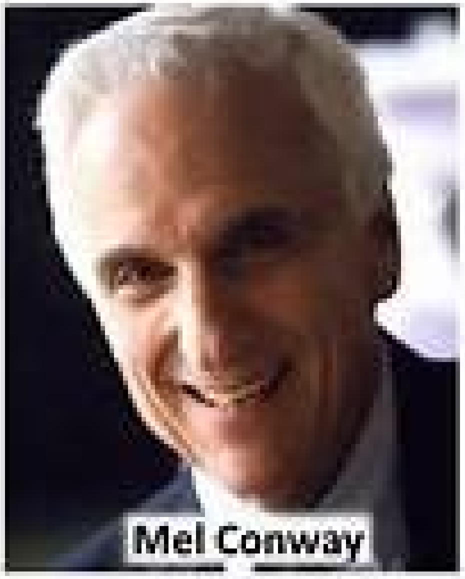
Conway's Law

"The structure of a software system reflects the structure of the organisation that built it"



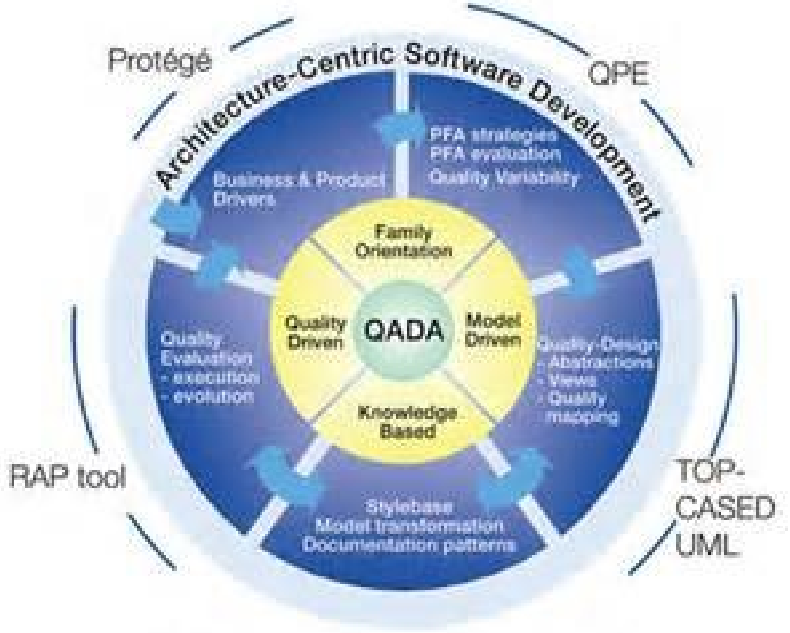
软件体系结构设计
Software Systems Architecture Design

清华大学软件学院 刘璘

Software Architecture

A software architecture defines:

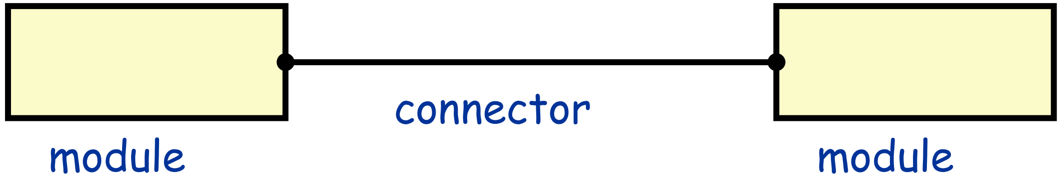
- the components of the software system
- how the components use each other's functionality and data
- How control is managed between the components



QADA® (Quality driven Architecture Desing and quality Analysis) is a registered trademark of VTT Technical Research Centre of Finland http://virtual.vtt.fi/qada

Coupling and Cohesion

- Architectural Building blocks:

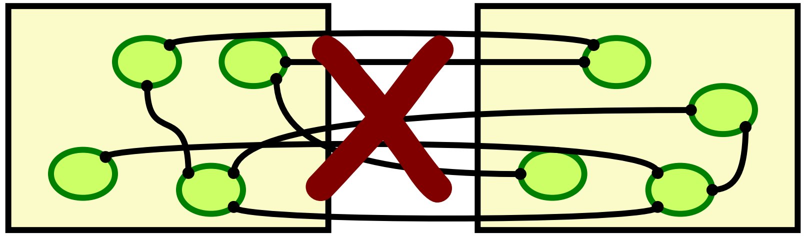


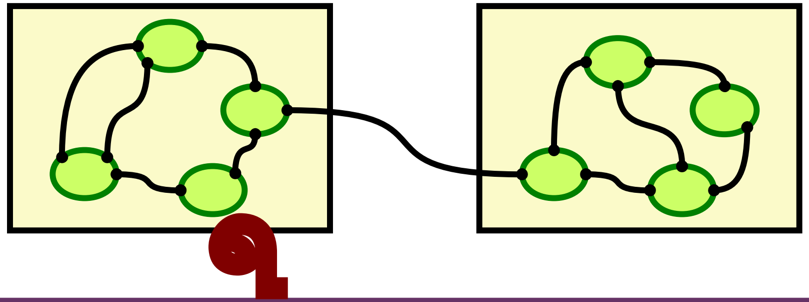
module

connector

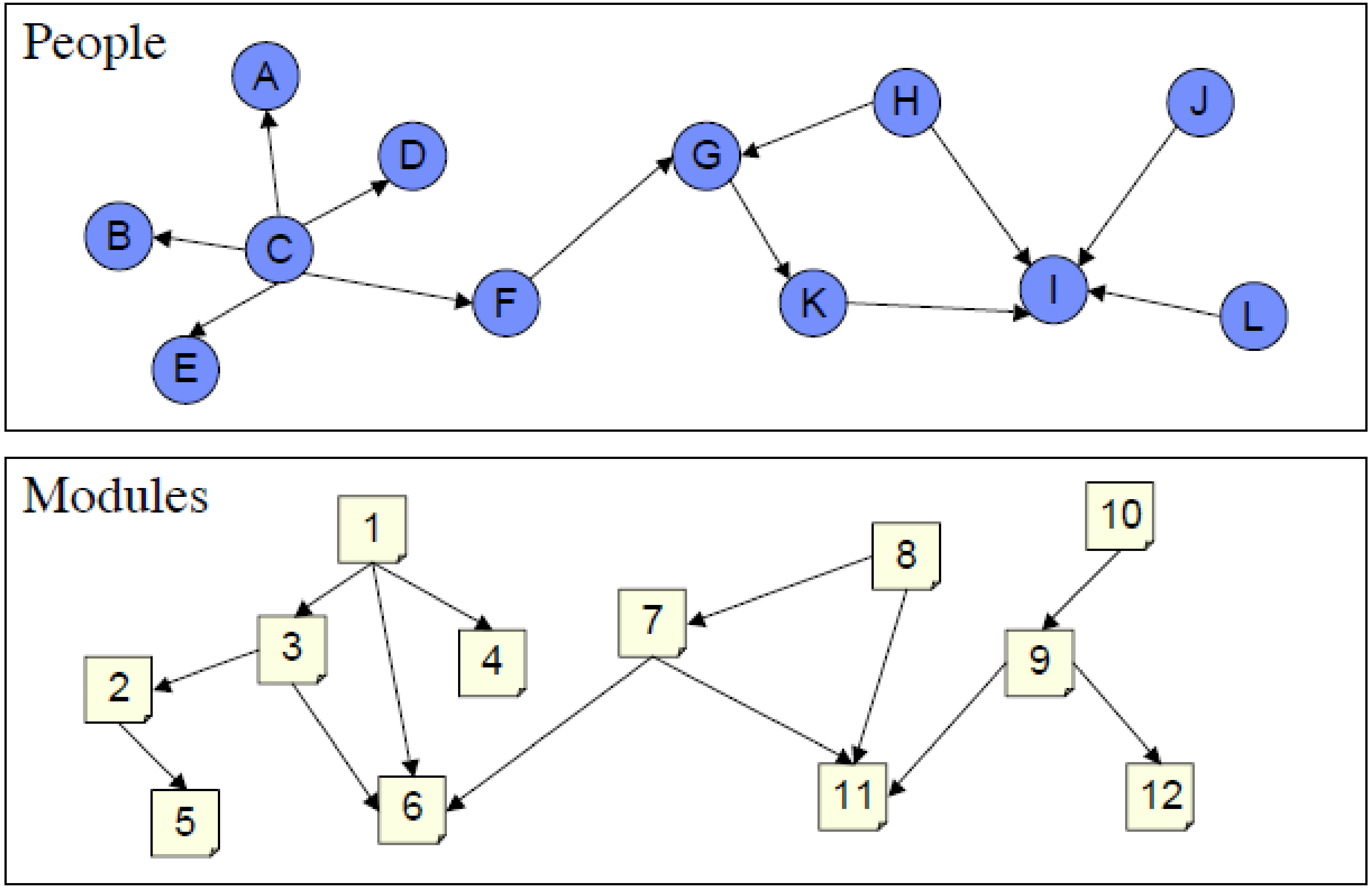
module

- A good architecture:
  - Minimizes coupling between modules:
    - Goal: modules don't need to know much about one another to interact
    - Low coupling makes future change easier
  - Maximizes the cohesion of each module
    - Goal: the contents of each module are strongly inter-related
    - High cohesion makes a module easier to understand



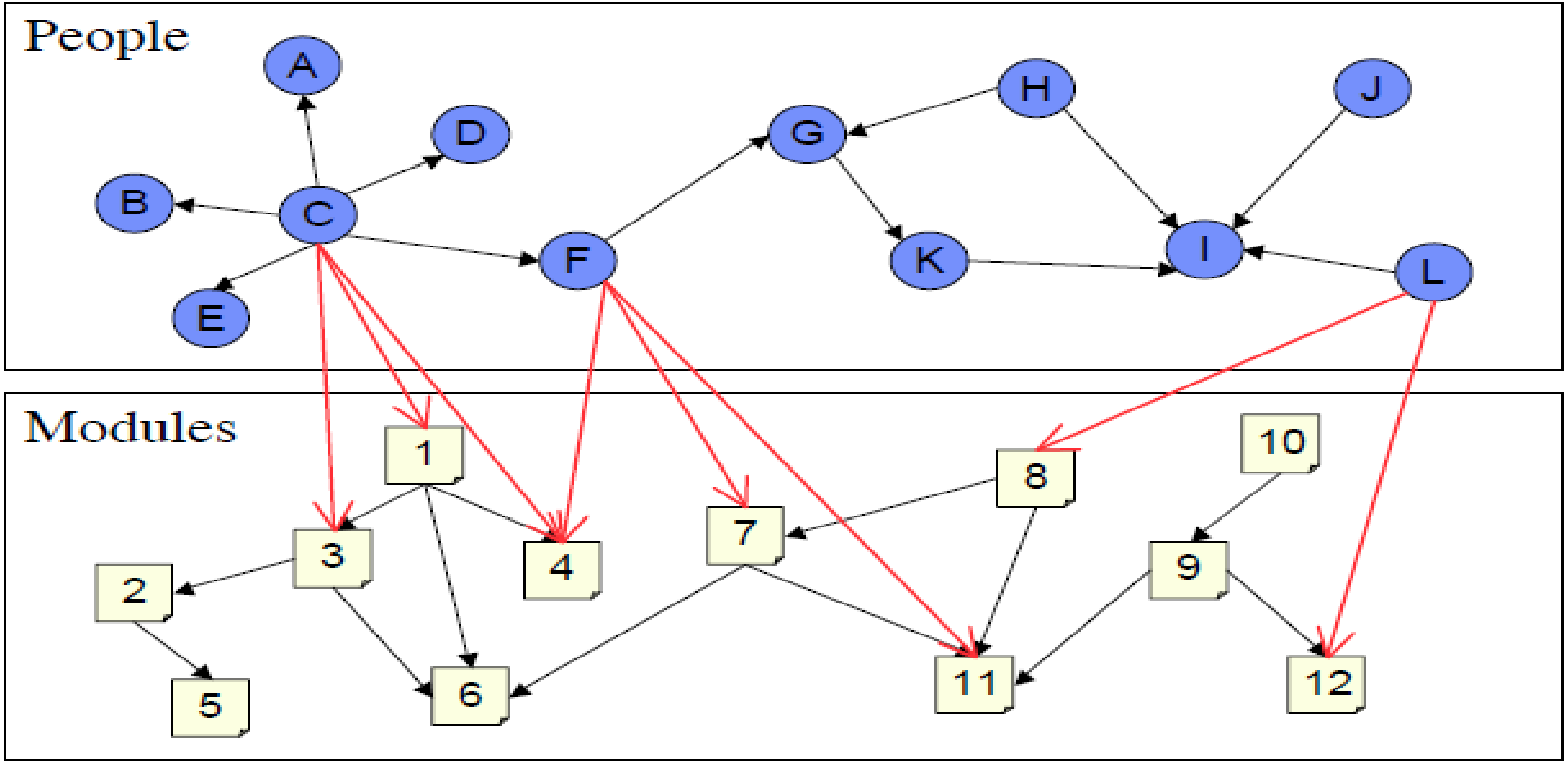


Socio-Technical Congruence



See: Valetto, et al., 2007.

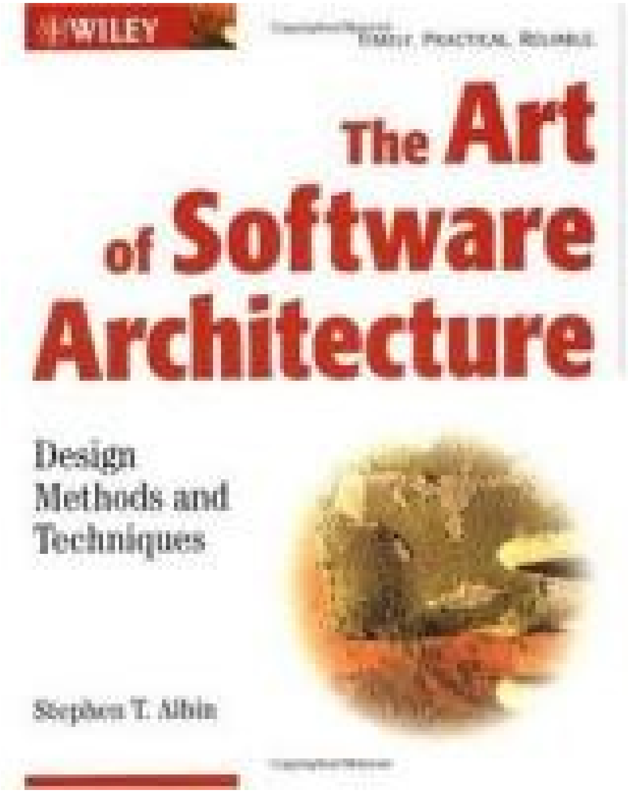
Socio-Technical Congruence



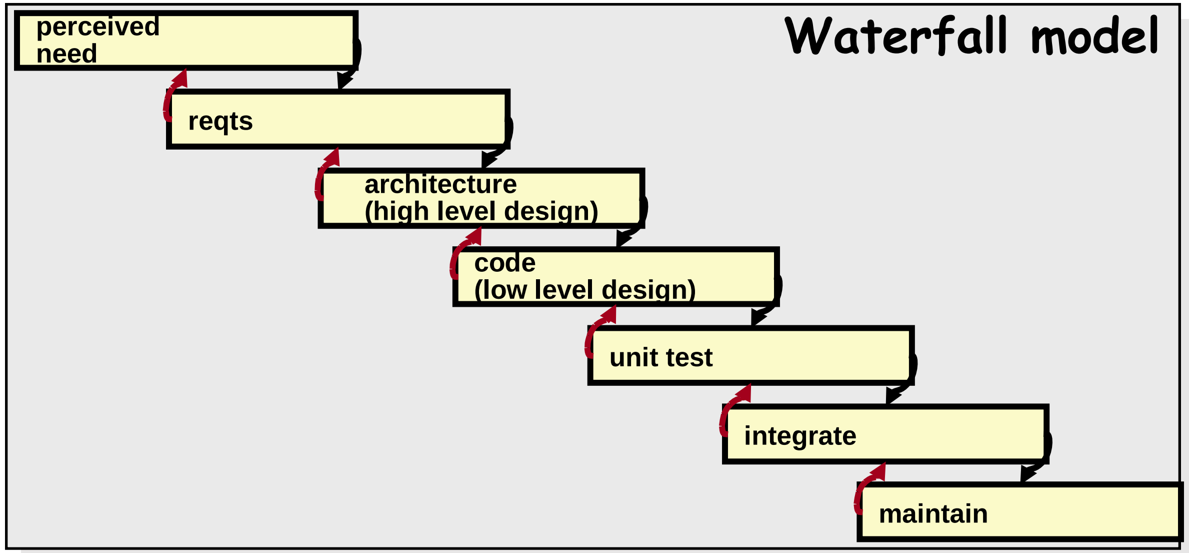
See: Valetto, et al., 2007.

