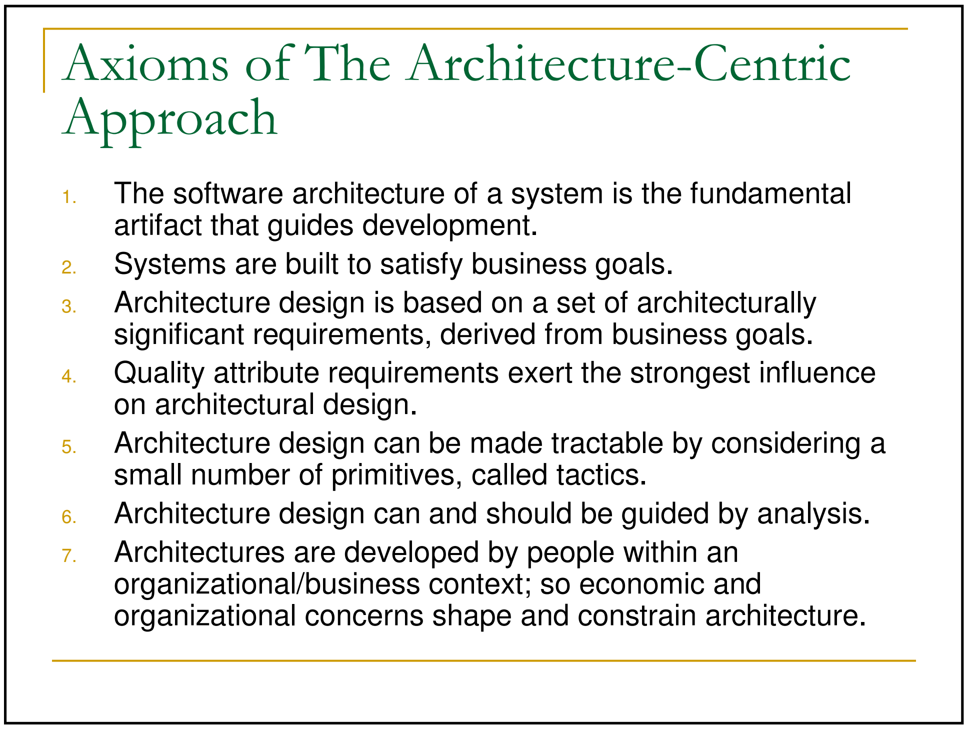
Analysis and Management of Software Architectures: *Design it Right, Build it Right*

Rick Kazman

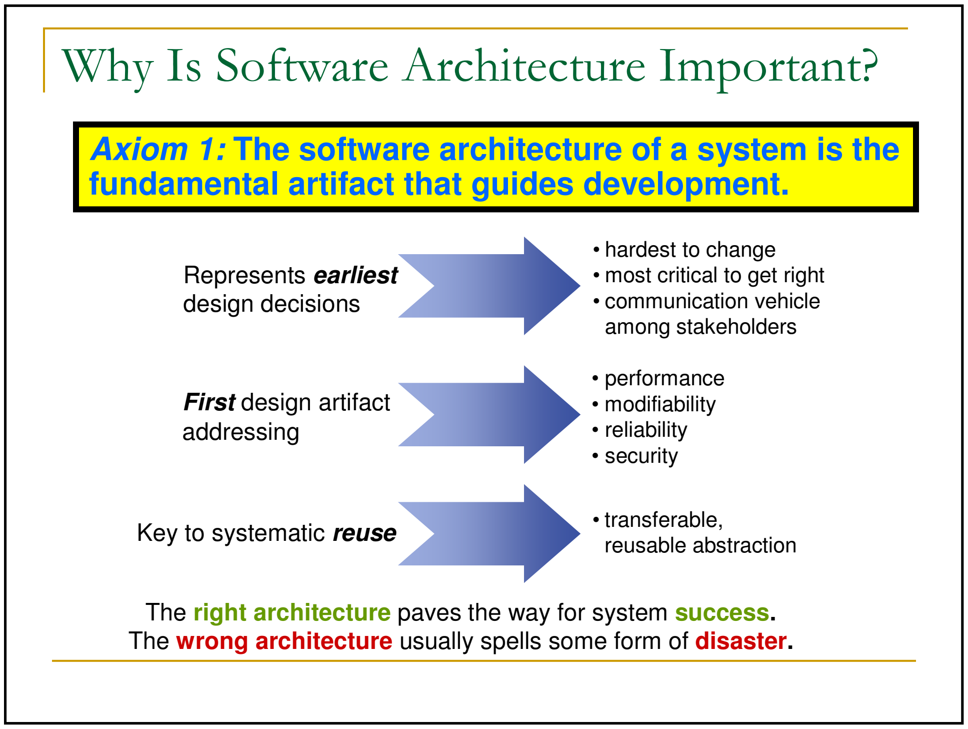
Rick.Kazman@gmail.com
Software Engineering Institute/CMU
and *University of Hawaii*

Part 1: Software Architecture Design



Axioms of The Architecture-Centric Approach

1. The software architecture of a system is the fundamental artifact that guides development.
2. Systems are built to satisfy business goals.
3. Architecture design is based on a set of architecturally significant requirements, derived from business goals.
4. Quality attribute requirements exert the strongest influence on architectural design.
5. Architecture design can be made tractable by considering a small number of primitives, called tactics.
6. Architecture design can and should be guided by analysis.
7. Architectures are developed by people within an organizational/business context; so economic and organizational concerns shape and constrain architecture.



Why Is Software Architecture Important?

Axiom 1: **The software architecture of a system is the fundamental artifact that guides development.**

Represents ***earliest*** design decisions
- hardest to change
- most critical to get right
- communication vehicle among stakeholders

First design artifact addressing
- performance
- modifiability
- reliability
- security

Key to systematic ***reuse***
- transferable, reusable abstraction

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

What Is Design and Analysis?

Design is making the decisions that lead to the creation of architecture.

- Which design decisions will lead to a software architecture that successfully addresses the desired system qualities?

Analysis ensures that the architecture used is the right one.

- How do you know if a given software architecture is deficient or at risk relative to its target system qualities?

Implications for Software Architecture Design and Analysis - 1

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

Consequences:

- These quality attributes should be designed into the architecture.

Caveats:

- A change in structure improving one quality often affects the other qualities.
- Architecture can only permit, not guarantee, any quality attribute.

Implications for Software Architecture Design and Analysis - 2

To be effective, the architecture-centric design and analysis activities must

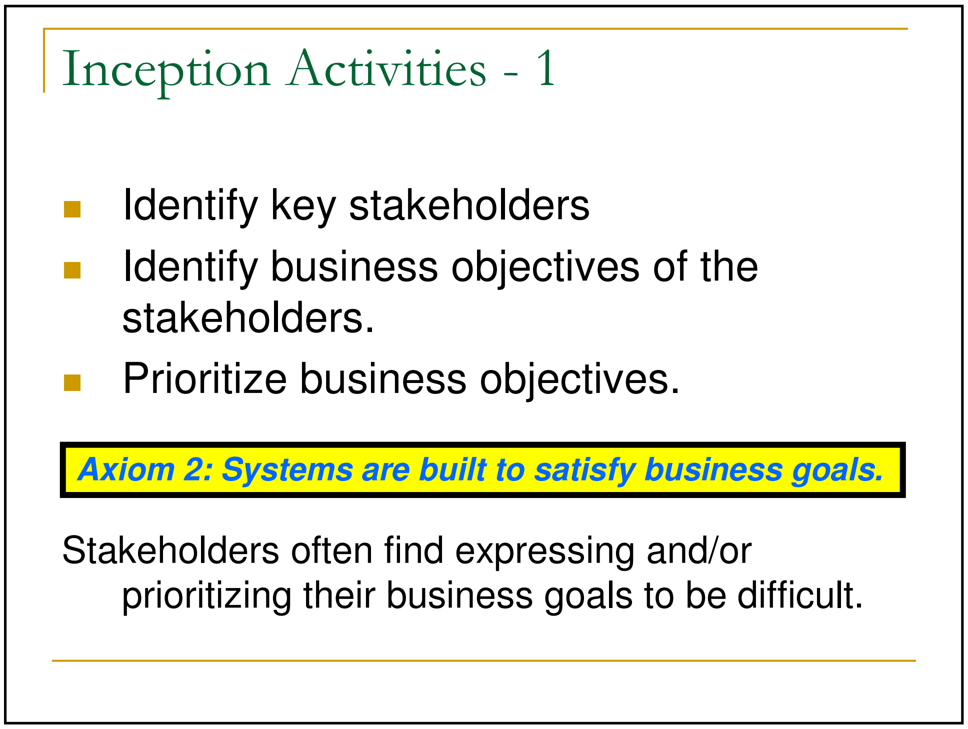
- directly link to business and mission goals
- explicitly focus on quality attributes
- explicitly involve system stakeholders

Sounds obvious. But how do we ensure this occurs?

We need *methods*, *processes*, and *tools* that enforce the axioms.

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - Identify, describe, and prioritize architecturally significant requirements (ASRs).
  - Design and document the architecture.
  - Validate the design decisions.
- *Review* Activities
  - Identify, describe, and prioritize ASRs.
  - Identify architecture description.
  - Analyze architecture description against ASRs.
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - Plan architecture improvements.
  - Refine review methods.

