Carnegie Mellon
Software Engineering Institute

Pittsburgh, PA 15213-3890

Best Practices in Software Architecture

Paul Clements
Software Engineering Institute / Carnegie Mellon University
and
Indian Institute of Technology - Bombay
26 July 2006

Sponsored by the U.S. Department of Defense

Carnegie Mellon
Software Engineering Institute

Software Engineering Institute

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

Established in 1984

Technical staff of 335

Offices in Pittsburgh, Pennsylvania, Arlington, Virginia, and Frankfurt, Germany

Purpose: Help others make measured improvements in their software engineering practices

Product Line Systems Program

One of 5-6 programs at the SEI, with about 30 people.
Our goal is to make improvements in

- Software product line engineering
- Predictable assembly of certifiable components
- Software architecture
    - Creation
    - Documentation
    - Evaluation
    - Use in system-building

Software architecture

The rise of software architecture has resulted from two trends:

- Recognition of the importance of quality attributes
- The development of very large and very complex systems

Building large, complex systems

Large-scale design decisions cannot be made by programmers.

- Have limited visibility and short-term perspectives
- Trained in technology solutions to specific problems.

Teams can only be coordinated, and QA's can only be achieved, by making broad design decisions that apply to the entire system – all of its elements.

Importance of quality attributes

If the only criterion for software was to get the right answer, we would not need architectures—*unstructured, monolithic systems would suffice*.

But other things also matter, such as

- modifiability
- time of development (time to market)
- performance
- coordination of work teams

These and other system quality attributes are largely dependent on architectural decisions.

- All design involves tradeoffs in system qualities.
- The earlier we reason about tradeoffs, the better.

Carnegie Mellon
Software Engineering Institute

What Is Software Architecture?

Software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of these elements, and the relationships among them.

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

Structures: Plural!

Systems can and do have many structures.

- No single structure can be *the* architecture.
- The set of candidate structures is not fixed or prescribed.
- Relationships and elements might be runtime related such as
  - “sends data to,” “invokes,” or “signals”
  - processes or tasks
- Relationships and elements might be nonruntime related such as
  - “is a submodule of,” “inherits from,” or “is allocated to team X for implementation”
  - a class or library
- Representations of structures are *views* of the architecture
  - All modern approaches to architecture embody the concept of multiple views.

A Picture of Architecture-Based Development

Development organizations who use architecture as a fundamental part of their way of doing business often define an architecture-based development process.

This talk will illuminate some parts of that process.

One of the early parts is understanding the architecturally significant requirements.

Quality attributes

If we accept the importance of quality attributes, then we need to understand how to specify and capture them…

- Our customer has to tell us what he wants
- Our architect and designers must understand it
- Our programmers have to achieve it
- Our testers have to test for it

…and how to design and build software to achieve them.

QA’s fall into two groups

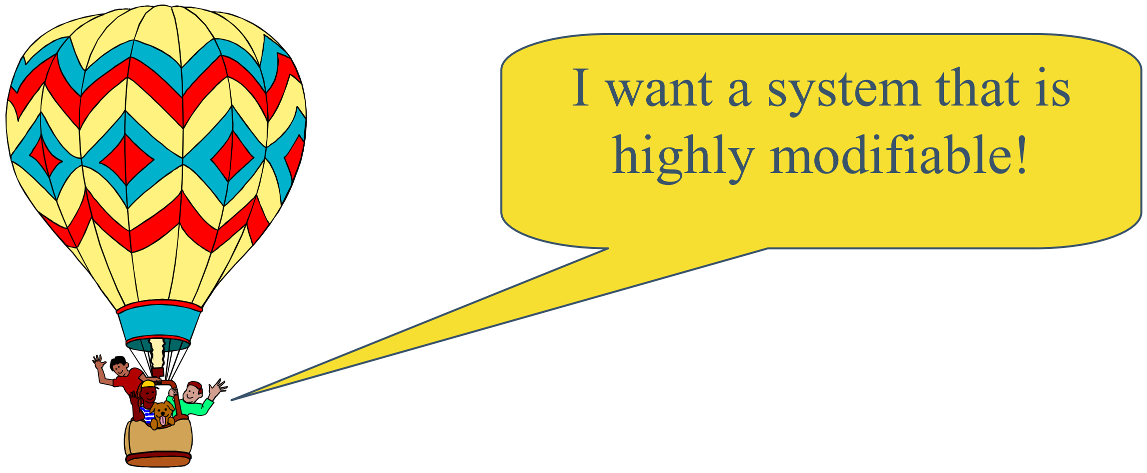
“Run-time” QA’s

- We can measure how well a system exhibits these by watching the system in operation
- Performance, security, availability, …

“Non-run-time” QA’s

- We can measure these by watching a team in operation
- Maintainability, portability, buildability, time to market…

Specifying quality attributes



Conclusion: Just naming a quality attribute doesn't help very much.

We can't build software with just that. We need to be more specific.

Most people use *quality attribute scenarios* to capture quality attributes.

Scenarios

A scenario is a little story describing an interaction between a stakeholder and a system.

A use case is a kind of scenario. The stakeholder is the user. The interaction is a functional use of the system.

- “The user pushes this button, and this result occurs.”

We can generalize the notion of a use case to come up with *quality attribute scenarios*.

A quality attribute scenario is a short description of how a system is required to respond to some stimulus.

QA Scenarios

A quality attribute scenario has six parts:

- source – an entity that generates a stimulus
- stimulus – a condition that affects the system
- artifact – the part of that was stimulated by the stimulus
- environment – the condition under which the stimulus occurred
- response – the activity that results because of the stimulus
- response measure – the measure by which the system's response will be evaluated

A QA Scenario for 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.*

1. source – external
2. stimulus – unanticipated message received
3. artifact – process
4. environment – during normal operation
5. response – system continues to operate
6. response measure – zero downtime

A QA Scenario for Modifiability

- *During maintenance, a change is made to the system's rules engine. The change is completed in one day.*

1. source – requestor of the change
2. stimulus – a change is made
3. artifact – rules engine
4. environment – during maintenance
5. response – the change is completed
6. response measure – …in one day

A QA Scenario for Security

- *During peak operation, an unauthorized intruder tries to download prohibited data via the system administrator’s interface. The system detects the attempt, blocks access, and notifies authorities within 15 seconds.*

1. source – an unauthorized intruder
2. stimulus – tries to download prohibited data
3. artifact – system administrator’s interface
4. environment – during peak operation
5. response – the attempt is detected, blocked, reported
6. response measure – …within 15 seconds

More about QAs

There is no standard set of quality attributes

- People disagree on names: Maintainability/modifiability/portability
- People come up with new ones: “calibrate-ability”
- There is no standard meaning of what it means to be “secure”

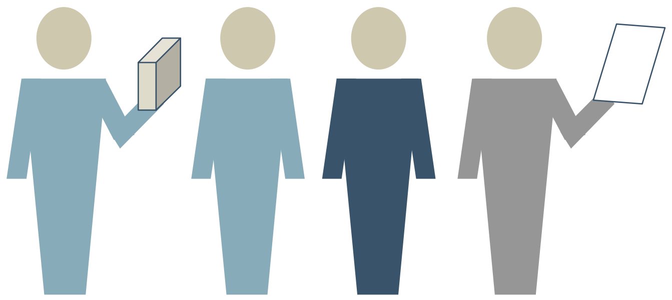
Scenarios let us avoid all of these problems!

The QAs are defined by the scenarios!

Who tells us what QA’s are important? **Stakeholders!**

Stakeholders

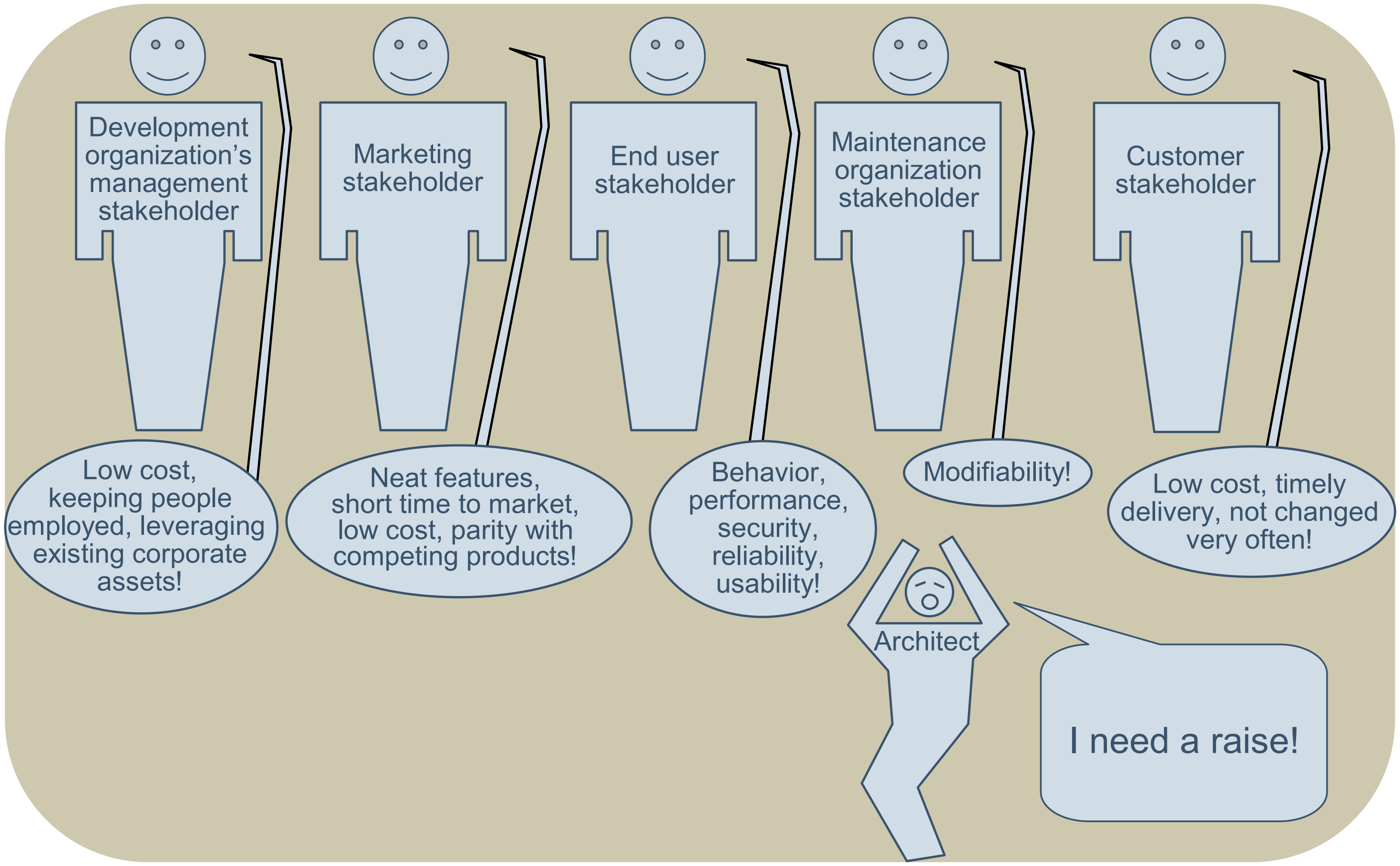
Stakeholders are people with a vested interest in the system. They are the people who can tell us what is needed. They are the people who can tell us if what we are building is the right thing.



We usually think of the user as telling us what is required, but there are many kinds of stakeholders.

Carnegie Mellon
Software Engineering Institute

Concerns of System Stakeholders


Development organization's management stakeholder
Low cost, keeping people employed, leveraging existing corporate assets!
Marketing stakeholder
Neat features, short time to market, low cost, parity with competing products!
End user stakeholder
Behavior, performance, security, reliability, usability!
Maintenance organization stakeholder
Modifiability!
Customer stakeholder
Low cost, timely delivery, not changed very often!
Architect
I need a raise!

Stakeholder Involvement

Stakeholders' quality attribute requirements are seldom documented, which results in

- goals not being achieved
- conflict between stakeholders

Architects must identify and actively engage stakeholders early in the life cycle to

- understand the real constraints of the system (many times, stakeholders ask for everything!)
- manage the stakeholders' expectations (they can't have everything!)
- negotiate the system's priorities
- make tradeoffs

SEI Quality Attribute Workshop (QAW)

The QAW is a facilitated method that engages system stakeholders early in the life cycle to discover the driving quality attributes of a software-intensive system.

