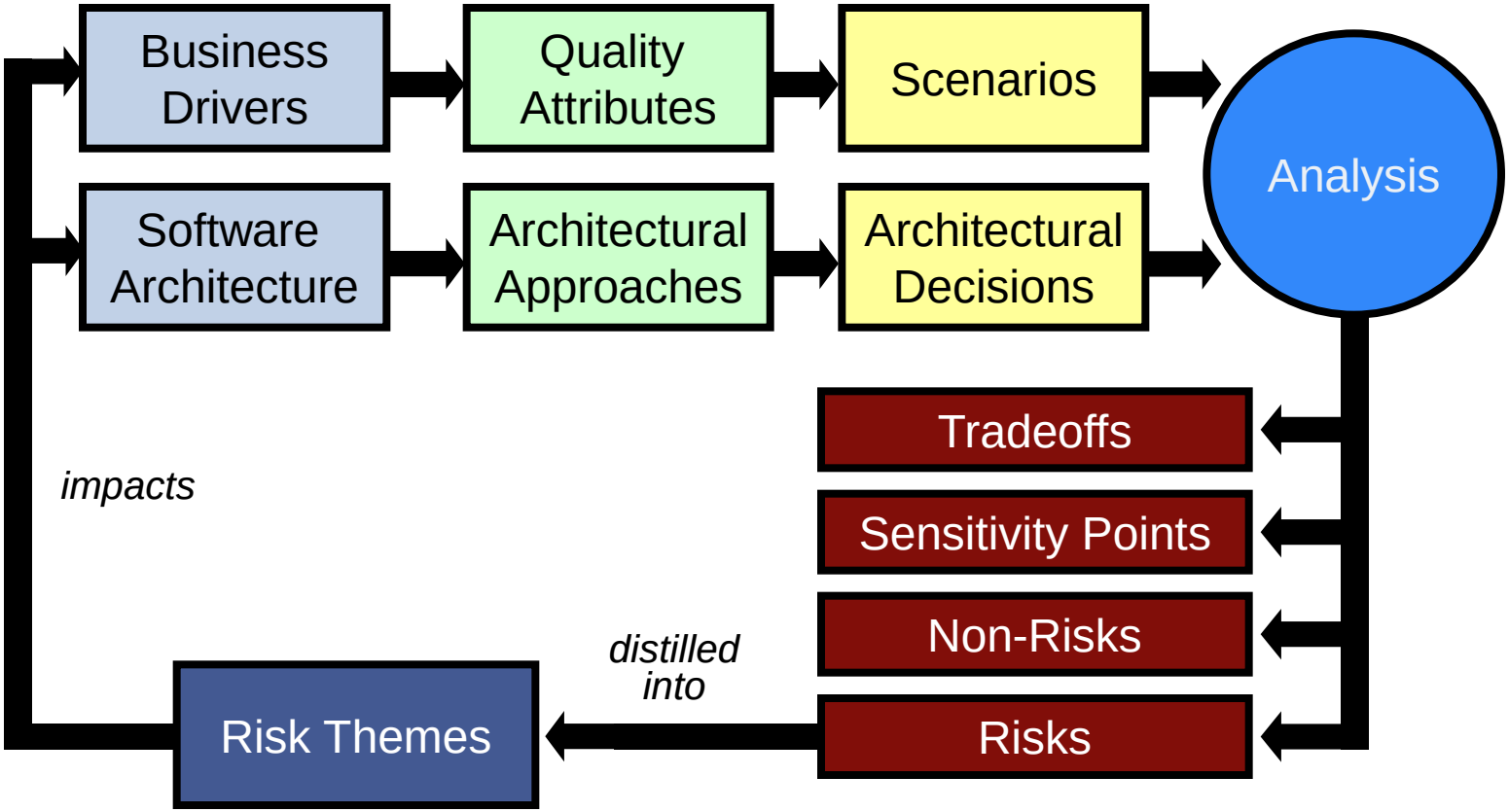
- modifiability
- performance
- security
- testability
- usability



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

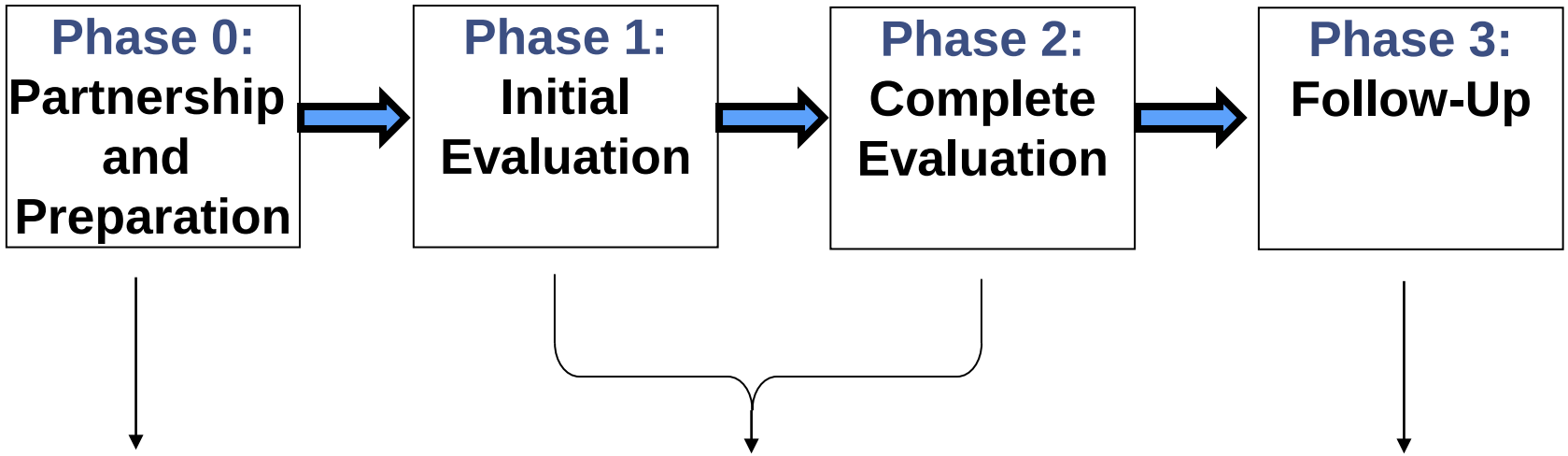
Analyzing the Architecture – SEI's Architecture Tradeoff Analysis Method® (ATAM®)