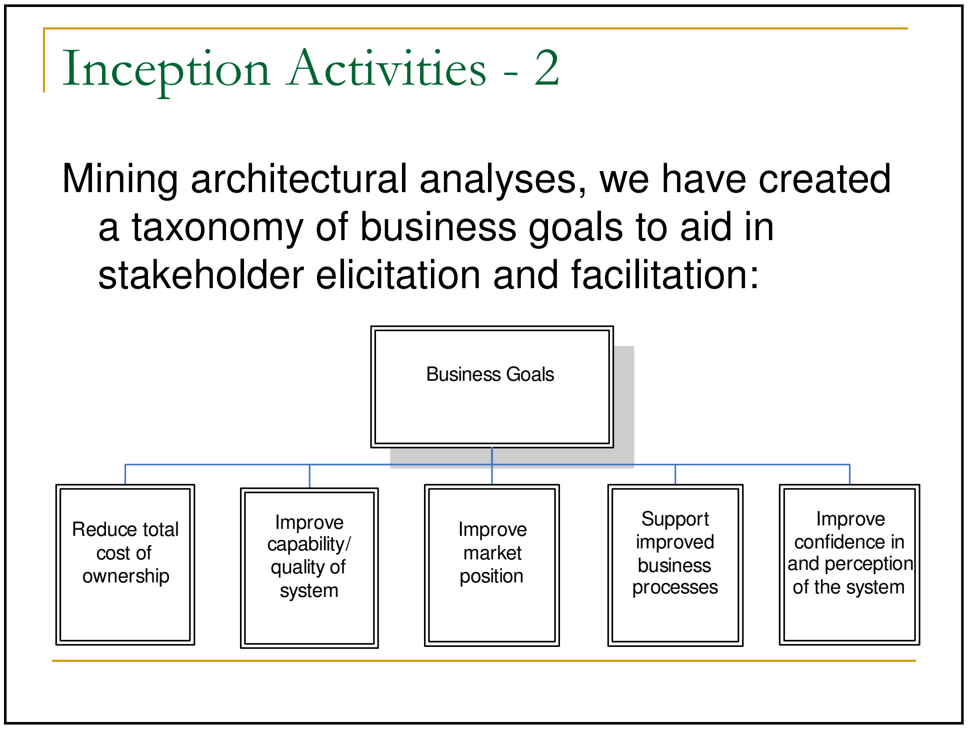


Inception Activities - 1

- Identify key stakeholders
- Identify business objectives of the stakeholders.
- Prioritize business objectives.

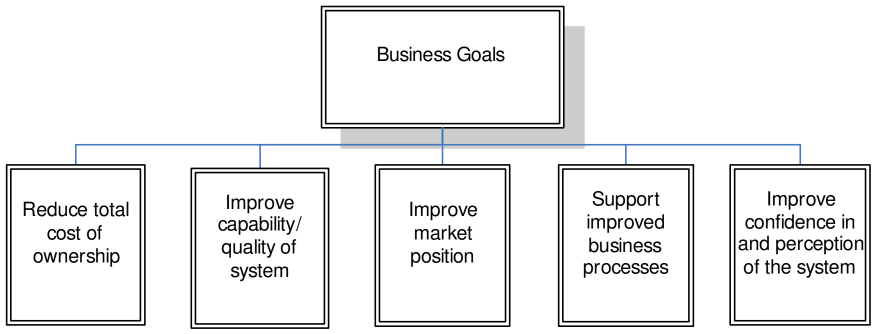
Axiom 2: Systems are built to satisfy business goals.

Stakeholders often find expressing and/or prioritizing their business goals to be difficult.



Inception Activities - 2

Mining architectural analyses, we have created a taxonomy of business goals to aid in stakeholder elicitation and facilitation:


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

Inception Activities - 3

Each of these categories is broken down into sub-categories, e.g.:

- Reduce Total Cost of Ownership:
  - reduce cost of development
  - reduce cost of deployment and operations
  - reduce cost of maintenance
  - reduce cost of retirement/moving to a new system

…and sub-sub-categories, e.g.:

- reduce cost of deployment and operations
  - ease of installation
  - ease of repair

Inception Activities - 4

These business goals must be *prioritized*.

But such goals are too vague for construction and analysis.

So we need to delve more deeply…

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - *Identify, describe, and prioritize architecturally significant requirements (ASRs).*
  - Design and document the architecture.
  - Validate the design decisions.
- *Review* Activities
  - Identify, describe, and prioritize ASRs.
  - Identify architecture description.
  - Analyze architecture description against ASRs.
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - Plan architecture improvements.
  - Refine review methods.

Architecturally Significant Requirements

***Axiom 3:* Architecture design is based on a set of architecturally significant requirements, derived from business goals.**

But where do we get ASRs from?

Stakeholder thinking about ASRs is often fuzzy.

One answer: a *Quality Attribute Workshop*

Quality Attribute Workshop (QAW)

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

Key points about the QAW are that it is

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

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

Step 1: QAW Presentation and Introductions

QAW Presentation

- QAW facilitators describe the motivation for the QAW and explain each step of the method.

Introductions

- QAW facilitators introduce themselves to the stakeholders.
- Stakeholders introduce themselves and briefly describe their background and relationship to the system.

Step 2: Business/Programmatic Presentation

A representative from the system stakeholder community presents the system's business and/or programmatic drivers.

- business/programmatic context for the system
- high-level functional requirements
- high-level constraints
- high-level quality attribute requirements
- plan for development

Facilitators capture information that may shed light on the quality attribute drivers.

Step 3: Architectural Plan Presentation

The system architect presents the architecture development plans including

- key business/programmatic requirements
- key technical requirements and constraints that will drive architectural decisions, such as
  - mandated operating systems, hardware, middleware, and so forth
  - other systems with which the system must interact
- existing context diagrams, high-level system diagrams, and descriptions

Step 4: Identification of Architectural Drivers

The QAW facilitators identify the architectural drivers that are key to realizing quality attribute goals. QAW facilitators

- present a distilled list of the architectural drivers they heard during the Business/Programmatic Presentation and the Architecture Plan Presentation
- ask for clarifications, additions, and/or deletions from the stakeholders to reach a consensus on the distilled list of architectural drivers

The final list of architectural drivers help focus the stakeholders during scenario brainstorming.

Step 5: Scenario Brainstorming

Stakeholders generate scenarios using a facilitated brainstorming process.

Each stakeholder generates a scenario in round-robin fashion or may opt to pass.

Depending on the number of stakeholders in the QAW and the allocated time for the workshop, stakeholders may have an opportunity to contribute one or more scenarios.

Describing Quality Attributes

Quality attribute names by themselves are not enough: performance, modifiability, security, …

Heated (and pointless) debates often revolve around the quality attribute to which a particular system behavior belongs.

The vocabulary describing quality attributes varies widely.

Quality Attribute Scenarios – 1

A solution to the problem of describing quality attributes is to use *quality attribute scenarios* to better characterize them.

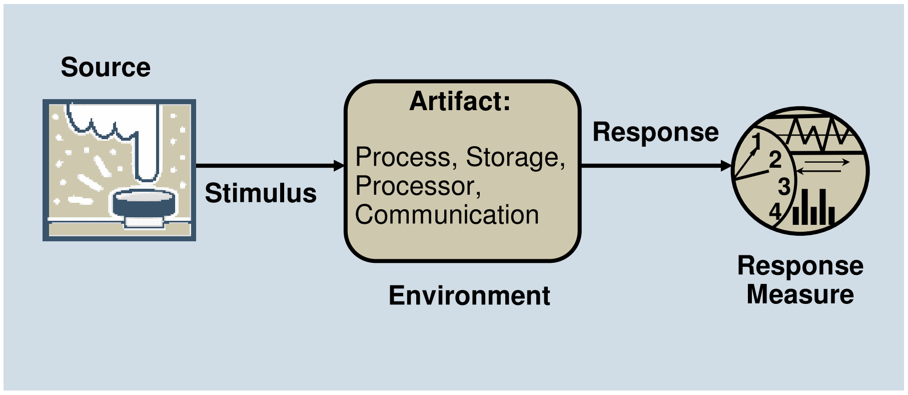
A quality attribute scenario is a short description of how a system is required to respond to some stimulus.

Quality Attribute Scenarios – 2

A quality attribute scenario consists of six parts:

1. **source** – an entity that generates a stimulus
2. **stimulus** – a condition that affects the system
3. **artifact** – the part of the system that was stimulated by the stimulus
4. **environment** – the condition under which the stimulus occurred
5. **response** – the activity that results because of the stimulus
6. **response measure** – the measure by which the system's response will be evaluated

Parts of a Quality Attribute Scenario



Guidance on Scenario Brainstorming

We focus on eliciting three types of scenarios:

- use cases – anticipated uses of the system
- growth – anticipated changes to the system
- exploratory – unanticipated stresses to the system (uses and/or changes)

Well-formed scenarios have: a stimulus, an environment, and a response specified.

Stimuli, Environment, Responses

Use case scenario

A remote user requests a database report via the Web during a 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.

These scenarios are "*falsifiable hypotheses*".

Step 6: Scenario Consolidation

The QAW facilitators ask stakeholders to identify scenarios that are similar in content.

- Similar scenarios are merged to prevent a "dilution" of votes when voting is done.
- QAW facilitators attempt to reach a consensus with the stakeholders before merging scenarios.

Example Scenario Consolidation

Scenarios that are similar in content are grouped together, e.g.

- In the event of a processor fault, the system can be rebooted/reinitialized.
- A processor failure or crash doesn’t adversely affect any other components.
- Software continues to operate even if the host fails.

Step 7: Scenario Prioritization

Each stakeholder is then allocated votes (30% of the number of scenarios generated).

- Voting occurs in two rounds; each stakeholder will “spend” half of the votes in each round.
- Stakeholders can spend any number of votes on any scenario they like.
- Votes are counted and the scenarios are prioritized.

Step 8: Scenario Refinement

The top scenarios are further refined. The number of scenarios refined depends on the time available.

- Typically the top five scenarios are refined.

For each scenario, the QAW facilitators further elaborate and document the following:

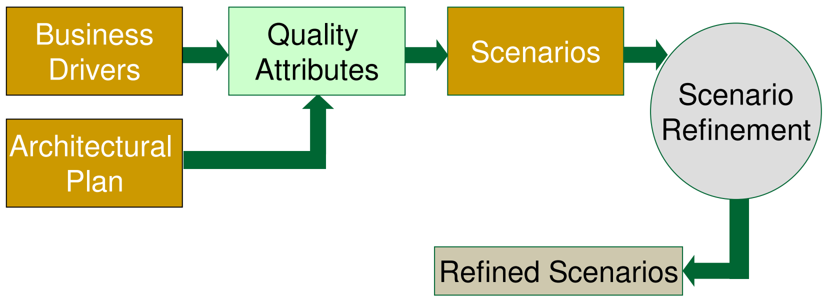
- business/mission goals affected by the scenario
- description of relevant quality attributes
- list of questions with respect to the scenarios that stakeholders would like to pose
- any issues that may arise during the scenario refinement

QAW Steps

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

Iterate as necessary with a broader or different stakeholder community.

QAW Conceptual Flow



QAW Results/Outputs

Increased stakeholder communication

Clarified quality attribute requirements

Informed basis for architectural design decisions

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - Identify, describe, and prioritize architecturally significant requirements (ASRs).
  - *Design and document the architecture.*
  - Validate the design decisions.
- *Review* Activities
  - Identify, describe, and prioritize ASRs.
  - Identify architecture description.
  - Analyze architecture description against ASRs.
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - Plan architecture improvements.
  - Refine review methods.