Key points about the QAW are that it is

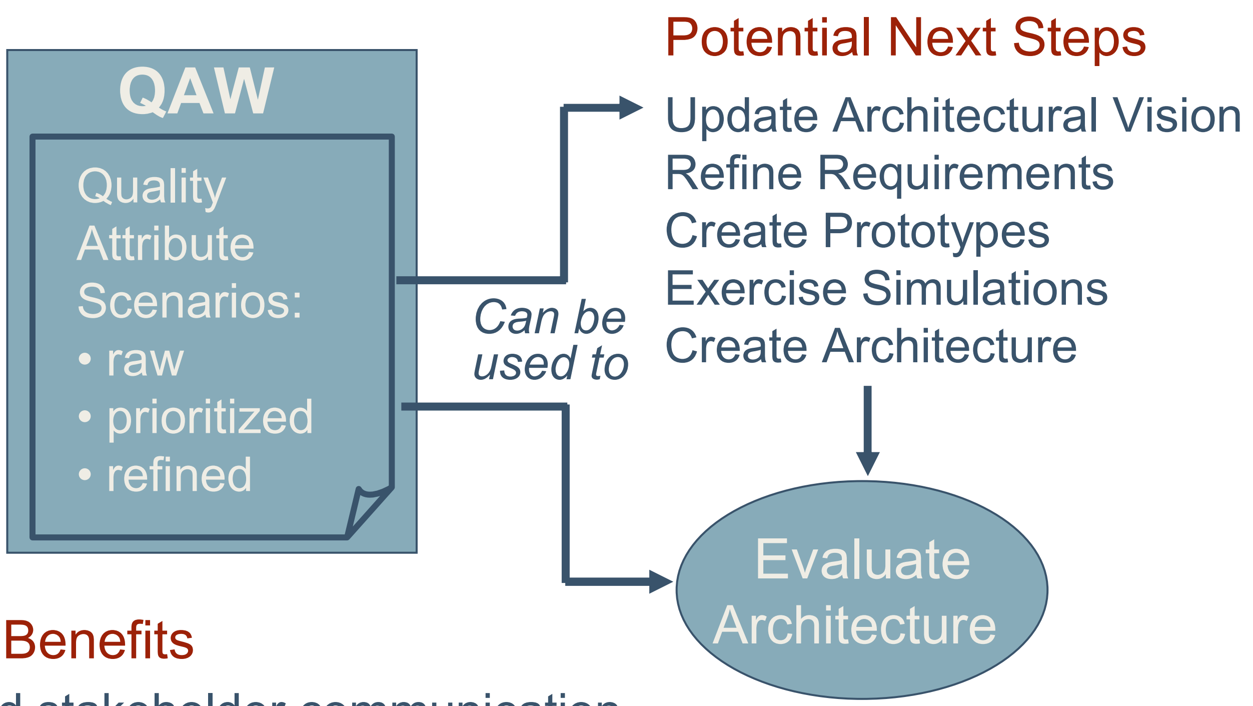
- system-centric
- stakeholder focused
- used before the software architecture has been created

QAW Steps

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

Iterate as necessary with broader stakeholder community

QAW Benefits and Next Steps



Potential Benefits

- increased stakeholder communication
- clarified quality attribute requirements
- informed basis for architectural decisions

Creating the architecture

Architects primarily work by using previously-tried solutions

- Large scale: Patterns and styles
- Small scale: Tactics

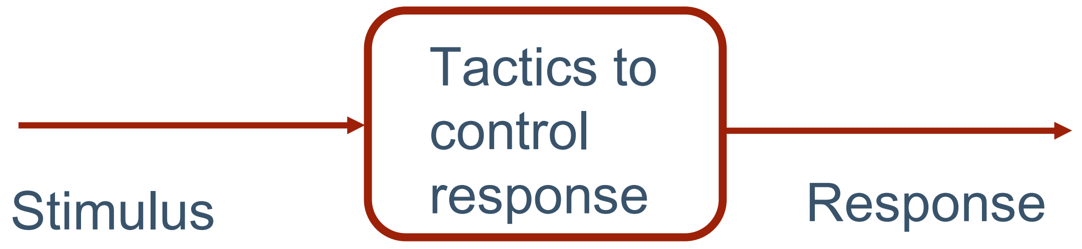
Styles, patterns, and tactics represent conceptual tools in the architect’s “tool bag.”

Professional architects always keep their tool bag up to date.

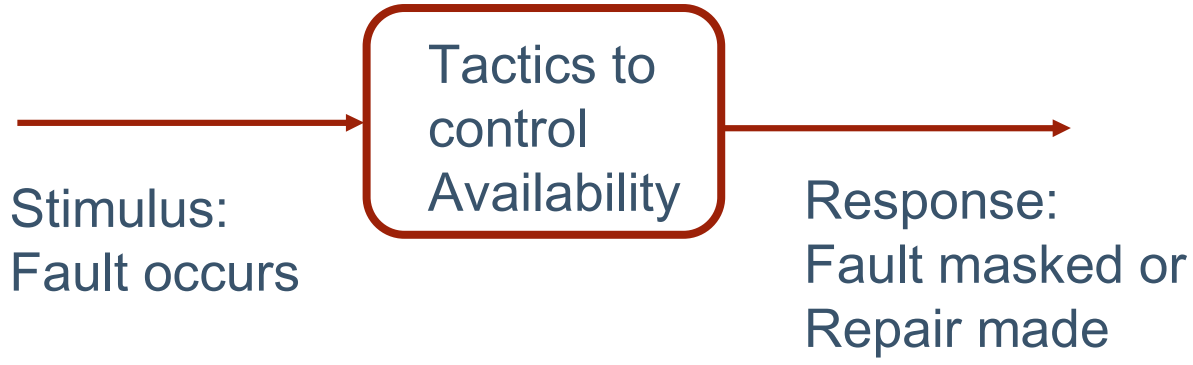
Tactics

An architectural *tactic* is a fine-grained design approach used to achieve a quality attribute response.

Tactics are the “building blocks” of design from which architectural patterns are created.

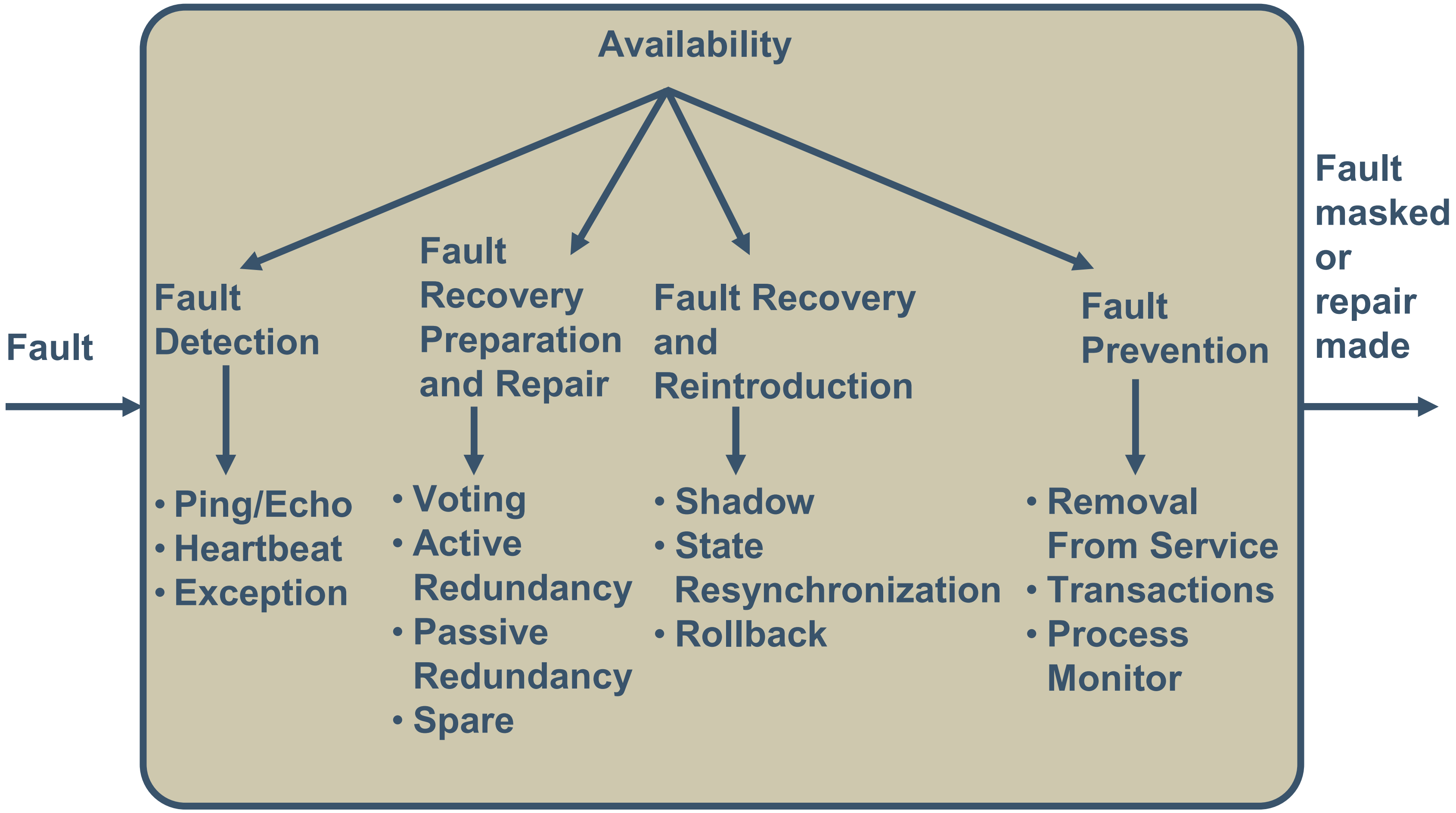


Tactics for Availability



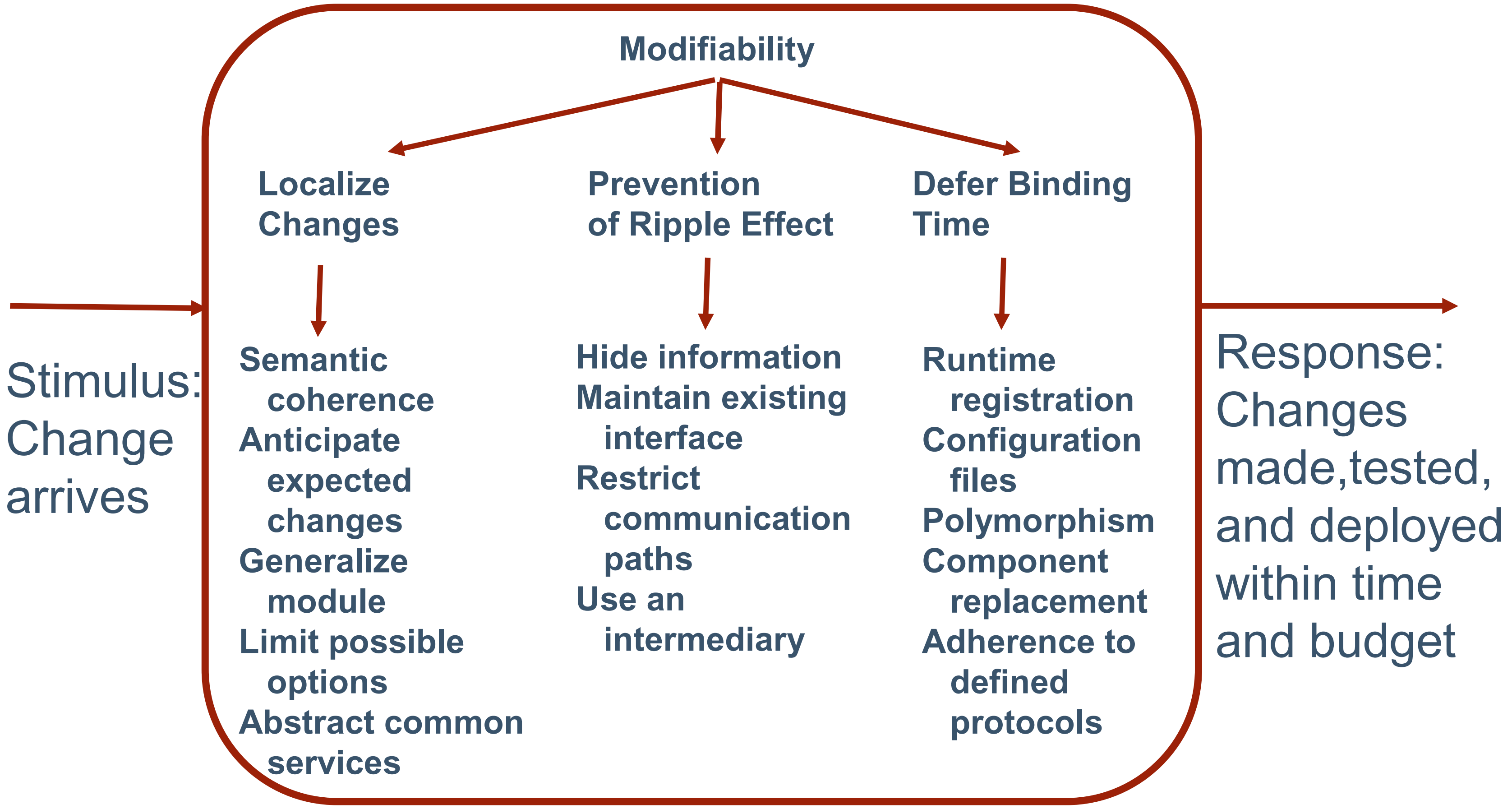
Carnegie Mellon
Software Engineering Institute