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

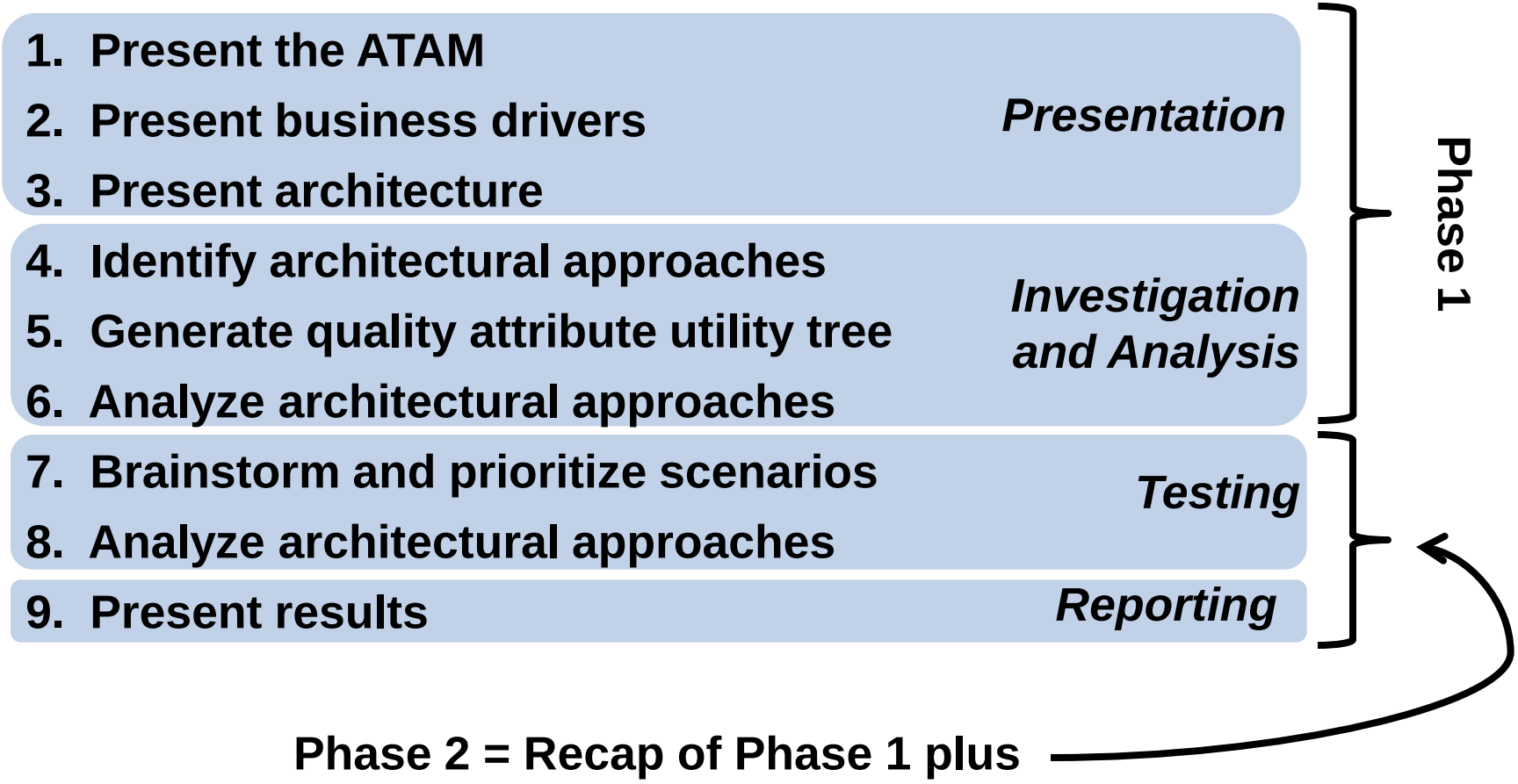


ATAM Phases

ATAM evaluations are conducted in four phases.



ATAM Evaluative Phases (1 & 2)



Documenting the 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.

Three Types of Views

Different types of views show different types of information:

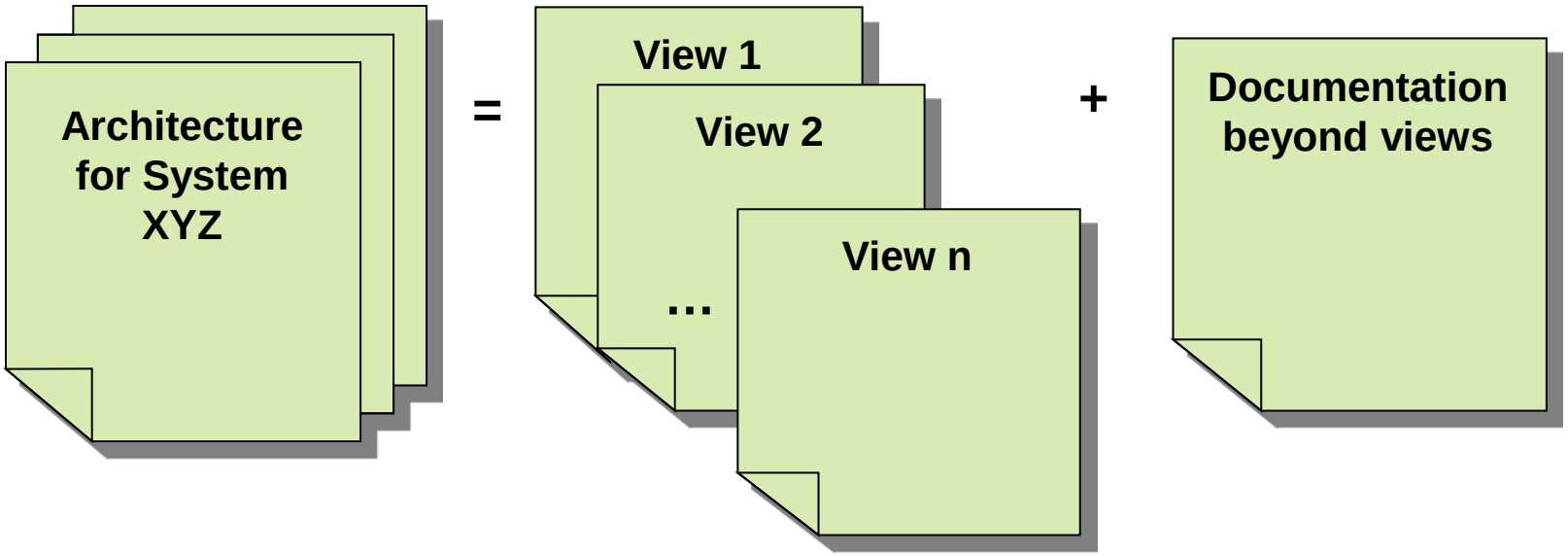
1. **Module views** show how the system is structured as a set of code units.
2. **Component-and-connector views** show how the system is structured as a set of elements with runtime behaviors and interactions.
3. **Allocation views** show how the system relates to non-software structures in its environment.

Every view contains information from at least one of these categories.

Some views contain information from more than one category, but these are often difficult to understand.

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

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 often not automated.