Software Architecture Design

The design for a system consists of a collection of decisions.

Design is the process of making decisions.

Decisions made early constrain ones made later.

So…

make decisions early that have the farthest reaching impact.

Software Architecture Design Decisions

Categories:

- Coordination model
- Data and object model
- Allocation of functionality
- Management of resources
- Binding time decisions
- Mapping among architectural elements

Coordination Model

What are the communication mechanisms between the system and external entities?

What are the inter-element communication mechanisms and what are their properties (e.g., synchronous, asynchronous, hybrid coupling)?

What are the intra-element communication mechanisms?

Data and Object Model

What *abstraction mechanisms* are used?

What are the *data models* on which element communication depend?

Allocation of Functionality

What are the major categories of system use?

What are the major modes of operation?

How is functionality divided and assigned to software elements?

Management of Resources

How much do system elements know about time?

What process/thread models will be employed?

What scheduling strategies will be employed?

What resources need to be managed?

What are the resource limits?

Binding Time Decisions

How and where are execution dependencies among elements resolved?

Which elements are stateful or stateless?

How are different variants of a system to be managed?

- Compile time (e.g., compiler switches)
- Build time (e.g., replace modules, pick from library)
- Load time (e.g., dynamic link libraries [DLLs])
- Initialization time (e.g., resource files)
- Run time (e.g., load balancing)

Mapping Among Architectural Elements

What execution dependencies exist among elements?

How do elements in different architectural structures map to each other, e.g.

- How are modules mapped to runtime elements?
- How are runtime elements mapped to processors?

Architectural Drivers

Axiom 4: Quality attribute requirements exert the strongest influence on architectural design.

Designing to satisfy all of the requirements at once is too difficult.

Some requirements have more influence than others on the architecture.

Architectural drivers are the combination of functional requirements, quality attribute requirements, and constraints that “shape” the architecture or the particular element under consideration.

How Are Early Design Decisions Made?

Early software architecture design decisions are made in the context of architectural drivers.

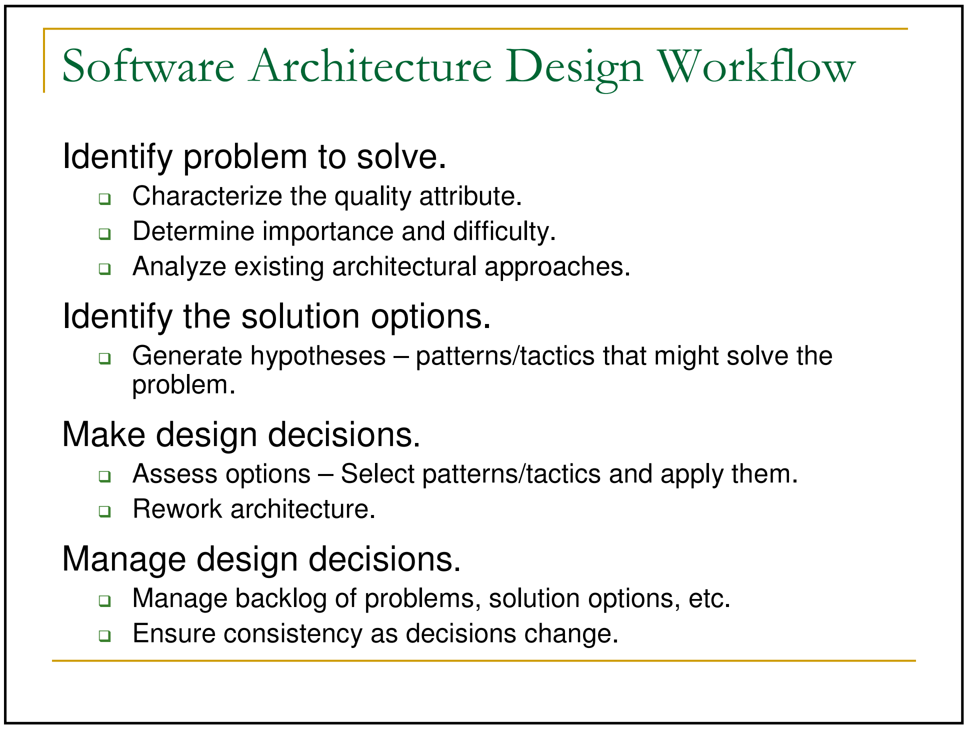
For each decision, consider whether the decision impacts any of the architectural drivers – either supports them or hinders them.

For example, consider the early design decisions associated with the Coordination model and the communication mechanisms between the system and external entities …

What Are the Communication Mechanisms?

How would quality attributes enter into this decision?

- Availability
  - Does the mechanism have to support failure of the external entity?
  - Does the mechanism have to guarantee delivery?
- Modifiability
  - Will the external entity change?
  - Will the information being communicated change?
- Performance
  - Is the communication with the external entity sensitive to system latency or throughput?
- Security
  - Is the communication with the external entity subject to a threat?
- Testability
  - How will the communication be tested?
  - Can the communication be played back for testing?
- Usability
  - If the external entity is a user, are any of the usability cross-cutting scenarios relevant?



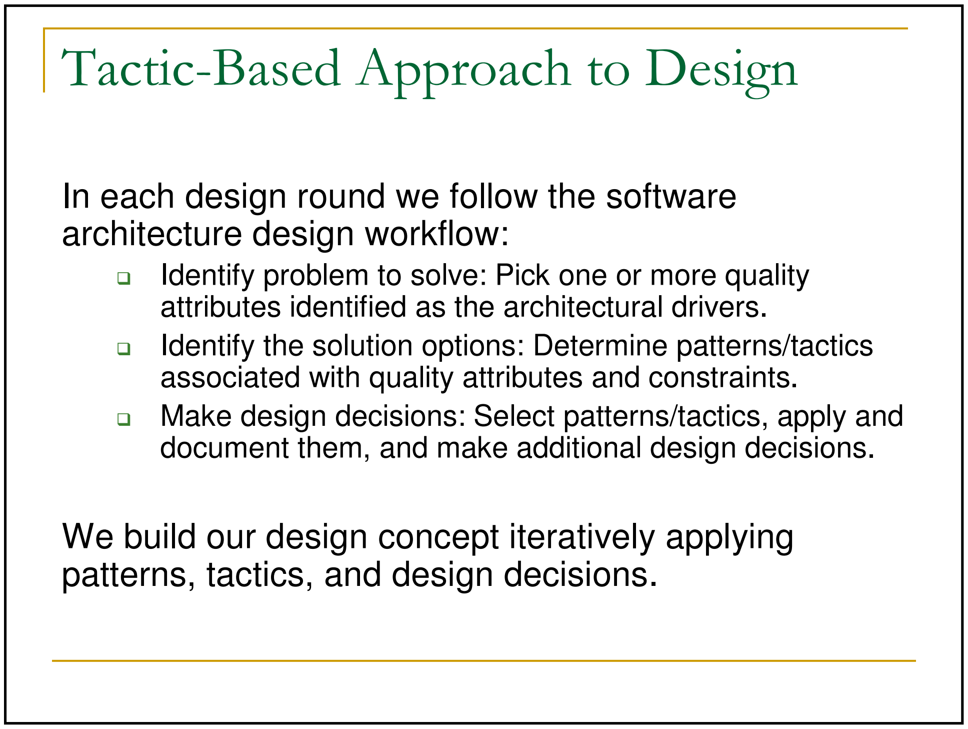
Software Architecture Design Workflow

Identify problem to solve.
- Characterize the quality attribute.
- Determine importance and difficulty.
- Analyze existing architectural approaches.

Identify the solution options.
- Generate hypotheses – patterns/tactics that might solve the problem.

Make design decisions.
- Assess options – Select patterns/tactics and apply them.
- Rework architecture.

Manage design decisions.
- Manage backlog of problems, solution options, etc.
- Ensure consistency as decisions change.



Tactic-Based Approach to Design

In each design round we follow the software architecture design workflow:
- Identify problem to solve: Pick one or more quality attributes identified as the architectural drivers.
- Identify the solution options: Determine patterns/tactics associated with quality attributes and constraints.
- Make design decisions: Select patterns/tactics, apply and document them, and make additional design decisions.

We build our design concept iteratively applying patterns, tactics, and design decisions.

Making Design Decisions

The core of the design process consists of the designer generating a hypothesis and then testing that hypothesis against some criteria.

- If the hypothesis does not pass the test, another hypothesis is generated and tested.

Tactics have a role in

- the validating of a design against criteria
- the generation of hypotheses

Let's examine tactics in detail, using availability and performance as examples.

Tactics – 1

Axiom 5: Architecture design can be made tractable by considering a small number of primitives, called tactics.

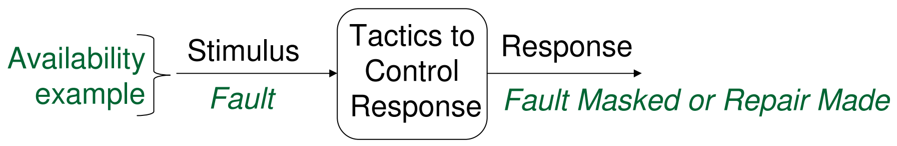
The design for a system consists of a collection of design decisions:

- Some decisions are intended to ensure the achievement of the system's functionality.
- Other decisions are intended to help control the quality attribute responses.

All of these decisions are *tactics*:

- A tactic is a design decision that is influential in the control of a quality attribute response.

Tactics – 2



Each tactic is a design option for the architect.

- For example, to promote availability, we might choose the Redundancy tactic.

One tactic can refine another tactic. For example, redundancy could be refined to data and/or computational redundancy tactics.

Patterns package tactics. For example, a pattern that supports availability will likely use the Redundancy tactic to achieve some level of availability.