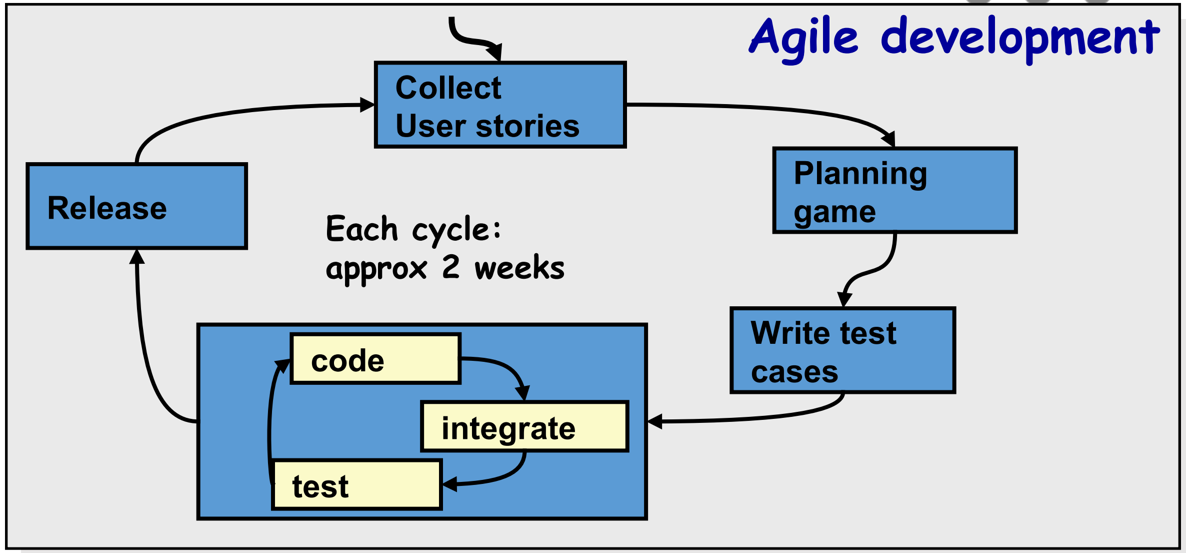
Moving into Design

- Analysis vs. Design
  - Why the distinction?
- Design Processes
  - Logical vs. Physical Design
  - System vs. Detailed Design
- Architectures
  - System Architecture
  - Software Architecture
  - Architectural Patterns
- Useful Notation
  - UML Packages and Dependencies

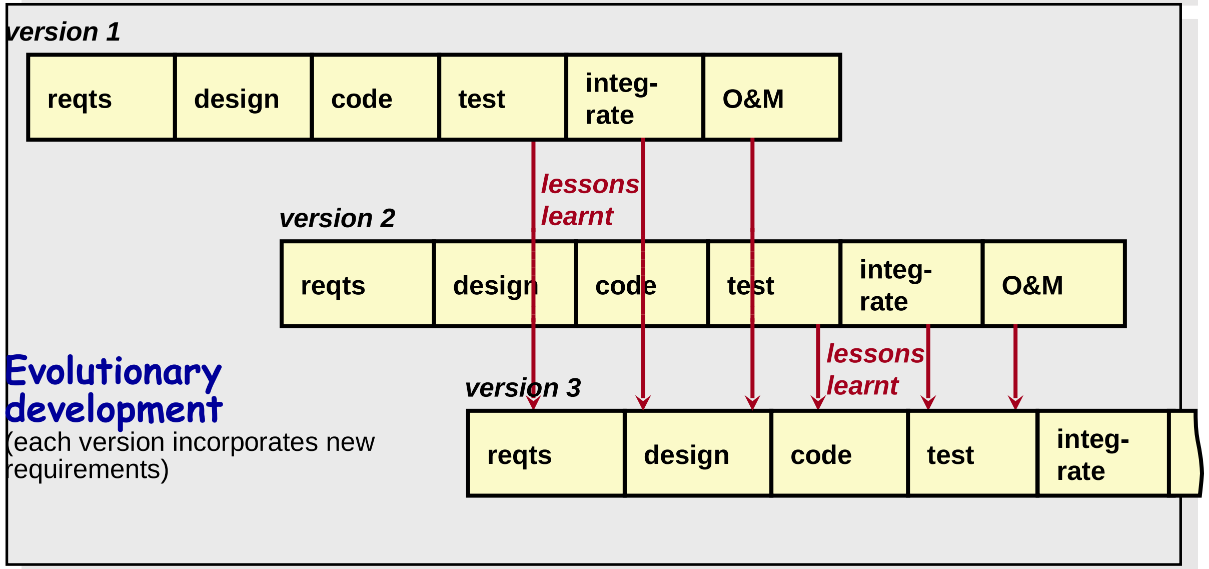

WILEY
The Art of Software Architecture
Design Methods and Techniques
Stephen T. Albin

Refresher: Lifecycle models

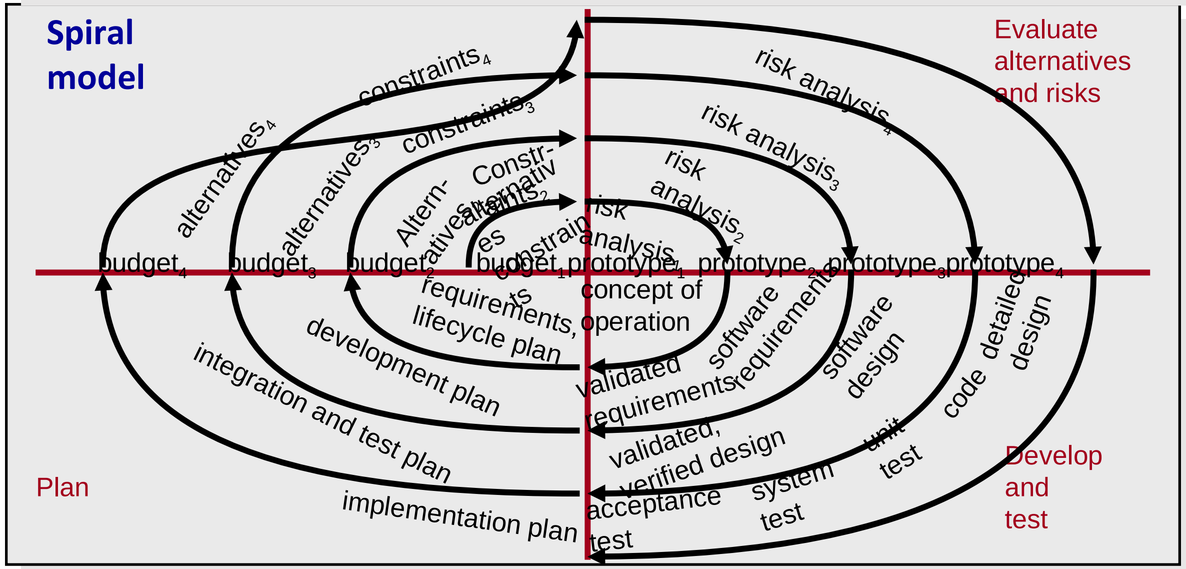

Waterfall model


Agile development

Each cycle: approx 2 weeks


Evolutionary development

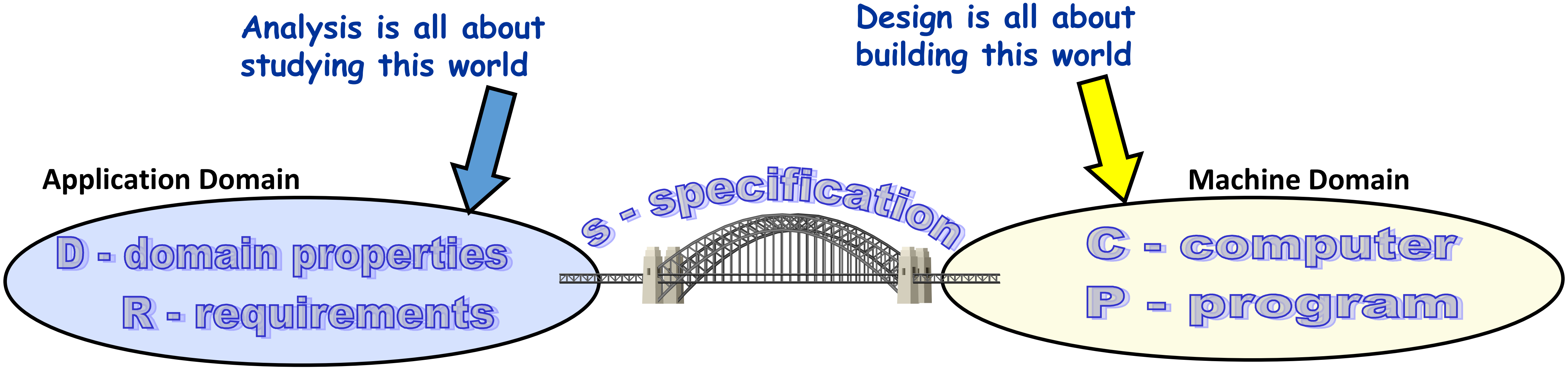
(each version incorporates new requirements)


Spiral model

Analysis vs. Design

- Analysis
  - Asks “what is the problem?”
    - what happens in the current system?
    - what is required in the new system?
  - Results in a detailed understanding of:
    - Requirements
    - Domain Properties
  - Focuses on the way human activities are conducted
- Design
  - Investigates “how to build a solution”
    - How will the new system work?
    - How can we solve the problem that the analysis identified?
  - Results in a solution to the problem
    - A working system that satisfies the requirements
    - Hardware + Software + Peopleware
  - Focuses on building technical solutions
- Separate activities, but not necessarily sequential
  - …and attempting a design usually improves understanding of the problem

Refresher: different worlds



But who builds the bridge?

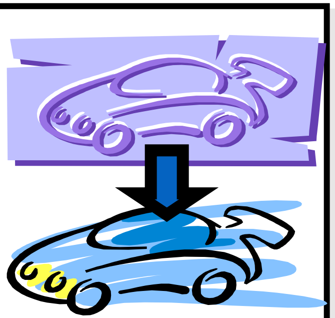
Four design philosophies

Decomposition & Synthesis

- Drivers:
  - Managing complexity
  - Reuse
- Example:
  - Design a car by designing separately the chassis, engine, drivetrain, etc. Use existing *components* where possible

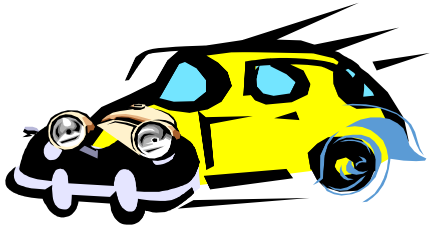
Search

- Drivers
  - Transformation
  - Heuristic Evaluation
- Example:
  - Design a car by *transforming* an initial rough design to get closer and closer to what is desired



Negotiation

- Drivers
  - Stakeholder Conflicts
  - Dialogue Process
- Example:
  - Design a car by getting *each stakeholder* to suggest (partial) designs, and then compare and discuss them

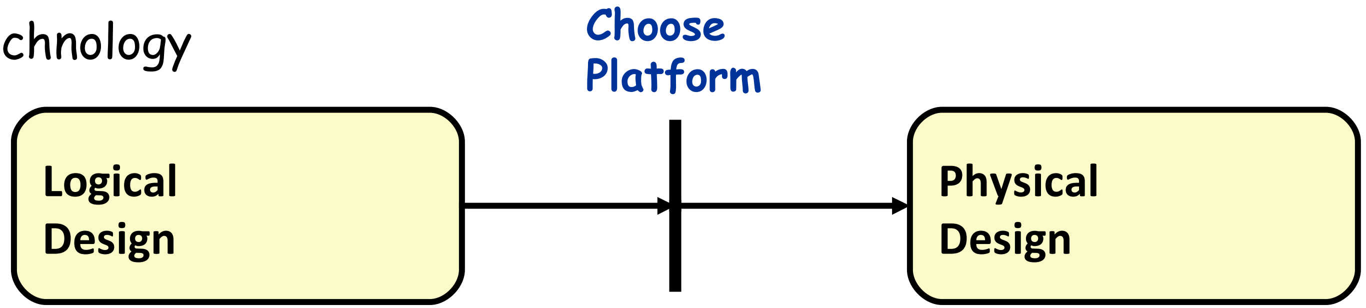


Situated Design

- Drivers
  - Errors in existing designs
  - Evolutionary Change
- Example:
  - Design a car by observing what's wrong with existing cars *as they are used*, and identifying improvements

Logical vs. Physical Design

- Logical Design concerns:
  - Anything that is platform-independent:
    - Interactions between objects
    - Layouts of user interfaces
    - Nature of commands/data passed between subsystems
  - Logical designs are usually portable to different platforms
- Physical Design concerns:
  - Anything that depends on the choice of platform:
    - Distribution of objects/services over networked nodes
    - Choice of programming language and development environment
    - Use of specialized device drivers
    - Choice of database and server technology
    - Services provided by middleware

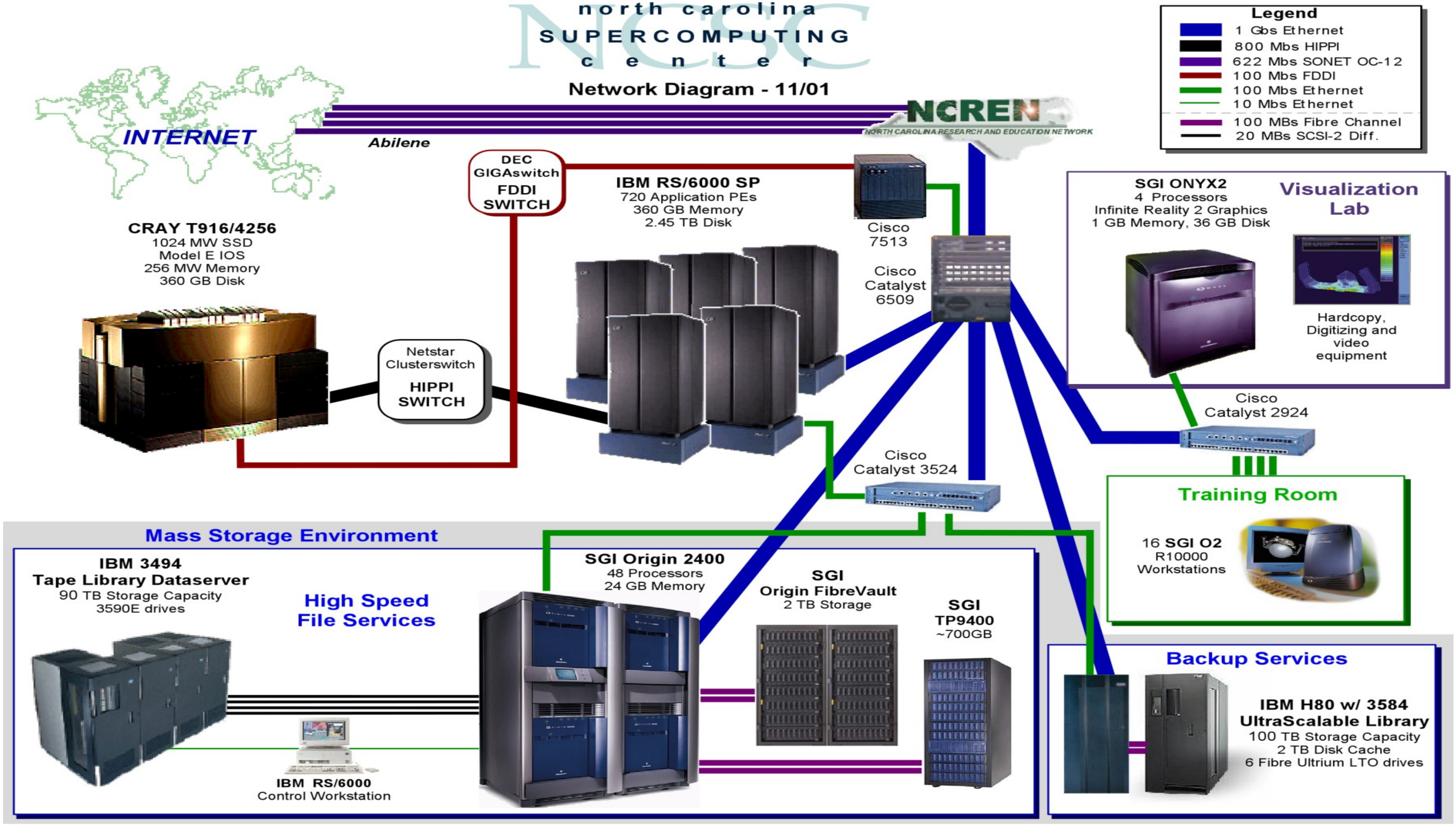


System Design vs. Detailed Design

- System Design
  - Choose a System Architecture
    - Networking infrastructure
    - Major computing platforms
    - Roles of each node (e.g. client-server; clients-broker-servers; peer-to-peer,…)
  - Choose a Software Architecture
  - Identify the subsystems
  - Identify the components and connectors between them
    - Design for modularity to maximize testability and evolveability
    - E.g. Aim for low coupling and high cohesion
- Detailed Design
  - Decide on the formats for data storage
    - E.g. design a data management layer
  - Design the control functions for each component
    - E.g. design an application logic layer
  - Design the user interfaces
    - E.g. design a presentation layer

Global System Architecture

- Choices:
  - Allocates users and other external systems to each node
  - Identify appropriate network topology and technologies
  - Identify appropriate computing platform for each node
- Example:
  - See next slide…



north carolina SUPERCOMPUTING center

Network Diagram - 11/01

Legend

🟦	1 Gbs Ethernet
■	800 Mbs HIPPI
🟪	622 Mbs SONET OC-12
🟥	100 Mbs FDDI
🟩	100 Mbs Ethernet
—	10 Mbs Ethernet
🟪	100 MBs Fibre Channel
—	20 MBs SCSI-2 Diff.