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

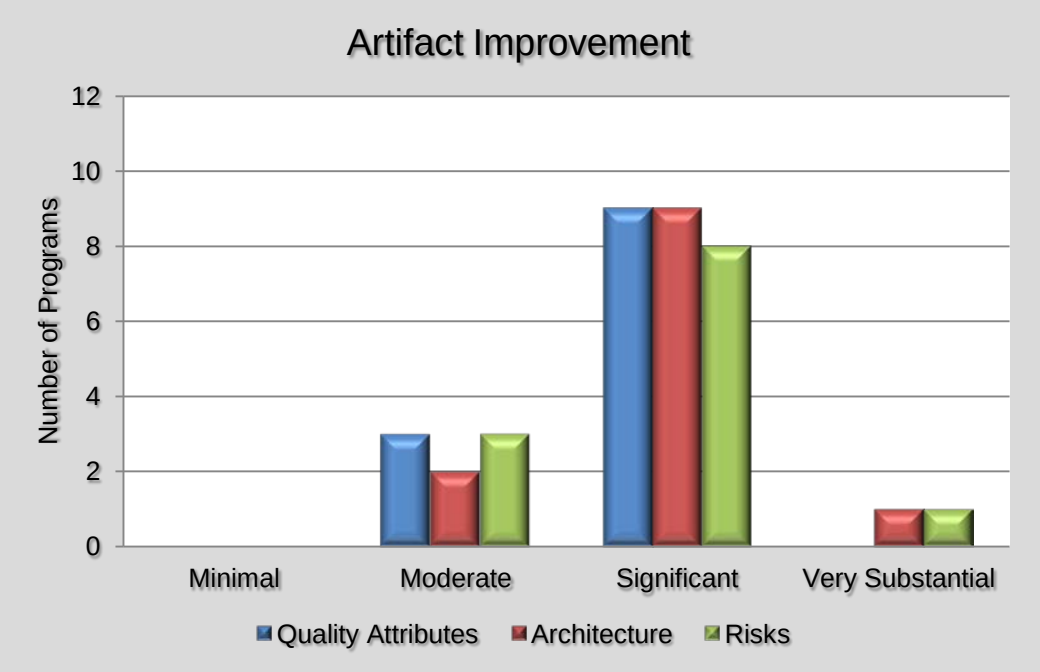
- 6/12: cost less than or equal to traditional techniques
- 10/12: quality of results greater than or equal to traditional techniques
- 10/12: helped understand and control cost and schedule
- 12/12: increased understanding of system's quality attribute requirements, design decisions, and risks
- 12/12: 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/QAW[2]



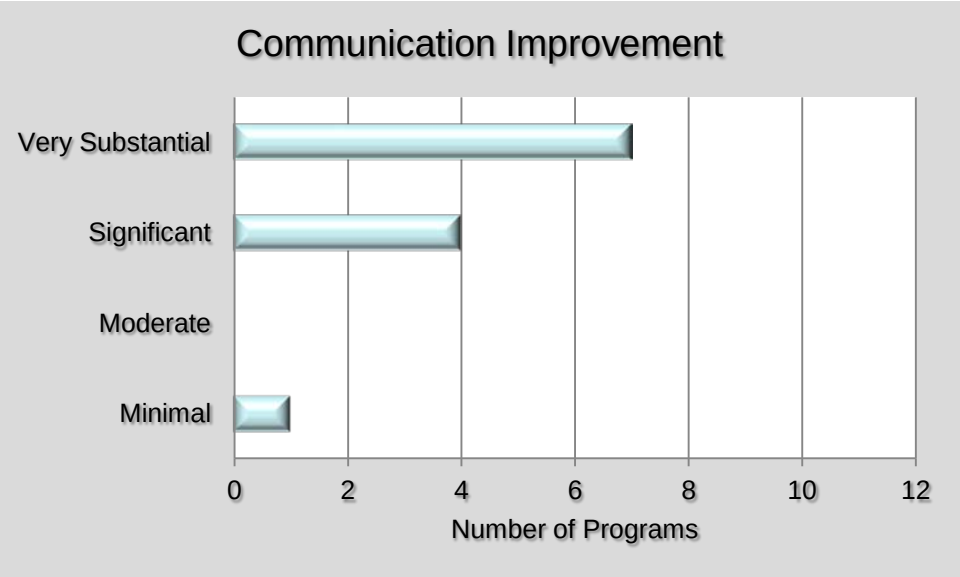
- Most reported *significant* improvement in their architecturally-significant artifacts
- Architecture teams were able to achieve understanding of stakeholder expectations and the implications of architectural decisions on user needs

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

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



What About System of Systems (SoS)? - 1

The software-intensive systems in an SoS are likely to have been developed independently of each other.

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

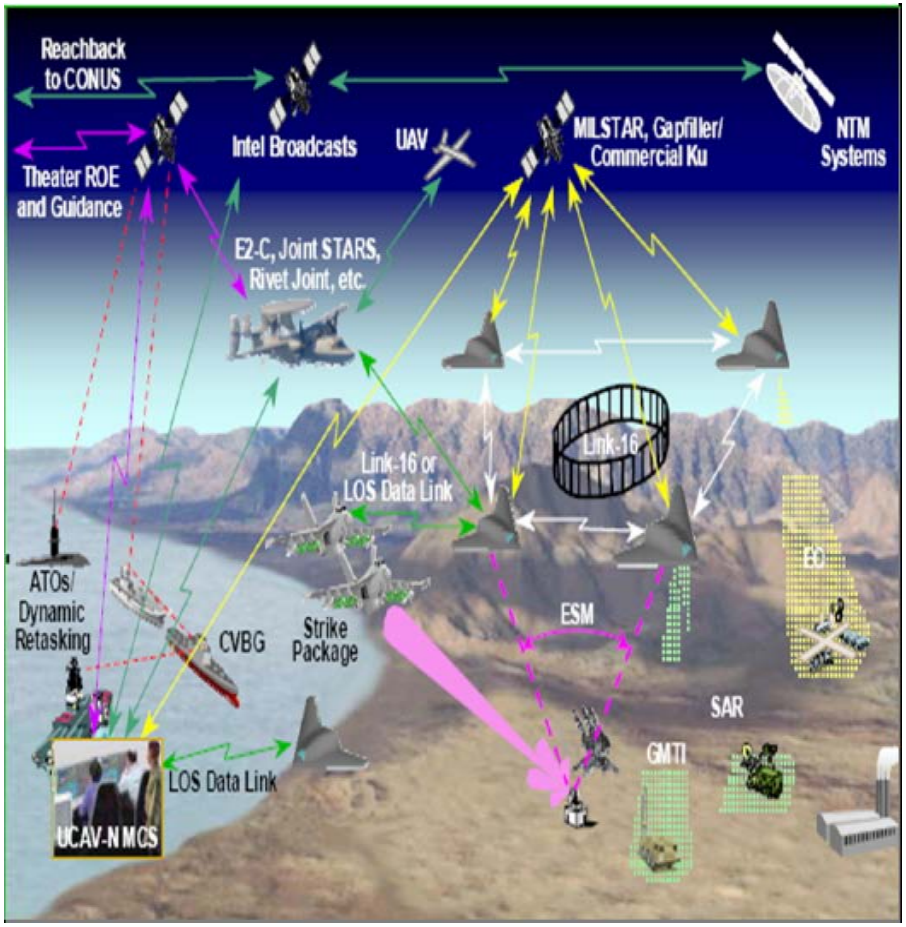
Thus, architecture is even more important in an SoS context, not less.