Summary of Availability Tactics



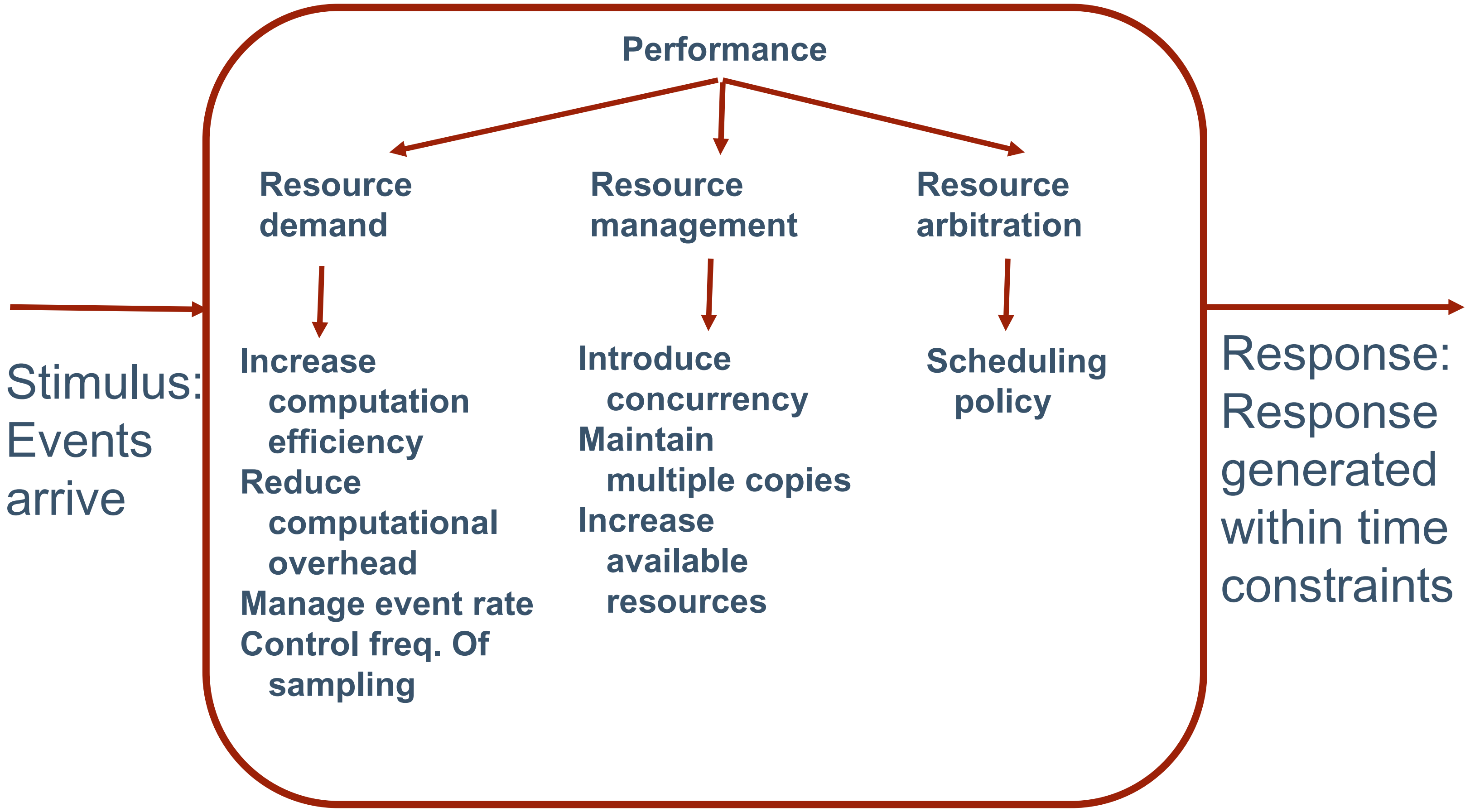
Carnegie Mellon
Software Engineering Institute

Summary of Modifiability Tactics

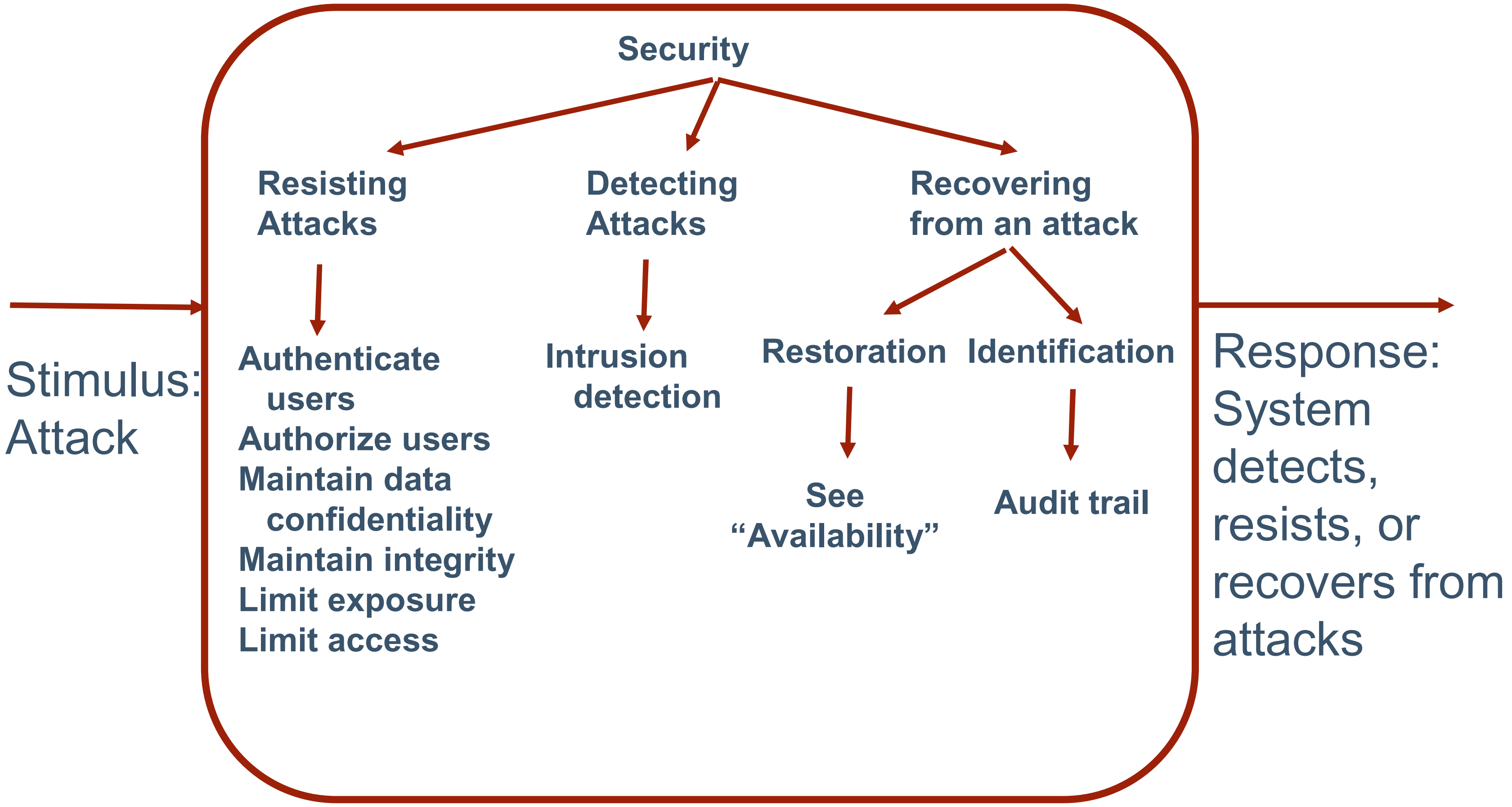


Carnegie Mellon
Software Engineering Institute

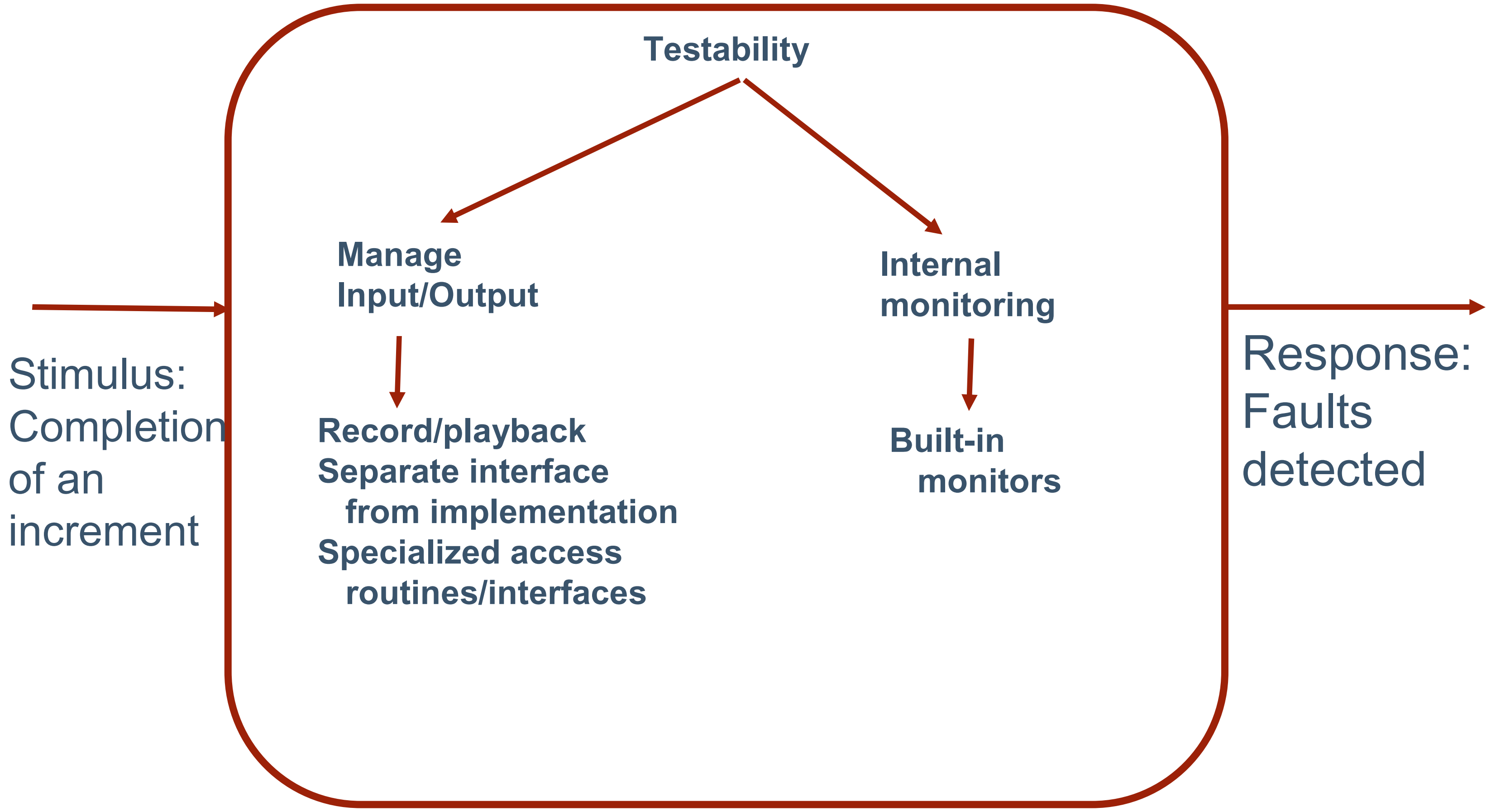
Tactics for Performance



Tactics for Security



Tactics for Testability


Testability
Manage Input/Output
Internal monitoring
Record/playback
Separate interface from implementation
Specialized access routines/interfaces
Built-in monitors
Stimulus: Completion of an increment
Response: Faults detected

Attribute-Driven Design (ADD) Method

ADD is a step-by-step method for systematically producing the first architectural designs for a system.