INTERNET

Abilene

NCREN
NORTH CAROLINA RESEARCH AND EDUCATION NETWORK

DEC
GIGAswitch
FDDI
SWITCH

IBM RS/6000 SP
720 Application PEs
360 GB Memory
2.45 TB Disk

Cisco
7513

Cisco
Catalyst
6509

CRAY T916/4256
1024 MW SSD
Model E IOS
256 MW Memory
360 GB Disk

Netstar
Clusterswitch
HIPPI
SWITCH

SGI ONYX2
4 Processors
Infinite Reality 2 Graphics
1 GB Memory, 36 GB Disk

Visualization Lab

Hardcopy, Digitizing and video equipment

Cisco
Catalyst 2924

Cisco
Catalyst 3524

Training Room

16 SGI O2
R10000
Workstations

Mass Storage Environment

IBM 3494
Tape Library Dataserver
90 TB Storage Capacity
3590E drives

High Speed
File Services

SGI Origin 2400
48 Processors
24 GB Memory

SGI
Origin FibreVault
2 TB Storage

SGI
TP9400
~700GB

IBM RS/6000
Control Workstation

Backup Services

IBM H80 w/ 3584
UltraScalable Library
100 TB Storage Capacity
2 TB Disk Cache
6 Fibre Ultrium LTO drives

System Architecture Questions

- Key questions for choosing platforms:
  - What hardware resources are needed?
    - CPU, memory size, memory bandwidth, I/O, disk space, etc.
  - What software/OS resources are needed?
    - application availability, OS scalability
  - What networking resources are needed?
    - network bandwidth, latency, remote access.
  - What human resources are needed?
    - OS expertise, hardware expertise, sys admin needs,
    - user training/help desk requirements.
  - What other needs are there?
    - security, reliability, disaster recovery, uptime requirements.
- Key questions constraining the choice:
  - What funding is available?
  - What resources are already available?
    - Existing hardware, software, networking
    - Existing staff and their expertise
    - Existing relationships with vendors, resellers, etc.

Data Management Questions

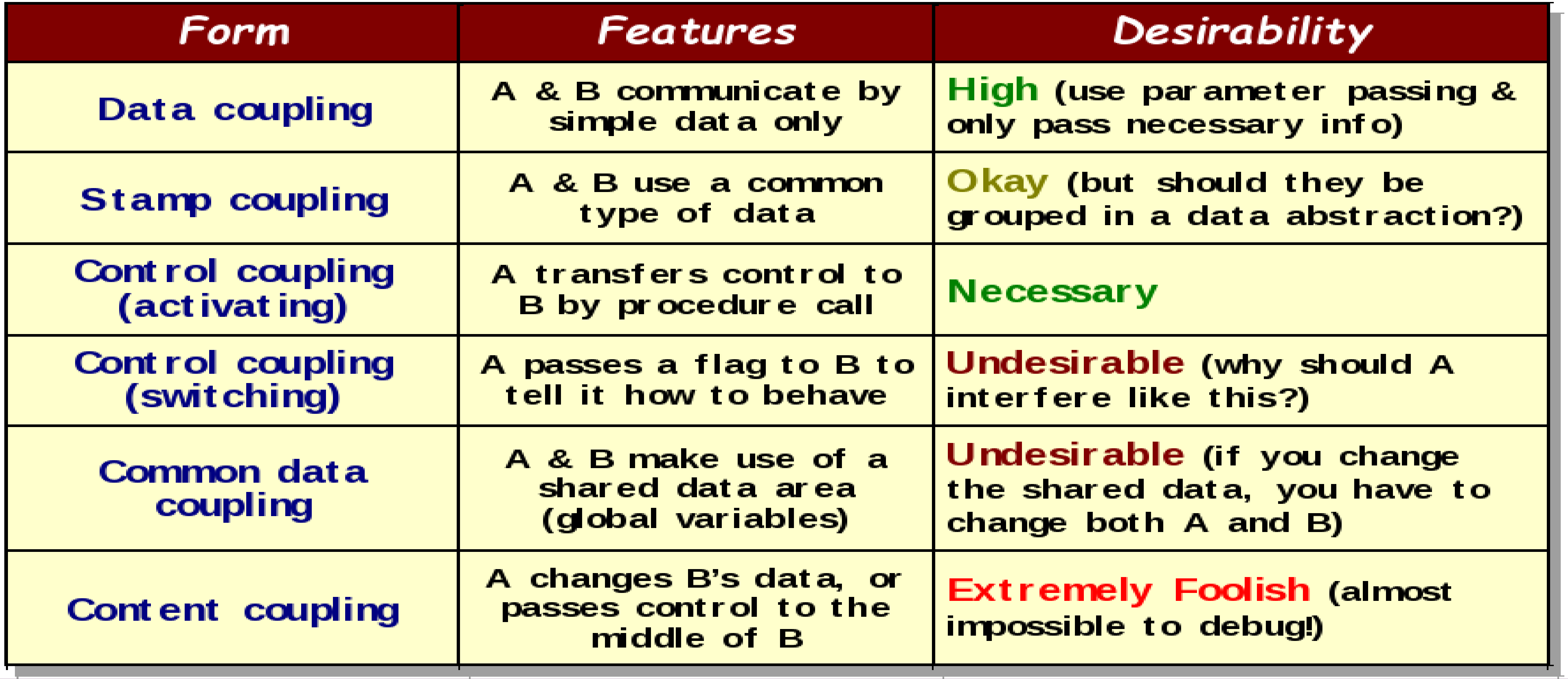
- How is data entry performed?
  - E.g. Keyless Data entry
    - bar codes; Optical Character Recognition (OCR)
  - E.g. Import from other systems
    - Electronic Data Interchange (EDI), Data interchange languages,…
- What kinds of data persistence is needed?
  - Is the operating system's basic file management sufficient?
  - Is object persistence important?
  - Can we isolate persistence mechanisms from the applications?
- Is a Database Management System (DBMS) needed?
  - Is data accessed at a fine level of detail
    - E.g. do users need a query language?
  - Is sophisticated indexing required?
  - Is there a need to move complex data across multiple platforms?
    - Will a data interchange language suffice?
  - Is there a need to access the data from multiple platforms?

Software Architecture

- A software architecture defines:
    - the components of the software system
    - how the components use each other’s functionality and data
    - How control is managed between the components
- An example: client-server
    - Servers provide some kind of service; clients request and use services
    - applications are located with clients
        - E.g. running on PCs and workstations;
    - data storage is treated as a server
        - E.g. using a DBMS such as DB2, Ingres, Sybase or Oracle
        - Consistency checking is located with the server
    - Advantages:
        - Breaks the system into manageable components
        - Makes the control and data persistence mechanisms clearer
    - Variants:
        - Thick clients have their own services, thin ones get everything from servers
    - Note: Are we talking about logical (s/w) or physical (h/w) architecture?

Coupling

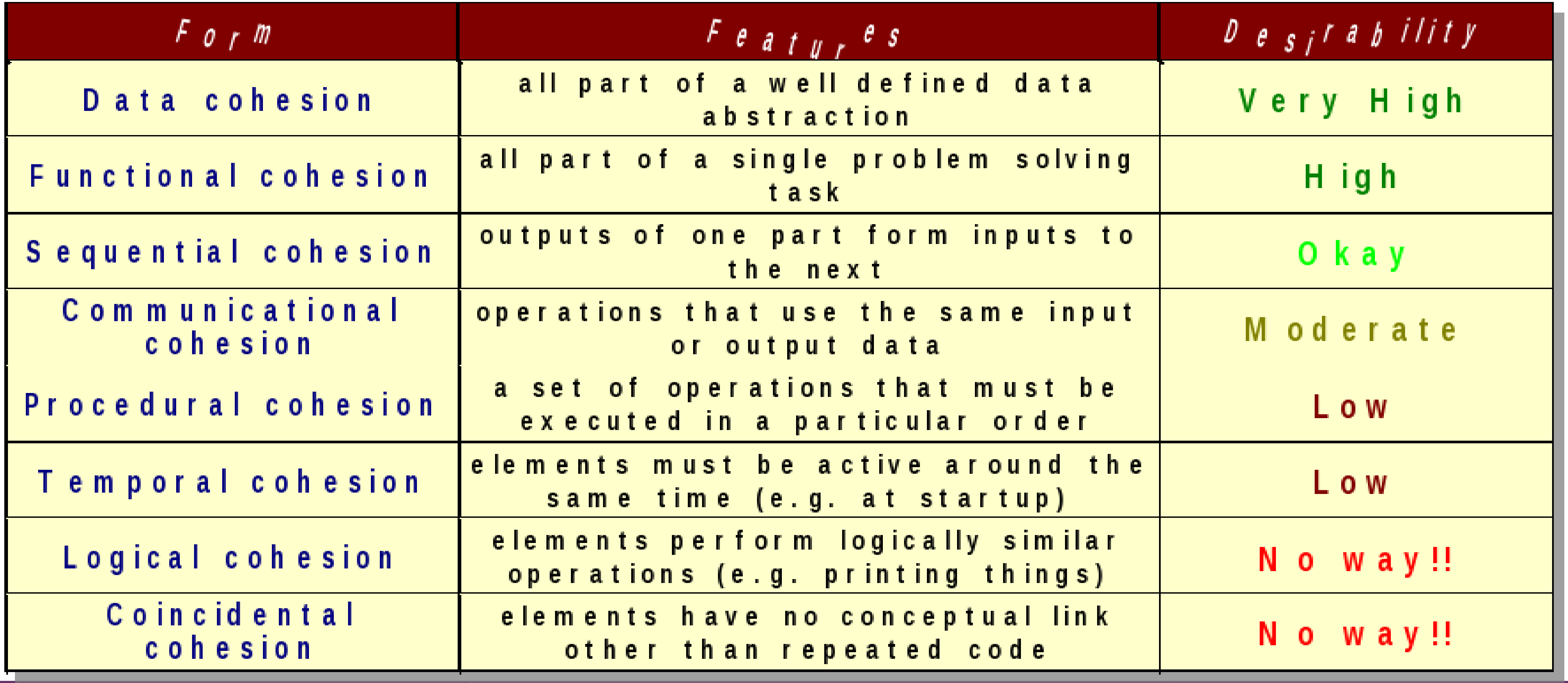
Given two units *(e.g. methods, classes, modules, …)*, A and B:



Form	Features	Desirability
Data coupling	A & B communicate by simple data only	High (use parameter passing & only pass necessary info)
Stamp coupling	A & B use a common type of data	Okay (but should they be grouped in a data abstraction?)
Control coupling (activating)	A transfers control to B by procedure call	Necessary
Control coupling (switching)	A passes a flag to B to tell it how to behave	Undesirable (why should A interfere like this?)
Common data coupling	A & B make use of a shared data area (global variables)	Undesirable (if you change the shared data, you have to change both A and B)
Content coupling	A changes B's data, or passes control to the middle of B	Extremely Foolish (almost impossible to debug!)

Cohesion

How well do the contents of an object *(module, package,…)* go together?

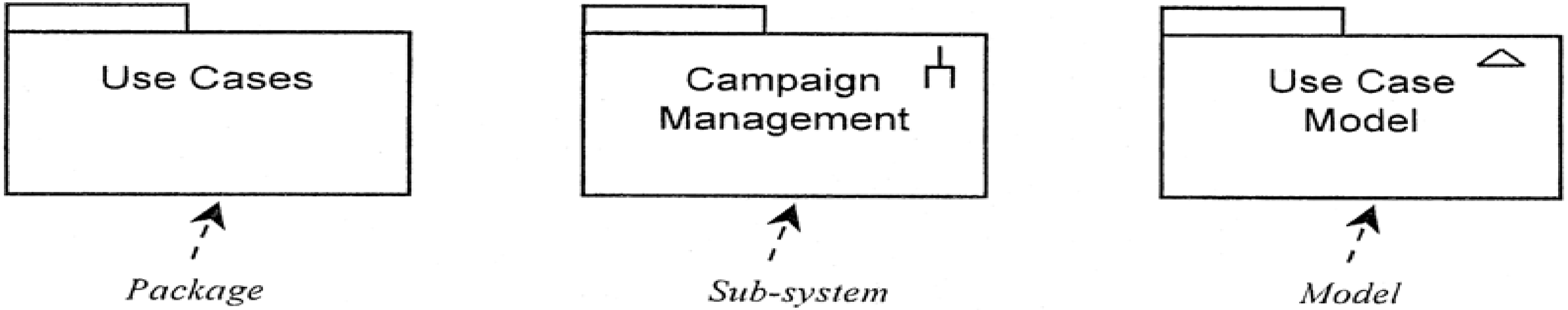


Form	Features	Desirability
Data cohesion	all part of a well defined data abstraction	Very High
Functional cohesion	all part of a single problem solving task	High
Sequential cohesion	outputs of one part form inputs to the next	Okay
Communicational cohesion	operations that use the same input or output data	Moderate
Procedural cohesion	a set of operations that must be executed in a particular order	Low
Temporal cohesion	elements must be active around the same time (e.g. at startup)	Low
Logical cohesion	elements perform logically similar operations (e.g. printing things)	No way!!
Coincidental cohesion	elements have no conceptual link other than repeated code	No way!!