What About System of Systems (SoS)? - 2

Therefore, in an SoS context:

- Each software-intensive system in an SoS has its own architecture that addresses both system and software aspects.
- The SoS itself has an architecture whose elements are the architectures of the individual software-intensive systems.

A uniform approach for specifying quality attribute requirements and evaluating SoS and software-intensive system architectures against such requirements is thus 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.

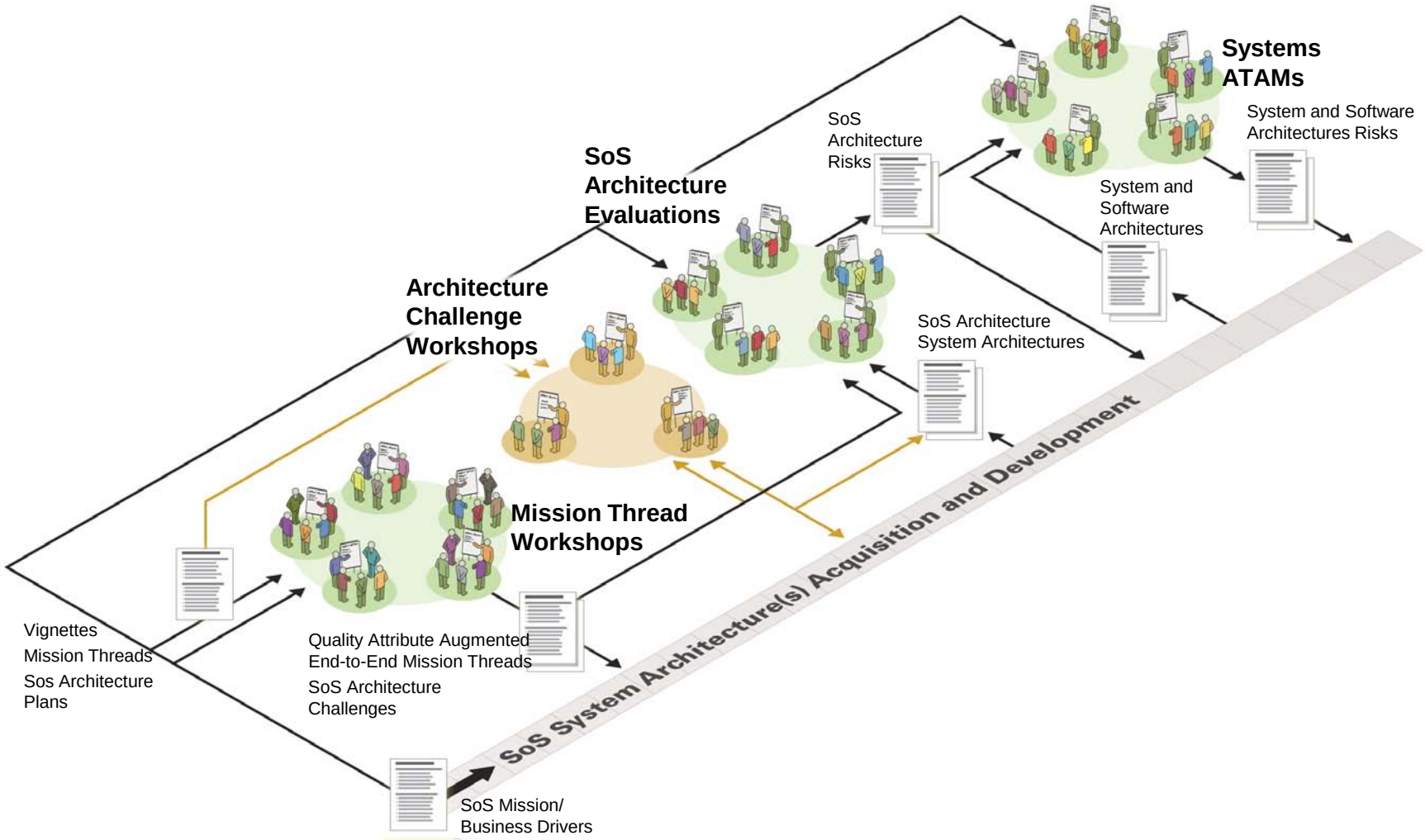
SoS end-to-end mission (operational or user) threads augmented with quality attribute considerations are needed to help develop, and later evaluate, the SoS architecture.

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

SoS and Quality Attribute Elicitation, Specification, and Analysis



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.