ADD results

- Overall structuring decisions
- Interconnection and coordination mechanisms
- Application of patterns and tactics to specific parts of architecture
- Explicit achievement of quality attribute requirements
- NOT detailed interfaces

ADD requires as input:

- Quality attribute requirements
- Functional requirements
- Constraints

Attribute-Driven Design (ADD) Steps

Step 1: Confirm there is sufficient requirements information

Step 2: Choose part of the system to decompose

Step 3: Prioritize requirements and identify architectural drivers

Step 4: Choose design concept – patterns, styles, tactics -- that satisfies the architectural drivers associated with the part of the system we've chosen to decompose.

Step 5: Instantiate architectural elements and allocate functionality

Step 6: Merge designs completed thus far

Step 7: Allocate remaining functionality

Step 8: Define interfaces for instantiated elements

Step 9: Verify and refine requirements and make them constraints for instantiated elements

Step 10: Repeat steps 2 through 9 for the next part of the system you wish to decompose

Now what?

How do we know that our architecture is appropriate for its intended purpose?

In a large development project, an enormous amount of money may be riding on the architecture.

The company’s future may be at stake.

We need to *evaluate* the architecture.

How can we do this?

The SEI has developed the Architecture Tradeoff Analysis Method (ATAM).

The purpose of ATAM is: *to assess the consequences of architectural decisions in light of quality attribute requirements and business goals*.

ATAM Benefits

There are a number of benefits from performing ATAM evaluations

- identified risks
- clarified quality attribute requirements
- improved architecture documentation
- documented basis for architectural decisions
- increased communication among stakeholders

The results are improved architectures.

ATAM Steps

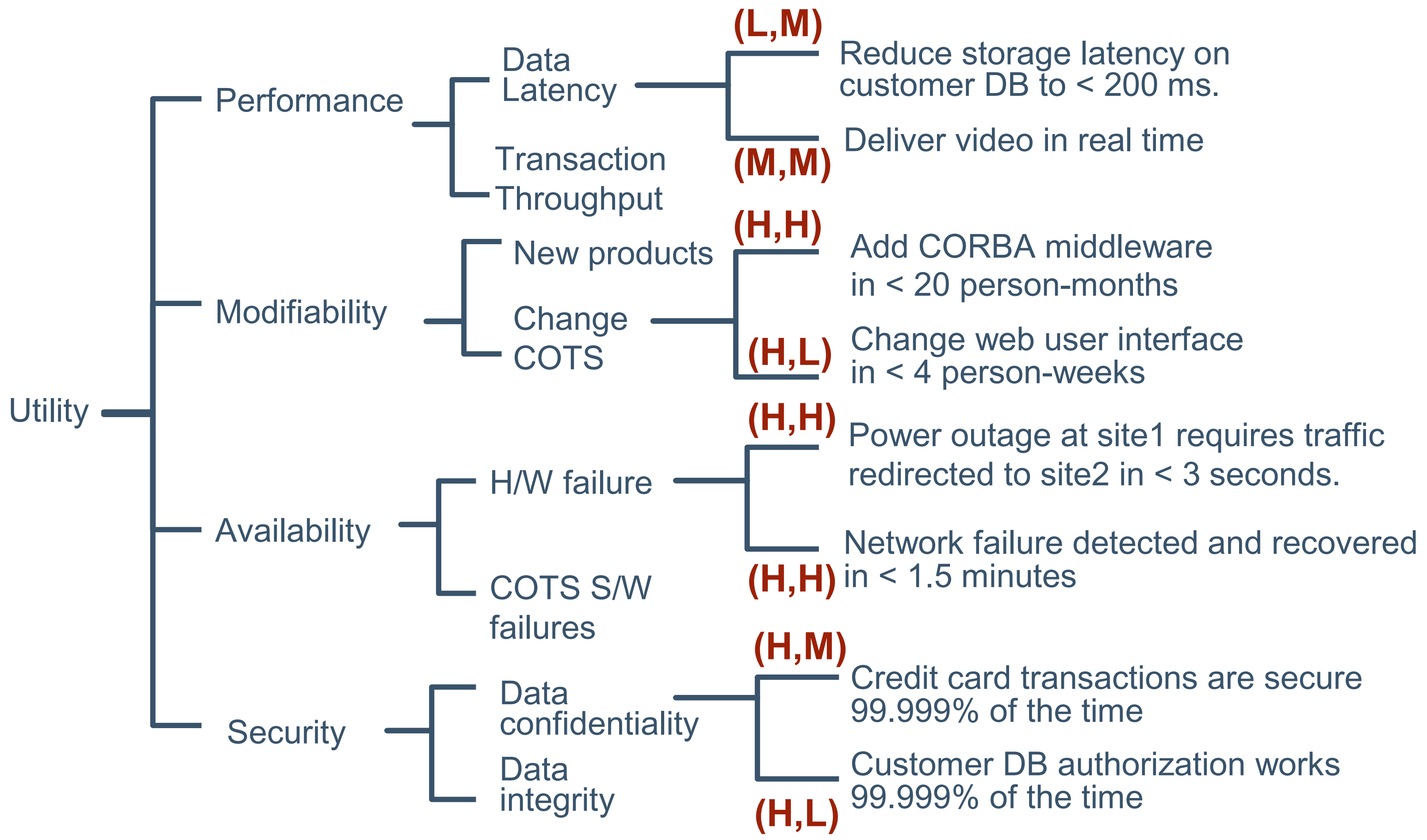
1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches

Phase 1

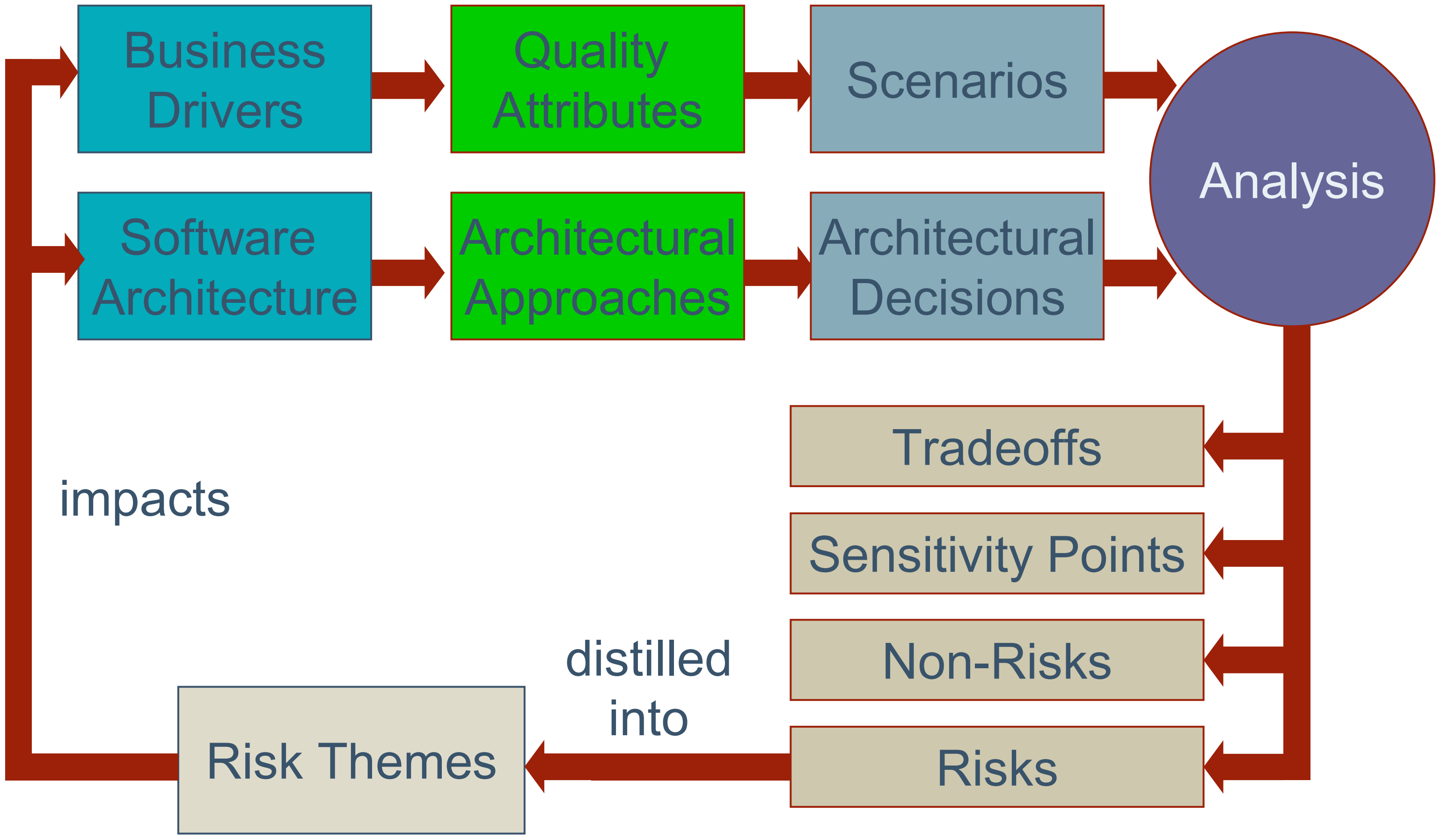
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results

Phase 2

Utility Tree Construction


Utility
Performance
Data Latency
Transaction Throughput
Modifiability
New products
Change COTS
Availability
H/W failure
COTS S/W failures
Security
Data confidentiality
Data integrity
(L,M)
Reduce storage latency on customer DB to < 200 ms.
(M,M)
Deliver video in real time
(H,H)
Add CORBA middleware in < 20 person-months
(H,L)
Change web user interface in < 4 person-weeks
(H,H)
Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
(H,H)
Network failure detected and recovered in < 1.5 minutes
(H,M)
Credit card transactions are secure 99.999% of the time
(H,L)
Customer DB authorization works 99.999% of the time

Conceptual Flow of ATAM



Documenting an architecture

Architecture serves as the blueprint for the system, and the project that develops it.

- It defines the work assignments.
- It is the primary carrier of quality attributes.
- It is the best artifact for early analysis.
- It is the key to post-deployment maintenance and mining.

Documenting the architecture is the crowning step to creating it.

Documentation speaks for the architect, today and 20 years from today.

What’s the answer?

“How do you document a software architecture?”

In industry, there seems to be a lack of systematic approaches to documentation. Instead, the emphasis has been on languages.

In the past, the answer seems to have been:

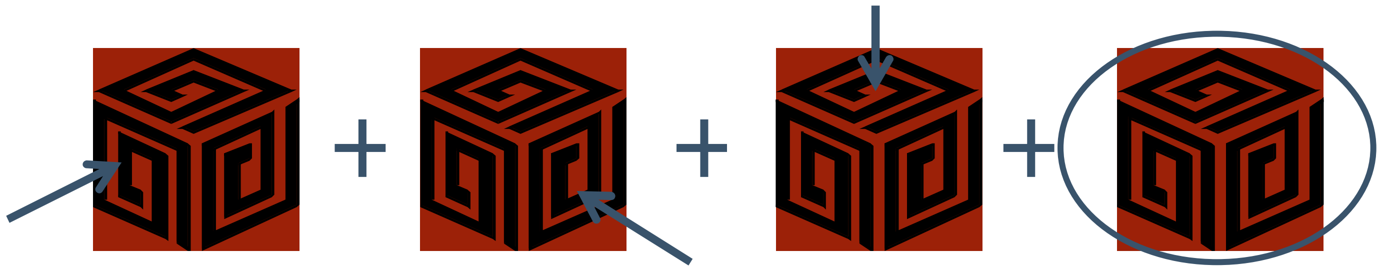
- “Use UML.”
- “Draw boxes and lines.”
- “What else do I need besides my class diagrams in Rose?”
- “Not very well.”
- “How do you document a *what*?”

Now, however, we have a much better answer.