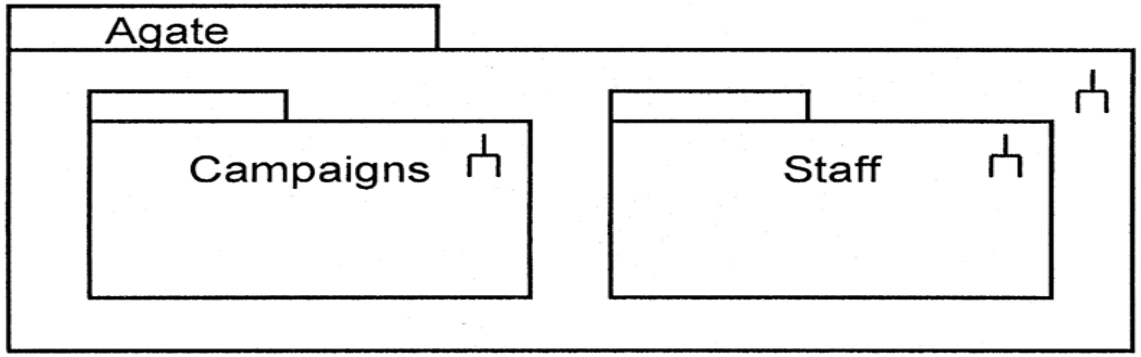
UML Packages

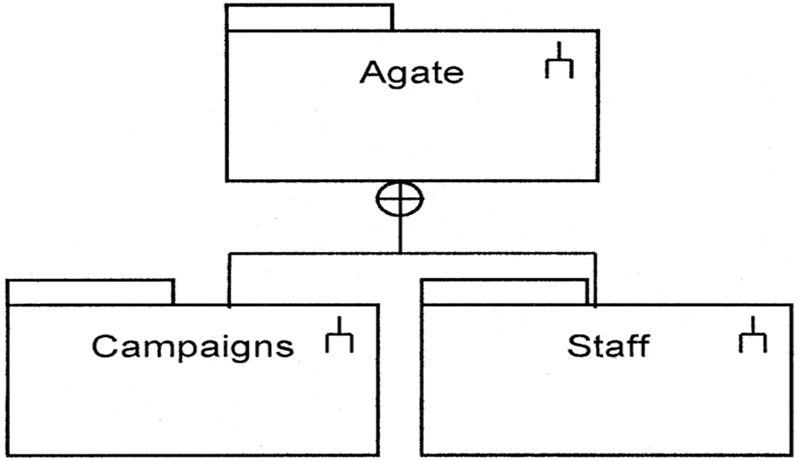
- We need to represent our architectures
  - UML elements can be grouped together in packages
  - Elements of a package may be:
    - other packages (representing subsystems or modules);
    - classes;
    - models (e.g. use case models, interaction diagrams, statechart diagrams, etc)
  - Each element of a UML model is owned by a single package
  - Packages need not correspond to elements of the analysis or the design
    - they are a convenient way of grouping other elements together
- Criteria for decomposing a system into packages:
  - Ownership
    - who is responsible for working on which diagrams
  - Application
    - each problem has its own obvious partitions;
  - Clusters of classes with strong cohesion
    - e.g., course, course description, instructor, student,…
  - Or use an architectural pattern to help find a suitable decomposition

Package notation

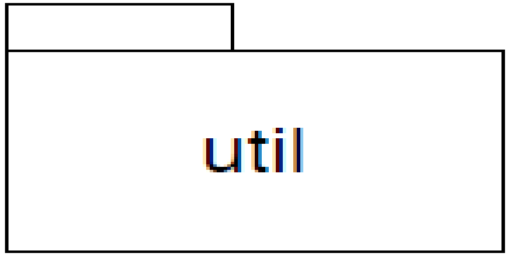


- 2 alternatives for showing package containment:

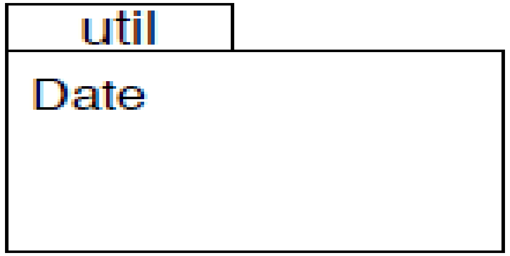




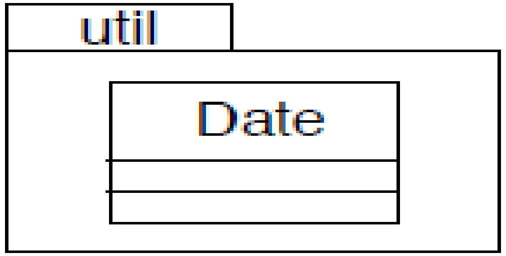
Package notation



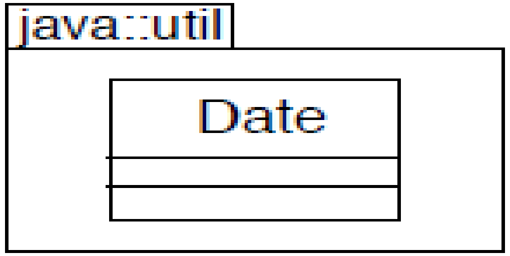
named package



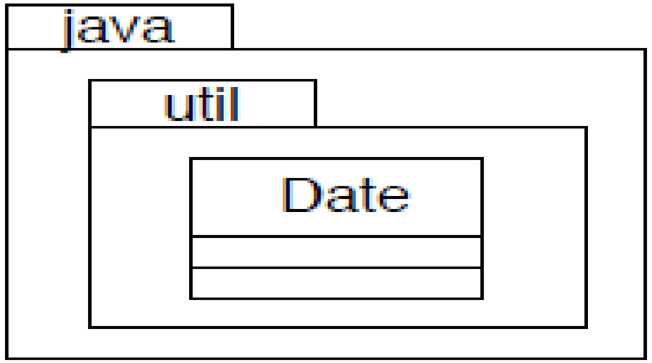
package with list of contained classes



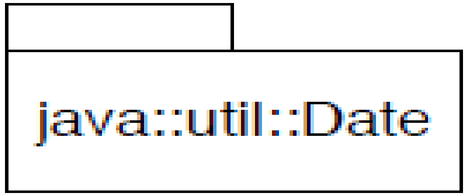
package containing a class diagram



package with qualified name



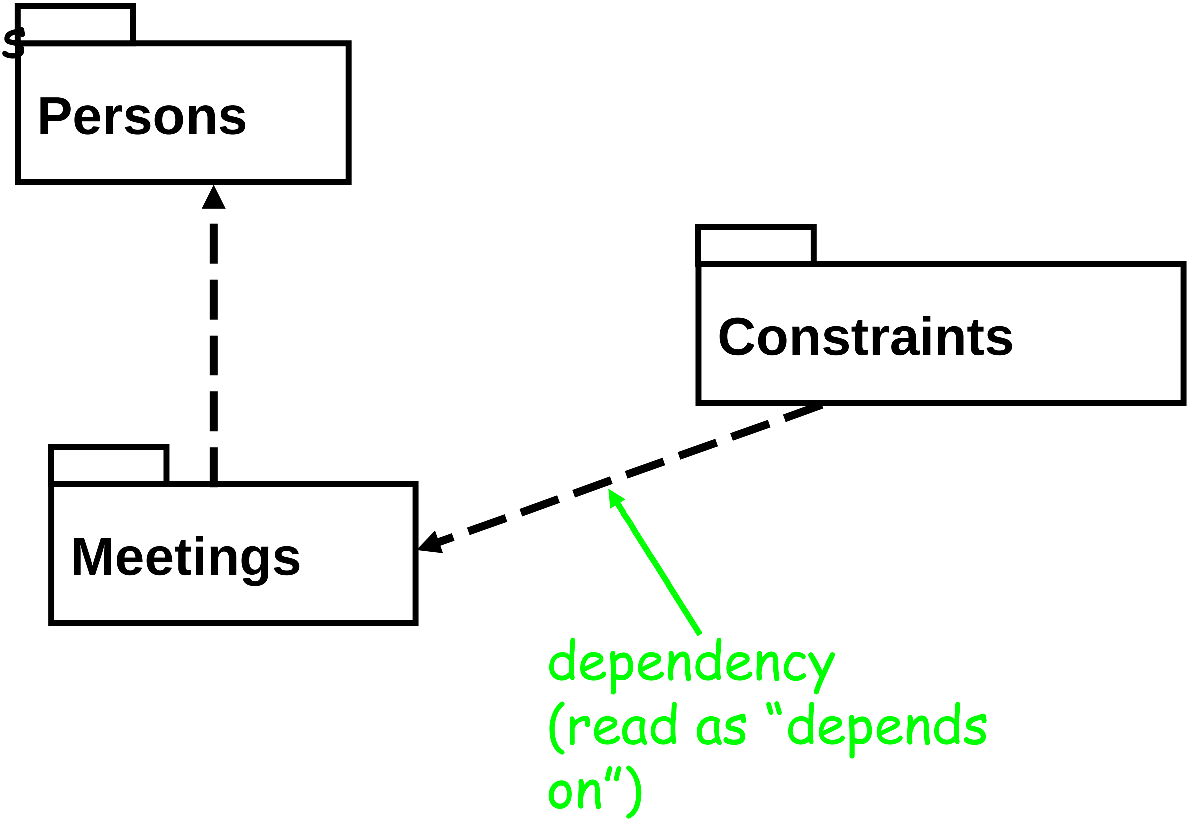
nested packages



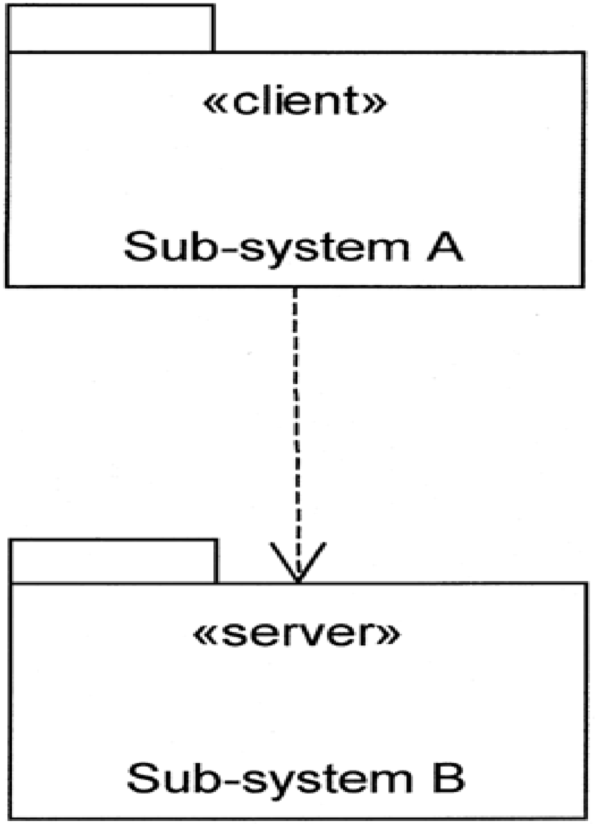
package with fully qualified name

Package Diagrams

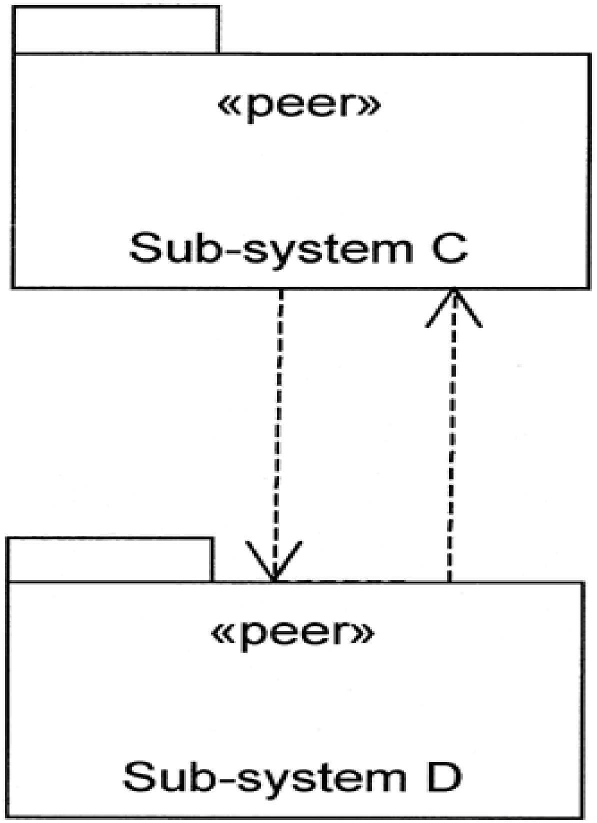
- Dependencies:
  - Similar to compilation dependencies
  - Captures a high-level view of coupling between packages:
    - If you change a class in one package, you may have to change something in packages that depend on it
- A good architecture minimizes dependencies
  - Fewer dependencies means lower coupling
  - Dependency cycles are especially undesirable



…Dependency Cycles



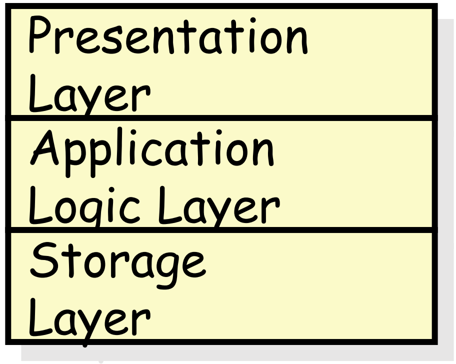
The server sub-system does not depend on the client sub-system and is not affected by changes to the client's interface.

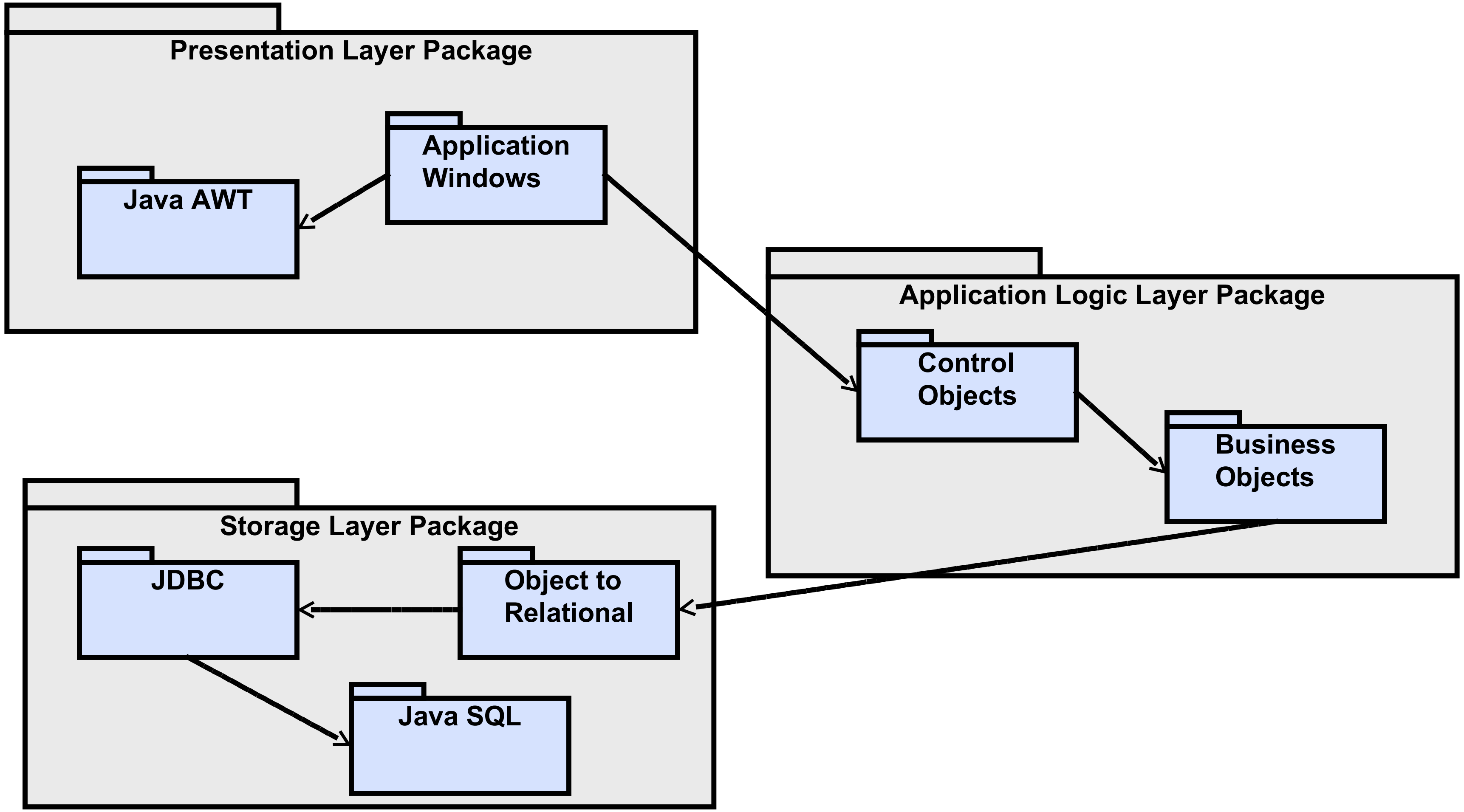


Each peer sub-system depends on the other and each is affected by changes in the other's interface.