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.***

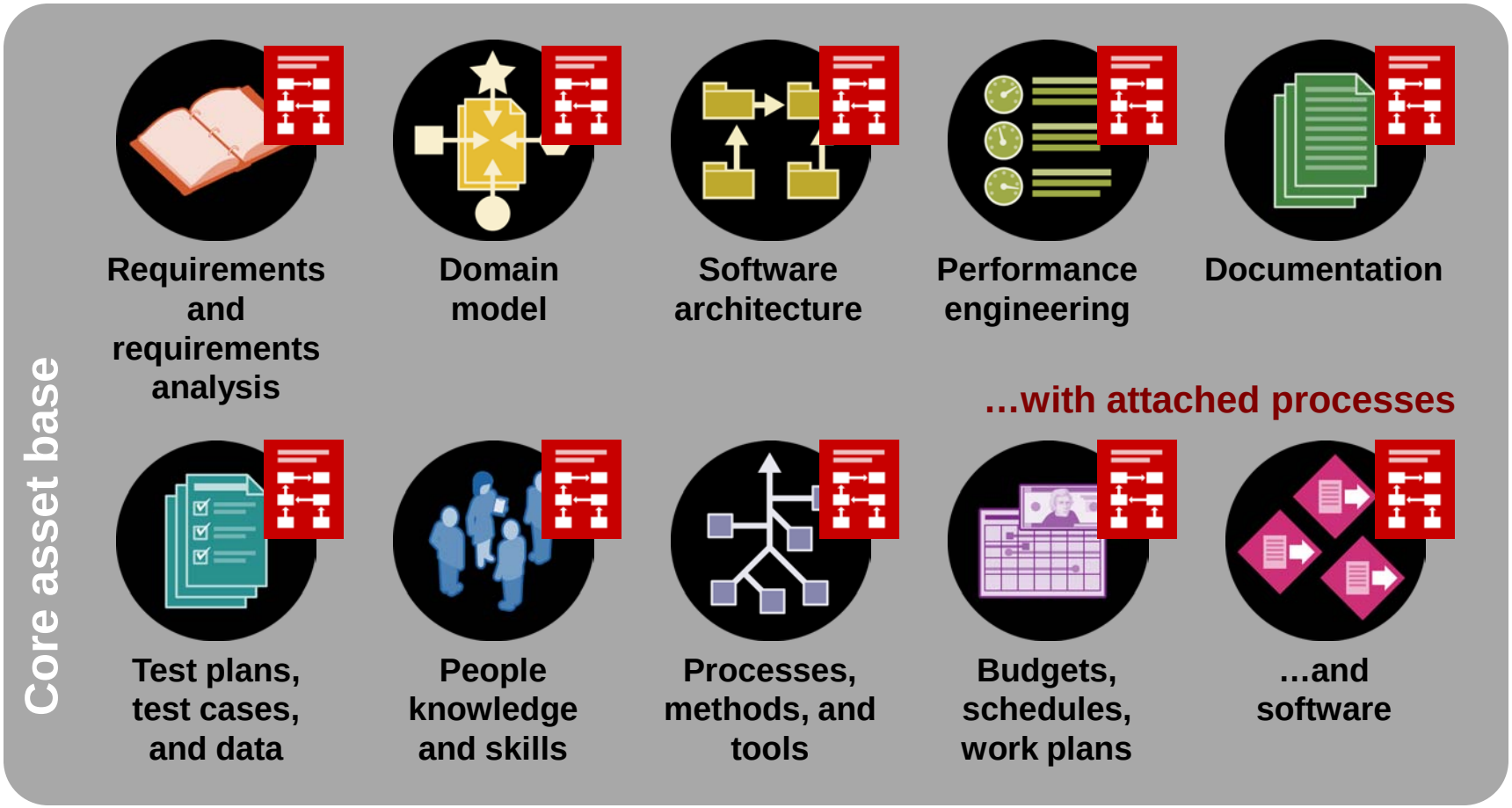
Successful Software Product Lines

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

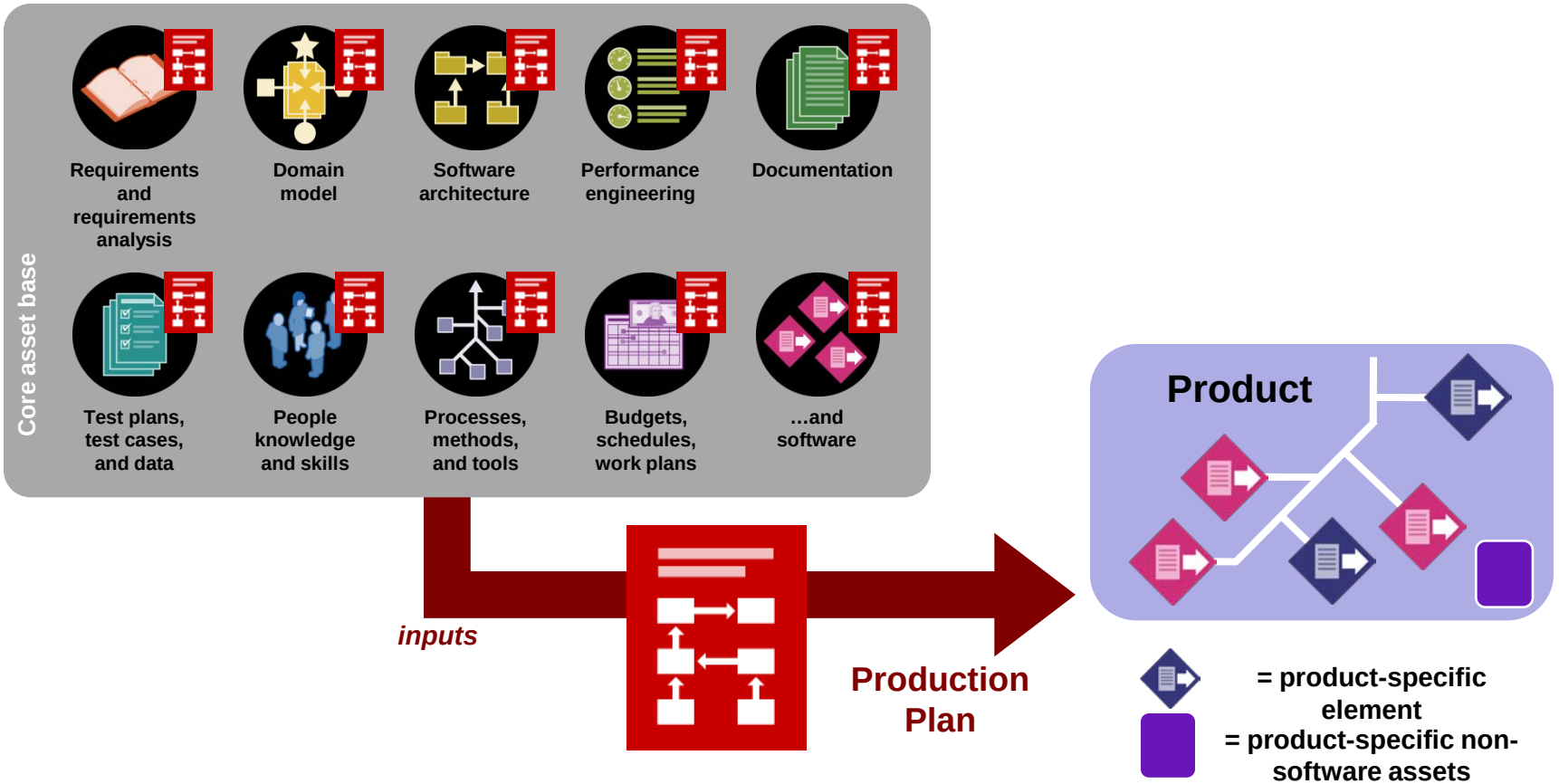
- **Cummins** reduced the time it takes to produce software for a diesel engine from one year to one week.
- **Motorola** realized a 400% productivity improvement in a family of one-way pagers.
- **Hewlett-Packard** reduced time to market by a factor of seven and increased productivity by a factor of 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.