Availability Tactics – 1

Fault detection

- Ping/Echo: when one component issues a ping and expects to receive an echo within a predefined time from another component
- Heartbeat: when one component issues a message periodically while another listens for it
- Exception: using exception mechanisms to raise faults when an error occurs

Availability Tactics – 2

Fault recovery preparation and repair

- Voting: when processes take equivalent input and compute output values that are sent to a voter
- Active Redundancy: when redundant components are used to respond to events in parallel
- Passive Redundancy: when a primary component responds to events and informs standby components of the state updates they must make
- Spare: when a standby computing platform is configured to replace failed components

Availability Tactics – 3

Fault recovery and reintroduction

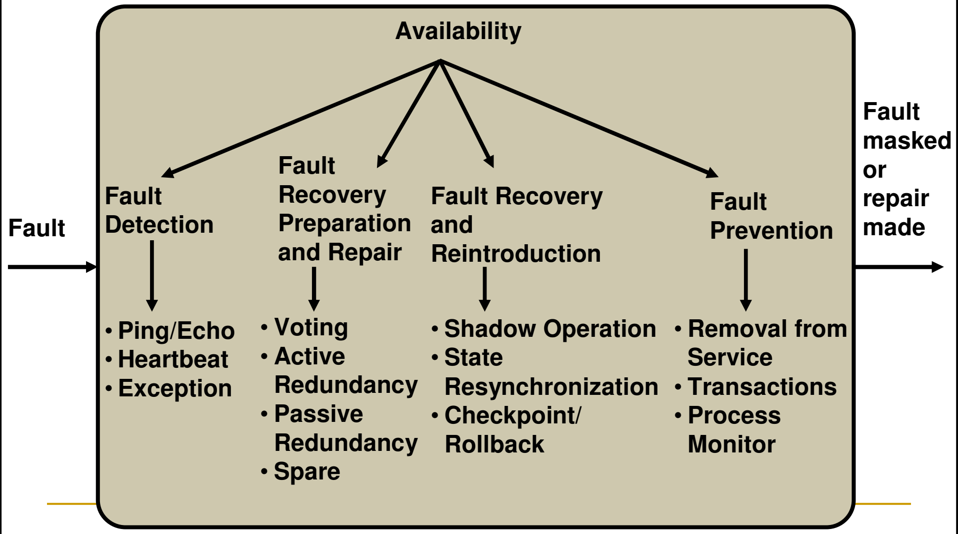
- Shadow Operation: running a previously failed component in “shadow mode” before it is returned to service
- State Resynchronization: saving a state periodically and then using it to resynchronize failed components
- Checkpoint/Rollback: recording a consistent state that is created periodically or in response to specific events

Availability Tactics – 4

Fault prevention

- Removal from Service: removing a system component from operation so it can undergo some procedure that will help it avoid failure in the future (e.g., rebooting a component prevents failures caused by memory leaks)
- Transactions: the bundling of several sequential steps such that the entire bundle can be undone at once
  - prevents data from being affected if one step in a process fails
  - prevents simultaneous access to data by concurrent threads
- Process Monitor: Monitoring processes are used to monitor critical components, remove them from service. and re-instantiate new processes in their place.

Summary of Availability Tactics



Thinking About Performance - 1

The goal of performance tactics is to generate a response to an event arriving at the system within some time constraint.

Two basic contributors to the response time are *resource consumption* and *blocked time*.

After an event arrives, either the system is processing on that event or the processing is blocked for some reason.

Thinking About Performance - 2

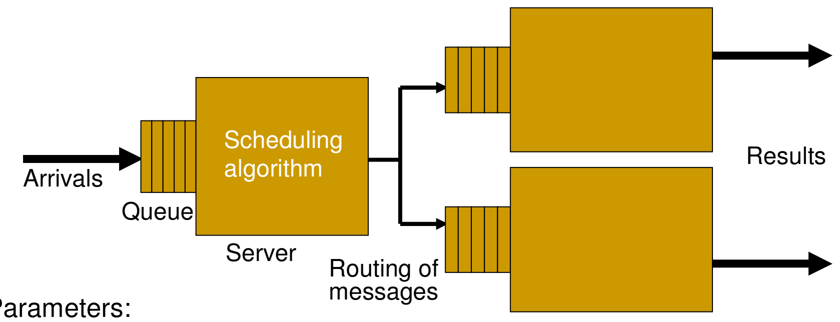
Resource consumption

- Resources include central processing unit (CPU), data stores, network communication bandwidth, and memory.

Blocked time

- A computation can be blocked from using a resource because
  - there is contention for the resource,
  - the resource is unavailable, or
  - the computation depends on the result of other computations that are not yet available.

Queuing Model for Performance



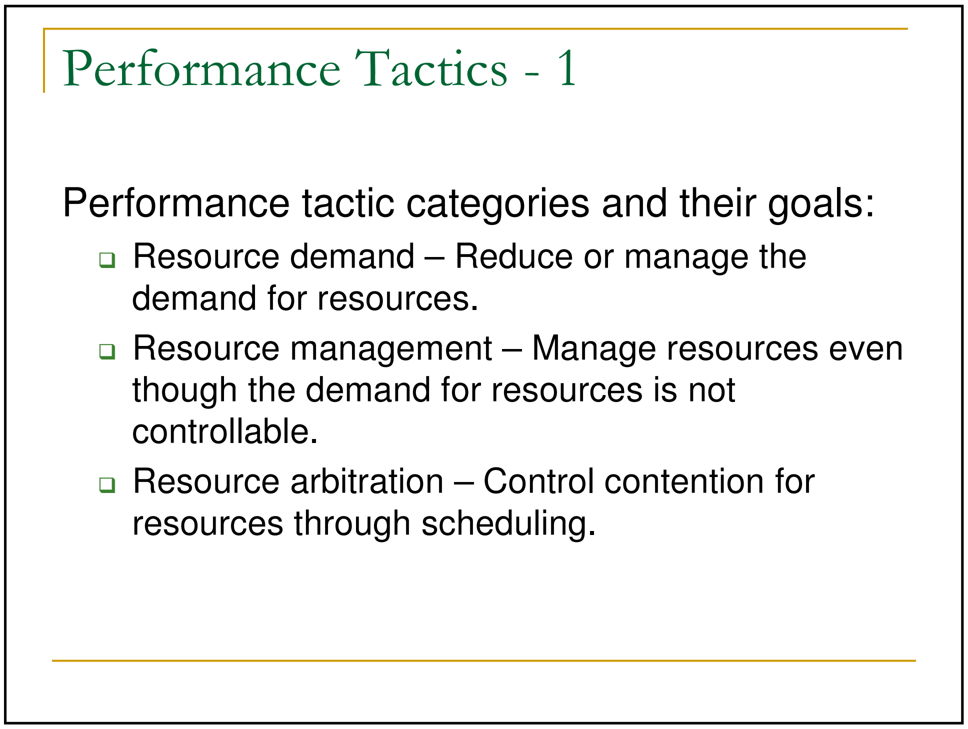
Parameters:
- Arrival rate
- Queuing discipline
- Scheduling algorithm
- Service time
- Topology
- Network bandwidth
- Routing algorithm

Latency (time to compute results) can *only* be affected by changing one of the parameters.

Managing Performance

Architectural means for controlling the parameters of a performance model:

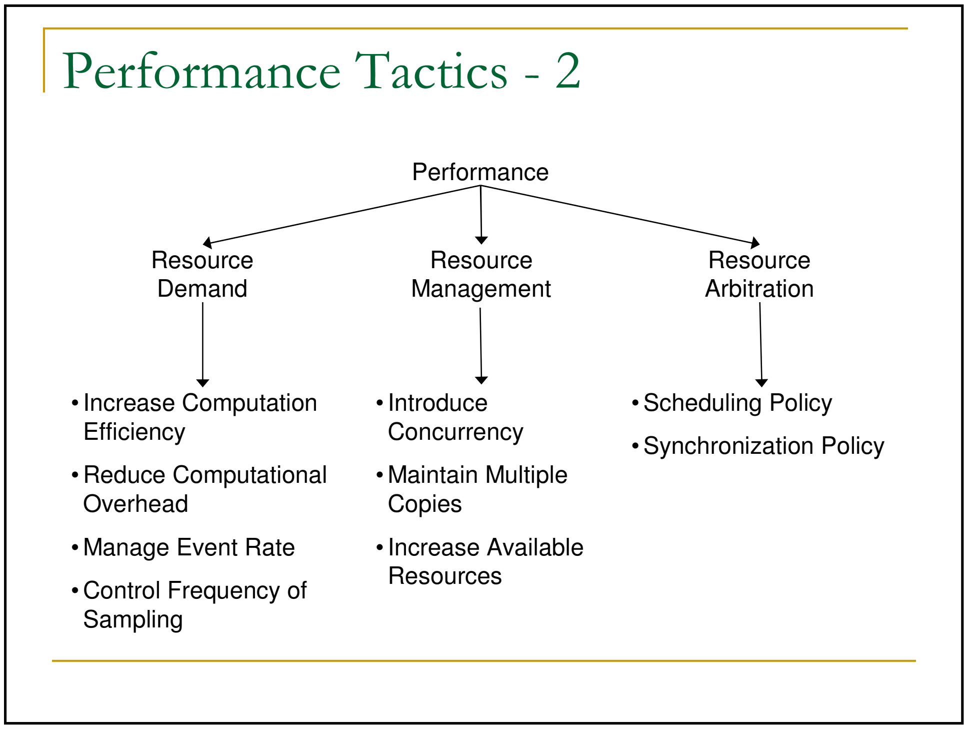
- Arrival rate – restrict access, differential rate/charging structure
- Queuing discipline – first-come first served (FCFS), priority queues, etc.
- Service time
  - Increase efficiency of algorithms.
  - Cut down on overhead (reduce inter-process communication, use thread pools, use pool of DB connections, etc.).
  - Use faster processor.
- Scheduling algorithm – round robin, service last interrupt first, etc.
- Topology – add/delete processors
- Network bandwidth – faster networks
- Routing algorithm – load balancing



Performance Tactics - 1

Performance tactic categories and their goals:

- Resource demand – Reduce or manage the demand for resources.
- Resource management – Manage resources even though the demand for resources is not controllable.
- Resource arbitration – Control contention for resources through scheduling.



Performance Tactics - 2

Performance

Resource Demand
- Increase Computation Efficiency
- Reduce Computational Overhead
- Manage Event Rate
- Control Frequency of Sampling

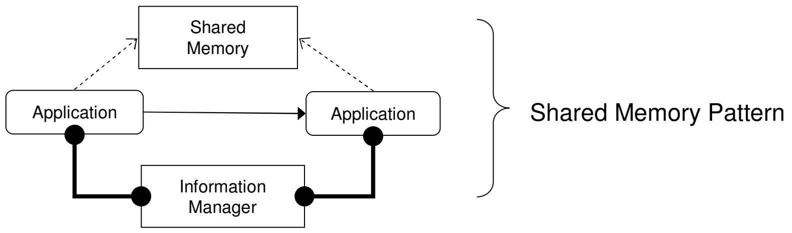
Resource Management
- Introduce Concurrency
- Maintain Multiple Copies
- Increase Available Resources

Resource Arbitration
- Scheduling Policy
- Synchronization Policy

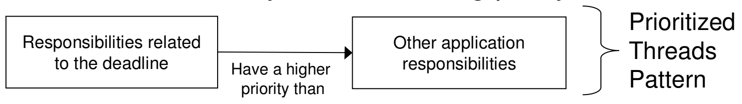
Tactics and Patterns - 1

Tactics can be implemented using patterns:

- Reduce computational overhead.