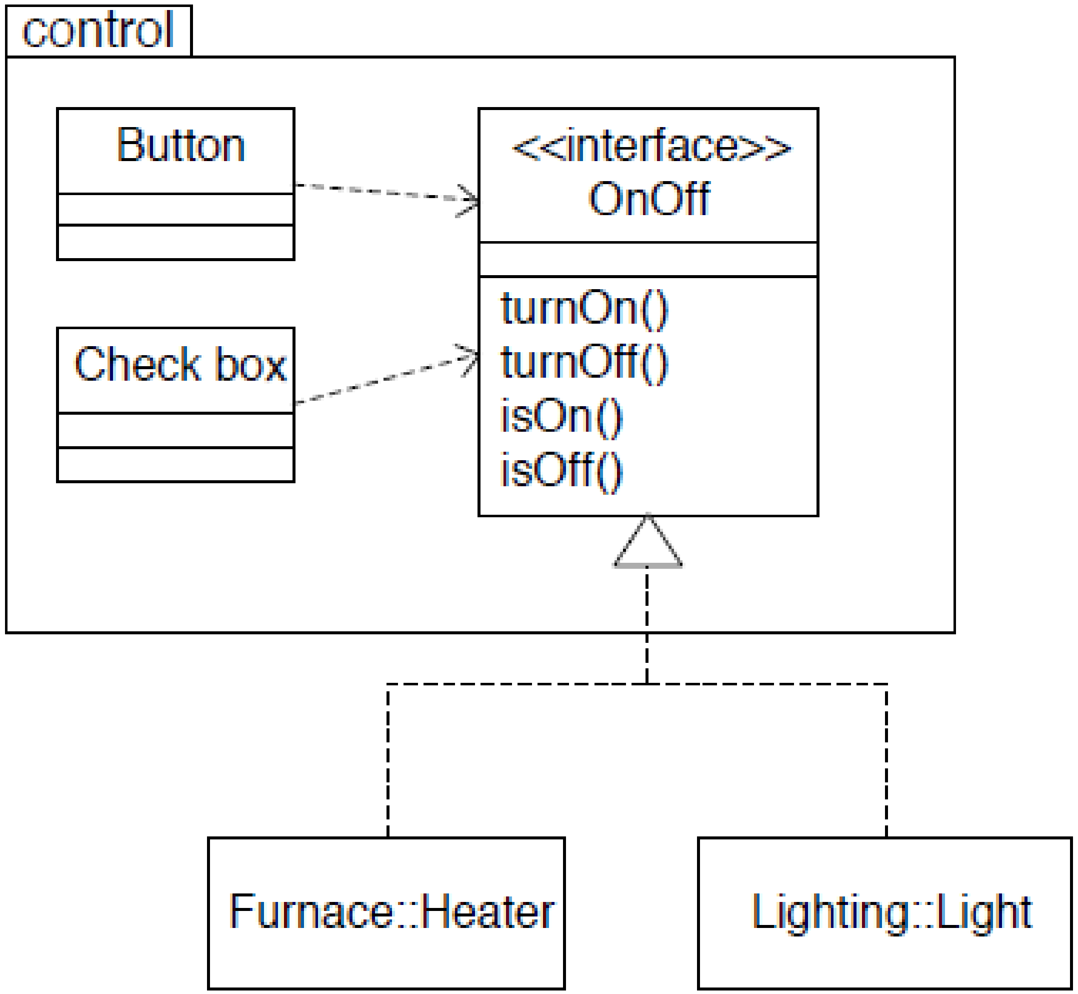
Architectural Patterns

E.g. 3 layer architecture:





Towards component-based design

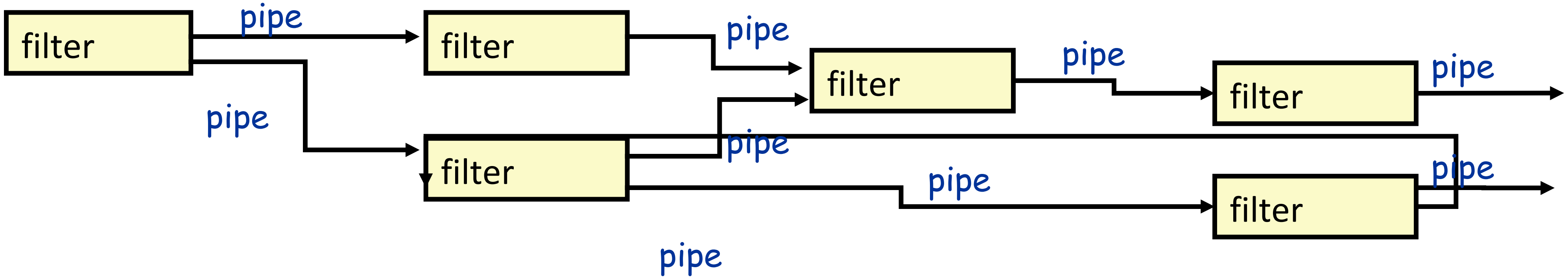


Software Architectures

- Architectural Styles
  - Pipe and filter
  - Object oriented:
    - Client-Server; Object Broker
  - Event based
  - Layered:
    - Designing Layered Architectures
  - Repositories:
    - Blackboard, MVC
  - Process control

Pipe-and-filter

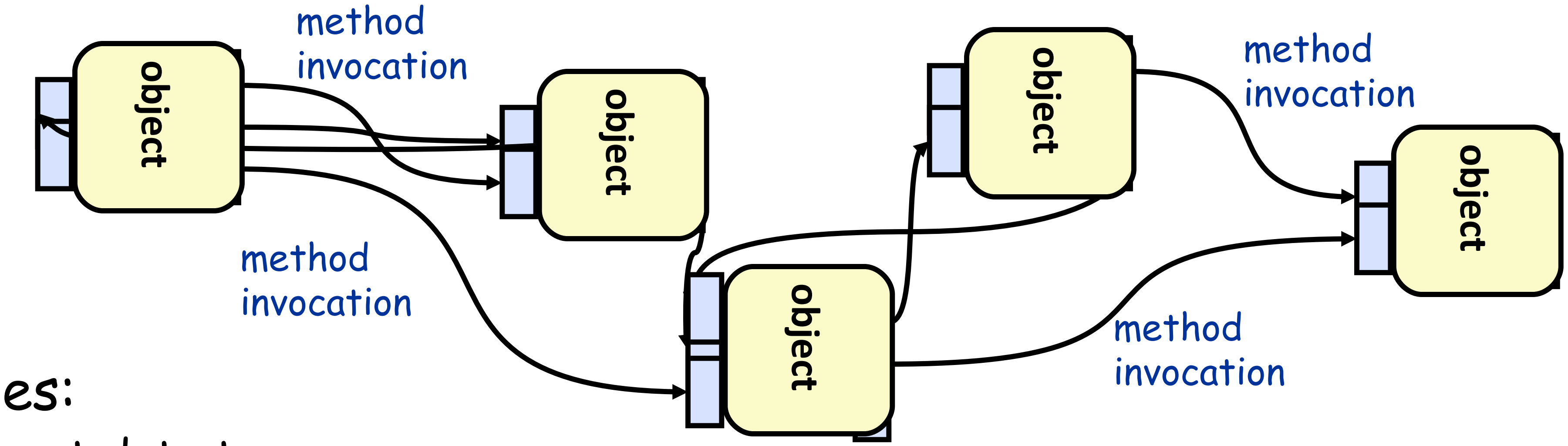
Source: *Adapted from Shaw & Garlan 1996, p21-2. See also van Vliet, 1999 Pp266-7 and p279*



- Examples:
  - UNIX shell commands
  - Compilers:
    - Lexical Analysis -> parsing -> semantic analysis -> code generation
  - Signal Processing
- Interesting properties:
  - filters don’t need to know anything about what they are connected to
  - filters can be implemented in parallel
  - behaviour of the system is the composition of behaviour of the filters
    - specialized analysis such as throughput and deadlock analysis is possible

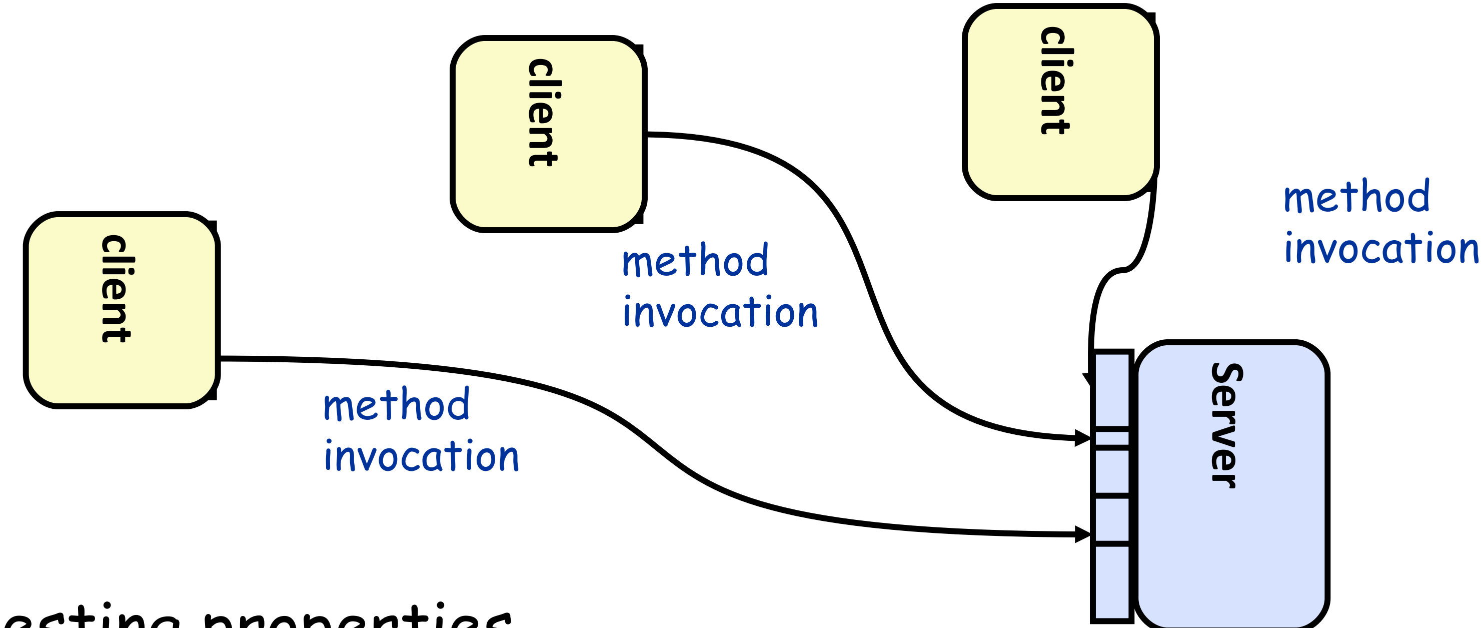
Object Oriented Architectures

Source: *Adapted from Shaw & Garlan 1996, p22-3.*



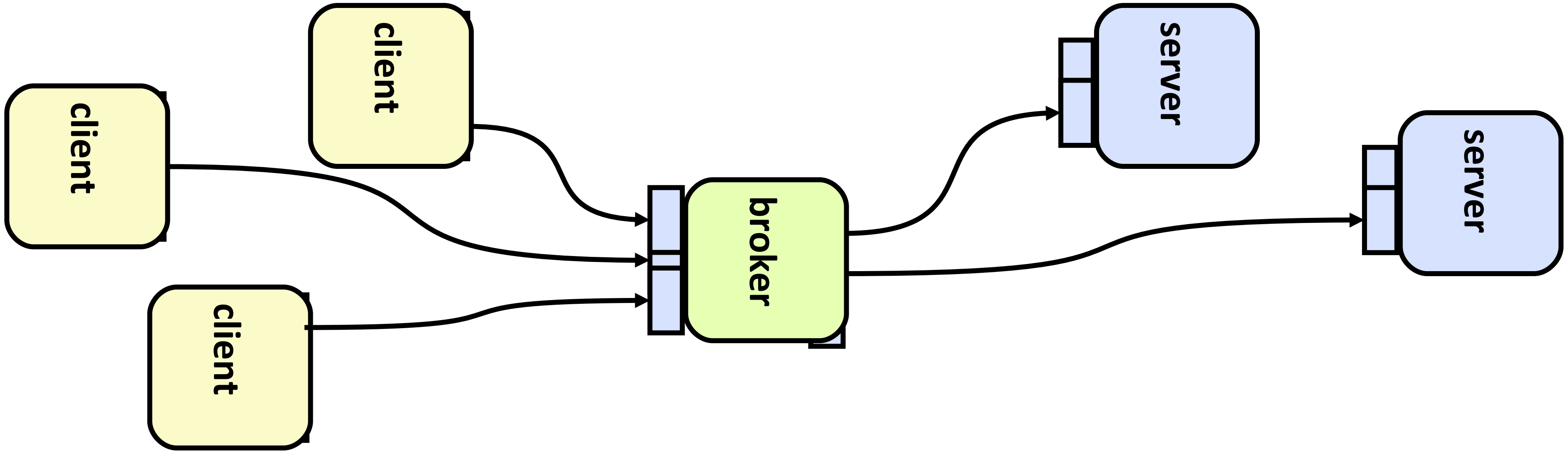
- Examples:
  - abstract data types
- Interesting properties
  - data hiding (internal data representations are not visible to clients)
  - can decompose problems into sets of interacting agents
  - can be multi-threaded or single thread
- Disadvantages
  - objects must know the identity of objects they wish to interact with

Variant 1: Client Server



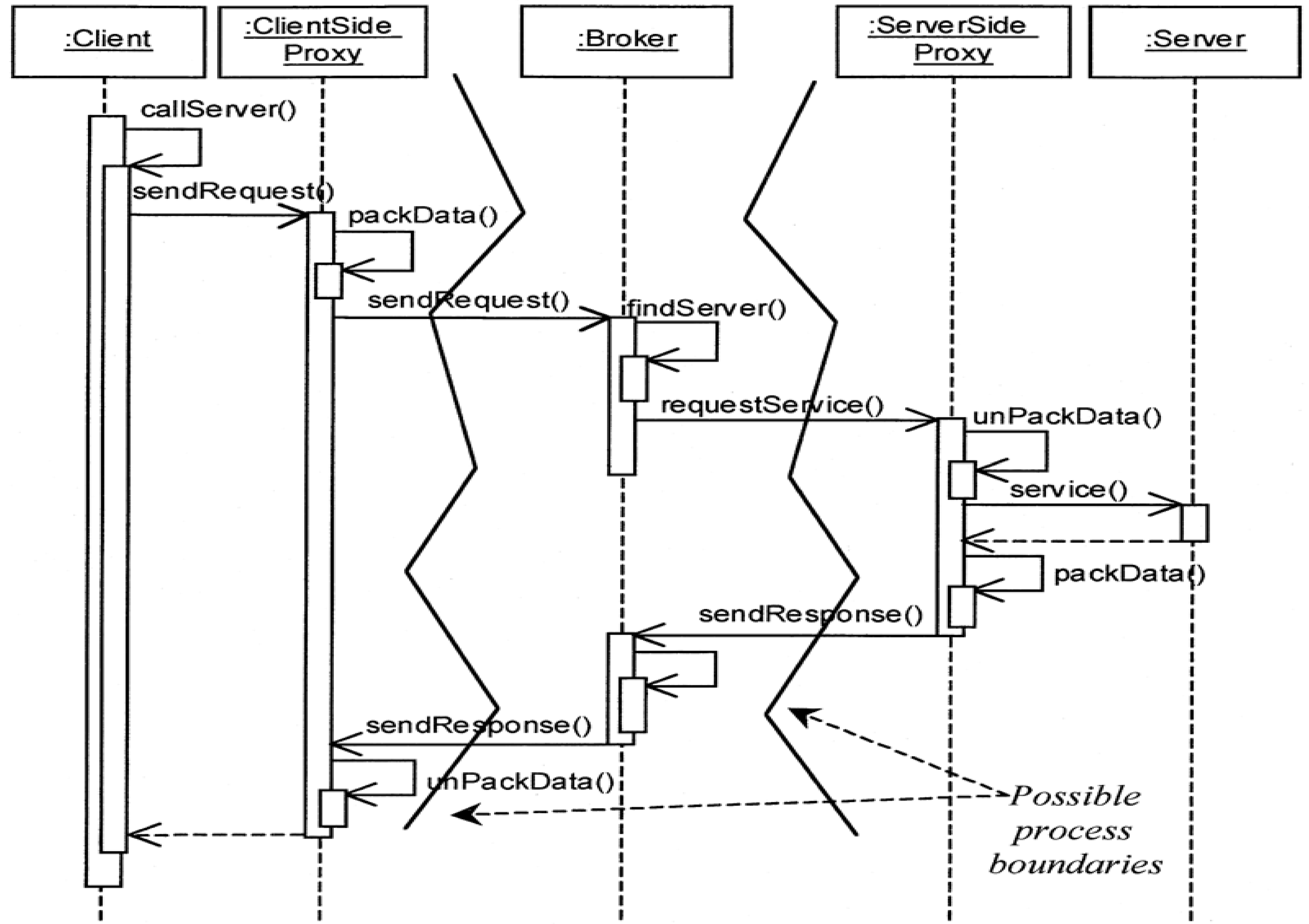
- Interesting properties
  - Is a special case of the previous pattern object oriented architecture
  - Clients do not need to know about one another
- Disadvantages
  - Client objects must know the identity of the server

