Current Best Practices in Software Architecture

Session 4: Evaluating Software Architectures

Paul Clements
Software Engineering Institute
Carnegie Mellon University
13 October 2005

Sponsored by the U.S. Department of Defense
© 2005 by Carnegie Mellon University
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

Large-scale design decisions cannot be made by programmers.
• Have limited visibility and short-term perspectives
• Trained in technology solutions to specific problems.

A software architecture
• Exists to achieve a system’s quality attributes
• Exists to allow parallel development by distributed teams (a special kind of quality attribute)
• Involves decomposing a whole into parts
• Involves system-wide design decisions such as
  - How the parts work together to achieve the system’s function and goals
Review: From Session 1

Definition: 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.

Architectures are created by engineering these software structures in way to achieve quality attributes.
Review: From Session 1

Module
- Uses
- Decomposition
- Class/Generalization
- Layers

Component-and-Connector
- Client-Server
- Concurrency
- Shared-Data
- Process

Allocation
- Work Assignment
- Deployment
- Implementation
Review: From Session 2

We need help capturing and expressing quality attributes. Quality attribute scenarios help.

Quality attributes come from stakeholders. Use a Quality Attribute Workshop to elicit them.

Other influences on the architecture are at work also:
- Developing organization
- Technical environment
- Architect’s experience

The architect must recognize and capture these.

Organizations must recognize that an architecture can influence these very factors: An Architecture Business Cycle exists.
Architecture Business Cycle (ABC)
Review: From Session 3

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.
Review: From Session 3

Authors such as Shaw and Garlan wrote “style catalogs”

Independent component patterns
- communicating-processes
- event systems
  - implicit invocation
  - explicit invocation

Data flow patterns
- batch sequential
- pipe-and-filter
- layers

Data-centered patterns
- blackboard
- repository

Virtual machine patterns
- interpreters
- rule-based systems

Call-return patterns
- main program and subroutine
- object oriented
Review: From Session 3

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.
Summary of Availability Tactics

- Fault Detection
  - Ping/Echo
  - Heartbeat
  - Exception

- Fault Recovery Preparation and Repair
  - Voting
  - Active Redundancy
  - Passive Redundancy
  - Spare

- Fault Recovery and Reintroduction
  - Shadow
  - State Resynchronization
  - Rollback

- Fault Prevention
  - Removal From Service
  - Transactions
  - Process Monitor

Fault masked or repair made
Summary of Modifiability Tactics

Stimulus: Change arrives

Response: Changes made, tested, and deployed within time and budget

Modifiability

- Localize Changes
  - Semantic coherence
  - Anticipate expected changes
  - Generalize module
  - Limit possible options
  - Abstract common services

- Prevention of Ripple Effect
  - Hide information
  - Maintain existing interface
  - Restrict communication paths
  - Use an intermediary

- Defer Binding Time
  - Runtime registration
  - Configuration files
  - Polymorphism
  - Component replacement
  - Adherence to defined protocols
Tactics for Performance

- **Stimulus:** Events arrive
- **Response:** Response generated within time constraints

**Performance**
- **Resource demand**
  - Increase computation efficiency
  - Reduce computational overhead
  - Manage event rate
  - Control freq. of sampling

- **Resource management**
  - Introduce concurrency
  - Maintain multiple copies
  - Increase available resources

- **Resource arbitration**
  - Scheduling policy

© 2005 by Carnegie Mellon University
Tactics for Security

Stimulus: Attack

Response: System detects, resists, or recovers from attacks

Security

Resisting Attacks
- Authenticate users
- Authorize users
- Maintain data confidentiality
- Maintain integrity
- Limit exposure
- Limit access

Detecting Attacks
- Intrusion detection

Recovering from an attack
- Restoration
- Identification
  - See “Availability”
  - Audit trail

Identification

Audit trail

See “Availability”
Tactics for Testability

Stimulus: Completion of an increment

Internal monitoring
- Manage Input/Output
- Record/playback
- Separate interface from implementation
- Specialized access routines/interfaces

Response: Faults detected
- Built-in monitors
Review: From Session 3

The Attribute-Driven Design method (ADD) is a step-by-step method for systematically producing the first architectural designs for a system.