- Introduce concurrency and scheduling policy.


Responsibilities related to the deadline
Have a higher priority than
Other application responsibilities
Prioritized Threads Pattern

Tactics and Patterns - 2

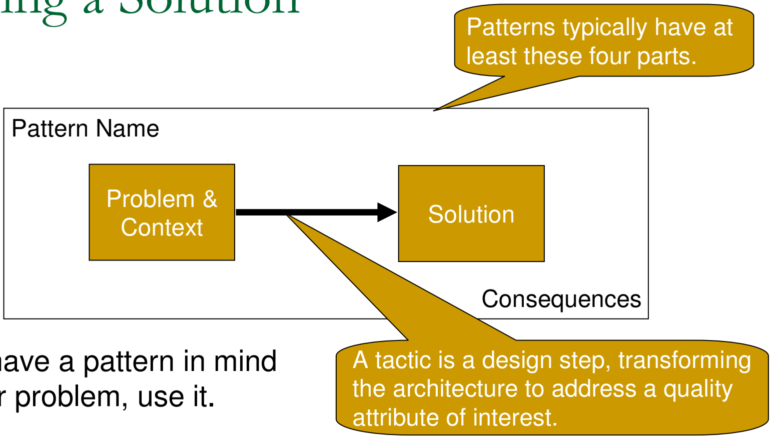
A *tactic* is a design decision that is influential in the control of a *single* quality attribute response.

A *pattern* is a prepackaged solution to a recurring problem that resolves *multiple forces*.

Where to find patterns:

- Design Patterns: Elements of Reusable Object Oriented Software
- Pattern-Oriented Software Architecture
- Core J2EE Patterns
- Hillside Group Pattern Library
- Conference on Pattern Languages of Programs (PLoP)

Finding a Solution


Patterns typically have at least these four parts.
Pattern Name
Problem & Context
Solution
Consequences
A tactic is a design step, transforming the architecture to address a quality attribute of interest.

If you have a pattern in mind for your problem, use it.

Use tactics when you need help coming up with a pattern, when an existing pattern isn't quite right and you need to tailor it, or when you want to validate the choice of a pattern.

Ideal Tactic Definition Process

Begin with an analytic model for the quality attribute of concern.

Identify parameters of that model.

Identify architectural techniques to manipulate the parameters of the model.

The World Is Not Ideal

Analytic models exist for performance, modifiability, and partially for a few other quality attributes.

What do we do if there is no analytic model?

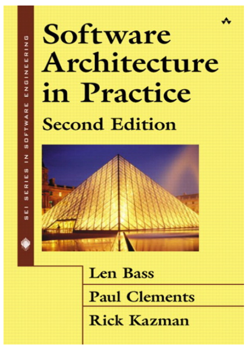
Interview experts in achieving the particular quality attribute and abstract the responses.

What Lists of Tactics Exist?

We have lists for

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

See


Software Architecture in Practice
Second Edition
SEI Series in Software Engineering
Len Bass
Paul Clements
Rick Kazman

КЛАССИКА COMPUTER SCIENCE
АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ПРАКТИКЕ
2-Е ИЗДАНИЕ
Л. БАСС, П. КЛЕМЕНТС, Р. КАЦМАН
ПИТЕР

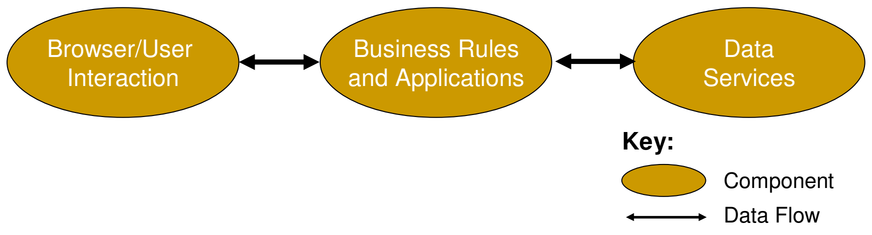
for more information

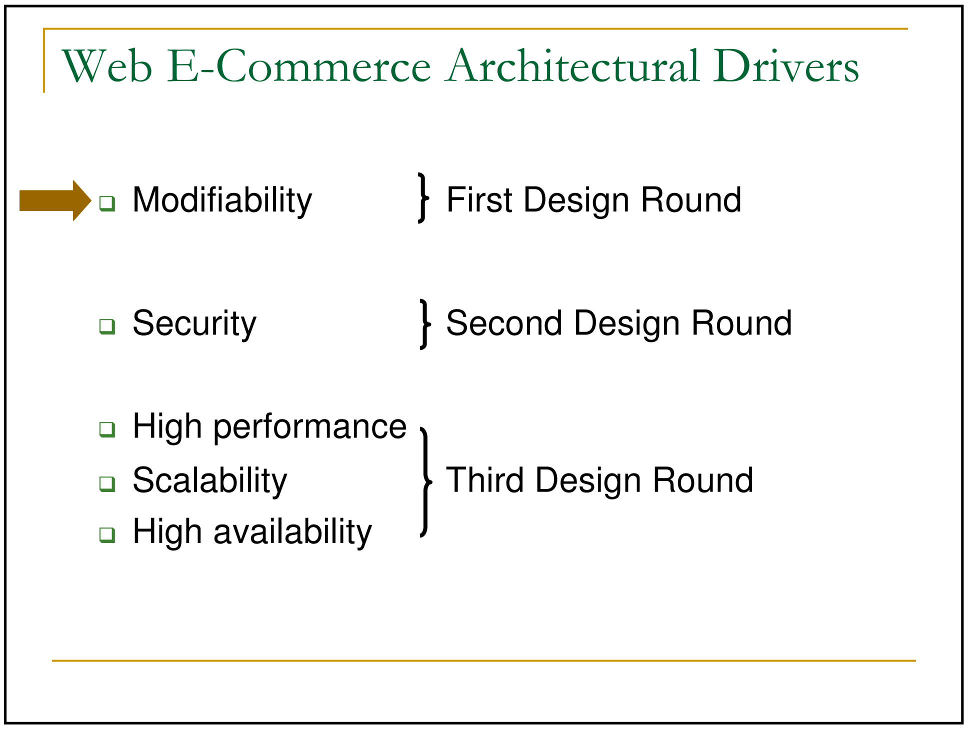
Software Architecture Design Example

We will now use an example to illustrate the software architecture design workflow.

Example: Web E-Commerce

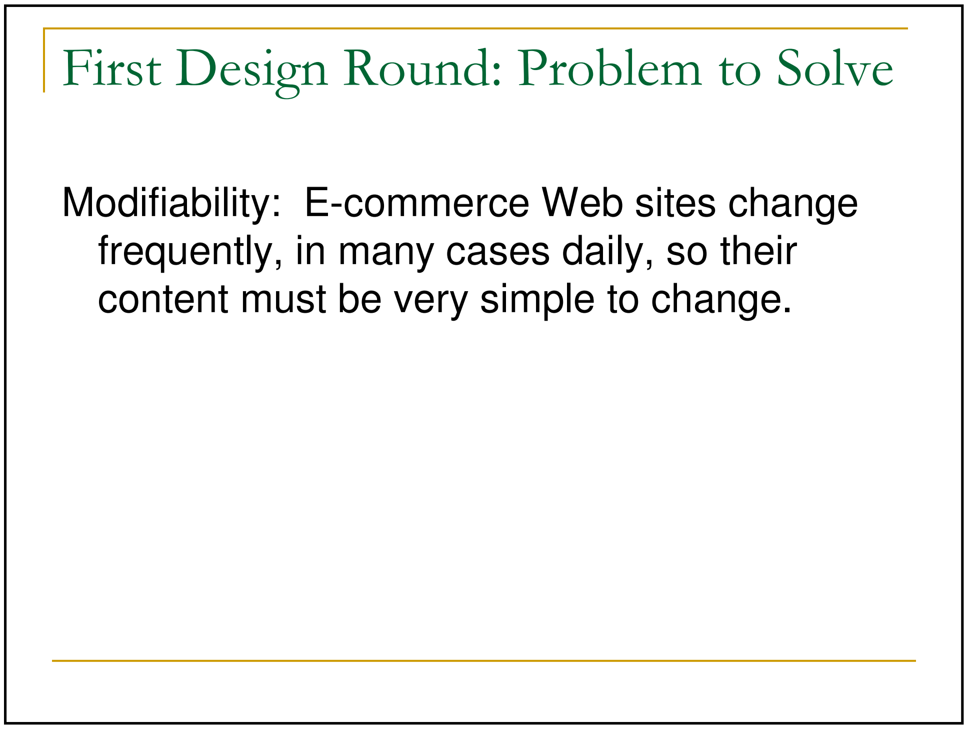
- System context: Internet
- Technical environment: e-commerce reference architecture
- Initial pattern: canonical e-commerce three-tier architecture





Web E-Commerce Architectural Drivers

➡ ❑ Modifiability	First Design Round
❑ Security	Second Design Round
❑ High performance ❑ Scalability ❑ High availability	Third Design Round



First Design Round: Problem to Solve

Modifiability: E-commerce Web sites change frequently, in many cases daily, so their content must be very simple to change.

First Design Round: Patterns and Tactics

The e-commerce pattern provides modifiability by virtue of separation of responsibilities into distinct *tiers*.

However, when later analyzing the architecture, it is helpful to understand the underlying tactics.

- Tactics
  - Abstract common services
  - Semantic coherence
  - Use an intermediary
  - Maintain existing interfaces
- How Achieved
  - Separation of browser functionality, database, and business logic into distinct tiers.