Building the Core Asset Base

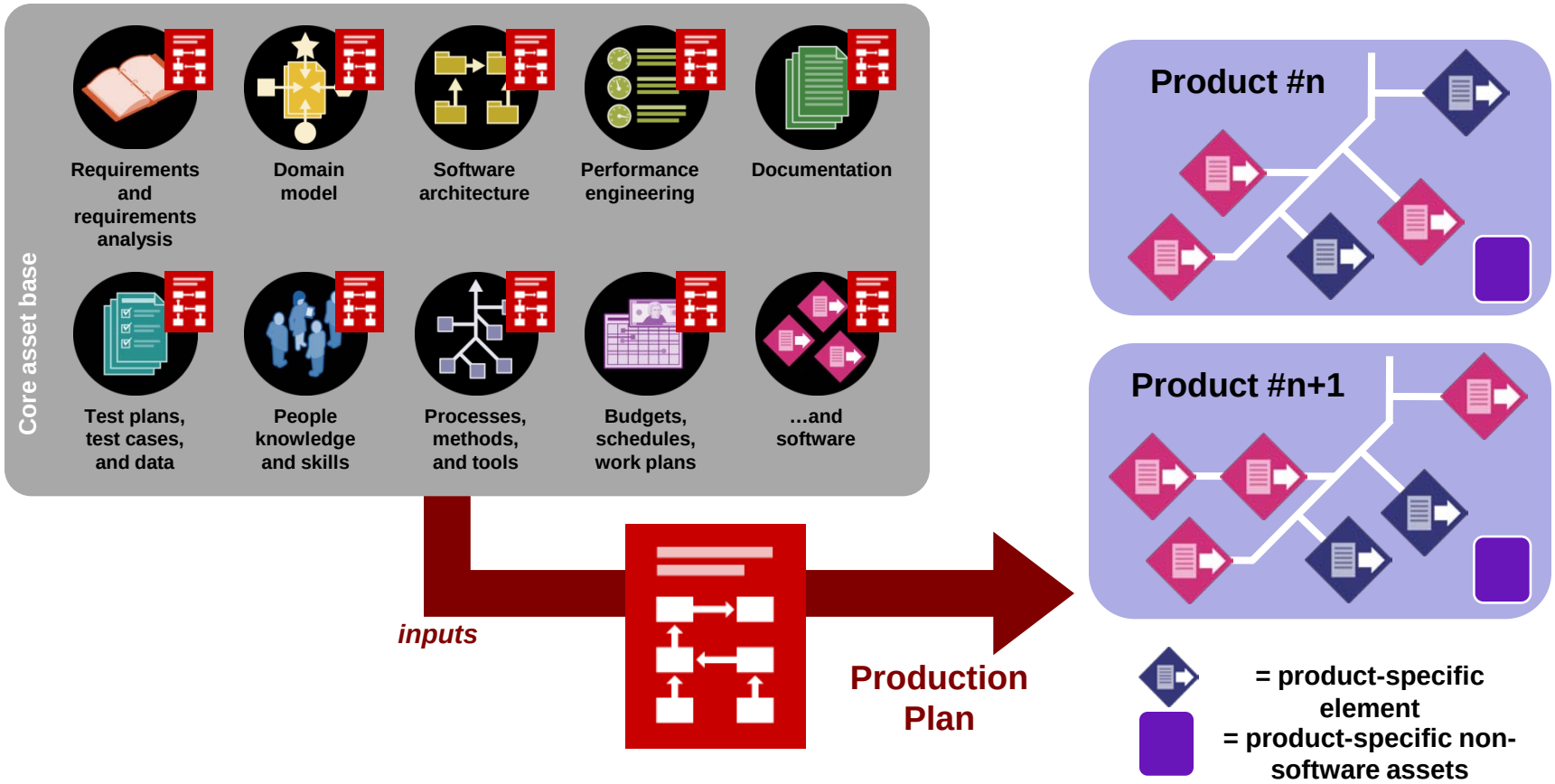
Core assets include:



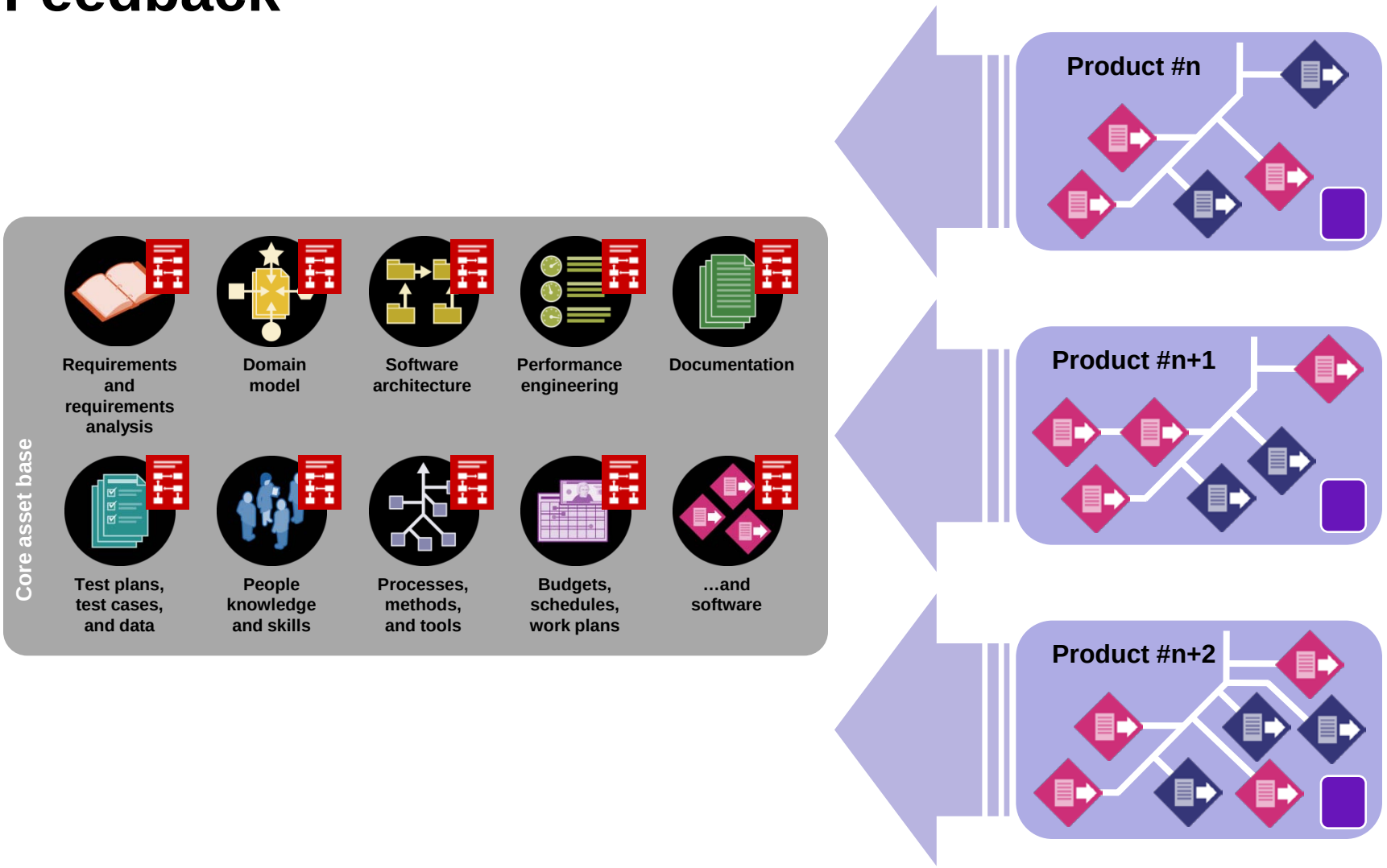
Building a Product...