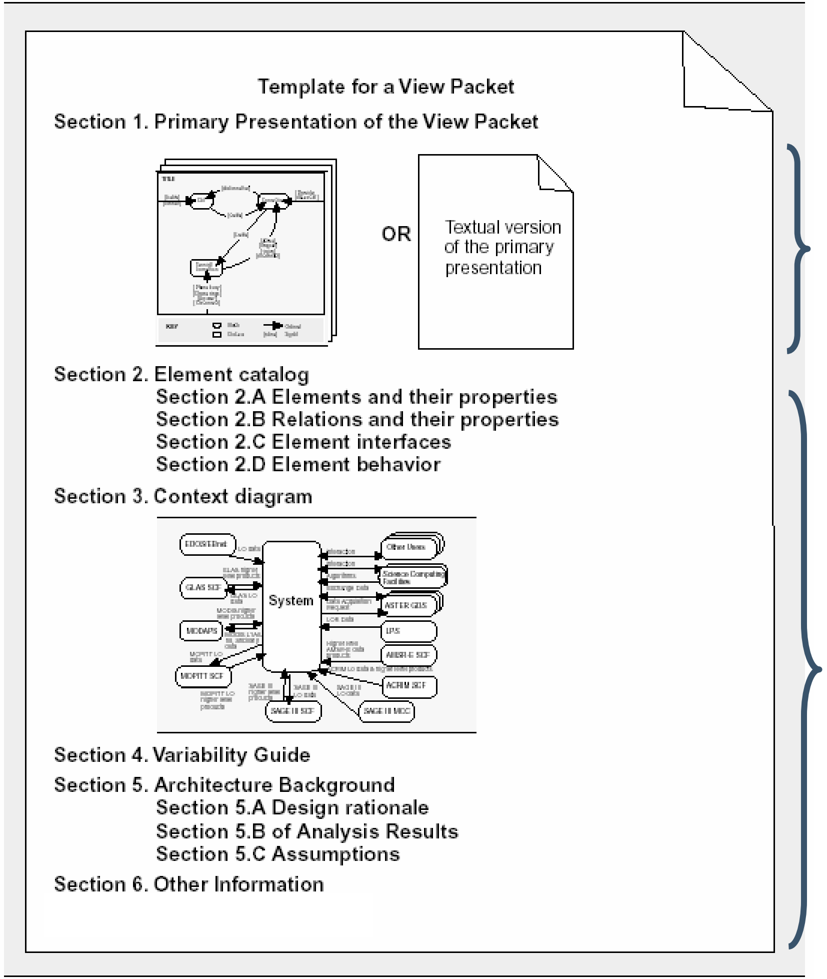
“Views and Beyond” approach to architecture documentation

The concept of a “view” gives us our main principle of architecture documentation:

*Document the relevant views,
and then add information
that applies to more than one view,
thus tying the views together.*

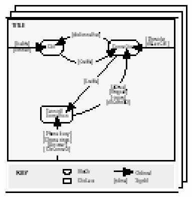


Summary: Documenting a View



Template for a View Packet

Section 1. Primary Presentation of the View Packet

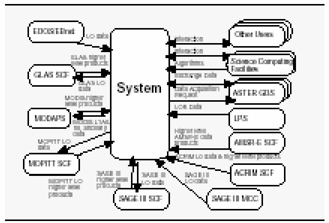


OR

Textual version of the primary presentation

Section 2. Element catalog
- Section 2.A Elements and their properties
- Section 2.B Relations and their properties
- Section 2.C Element interfaces
- Section 2.D Element behavior

Section 3. Context diagram



Section 4. Variability Guide

Section 5. Architecture Background
- Section 5.A Design rationale
- Section 5.B of Analysis Results
- Section 5.C Assumptions

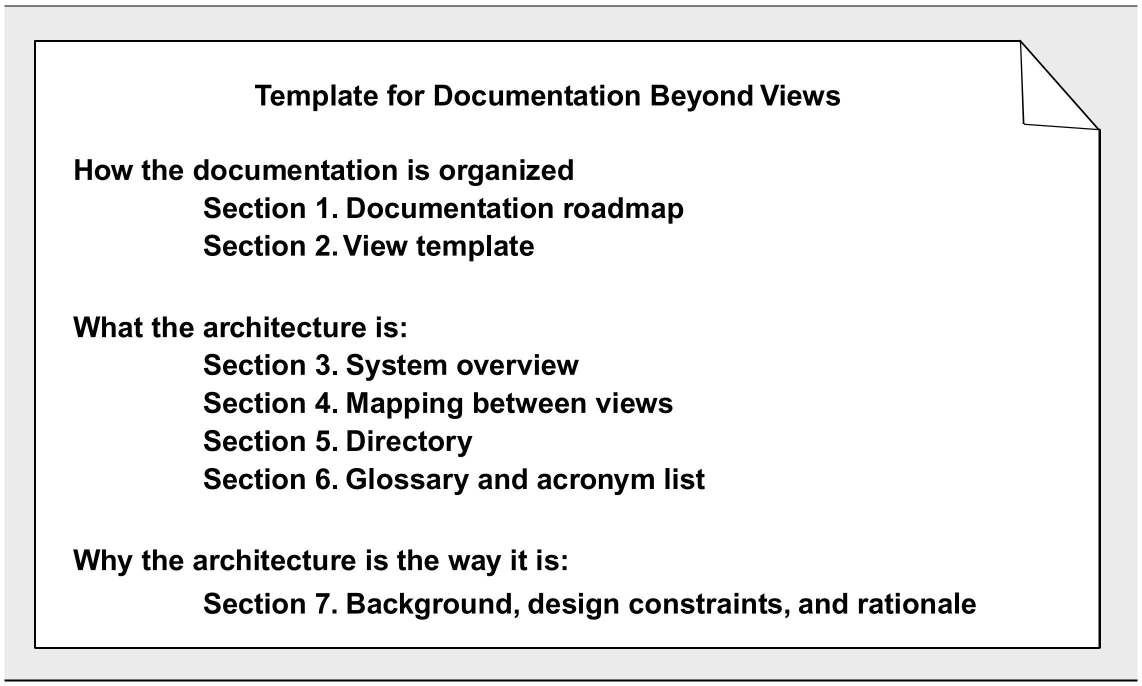
Section 6. Other Information

Section 1: Primary presentation

Sections 2-6: Supporting documentation

Summary: Documentation Beyond Views

A template for putting Documentation Beyond Views in a volume of its own:



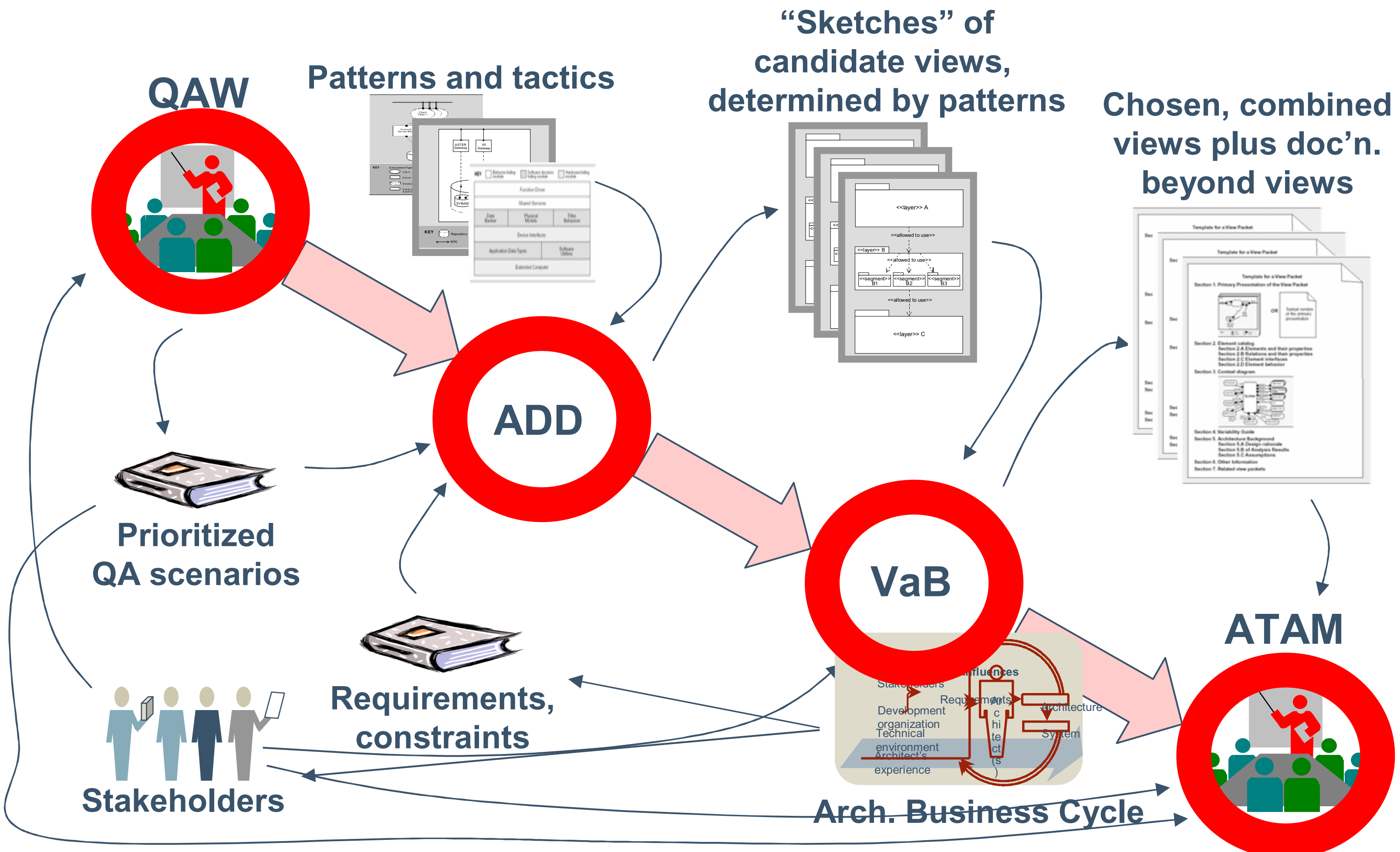
Template for Documentation Beyond Views

How the documentation is organized
- **Section 1. Documentation roadmap**
- **Section 2. View template**

What the architecture is:
- **Section 3. System overview**
- **Section 4. Mapping between views**
- **Section 5. Directory**
- **Section 6. Glossary and acronym list**

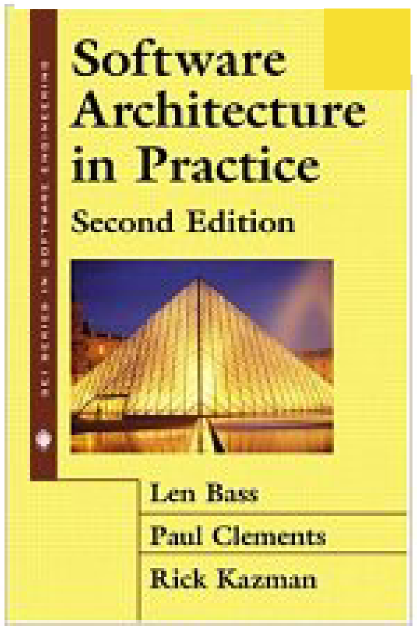
Why the architecture is the way it is:
- **Section 7. Background, design constraints, and rationale**

A Picture of Architecture-Based Dev.

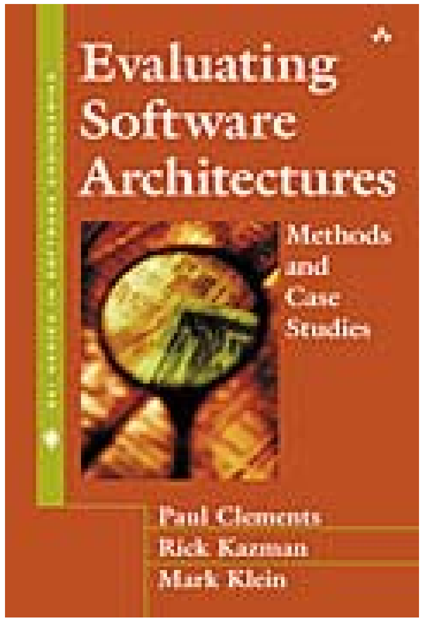


E-mail: clements@sei.cmu.edu

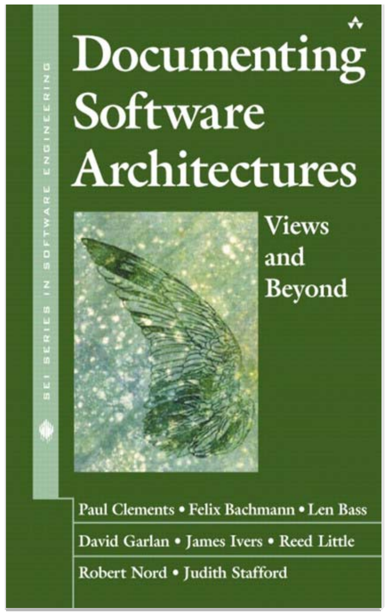
Source of material



Software Architecture in in Practice,
Len Bass, Paul Clements, Rick Kazman, Addison Wesley 2003

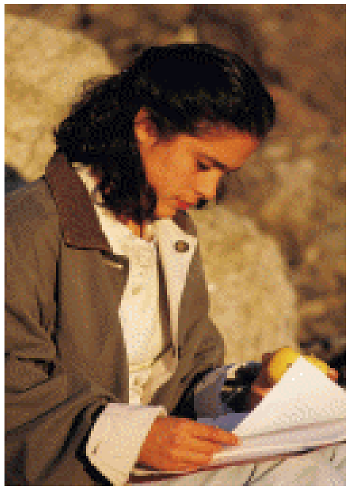


Evaluating Software Architectures: Methods and Case Studies,
Paul Clements, Rick Kazman, Mark Klein, Addison Wesley 2001


Documenting Software Architectures
Views and Beyond
Paul Clements • Felix Bachmann • Len Bass
David Garlan • James Ivers • Reed Little
Robert Nord • Judith Stafford

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

New project: Improving Software Architecture Competence



Architectures are created by *architects*.