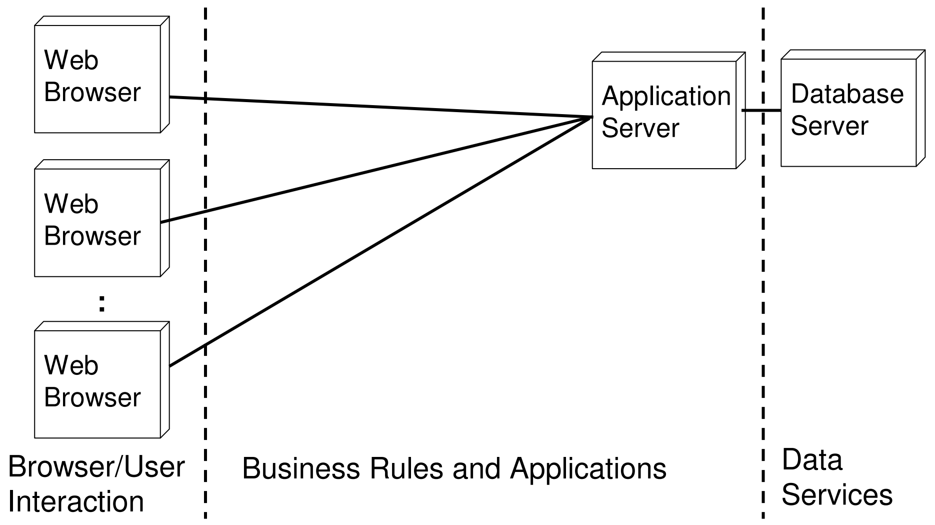
First Design Round: Design Decisions

The e-commerce pattern does not exempt the architect from having to make other early design decisions such as

- state management (which elements are stateful and which are stateless).

This affects whether clients are “thick” or “thin” the choice to use cookies, etc.

First Design Round: Design Concept



Key: UML

Web E-Commerce Architectural Drivers

❑ Modifiability	} First Design Round
➡ ❑ Security	} Second Design Round
❑ High performance ❑ Scalability ❑ High availability	} Third Design Round

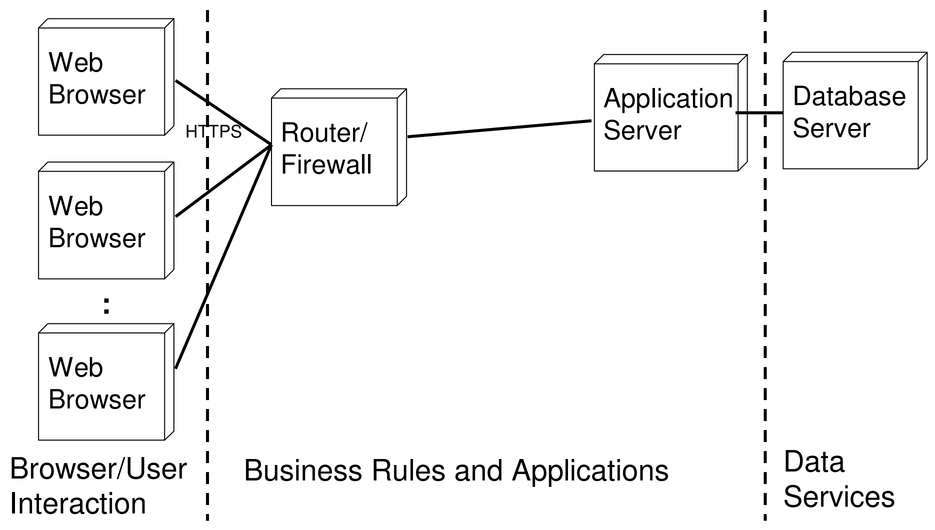
Second Design Round: Problem to Solve

- Security. Users must be assured that any sensitive information they send across the Web is secure from snooping. Operators of Web sites must be assured that their system is secure from attack (stealing or modifying data, rendering data unusable by flooding it with requests, crashing it, etc.).

Second Design Round: Tactics

- Tactics
  - Limit access
  - Maintain integrity
  - Limit exposure
  - Maintain data confidentiality
- How Achieved
  - Router/Firewall
  - Encryption across public networks (HTTPS)

Second Design Round: Design Concept



Key: UML

Web E-Commerce Architectural Drivers

- Modifiability } First Design Round
- Security } Second Design Round
- High performance
- Scalability
- High availability

} Third Design Round

Third Design Round: Problem to Solve

High performance. A popular Web site will typically have tens of millions of "hits" per day, and users expect low latency from it. Customers will not tolerate the site simply refusing their requests.

Scalability. As Web sites grow in popularity, their processing capacity must be able to similarly grow, to both expand the amount of data they can manage and maintain acceptable levels of customer service.

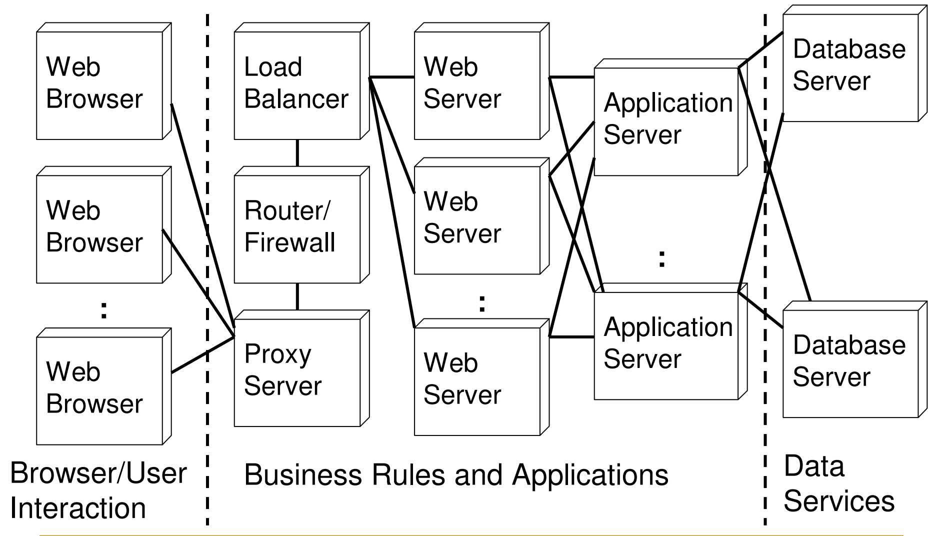
High availability. E-commerce sites are expected to be available "24/7." They never close, so must have minimal downtime-perhaps a few minutes a year.

Third Design Round: Patterns and Tactics

To achieve high performance and availability in the e-commerce architecture we need to make some further architectural changes.

- Tactics
  - Introduce Concurrency
  - Maintain Multiple Copies
  - Increase Available Resources
  - Scheduling Policy
- How Achieved
  - Replicated servers
  - Load balancing

Third Design Round: Design Concept



Key: UML

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - Identify, describe, and prioritize architecturally significant requirements (ASRs).
  - Design and document the architecture.
  - *Validate the design decisions.*
- *Review* Activities
  - Identify, describe, and prioritize ASRs.
  - Identify architecture description.
  - Analyze architecture description against ASRs.
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - Plan architecture improvements.
  - Refine review methods.

Validate Design Decisions

Axiom 6: Architecture design can and should be guided by analysis.

Design and analysis are two sides of the same coin.

To validate a design, it must be analyzed.

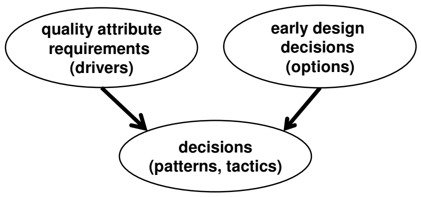
That is the subject of the next part of the workshop…

Interim Summary

Make important decisions early. Software architecture focuses on design decisions that help control a quality attribute response.

Choose the most influential (few) ASRs on which to focus. These are the "architectural drivers."

Choose a pattern, if you can find one, and then adjust the pattern based on tactics.


quality attribute requirements (drivers)
early design decisions (options)
decisions (patterns, tactics)

Part 2: Software Architecture Analysis

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - Identify, describe, and prioritize architecturally significant requirements (ASRs).
  - Design and document the architecture.
  - Validate the design decisions.
- *Review* Activities
  - *Identify, describe, and prioritize ASRs.*
  - *Identify architecture description.*
  - *Analyze architecture description against ASRs.*
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - Plan architecture improvements.
  - Refine review methods.

Why Evaluate an Architecture?

Because so much depends on it!

- An unsuitable architecture will precipitate disaster.
- Architecture determines the structure of the project.

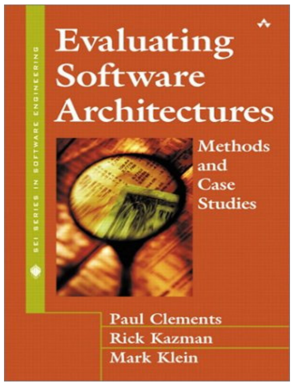
Because we can!

- Repeatable, structured methods offer a low-cost risk mitigation capability that can be employed early in the development life cycle.
- Making sure an architecture is the right one simply makes good sense.

Architecture evaluation should be a standard part of every architecture-based development methodology.

Evaluation Techniques

- *Questioning techniques* use questionnaires, checklists, and scenarios to investigate the way an architecture addresses it quality requirements.
- *Measuring techniques* apply some measurement tool to a software artifact.
- Our focus today, a hybrid technique: *the ATAM*
- These are all described in:


Evaluating Software Architectures
Methods and Case Studies
SEI Series in Software Engineering
Paul Clements
Rick Kazman
Mark Klein

Pedigree of the ATAM

The ATAM has existed for over 10 years.

- Well-defined, documented process.
- Books, courses focused on the ATAM.

It has been used in countless evaluations by major companies and government organizations:

- Boeing, Raytheon, GM, Ford, US Army, Siemens, Fidelity Investments, Bosch, Pitney-Bowes, HP, General Dynamics, Philips, Visteon, Wells Fargo, UPS, Daimler, Mellon Financial, …

The ATAM

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

The ATAM brings together three groups in an evaluation:

- a trained evaluation team
- an architecture's "decision makers" (architect, senior designers, project managers, customers)
- representatives of the architecture's stakeholders

Purpose of the ATAM

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

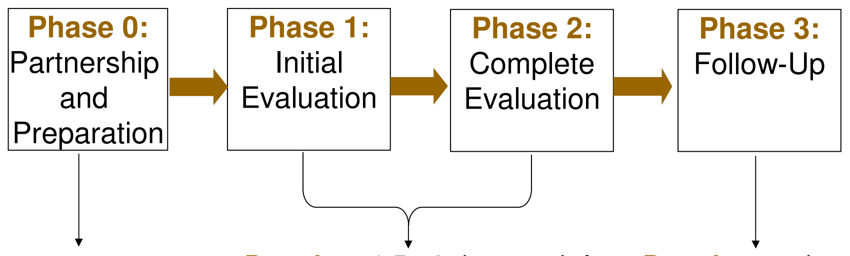
Discovered risks can then be made the focus of mitigation activities such as further design, further analysis, and prototyping.