Building Subsequent Products...



Feedback



Widespread Application - 1

akvasmart

Feed control and farm management software

BOEING

Bold Stroke Avionics

E-COM Technology Ltd.

Medical imaging workstations

hp invent

Firmware for computer peripherals

Lucent Technologies
Bell Labs Innovations

5ESS telecommunications switch

ABB **Asea Brown Boveri**

Gas turbines, train control, semantic graphics framework

Dialect

Internet payment gateway infrastructure products

ERICSSON

AXE family of telecommunications switches

LG

Elevator control systems

NOKIA

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

AXIS COMMUNICATIONS

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

DNV

Customized solutions for transportation industries

GM

Software for engines, transmissions and controllers

LSI LOGIC

RAID controller firmware for disk storage units

NASA

Interferometer product line

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

NAVSEA WARFARE CENTERS

Test range facilities

RICOH

Office appliances

SALION TARGET. WIN. DELIVER.

Revenue acquisition
management systems

TELVENT

Industrial supervisory control
and business process
management systems

U.S.ARMY

Command and control
simulator for Army fire
support

BOSCH

Automotive gasoline systems

SIEMENS

Software for viewing and quantifying
radiological images

testo

Climate and flue gas
measurement devices

alltel FIDELITY NATIONAL FINANCIAL

Support software

MOTOROLA

Pagers product line

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

Modern Engineering Approaches in CMMI - 1

For Version 1.3, CMMI provides better coverage of architecture-centric practices (mostly by changes to informative material):

- creating the **business case** for the system (partially in RD and TS)
- 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”)

RD = Requirements Development PA

TS = Technical Solution PA

VER = Verification PA

VAL = Validation Pa

A/PL = Agile and Product Lines notes (primarily in the Introductory Notes of a PA)

Modern Engineering Approaches in CMMI - 2

CMMI V1.3 also provides an improved terminology to support understanding and use of 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.

Modern Engineering Approaches in CMMI - 3

Added and revised the informative material throughout the Engineering PAs (in particular) to appropriately mention the following engineering concepts:

- quality attributes (i.e., non-functional requirements or “ilities”)
- architecture-centric practices
- product lines, system of systems
- 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 discussions that mention various approaches that can be used.

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

Requirements Development

SG 1: Develop Customer Requirements
- SP 1.1 Elicit Needs
- SP 1.2 Transform Stakeholder Needs into [Prioritized] 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

In SP1.2, added that customer requirements should be **prioritized** based on their criticality to the **customer** and other stakeholders "representing all phases of the product's lifecycle … including *business* as well as technical functions."

In SP 2.1, added a focus on **architectural requirements** and quality attribute **measures**.

In SP 2.2, added a subpractice allocating requirements to **delivery increments**.

Addressed "**Quality attributes**" (QAs) as well as functionality in SG3 and SP 3.2 statements.

In SP 3.1, broadened emphasis to **"operational, sustainment,** and **development"** scenarios.

In SP 3.2, determined **architecturally-significant QAs** from mission and business drivers.

Technical Solution

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

SG 3: Implement the Product Design

- SP 3.1 Implement the Design
- SP 3.2 Develop Product Support Documentation

Intro Notes: "QA **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."

In SP 1.1, Added an example selection criterion, "Achievement of key quality attribute requirements" and a new subpractice: "Identify **re-usable solution** components or applicable **architecture patterns.".**

In SP 2.1, described architecture definition tasks such as selecting **architectural patterns** and formally defining component behavior and interactions using an **architecture description language.**

In SP 2.2, added subpractice to determine **views** to document structures and address stakeholder concerns.

In SP 2.3, mentioned exception and error handling,

Product Integration

SG 1: Prepare for Product Integration
- SP 1.1 Establish an Integration Strategy
- SP 1.2 Establish the Product Integration Environment
- SP 1.3 Establish Product Integration Procedures and Criteria

SG 2: Ensure Interface Compatibility
- SP 2.1 Review Interface Descriptions for Completeness
- SP 2.2 Manage Interfaces

SG 3: Assemble Product Components and Deliver the Product
- SP 3.1 Confirm Readiness of Product Components for Integration
- SP 3.2 Assemble Product Components
- SP 3.3 Evaluate Assembled Product Components
- SP 3.4 Package and Deliver the Product or Product Component

Revised the purpose to ensure proper **behavior** instead of proper function, thereby more implicitly including **quality attributes** as well as functionality.

Changed emphasis from integration sequence to an emphasis on **integration strategy**, i.e., the approach to receiving, assembling, and evaluating product components. The **architecture** will significantly influence the selection of a product integration strategy.

In the PA notes, addressed: **interfaces** to data sources and middleware; APIs, **automated builds, continuous integration**

Validation

SG 1: Prepare for Validation

- SP 1.1 Select Products for Validation
- SP 1.2 Establish the Validation Environment
- SP 1.3 Establish Validation Procedures and Criteria

SG 2: Validate Product or Product Components

- SP 2.1 Perform Validation
- SP 2.2 Analyze Validation Results

Reinforced when validation occurs in the product lifecycle: "validation is performed early (concept/exploration phases) and incrementally throughout the product lifecycle (including transition to operations and sustainment)."

In VAL SP 1.1, included **access protocols** and data interchange reporting formats as examples of what to validate.

Also, included **incremental delivery of working and potentially acceptable product** as an example validation method.

Verification

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

In SP 1.1, added example verification methods**: software architecture evaluation and implementation conformance evaluation** and continuous integration.

In SP 1.3, added example sources of verification criteria: customers reviewing work products collaboratively with developers.

In SP 2.1, added example type of peer review: **architecture implementation conformance evaluation**

In SP 2.3, added 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 defects

Addressing Agile - 1

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 - 2

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 - 3

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 - 4

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 Product Lines - 1

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 - 2

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]*

Changes in CMMI Terminology - 1

Allocated requirement

DEFINITION

Requirement that ~~levies~~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.

The improvements to the above definition make the substance of the solution space and the allocation of requirements to it more explicit, enabling insightful analyses (including verification) of requirements, architectures, and implementations.

Changes in CMMI Terminology - 2

Architecture

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.”)

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.

Changes in CMMI Terminology - 3

Definition of required functionality and quality attributes

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].

The term "definition of required functionality" that appeared in V1.2 has been removed from CMMI because of the implicit suggestion that functionality be addressed first or has higher priority. The term has been replaced with the one above, which is intended to help ensure a sufficiently balanced and concurrent focus (functional *and* non-functional) in requirements analysis.

Changes in CMMI Terminology - 4

“Functional analysis” and “functional architecture”

These terms, which appeared in earlier versions of CMMI, are now “cul de sacs” in the model.

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

The note in RD SP 3.2 contrasts the approaches implied by these terms with “modern engineering approaches” that encourage a more balanced and concurrent 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

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.