- How can we help them do their best work?
- What does it mean for an architect to be competent?
- How can an architect improve his/her competence?

Architects work in *organizations*.

- How can we help an organization help their architects do their best work?
- What does it mean for an organization that produces architectures to be competent?
- How can an organization improve its competence in architecture?

Carnegie Mellon
Software Engineering Institute

What do architects do?

To understand how to help architects do what they do, we need to understand what they do.

- What are their duties?
- What skills and knowledge made them “capable of performing their allotted or required function?”

Philippe Kruchten writes that he requires architects working for him to spend 50% of their time on the architecture.

What do they do with the *other* 50%?

We can survey the “community”

Sources of information

- “Broadcast” sources: Information written by self-styled experts for mass anonymous consumptions
  - Web sites: e.g., Bredemeyer, SEI, HP, IBM (16)
  - Blogs and essays (16)
  - “Duties” list on SEI web site
  - Books on software architecture (25 top-sellers)
- Education and training sources:
  - University courses in software architecture (29)
  - Industrial/non-university public courses (22)
  - Certificate and certification programs in architecture; e.g., SEI, Open Group, Microsoft (7)
- “Architecture for a living” sources
  - Position descriptions for software architects (60)
  - Résumés of software architects

Survey results to date

This is a work in progress.

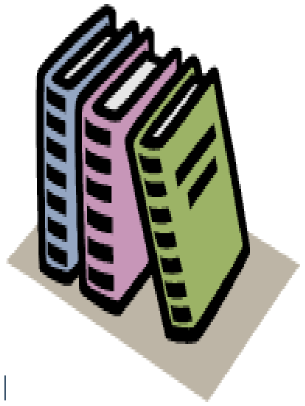
To date, we have surveyed over 200 sources.

We have cataloged

- 201 duties
- 85 skills
- 96 knowledge areas

Carnegie Mellon
Software Engineering Institute

Example duties

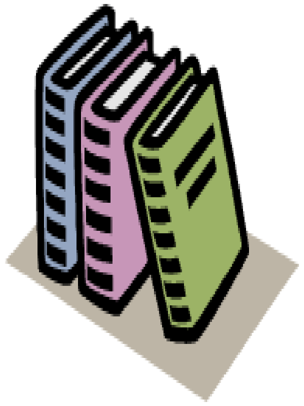


<snip>

- Duties related to documentation
  - Thoroughly understand and document the a (domains) for which the system will be buil
  - Prepare architectural documents and presentations
  - Document software interfaces
  - Produce a comprehensive documentation package for architecture useful to stakeholders
  - Keeping reader's point of view in mind while documentation
  - Creating, standardizing and using architectural descriptions
  - Use a certain documentation standard
  - Document variability and dynamism
  - Create conceptual architectural view

<snip>

Example skills

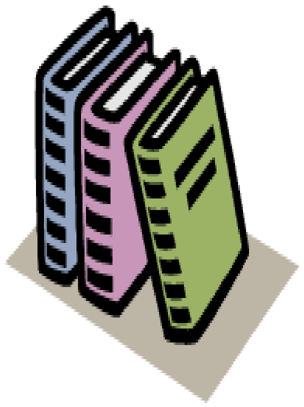


<snip>

- Inspire creative collaboration
- Interpersonal skills
- Interviewing
- Investigative
- Leadership
- Learning
- Listening skills
- Maintains constructive working relationships
- Mentoring
- Negotiation skills
- Observation power
- Open minded
- Oral and written communication skills
- Organizational and workflow skills
- Patient
- Planning skills
- Political sagacity

<snip>

Example knowledge



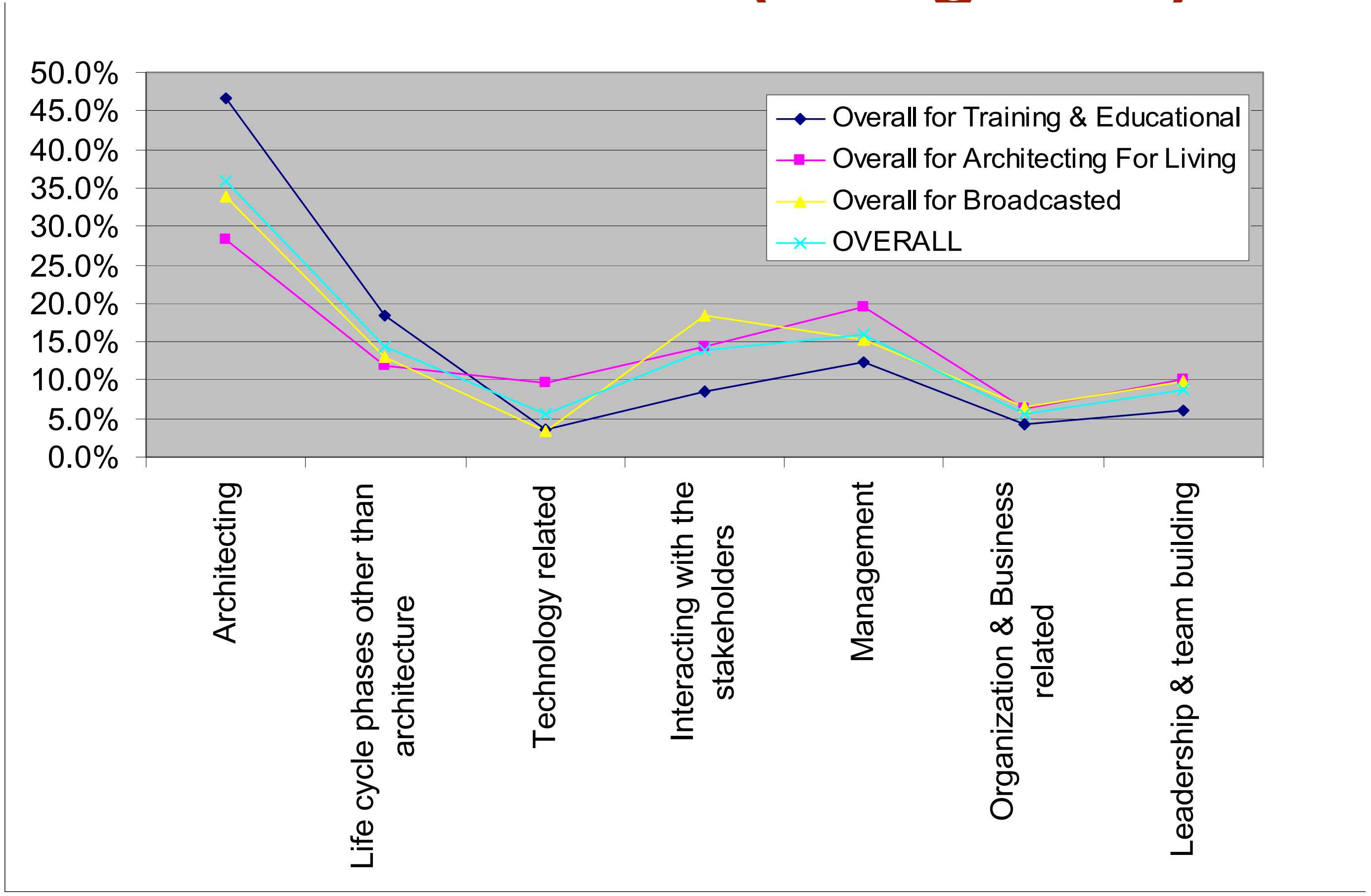
<snip>

Software Architecture concepts
UML diagrams and UML analysis modeling
Basic knowledge of Software Engineering
Specialized knowledge of software engineering
Knowledge about IT industry future directions
Understanding of web-based applications
Experience with Web Services Technologies
Business re-engineering principles and processes
Knowledge of industry's best practices
Experience in testing
Knowledge of testing/debugging tools
Experience with Real-time systems, Video systems
Security domain Experience

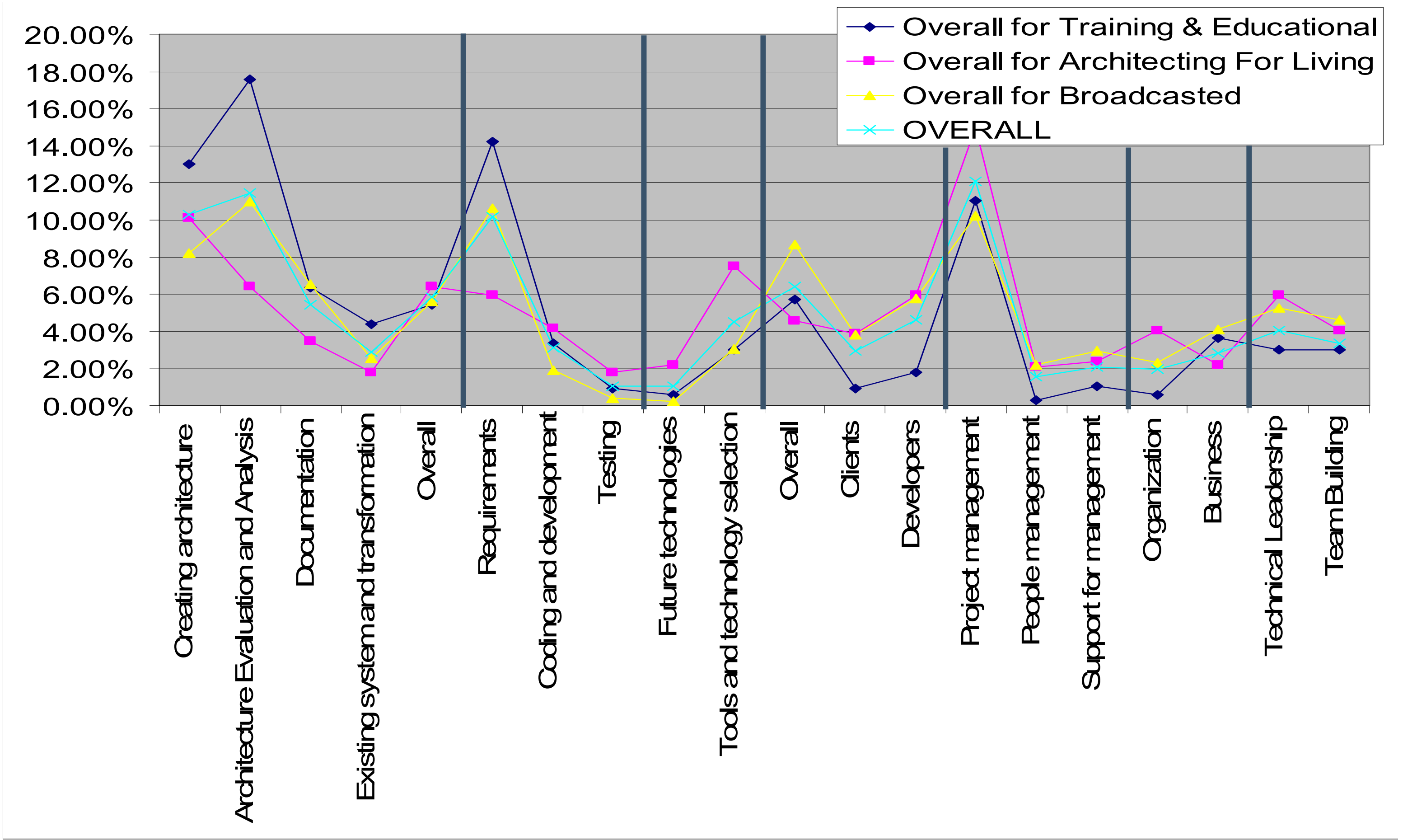
<snip>

Carnegie Mellon
Software Engineering Institute

Architecture Duties (categories)