Variant 2: Object Brokers



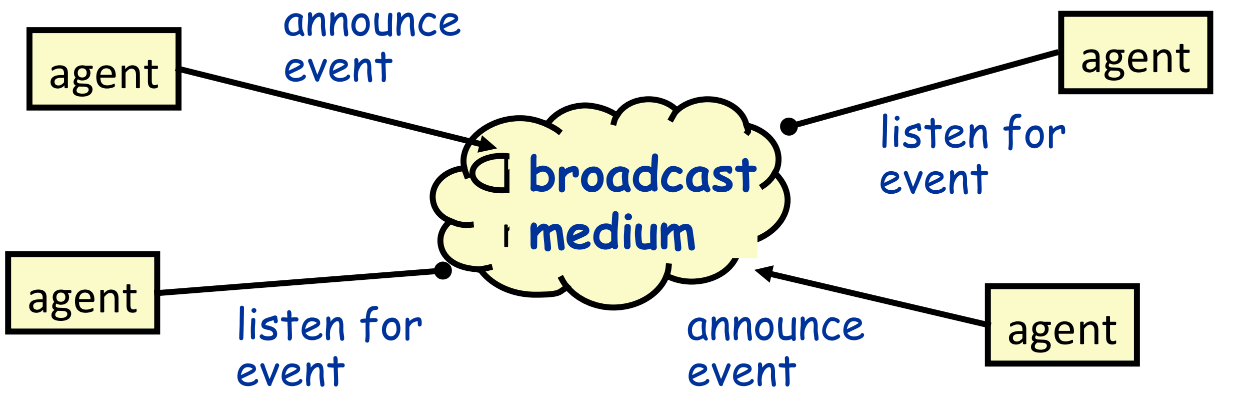
- Interesting properties
  - Adds a broker between the clients and servers
  - Clients no longer need to know which server they are using
  - Can have many brokers, many servers.
- Disadvantages
  - Broker can become a bottleneck
  - Degraded performance

Broker Architecture Example



Event based (implicit invocation)

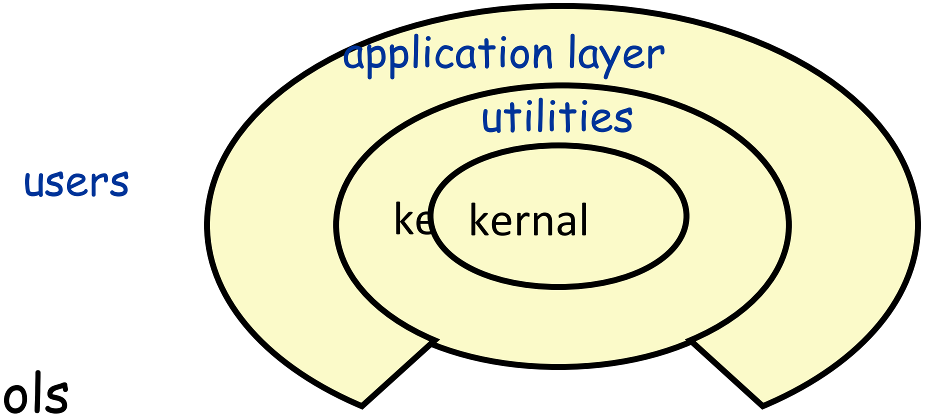
Source: *Adapted from Shaw & Garlan 1996, p23-4. See also van Vliet, 1999 Pp264-5 and p278*



- Examples
  - debugging systems (listen for particular breakpoints)
  - database management systems (for data integrity checking)
  - graphical user interfaces
- Interesting properties
  - announcers of events don't need to know who will handle the event
  - Supports re-use, and evolution of systems (add new agents easily)
- Disadvantages
  - Components have no control over ordering of computations

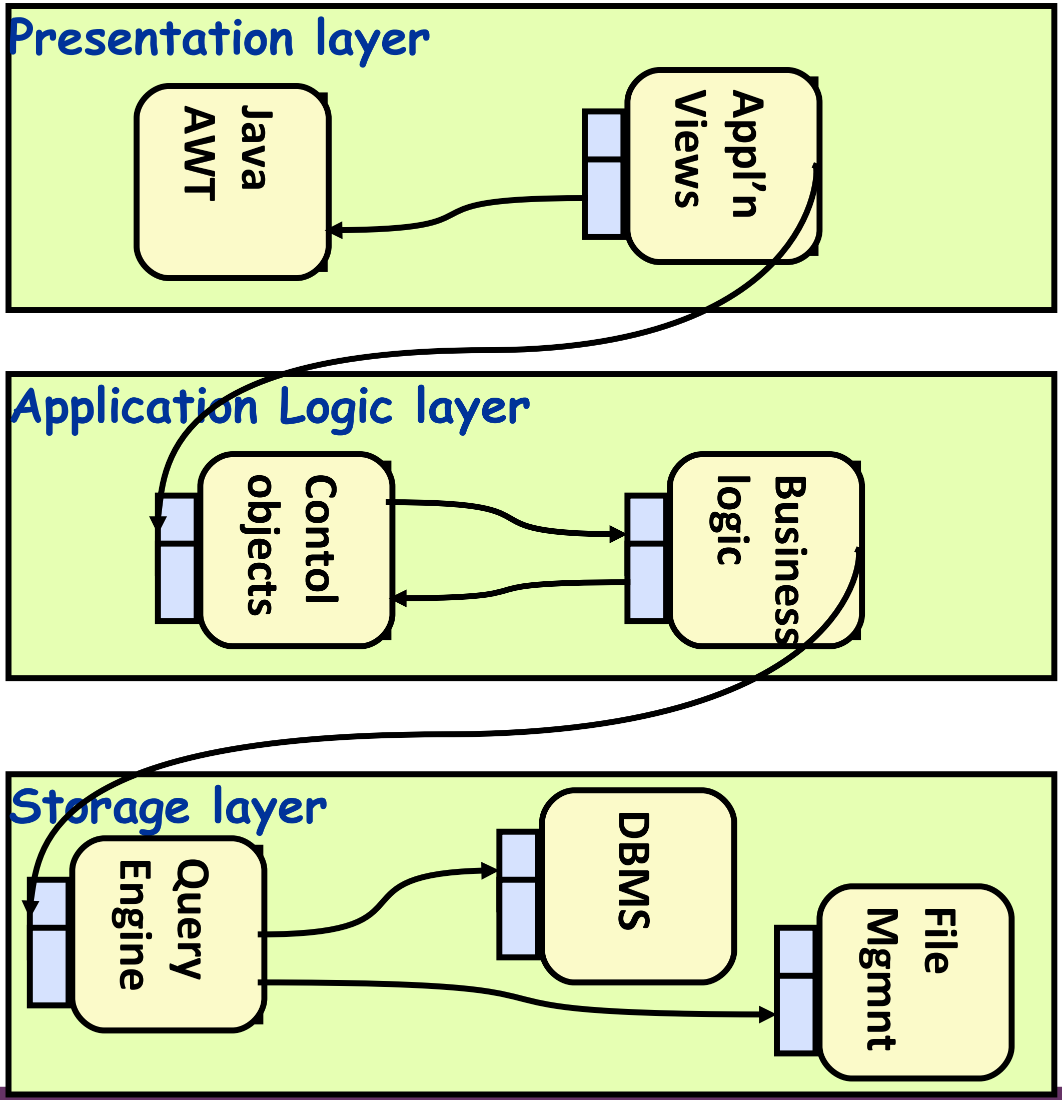
Layered Systems

Source: *Adapted from Shaw & Garlan 1996, p25. See also van Vliet, 1999, p281.*



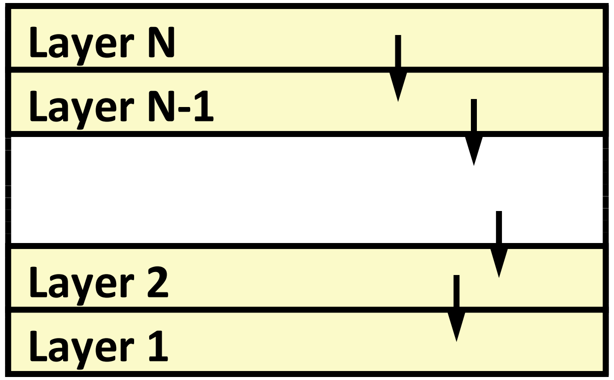
- Examples
  - Operating Systems
  - communication protocols
- Interesting properties
  - Support increasing levels of abstraction during design
  - Support enhancement (add functionality) and re-use
  - can define standard layer interfaces
- Disadvantages
  - May not be able to identify (clean) layers

Variant: 3-layer data access

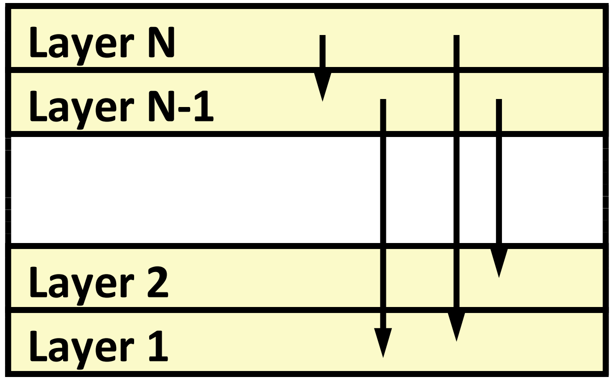


Open vs. Closed Layered Architecture

- closed architecture
  - each layer only uses services of the layer immediately below;
  - Minimizes dependencies between layers and reduces the impact of a change.

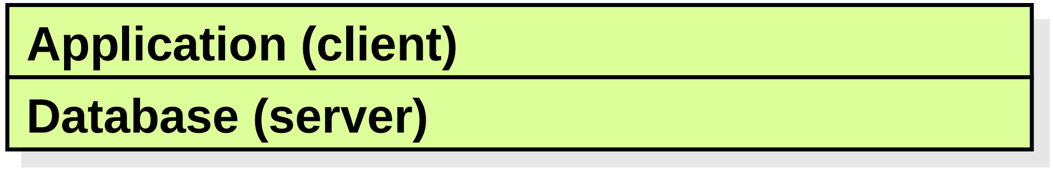


- open architecture
  - a layer can use services from any lower layer.
  - More compact code, as the services of lower layers can be accessed directly
  - Breaks the encapsulation of layers, so increase dependencies between layers



How many layers?

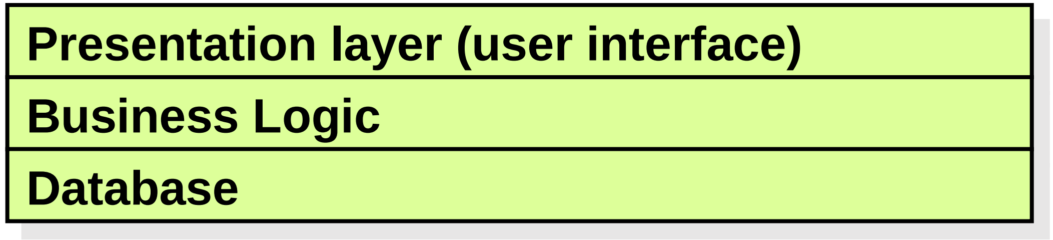
- 2-layers:
  - application layer
  - database layer
  - e.g. simple client-server model



Application (client)

Database (server)

- 3-layers:
  - separate out the business logic
    - helps to make both user interface and database layers modifiable

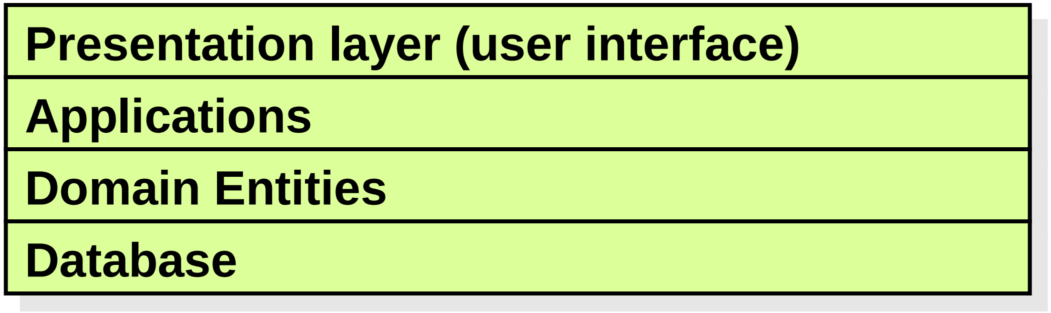


Presentation layer (user interface)

Business Logic

Database

- 4-layers:
  - Separates applications from the domain entities that they use
    - boundary classes in presentation layer
    - control classes in application layer
    - entity classes in domain layer



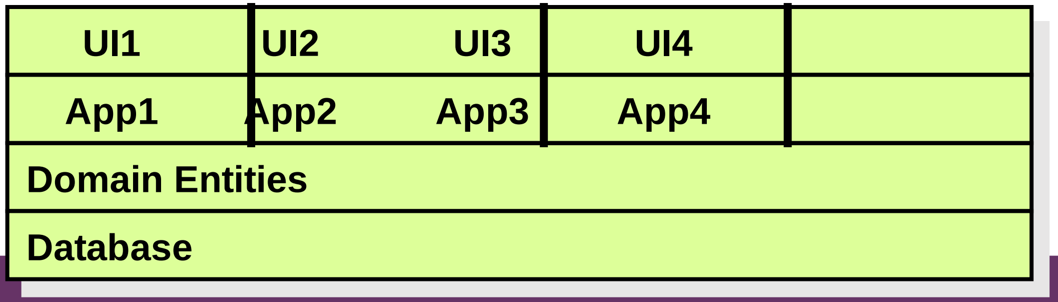
Presentation layer (user interface)