Surfaced tradeoffs can be explicitly identified and documented.

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

ATAM Phases

ATAM evaluations are conducted in four phases.


Phase 0: Partnership and Preparation
Phase 1: Initial Evaluation
Phase 2: Complete Evaluation
Phase 3: Follow-Up

Duration: varies
Meeting: primarily phone, email

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

Duration: varies
Meeting: primarily phone, email

ATAM Phase 0

Phase 0 precedes the technical part of the evaluation:

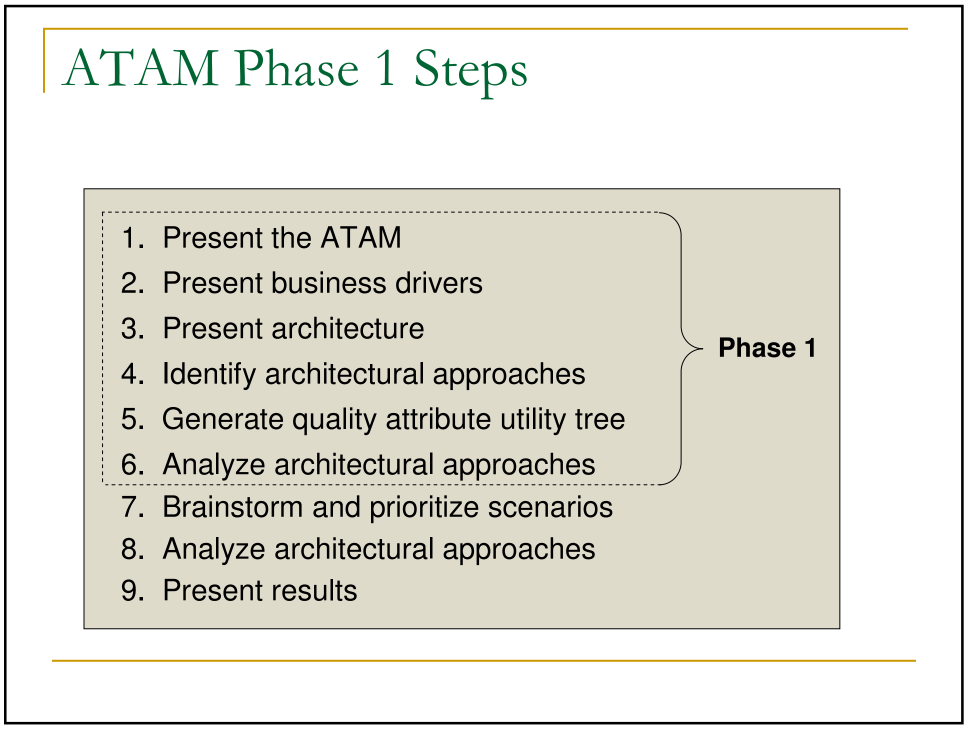
- The customer and a subset of the evaluation team exchange their understanding about the method and the system whose architecture is to be evaluated.
- An agreement to perform the evaluation is worked out.
- A core evaluation team is fielded.

ATAM Phase 1

Phase 1 involves a small group of predominantly technically oriented stakeholders.

Phase 1 is

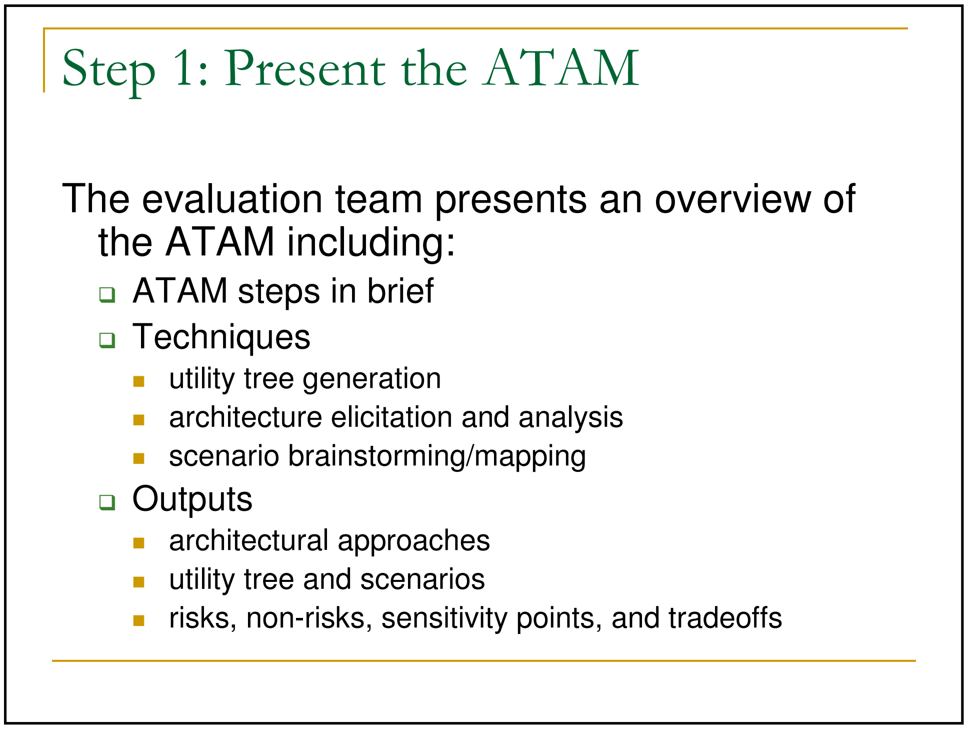
- architecture-centric
- focused on eliciting detailed architectural information and analyzing it
- top-down analysis



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

Phase 1



Step 1: Present the ATAM

The evaluation team presents an overview of the ATAM including:

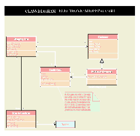
- ATAM steps in brief
- Techniques
  - utility tree generation
  - architecture elicitation and analysis
  - scenario brainstorming/mapping
- Outputs
  - architectural approaches
  - utility tree and scenarios
  - risks, non-risks, sensitivity points, and tradeoffs

Step 2: Present Business Drivers

The customer representative describes the system's business drivers including its

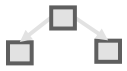
- business context
- high-level functional requirements
- high-level quality attribute requirements
  - architectural drivers: quality attributes that "shape" the architecture
  - critical requirements: quality attributes most central to the system's success

Step 3: Present Architecture



The architect presents an overview of the architecture including

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



The evaluation team begins probing for and capturing risks.

Step 4: Identify Architectural Approaches

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.

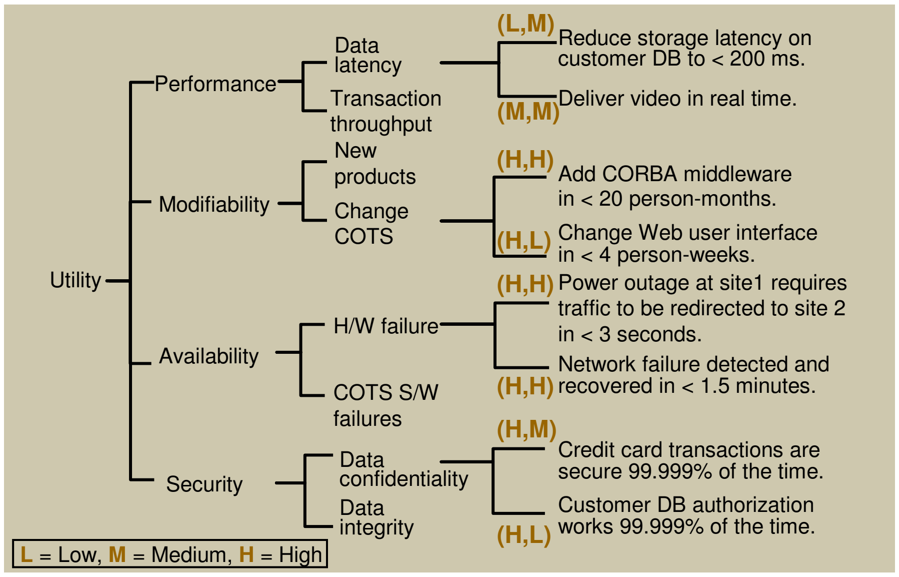
Step 5: Generate Quality Attribute Utility Tree

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 the “driving” attribute-specific requirements.
- The most important quality goals are the high-level nodes (typically performance, modifiability, security, and availability).
- Scenarios are the leaves of the utility tree.

The outputs of this step are a characterization and a prioritization of specific quality attribute requirements.

Example Quality Attribute Utility Tree



Utility
Performance
Data latency
Transaction throughput
(L,M) Reduce storage latency on customer DB to < 200 ms.
(M,M) Deliver video in real time.
Modifiability
New products
Change COTS
(H,H) Add CORBA middleware in < 20 person-months.
(H,L) Change Web user interface in < 4 person-weeks.
Availability
H/W failure
COTS S/W failures
(H,H) Power outage at site1 requires traffic to be redirected to site 2 in < 3 seconds.
(H,H) Network failure detected and recovered in < 1.5 minutes.
Security
Data confidentiality
Data integrity
(H,M) Credit card transactions are secure 99.999% of the time.
(H,L) Customer DB authorization works 99.999% of the time.
L = Low, M = Medium, H = High

How Scenarios Are Used – 1

We use six-part scenarios as described earlier:

1. **source** – an entity that generates a stimulus
2. **stimulus** – a condition that affects the system
3. **artifact** – the part of the system that was stimulated by the stimulus
4. **environment** – the condition under which the stimulus occurred
5. **response** – the activity that results because of the stimulus
6. **response measure** – the measure by which the system's response will be evaluated

How Scenarios Are Used – 2

Recall…

Scenarios are used to

- represent stakeholders' interests
- understand quality attribute requirements

Scenarios should cover a range of

- anticipated uses of the system (*use case scenarios*)
- anticipated changes to the system (*growth scenarios*)
- unanticipated stresses on the system (*exploratory scenarios*)

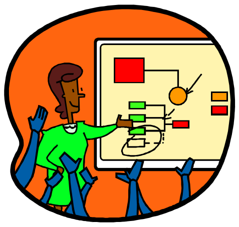
Scenario Analysis Outputs

As each scenario is analyzed against the architecture, the evaluation team identifies risks, non-risks sensitivity points, and tradeoffs.

- 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 place in the architecture that significantly affects whether a particular quality attribute response is achieved.
- A tradeoff is a property that affects more than one attribute and is a sensitivity point for more than one attribute.