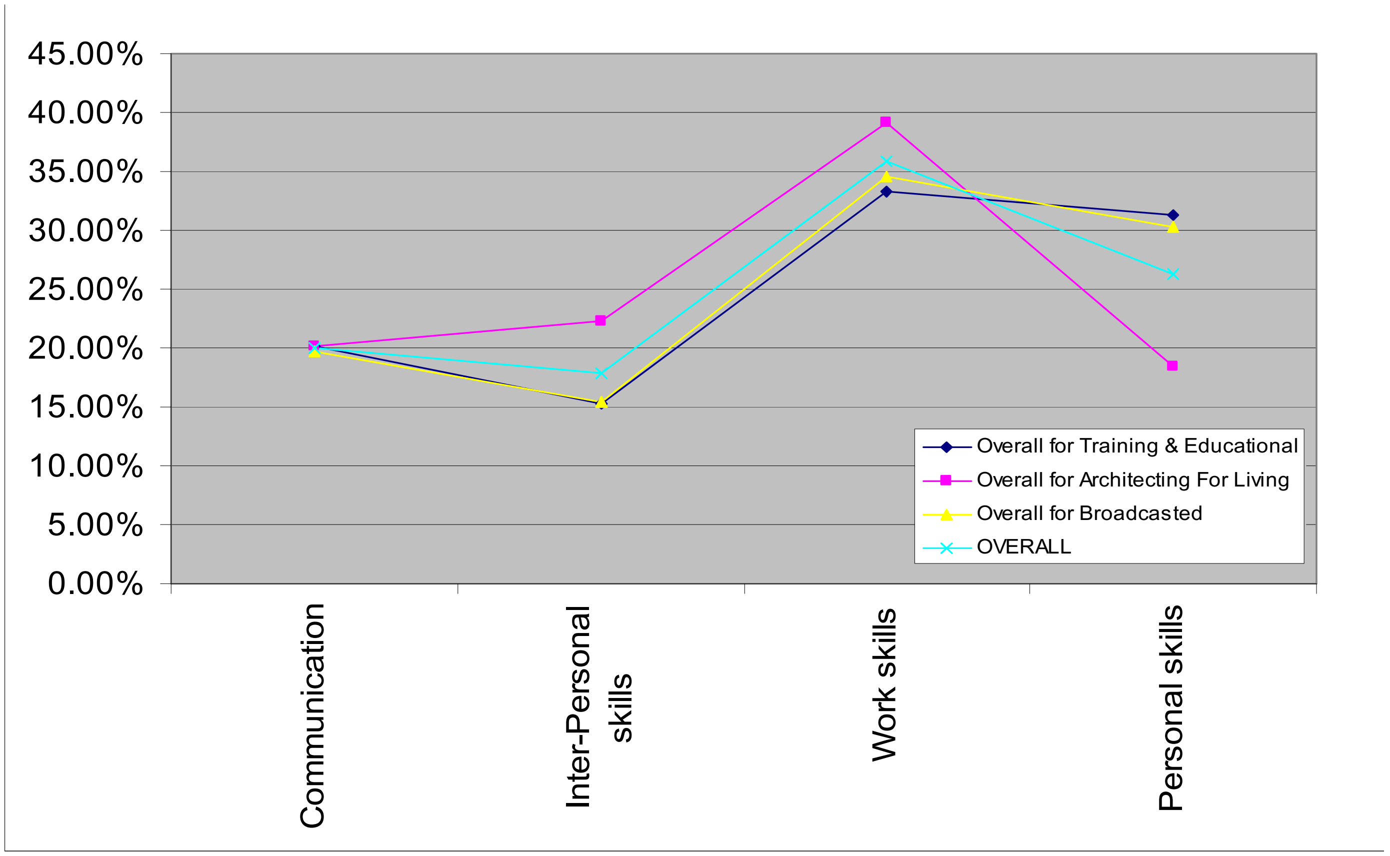


Architecture Duties (sub-categories)



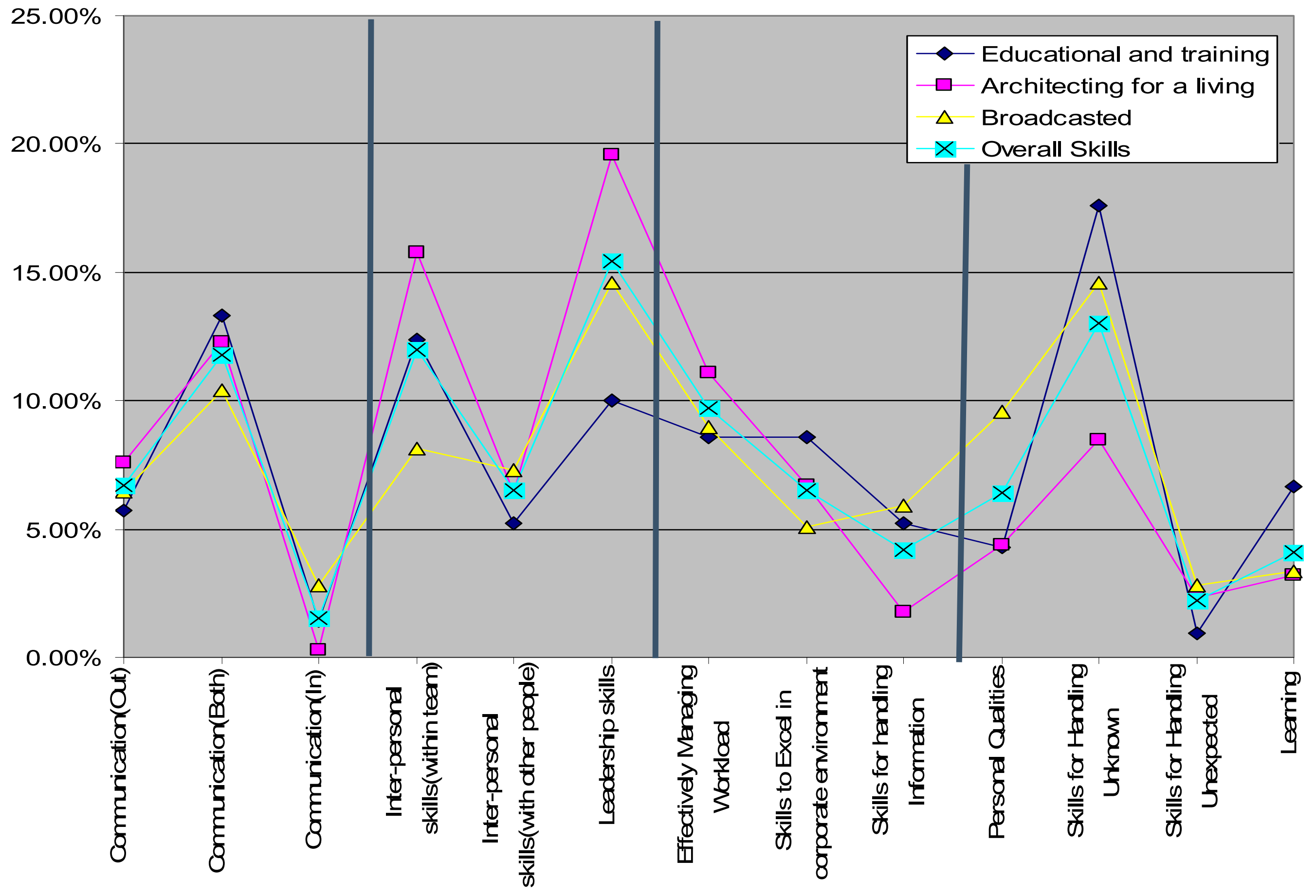
Carnegie Mellon
Software Engineering Institute

Architecture Skills (categories)

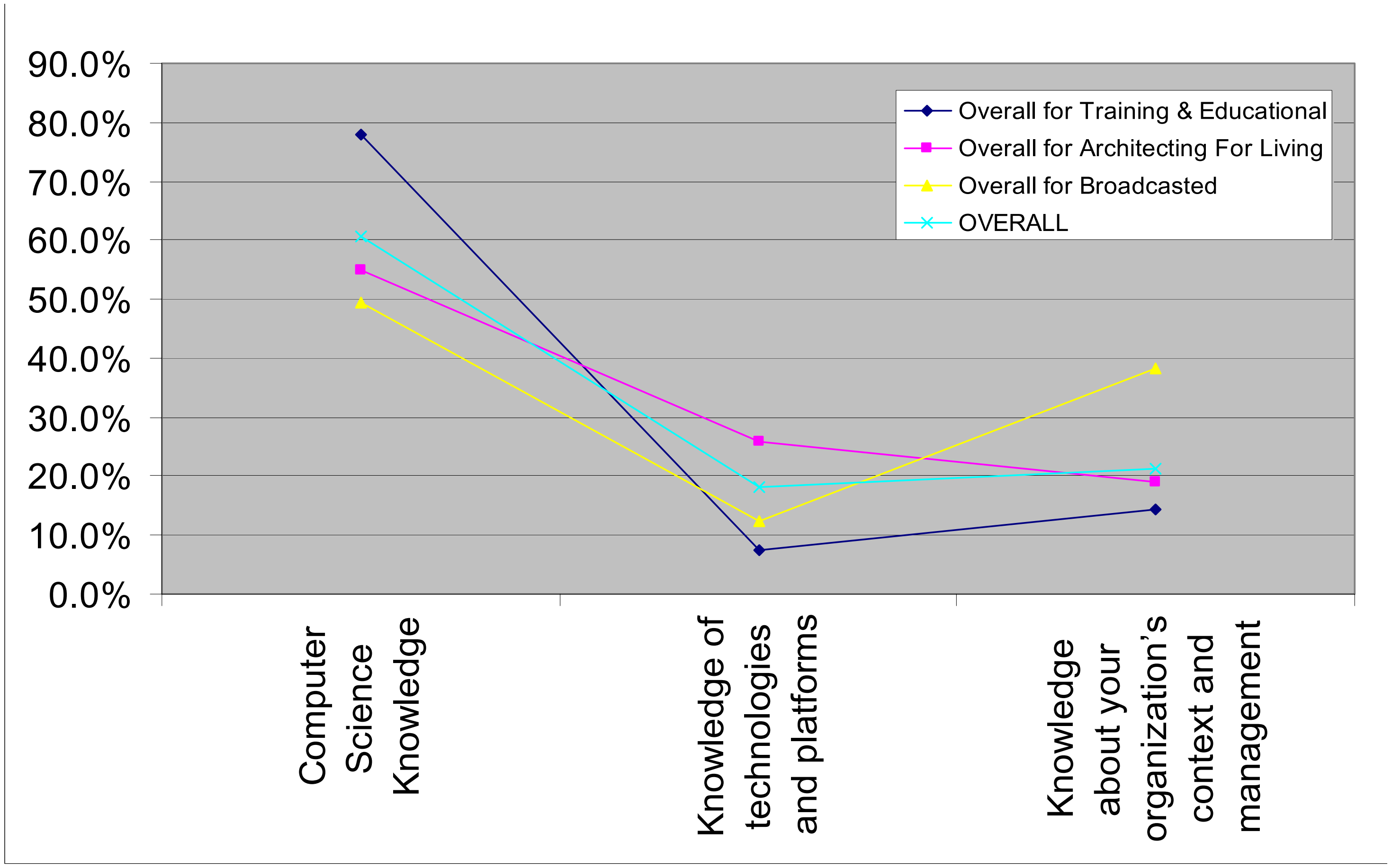


Carnegie Mellon
Software Engineering Institute

Architecture Skills (sub-categories)