Applications

Domain Entities

Database

- Partitioned 4-layers
  - identify separate applications



UI1 UI2 UI3 UI4

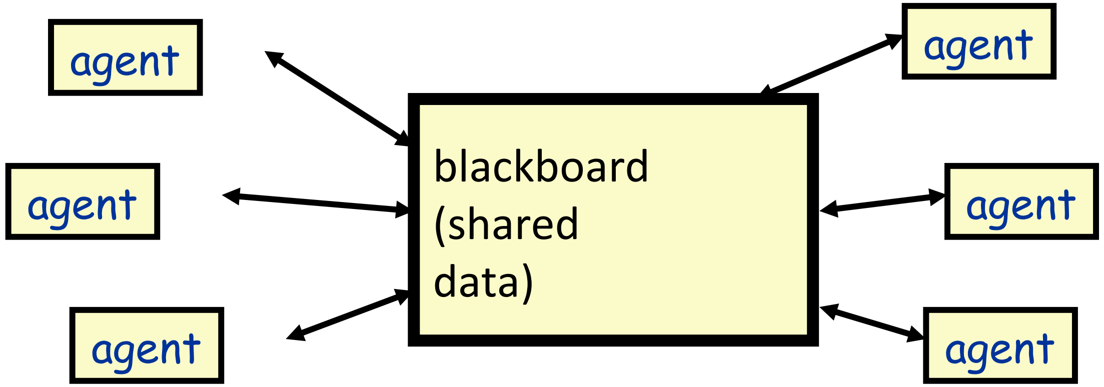
App1 App2 App3 App4

Domain Entities

Database

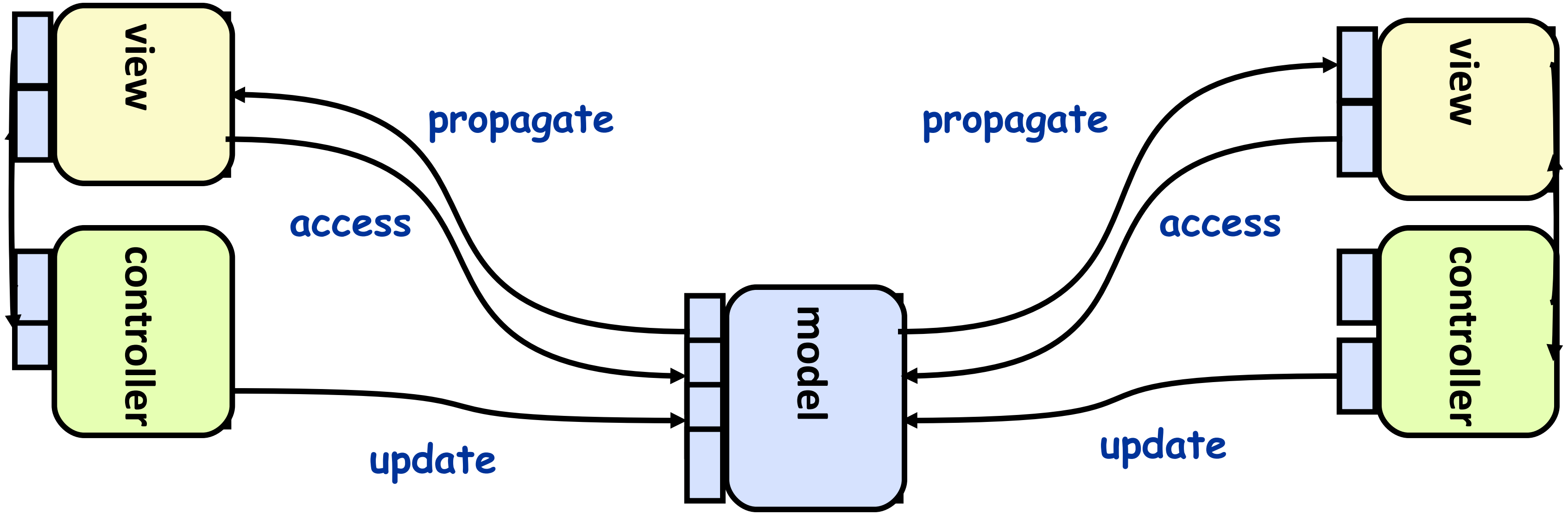
Repositories

Source: *Adapted from Shaw & Garlan 1996, p26-7. See also van Vliet, 1999, p280*



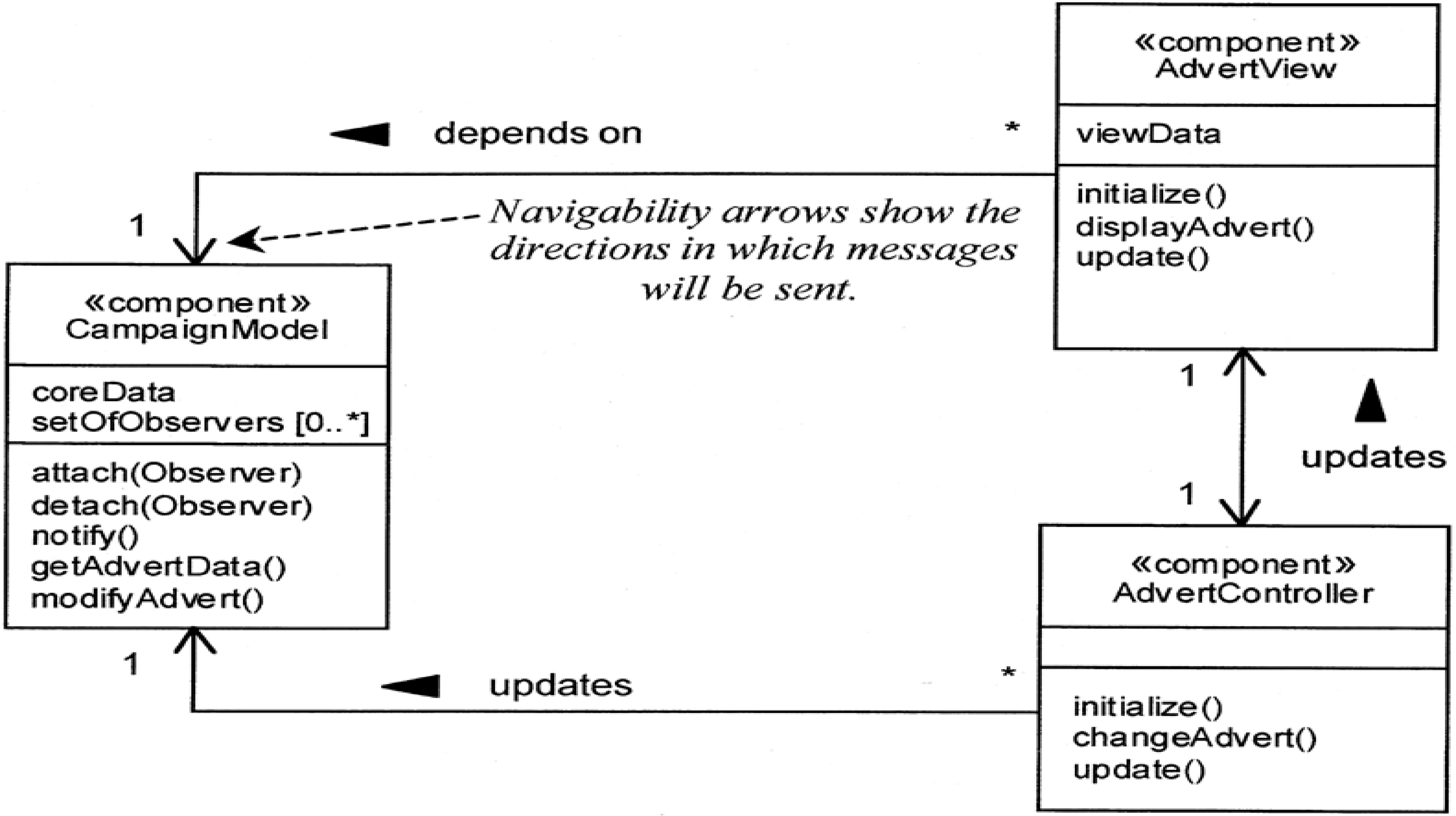
- Examples
  - databases
  - blackboard expert systems
  - programming environments
- Interesting properties
  - can choose where the locus of control is (agents, blackboard, both)
  - reduce the need to duplicate complex data
- Disadvantages
  - blackboard becomes a bottleneck

Variant: Model-View-Controller

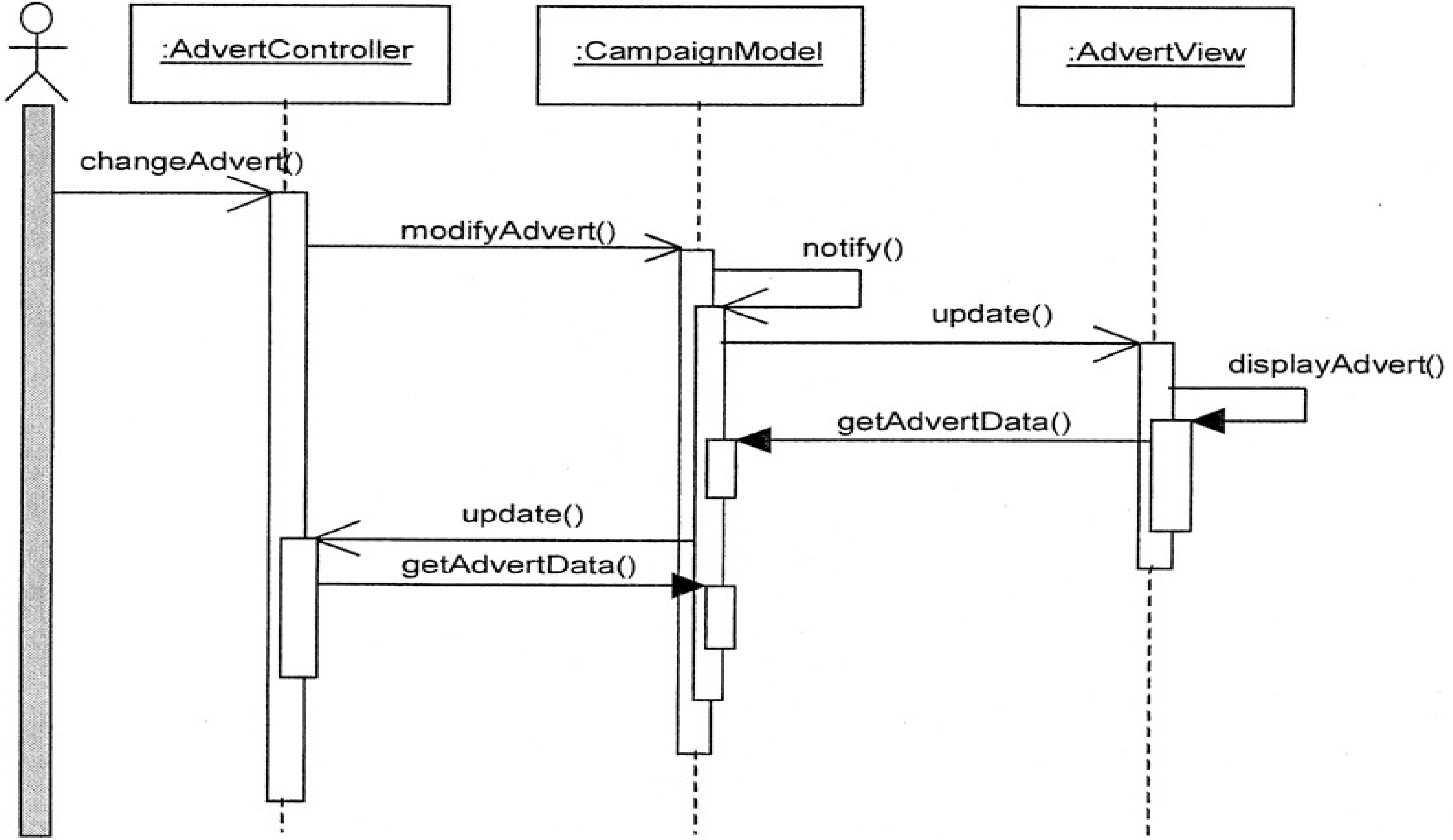


- Properties
  - One central model, many views (viewers)
  - Each view has an associated controller
  - The controller handles updates from the user of the view
  - Changes to the model are propagated to all the views

Model View Controller Example

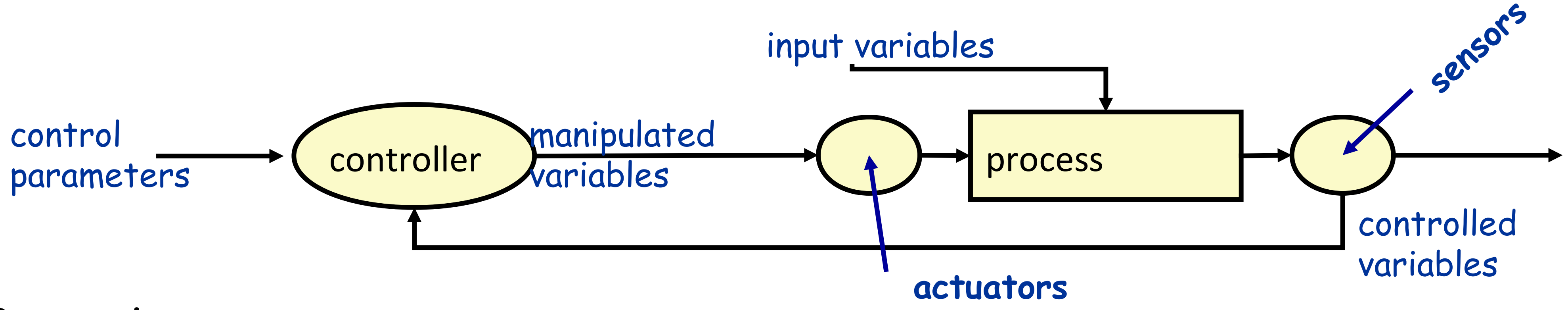


MVC Component Interaction



Process Control

Source: *Adapted from Shaw & Garlan 1996, p27-31.*



- Examples
  - aircraft/spacecraft flight control systems
  - controllers for industrial production lines, power stations, etc.
  - chemical engineering
- Interesting properties
  - separates control policy from the controlled process
  - handles real-time, reactive computations
- Disadvantages
  - Difficult to specify the timing characteristics and response to disturbances

验证非功能性需求
The Architecture Tradeoff Analysis Method

清华大学软件学院 刘璘

Why evaluate an Architecture ?

- All design involves tradeoffs
- A software architecture is the earliest life-cycle artifact that embodies significant design decisions: choices and tradeoffs

The ATAM$^{(sm)}$ is a service marked product of the Software Engineering Institute of Carnegie Mellon University

http://www.sei.cmu.edu/ata/ata_method.html

Purpose of the ATAM

- To assess the consequences of architectural decision tradeoffs in the light of quality attribute requirements
  - Discover Risks
    - Alternatives that might create problems in some future quality attribute
  - Discover Sensitivity Points
    - Alternatives for which a slight change makes a big difference in some quality attribute
  - Discover Tradeoffs
    - Decisions affecting more than one quality attribute
- The point of the ATAM analysis is not to provide precise analysis ---- rather discover risks created by architectural decisions

Purpose of the ATAM cont’d

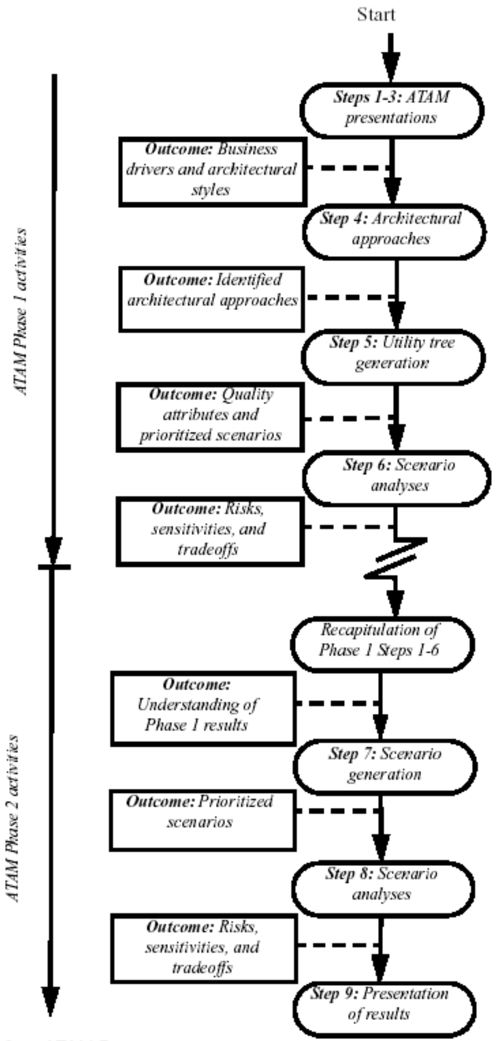
- Discover trends – the correlation between architectural decisions and predictions of system properties
- Discovered risks and then be made subjects of mitigation activities:
    - FURTHER DESIGN
    - FURTHER ANALYSIS
    - PROTOTYPING

Quality Attributes

- The ATAM focuses on quality attribute requirements. Therefore, it is critical to have precise characterizations for each quality attribute.
- Quality attribute characterizations answer the following questions about each attribute:
  - What are the stimuli to which the architecture must respond?
  - What is the measurable or observable manifestation of the quality attribute by which its achievement is judged?
  - What are the key architectural decisions that impact achieving the attribute requirement?

ATAM Steps

- 1. Present ATAM
- 2. Present business drivers
- 3. Present architecture
- 4. Identify architectural styles
- 5. Generate quality attribute utility tree
- 6. Elicit and analyze architectural styles
- 7. Generate seed scenarios
- 8. Brainstorm and prioritize scenarios
- 9. Map scenarios onto styles
- 10. Present out-brief and/or write report