Step 6: Analyze Architectural Approaches

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



The team

- identifies the architectural approaches
- asks quality-attribute-specific questions for the highest priority scenarios
- identifies and records risks, non-risks, sensitivity points, and tradeoffs

Risks and Tradeoffs

Example risk:

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

Example tradeoff:

- *Increasing the level of encryption will significantly increase security but decrease performance.*

Sensitivity Points and Non-Risks

Example sensitivity point:

- *The response time to system events is sensitive to the number of processes running on the main processor.*

Example non-risk:

- *Assuming message-arrival rates of no more than once per second and a processing time of less than 30 ms, the architecture should meet the 1-second soft deadline requirement.*

Scenario Analysis Template - Part 1

ATAM: Scenario Analysis		
Scenario		
Business Goal(s)		
Attribute		
Attribute Concern		
Scenario Refinement	Stimulus	
	Stimulus Source	
	Environment	
	Artifact	
	Response	
	Response Measure	

Scenario Analysis Template - Part 2

Architectural Decisions and Reasoning	
Risks	1.
Sensitivities	1.
Tradeoffs	1.
Non-Risks	1.
Other Issues	1.

ATAM Phase 2

Phase 2 involves a larger group of stakeholders.

Phase 2 is

- stakeholder-centric
- focused on eliciting diverse stakeholders' points of view and on verifying the results of Phase 1
- bottom-up analysis

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

Recap Phase 1 Work

Do this

Phase 2

Step 7: Brainstorm and Prioritize Scenarios

Scenarios are brainstormed in a round-robin manner.

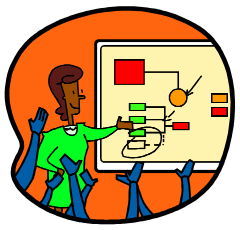
As in the QAW, each stakeholder is given 30% of the number of scenarios as votes.

Stakeholders can "spend" any number of votes on any scenario they like.

Votes are counted and the scenarios are prioritized.

Step 8: Analyze Architectural Approaches

The evaluation team now asks the architect to map these new scenarios on to the architecture.



These scenarios are “test cases”.

The evaluation team continues to probe for risks, sensitivities, and tradeoffs.

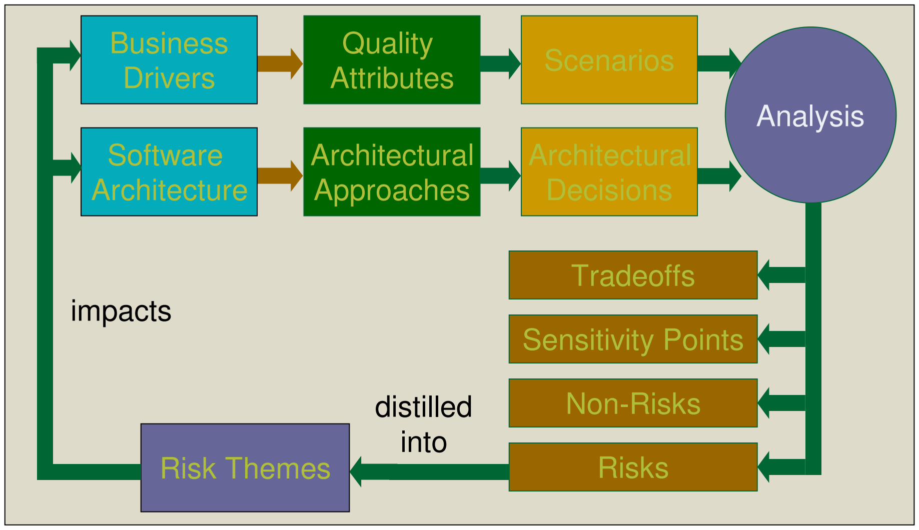
ATAM Phase 3

Phase 3 primarily involves producing a final report for the customer.

Typically a written report and a presentation are created.

Follow-on activities may also be scheduled.

Conceptual Flow of the ATAM



Benefits of the ATAM

The benefits of performing ATAM evaluations include

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

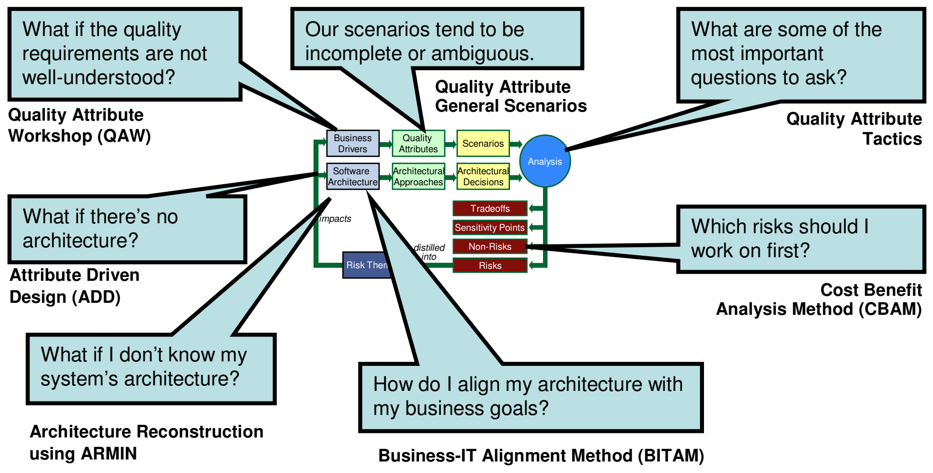
The end result is *improved architectures*.

Interim 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

ATAM Led to the Development of Other Methods and Techniques



What if the quality requirements are not well-understood?

Quality Attribute Workshop (QAW)

Our scenarios tend to be incomplete or ambiguous.

Quality Attribute General Scenarios

What are some of the most important questions to ask?

Quality Attribute Tactics

What if there's no architecture?

Attribute Driven Design (ADD)

Which risks should I work on first?

Cost Benefit Analysis Method (CBAM)

What if I don't know my system's architecture?

Architecture Reconstruction using ARMIN

How do I align my architecture with my business goals?

Business-IT Alignment Method (BITAM)

The Design and Analysis Process

- *Inception* Activities
  - Identify key stakeholders
  - Identify business objectives of the stakeholders.
  - Prioritize business objectives.
- *Design* Activities
  - Identify, describe, and prioritize architecturally significant requirements (ASRs).
  - Design and document the architecture.
  - Validate the design decisions.
- *Review* Activities
  - Identify, describe, and prioritize ASRs.
  - Identify architecture description.
  - Analyze architecture description against ASRs.
- *Post-Review* Activities
  - Summarize findings and review them with architecture owners.
  - *Plan architecture improvements.*
  - Refine review methods.

Planning Architecture Improvements

For a single evolutionary step, we use an Architecture Improvement Workshop

- borrows techniques from the ATAM, from Cost-Benefit Analysis (CBAM) and from Attribute-Driven Design (ADD)

For the long-term, we focus on Business-IT Alignment, using the Business-IT Alignment Method (BITAM)

The Purpose of the BITAM

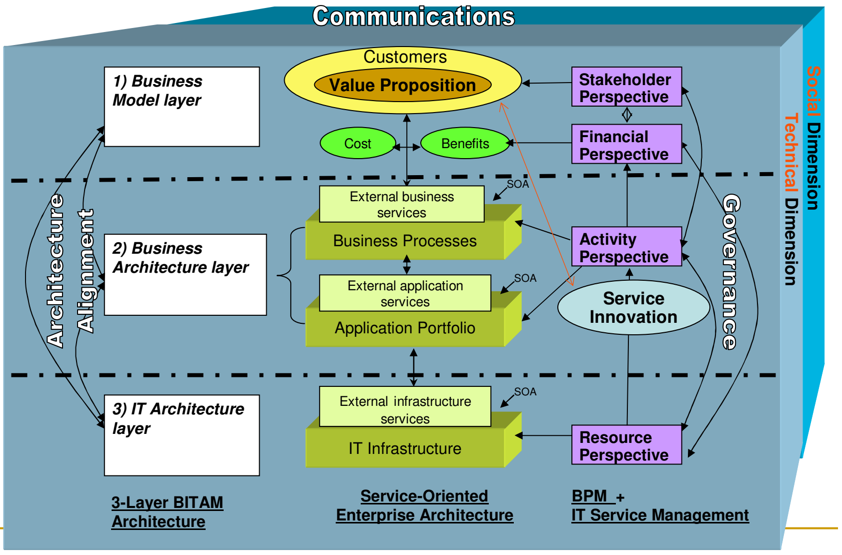
To systematically detect and correct misalignments between business models, business architectures, and IT architectures.

Benefits of the BITAM

The BITAM:

- provides a way of eliciting, collecting, prioritizing, and organizing information needed by the alignment/realignment process
- invites stakeholders to consider a range of realignment strategies and provides a decision procedure for choosing among the alternatives
- affords traceable and repeatable procedures for easier maintenance of alignment or faster re-alignment

The BITAM in Context



The BITAM-SOA Framework [Chen 2007]

Steps of the BITAM

1. Elicit business drivers from key management stakeholders.
2. Elicit a set of operational scenarios from the entire group of stakeholders.
3. Elicit a set of change scenarios from the entire group of stakeholders.
4. Prioritize the collected scenarios based on risk/value.
5. Elicit the business architecture from the key information architects.
6. Elicit the IT architecture from the key technical architects.
7. Map the operational scenarios onto the business architecture.
8. Map the operational and change scenarios onto the IT architecture.
9. Assess the misalignments.

Post Alignment…

Once alignment has been determined, the cycle begins again, starting with Inception Activities.

Summary

Design and analysis of architectures are mirror activities.

These activities should reflect the axioms of the architecture-centric approach.

To do them well you need:

- active stakeholder involvement
- clear characterizations and prioritizations of business goals and architectural drivers (described as quality attribute scenarios)
- an understanding of tactics and patterns
- methods that keep you focused

Further Reading

General Software Architecture:

- L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003.

Software Architecture Analysis:

- P. Clements, R. Kazman, M. Klein, *Evaluating Software Architectures: Methods and Case Studies*, Addison-Wesley, 2001.

BITAM:

- H-M Chen, R. Kazman, A. Garg, “BITAM: An Engineering-principled Method for Managing Misalignments between Business and IT Architectures”, *Journal of Science of Computer Programming*, 57:1, 2005, 5-26.

Or contact me:

Rick.Kazman@gmail.com

Questions? Comments?