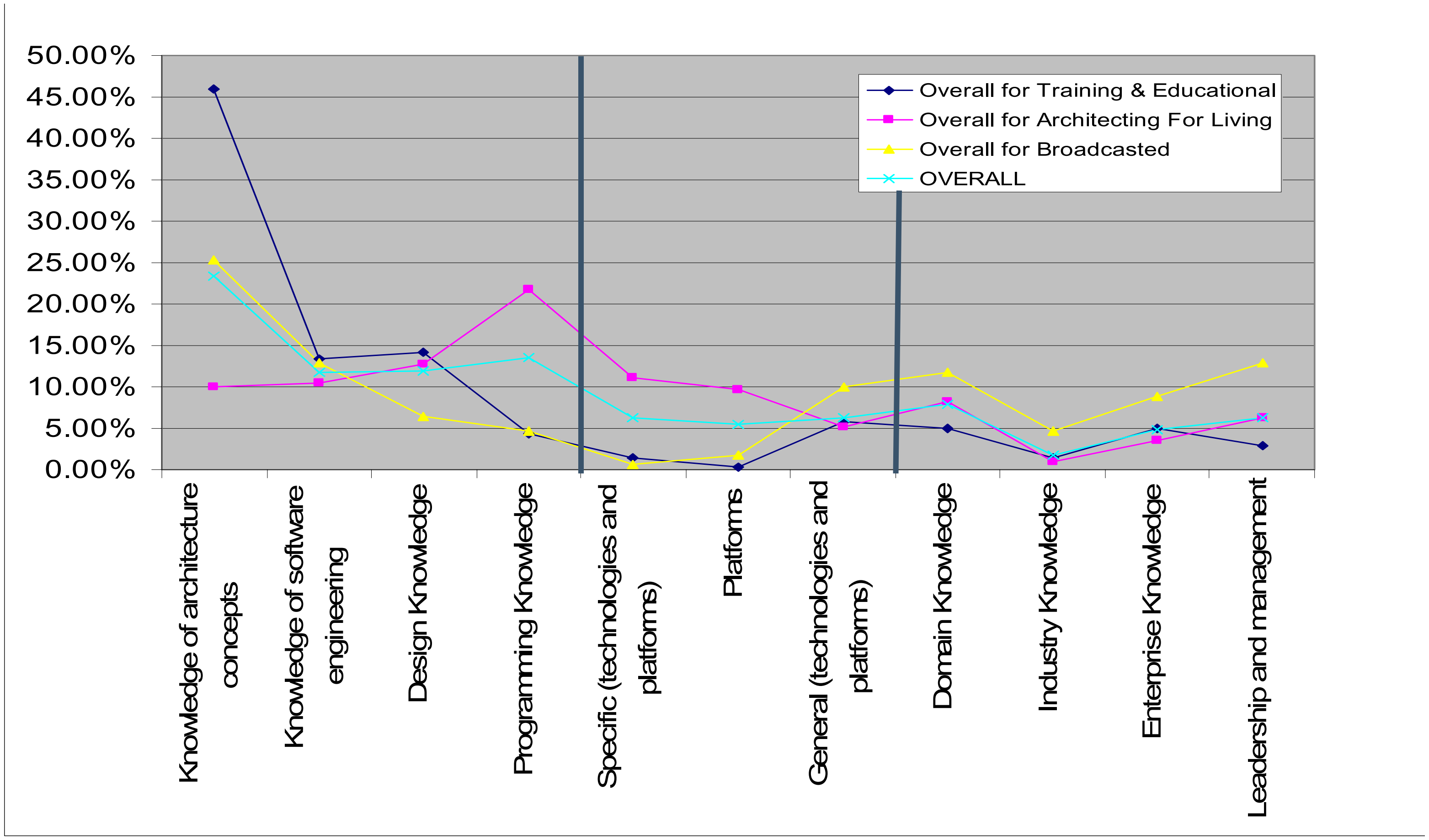


Architecture Knowledge (categories)



Carnegie Mellon
Software Engineering Institute

Architecture Knowledge (sub-cat's)



Duties/Skills/Knowledge

This work lets us propose a "duties/skills/knowledge" model of competence.

Knowledge and *skills* support carrying out the *duties*.

Competence is
- Carrying out the duties
- Having the skills
- Knowing the knowledge

Duties/Skills/Knowledge

Advantages

- It applies equally well to individuals, teams, and organizations.

- It straightforwardly suggests an assessment instrument.

- It straightforwardly suggests an improvement strategy
  - Improve your duties
  - Improve your skills
  - Improve your knowledge

Future work (1): Grow this body of work

Survey communities of practicing architects

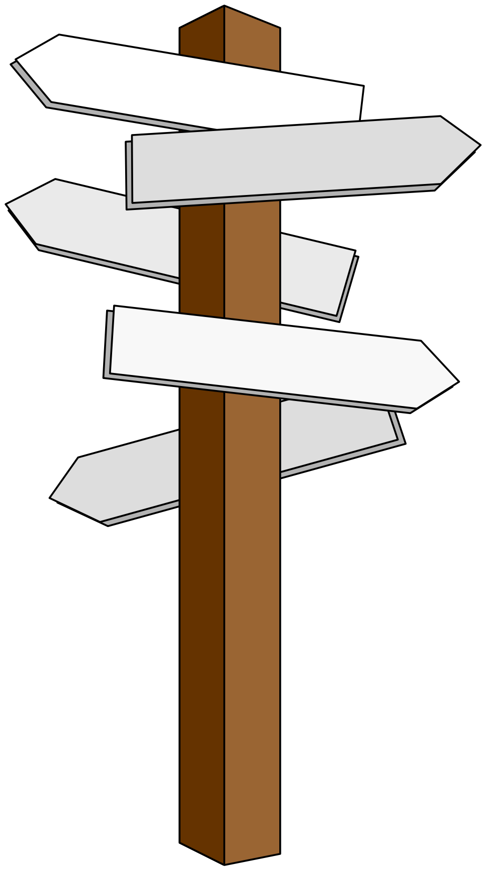
- E.g., WWISA, IASA, architects within a company
- First questionnaire: 15-minute survey of duties, skills, and knowledge *and* organizational duties.

Tie specific skills and knowledge to duties

- If knowledge or a skill doesn't support a duty, does it matter?

Survey more sources

- especially more position descriptions



Future work (2): Apply model to organizations

Case studies and surveys

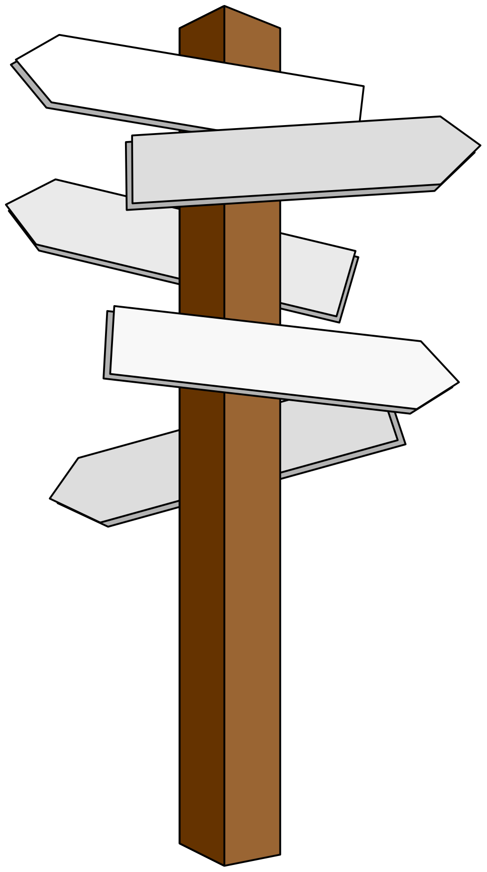
- organizational excellence in architecture
- organizational improvement in architecture
- surveys of organizational practices

Surveys of practicing architects

- E.g., WWISA, IASA, architects within a company
- First questionnaire: 15-minute survey of duties, skills, and knowledge *and* organizational duties.

Investigation of team practices

Assessment of past performance to find targeted areas of improvement

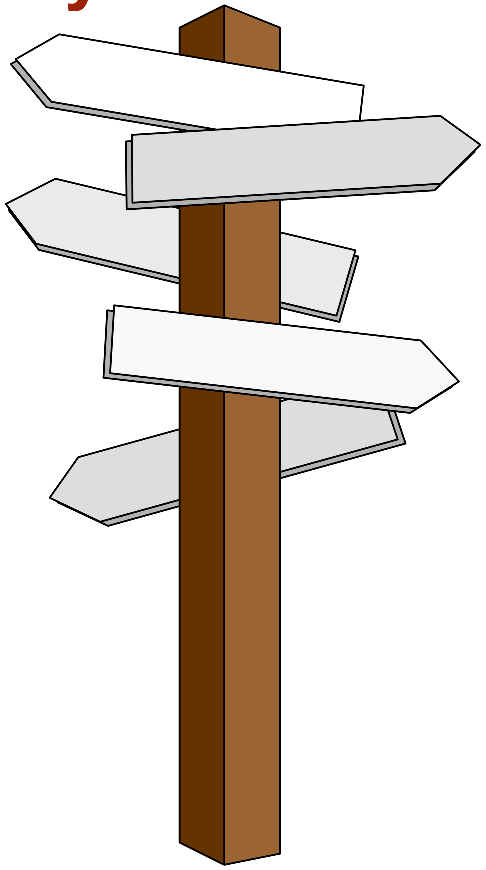


What are an *organization's* duties, skills, and knowledge?

List may include:

- Hire talented architects
- Establish a career track for software architects
- Make the position of architect highly regarded through visibility, reward, and prestige
- Establish a clear statement of duties, responsibilities, and authority for software architects
- Establish a mentoring program for architects
- Start an architecture training and education program
- Track how architects spend their time
- Establish an architect certification program
- Measure architects' performance
- Provide a forum for architects to communicate, and share information and experience
- Put in a place organization-wide development practices centered around architecture
- Establish and empower an architecture review board
- Measure quality of architectures produced
- Initiate software process improvement or software quality improvement practices

Future work (3): Tie duties, skills, and knowledge to architecture quality



Case studies of successful and failed architectures, relating cause to effect.

Pilot assessment instruments

Pilot improvement strategies

Other models of competence

- Organizations as architectures: They have elements and relations and behaviors. Perhaps we can "evaluate" them as we evaluate architectures.
- Design for Six Sigma techniques

Research collaborators wanted!

Trends in Software Architecture

Predictions are risky, and usually worth less than what you paid for them.

"Basic building blocks" of software

- Will continue becoming more sophisticated, complex, domain-specific, interoperable, and stand-alone (continuing a 40-year-old trend)
- "Service" is the current form of this, but will be replaced by something else in five years
- SLAs will be come more sophisticated, generalized, and dependable. "Credentials" will be the watchword, especially in services. Watch for an ebay-like model where consumers leave feedback, especially in ubiquitous computing environments.

Trends in Software Architecture

Process of architecting

- Will be come more standardized, more repeatable, more teachable, more methodical – less “magic”
- Evaluation of architectures will continue becoming a widespread practice
- Stakeholder-based documentation will become the norm
- Gaps in the conceptual representations (e.g., between ADLs and downstream design languages such as UML, or between business goals and architecture) will be bridged
- More automated traceability, from business processes/goals to architecture to design to code to testing, possibly by automated design assistants and tooling, possibly by better language support

Trends in Software Architecture

Architecture as a practice

- We can view the history of software engineering as producing simpler ways to specify much more complex software. Our *languages* to describe solutions have become steadily more sophisticated.
  - 1960 atoms: + - / * SQRT, simply-structured reports
  - 2006 atoms: shopping cart, GUI, rules engine, workflow, auction…
- When our languages achieve a “new plateau” of expressiveness, the programs we write suddenly become very simple, even though the software systems are much more complex. The complexity is carried in the language.
- The need for architecture is not as great when the programs are very short and simple.
- But we always learn to exploit our new capabilities to the fullest. Our programs quickly become more and more complex. Soon, we cannot understand them, nor can we understand how they will deliver the ever-higher quality attributes we require.
- And that is the point where architecture steps in to help us structure the solutions and lend understandability and buildability.

Trends in Software Architecture

Architecture as a practice (continued)

- Right now there seem to be two kinds of organizations.
- One kind designs extensive architectures.
  - These are application builders solving unprecedented problems, or in domains without large “platform” vendors (they may *be* the platform vendors) or standardized solutions
  - They perform extensive architectural design
- The other kind makes major architectural decisions *by the process of selecting a platform vendor.*
  - Here, “architecture” means putting big vendor-supplied pieces together. “Design” is de-emphasized.
  - The “space” of what they can specify is not that large.
  - You can view the things they get to choose as a specification language.
  - They’ve just arrived at the latest “new plateau”
  - They don’t view architecture design as that important.
  - Last prediction: They will!

Carnegie Mellon
Software Engineering Institute

Questions—Now or Later

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

U.S. Mail:
Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-3890

World Wide Web:
http://www.sei.cmu.edu/ architecture

SEI Fax: 412-268-5758

Paul Clements
Email: clements@sei.cmu.edu

A Microsoft Word template for a software architecture document based on the Views and Beyond approach is available at

http://www.sei.cmu.edu/architecture/arch_doc.html

or

http://www.sei.cmu.edu/architecture
Click on documentation
Click on download