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
A Picture of Architecture-Based Development

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

This seminar series is illuminating the usual parts of that process.

Typically, the steps include
- Analyze the business case
- Understand the architecturally significant requirements
- Create an architecture to satisfy those requirements
A Picture of Architecture-Based Development -2

We now have tools in hand to carry out these steps.

- Architecture Business Cycle (ABC) – helps us identify business case factors that will shape the architecture.
- Quality Attribute Workshop (QAW) – first way to engage the stakeholders.
- QA scenarios – the way to capture QA requirements.
- ADD – a method to design an architecture to meet its functional and QA requirements.
A Picture of Architecture-Based Dev.

Patterns and tactics

“Sketches” of candidate views, determined by patterns

Prioritized QA scenarios

Requirements

Stakeholders

QAW

ADD
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.
Purpose of the ATAM – 1

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

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

Tradeoffs can be explicitly identified and documented.
The purpose of the ATAM is **NOT** to provide precise analyses . . . the purpose **IS** to discover risks created by architectural decisions.

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

ATAM evaluations are conducted in four phases.

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

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

- **Phase 2**: Complete Evaluation

- **Phase 3**: Follow-up
  - Duration: varies
  - Meeting: primarily phone, email
ATAM Phase 0

Phase 0: This phase precedes the technical evaluation.

• The customer and a subset of the evaluation team exchange understanding about the method and the system whose architecture is to be evaluated.
• An agreement to perform the evaluation is worked out.
• A core evaluation team is fielded.
ATAM Phase 1

Phase 1: involves a small group of predominantly technically-oriented stakeholders

Phase 1 is

• architecture centric
• focused on eliciting detailed architectural information and analyzing it
• top down analysis
ATAM Phase 1 Steps

1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results
1. Present the ATAM

The evaluation team presents an overview of the ATAM including:

• ATAM steps in brief
• Techniques
  - utility tree generation
  - architecture elicitation and analysis
  - scenario brainstorming/mapping
• Outputs
  - architectural approaches
  - utility tree and scenarios
  - risks, non-risks, sensitivity points, and tradeoffs
2. Present Business Drivers

ATAM customer representative describes the system’s business drivers including:

- business context for the system
- high-level functional requirements
- high-level quality attribute requirements
  - architectural drivers: quality attributes that “shape” the architecture
  - critical requirements: quality attributes most central to the system’s success
3. Present Architecture

Architect presents an overview of the architecture including:

• technical constraints such as an OS, hardware, or middleware prescribed for use
• other systems with which the system must interact
• architectural approaches used to address quality attribute requirements

Evaluation team begins probing for and capturing risks.
Identify predominant architectural approaches such as

- client-server
- 3-tier
- watchdog
- publish-subscribe
- redundant hardware

The evaluators begin to identify places in the architecture that are key to realizing quality attribute goals.
Identify, prioritize, and refine the most important quality attribute goals by building a utility tree.

- A utility tree is a top-down vehicle for characterizing and prioritizing the “driving” attribute-specific requirements.
- The driving quality attributes are the high-level nodes (typically performance, modifiability, security, and availability).
- Scenarios are the leaves of the utility tree.

Output: a characterization and a prioritization of specific quality attribute requirements.
Utility Tree Construction

Performance
- Data Latency
- Transaction Throughput
- New products
- Change COTS

Modifiability

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

Security
- Data confidentiality
- Data integrity

Utility

(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
Scenarios

Scenarios are used to

• represent stakeholders’ interests
• understand quality attribute requirements

Scenarios should cover a range of

• use case scenarios: anticipated uses of the system
• growth scenarios: anticipated changes to the system
• exploratory scenarios: unanticipated stresses to the system

A good scenario makes clear what the stimulus is that causes it and what responses are of interest.
Example Scenarios

Use case scenario
   Remote user requests a database report via the Web during peak period and receives it within 5 seconds.

Growth scenario
   Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week.

Exploratory scenario
   Half of the servers go down during normal operation without affecting overall system availability.

Scenarios should be as specific as possible.
Stimuli, Environment, Responses

Use Case Scenario
Remote user requests a database report via the Web during peak period and receives it within 5 seconds.