This term is now included in the Glossary for the first time. This 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.

Changes in CMMI Terminology - 7

“Performance” used by itself can be ambiguous

A “quality attribute” for CMMI is clarity ☺. One term that has repeatedly caused problems in translations was “performance.” For V1.3, each use of the term was examined to ensure it was unambiguous, correctly used, and where appropriate the term was qualified:

- supplier 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

Changes in CMMI Terminology - 8

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.

The above term appears in many CMMI practices. This term was changed in V1.3 to emphasize that artifacts that have a long-term role need to evolve to remain useful. Example from RD SP 2.1 note: "The modification of requirements due to approved requirement changes is covered by the "maintain" aspect of this specific practice…" The issue of how much to document is also addressed, e.g., TS SP 2.2.

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.

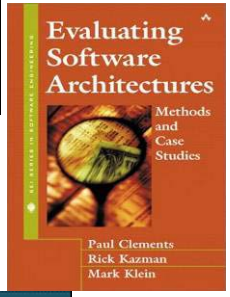
References - 1

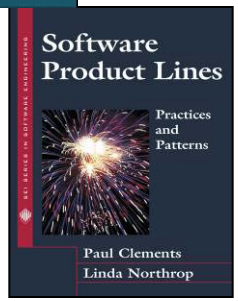
Software Architecture in Practice, Second Edition
Bass, L.; Clements, P.; & Kazman, R. Reading, MA: Addison-Wesley, 2003.

Evaluating Software Architectures: Methods and Case Studies
Clements, P.; Kazman, R.; & Klein, M. Reading, MA: Addison- Wesley, 2002.

Documenting Software Architectures: Views and Beyond, Second Ed.
Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. Reading, MA: Addison-Wesley, 2010.

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

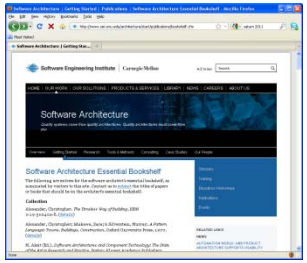




References - 2

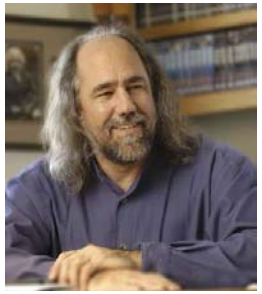
You can find a moderated list of references on the “Software Architecture Essential Bookshelf”

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

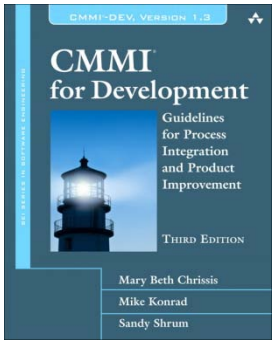


CMMI for Development, Version 1.3

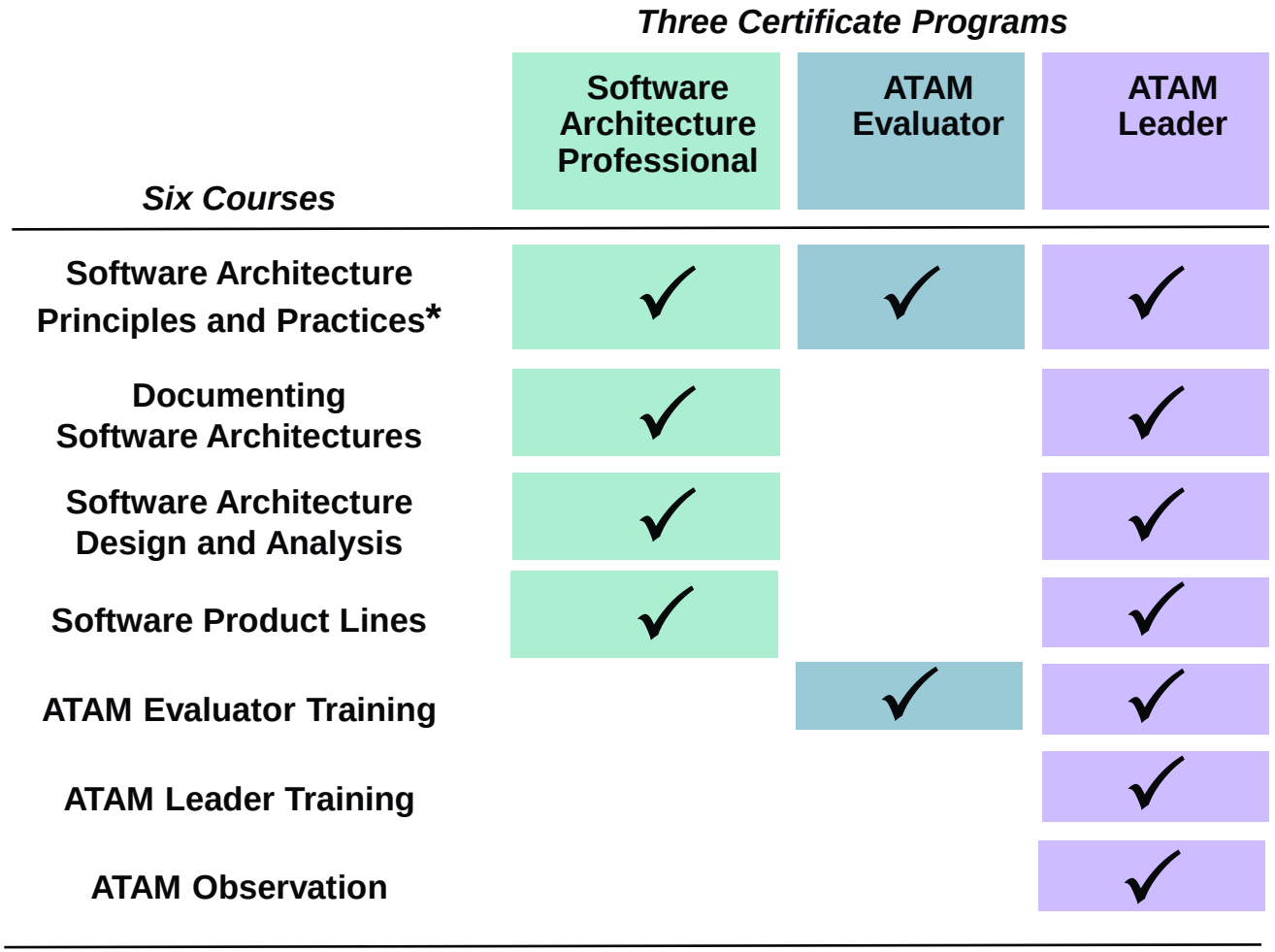
http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm

(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



Contact Information

Larry Jones

Research, Technology, and Systems Solutions Program

Telephone: 719-481-8672

Email: lgj@sei.cmu.edu

Mike Konrad

SEPM/CMMI

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.