Growth Scenario
Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week.

Exploratory Scenario
Half of the servers go down during normal operation without affecting overall system availability.
6. Analyze Architectural Approaches

Evaluation team probes architectural approaches from the point of view of specific quality attributes to identify risks.

- identify the architectural approaches
- ask quality attribute specific questions for highest priority scenarios
- identify and record risks and non-risks, sensitivity points and tradeoffs
Quality Attribute Questions

Quality attribute questions probe architectural decisions that bear on quality attribute requirements.

Performance

• How are priorities assigned to processes?
• What are the message arrival rates?

Modifiability

• Are there any places where layers/facades are circumvented?
• What components rely on detailed knowledge of message formats?
Risks, Tradeoffs, Sensitivities, and Non-Risks

A risk is a potentially problematic architectural decision. Non-risks are good architectural decisions that are frequently implicit in the architecture.

A sensitivity point is a property of one or more components (and/or component relationships) that is critical for achieving a particular quality attribute response.

A tradeoff point is a property that affects more than one attribute and is a sensitivity point for more than one attribute.
Risks and Tradeoffs

Example Risk:

• “Rules for writing business logic modules in the second tier of your 3-tier architecture are not clearly articulated. This could result in replication of functionality thereby compromising modifiability of the third tier.”

Example Tradeoff:

• “Changing the level of encryption could have a significant impact on both security and performance.”
Sensitivity Points and Non-Risks

Example Sensitivity Point:

• “The average number of person-days of effort it takes to maintain a system might be sensitive to the degree of encapsulation of its communication protocols and file formats.”

Example Non-Risk:

• “Assuming message arrival rates of once per second, a processing time of less than 30 ms, and the existence of one higher priority process, a 1 second soft deadline seems reasonable.”
Phase 2: involves a larger group of stakeholders

Phase 2 is

- stakeholder centric
- focused on eliciting diverse stakeholder points of view and on verification of the Phase 1 results
ATAM Phase 2 Steps

1. Present the ATAM
2. Present business drivers
3. Present architecture
4. Identify architectural approaches
5. Generate quality attribute utility tree
6. Analyze architectural approaches
7. Brainstorm and prioritize scenarios
8. Analyze architectural approaches
9. Present results
7. Brainstorm and Prioritize Scenarios

Stakeholders generate scenarios using a facilitated brainstorming process.

- Scenarios at the leaves of the utility tree serve as examples to facilitate the step.

In phase 2, each stakeholder is allocated a number of votes roughly equal to 0.3 x #scenarios.
8. Analyze Architectural Approaches

Identify the architectural approaches impacted by the scenarios generated in the previous step.

- This step continues the analysis started in step 6 using the new scenarios.
- Continue identifying risks and non-risks.
- Continue annotating architectural information.
9. Present Results

Recapitulate all the steps of the ATAM and present the ATAM outputs, including

• architectural approaches
• utility tree
• scenarios
• risks and non-risks
• sensitivity points and tradeoffs
• risk themes
Phase 3: primarily involves producing a final report for the customer as well as reflecting upon the quality of the evaluation and the ATAM materials.
The Final Report

The evaluation team will typically create the final report which includes:

- Executive summary
- Description of ATAM
- Description of business drivers and architecture
- List of phase 1 and phase 2 scenarios and utility tree
- Phase 1 and phase 2 analysis: architectural approaches, decisions, risks, sensitivities, tradeoffs, and non-risks
- Risk themes
- Next steps
Summary

The ATAM is

• a method for evaluating an architecture with respect to multiple quality attributes
• an effective strategy for discovering the consequences of architectural decisions
• a method for identifying *trends*, not for performing precise analyses
A Picture of Architecture-Based Dev.

QAW

Patterns and tactics

“Sketches” of candidate views, determined by patterns

ADD

Prioritized QA scenarios

Requirements

ATAM

Stakeholders
Source of material

Evaluating Software Architectures: Methods and Case Studies

Paul Clements
Rick Kazman
Mark Klein

Addison Wesley 2001
Next time

Documenting software architectures:

How do we write down our architecture so that others can use it, understand it, and build a system from it?

Date: October 19