Start

ATAM Phase 1 activities

Steps 1-3: *ATAM presentations*

Outcome: *Business drivers and architectural styles*

Step 4: *Architectural approaches*

Outcome: *Identified architectural approaches*

Step 5: *Utility tree generation*

Outcome: *Quality attributes and prioritized scenarios*

Step 6: *Scenario analyses*

Outcome: *Risks, sensitivities, and tradeoffs*

ATAM Phase 2 activities

Recapitulation of Phase 1 Steps 1-6

Outcome: *Understanding of Phase 1 results*

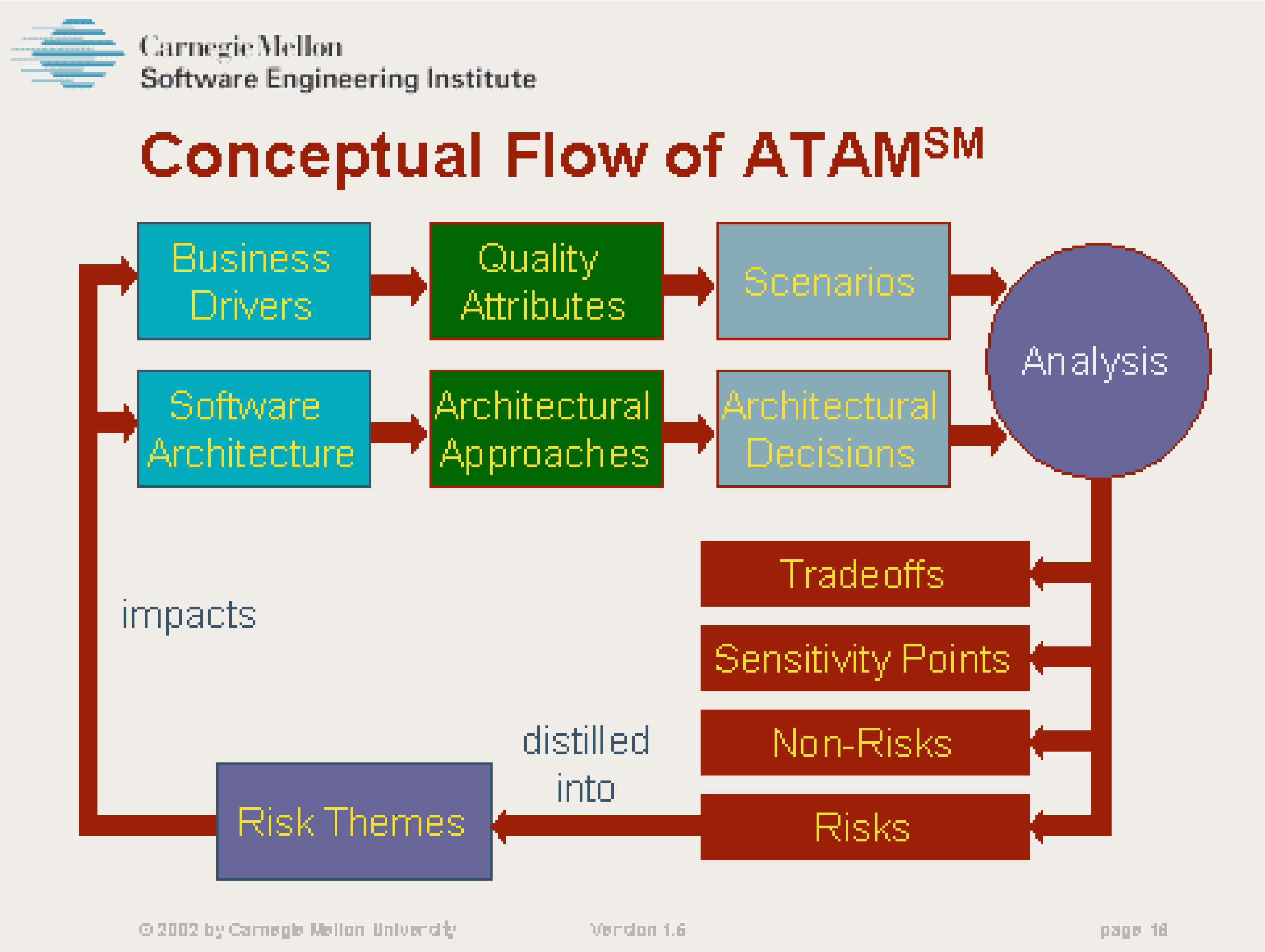
Step 7: *Scenario generation*

Outcome: *Prioritized scenarios*

Step 8: *Scenario analyses*

Outcome: *Risks, sensitivities, and tradeoffs*

Step 9: *Presentation of results*



Carnegie Mellon
Software Engineering Institute

Conceptual Flow of ATAM^{SM}

Business Drivers

Quality Attributes

Scenarios

Software Architecture

Architectural Approaches

Architectural Decisions

Analysis

Tradeoffs

Sensitivity Points

Non-Risks

Risks

distilled into

Risk Themes

impacts

Step 1. Present ATAM

- **Evaluation team presents an overview of ATAM including:**
  - ATAM steps in brief
  - techniques
    - utility tree generation
    - style-based elicitation/analysis
    - scenario brainstorming/mapping
  - outputs
    - scenarios
    - architectural styles
    - quality attribute questions
    - risks and “non-risks”
    - utility tree

Step 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 system
  - Major stakeholders
  - Any relevant technical, political, economic or managerial constraints

Step 3 Present Architecture

- **Architect presents an overview of the architecture including:**
  - Driving architectural requirements and any existing standards/models approaches
  - High level architectural views
    - Functional – Key abstractions. Domain elements and or data flows
    - Decomposition
    - Concurrency, flows, events
    - Physical (deployment)
  - Architectural styles, mechanisms employed and how they address quality attributes
  - Use of COTS
  - Trace of most important scenarios
  - Trace of most important growth scenarios
  - Architectural issues

- **Evaluation team begins probing for:**
  - risks
  - architectural styles

Step 4 Identify Architectural Styles

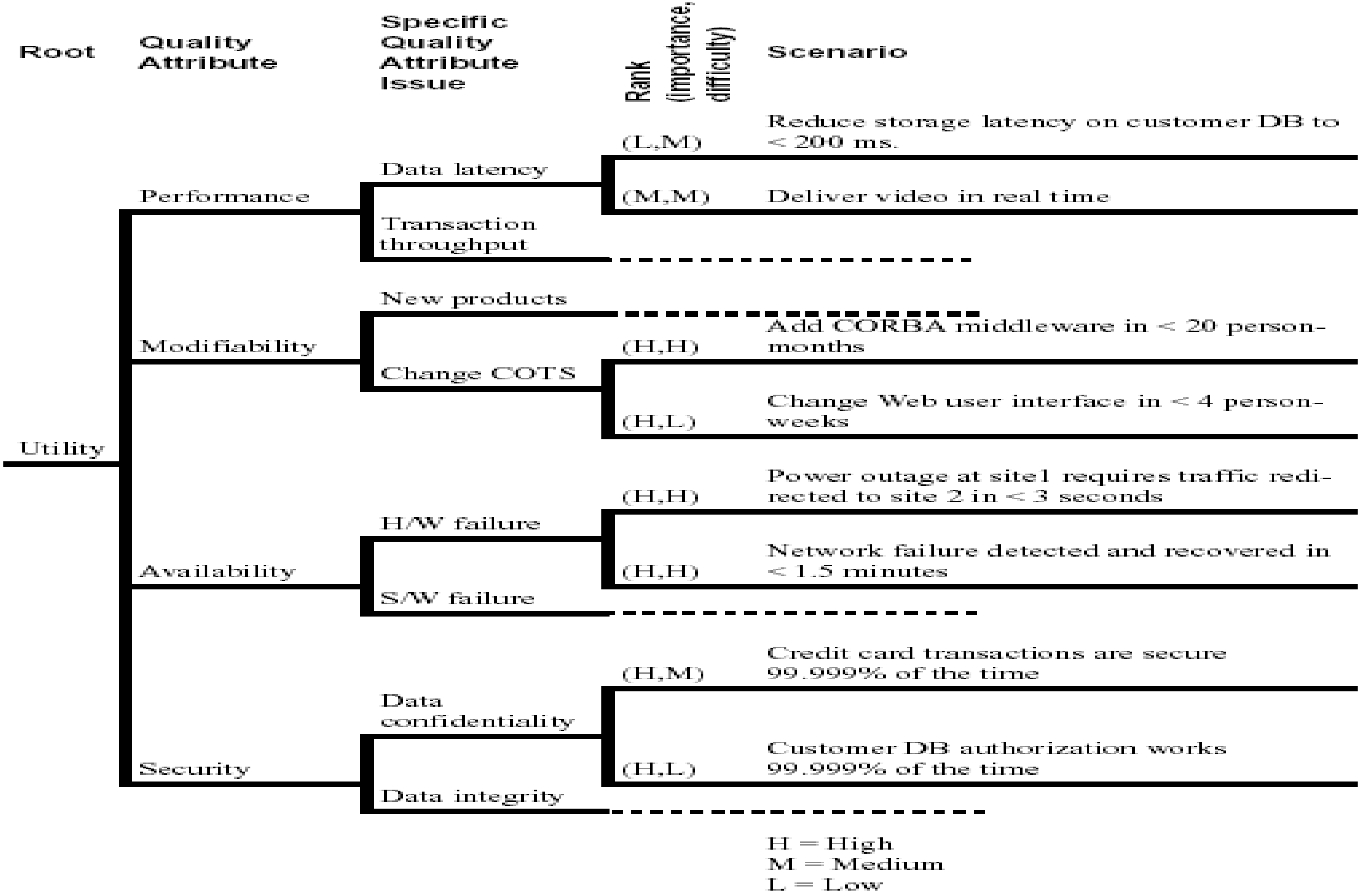
- **High-level overview of architecture is completed by itemizing architectural styles found in the architecture.**
- **Examples:**
  - client-server
  - 3-tier
  - pipeline
  - publisher-subscriber

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 an AHP (analytic hierarchy process)-like model of the “driving” attribute-specific requirements
  - typically performance, modifiability, security, and availability are the high-level nodes
  - scenarios are leaves of utility tree
- **Output: a prioritization of specific quality attribute requirements.**

Quality Attribute Goals

- In Step 5, the participants 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,
        - Scenarios are used to represent stakeholders’ interests and should cover a range of anticipated
            - uses of the system (use case scenarios),
            - anticipated changes to the system (growth scenarios),
            - or unanticipated stresses to the system (exploratory scenarios).
- A good scenario clearly indicates
    - which stimulus causes it and what responses are important.
- During scenario prioritization, scenarios are categorized by two parameters,
    - importance and
    - difficulty,
    - using a scale of
        - Low (L)
        - Medium (M)
        - -High (H).

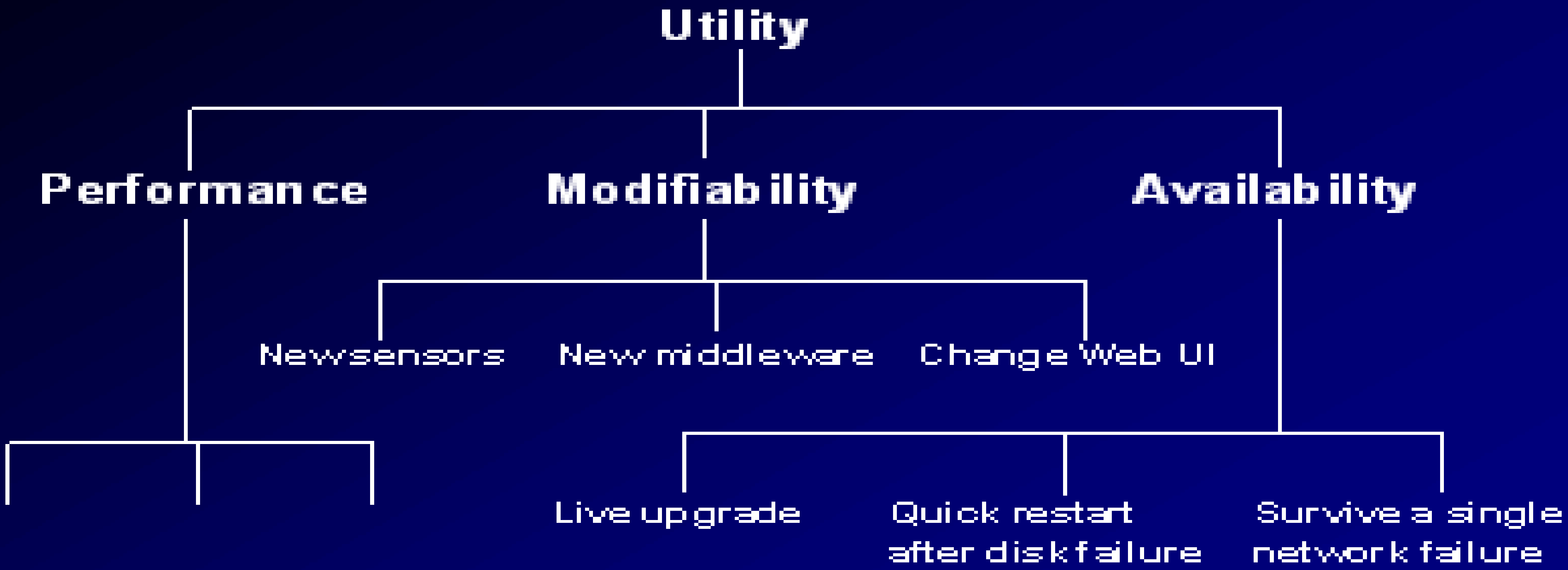


Root	Quality Attribute	Specific Quality Attribute Issue	Rank (importance, difficulty)	Scenario
Utility	Performance	Data latency	(L,M)	Reduce storage latency on customer DB to < 200 ms.
			(M,M)	Deliver video in real time
		Transaction throughput		
	Modifiability	New products		
		Change COTS	(H,H)	Add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

H = High
M = Medium
L = Low

Figure 4: Example ATAM Utility Tree

Utility Tree Construction -1



Utility

Performance

Modifiability

Availability

New sensors

New middleware

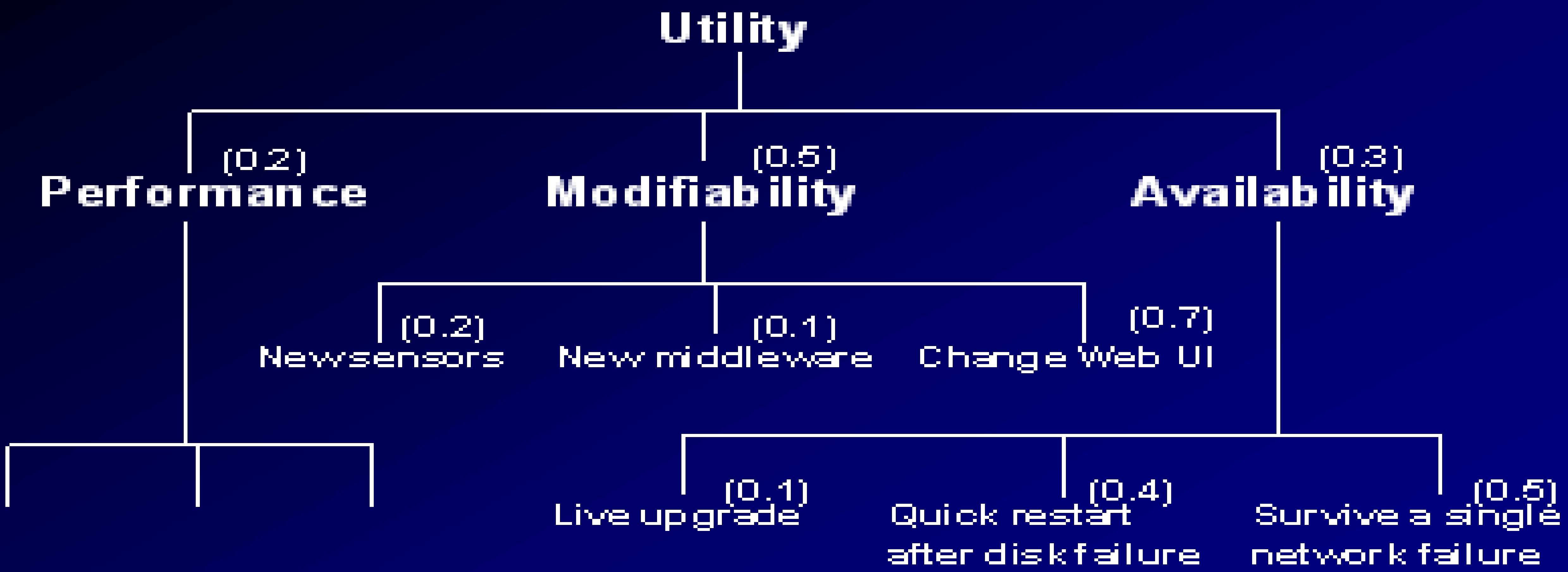
Change Web UI

Live upgrade

Quick restart after disk failure

Survive a single network failure

Utility Tree Construction -2



Step 6 Elicit and Analyze Architecture Styles

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

- Identify the styles which pertain to the highest priority quality attribute requirements
- Generate quality-attribute specific questions for highest priority quality attribute requirement
- Ask quality-attribute specific questions
- Identify and record risks and non-risks

Categories

- **Each quality attribute characterization is divided into three categories:**
  - ***external stimuli,***
  - ***environment,***
  - **and *responses*.**
- ***External stimuli* (or just *stimuli* for short) are the events that cause the architecture to respond or change.**
  - **What the stakeholder does to initiate interaction; invoke a function, request a change, etc.**
- ***Environment* what's going on at the time of the stimulus – what is the system state, heavily loaded? A processor down? – under "normal conditions" not described**
- ***Responses* – how the system responds to the stimulus; is the function successful? Does reconfiguration happen?**

Example Scenarios

- **Use case**
  - Remote user comes via the web to access report database.
- **Growth scenario**
  - Add a new data server to reduce latency by 50%.
- **Exploratory scenario**
  - Half of the servers go down during operation.
- **=> Scenarios should be as specific as possible.**

Quality Attribute Questions

- **Quality attribute questions probe styles to elicit architectural decisions which 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 and Non-Risks

- **Risks are potentially problematic architectural decisions,**
- **Non-risks are good decisions relying on implicit assumptions.**
- **Risk and non-risk constituents**
  - architectural decision
  - quality attribute requirement
  - rationale
- **Sensitivity points – a property of one or more components that is critical for achieving a particular quality attribute**
  - **Example level of confidentiality sensitive to the number of bits of encryption**
- **Tradeoff points – a property that affects more than one attribute and is a sensitivity point for more than one attribute**
  - **Example changing level of encryption could have an effect on both security and performance**
- **Sensitivity points are candidate risks and risks are candidate tradeoff points.**

Step 7 Generate Seed Scenarios

- **Scenarios are used to**
  - Represent stakeholders’ interests
  - Understand quality attribute requirements
- **Seed scenarios are sample scenarios**
- **Scenarios are specific,**
  - anticipated uses of (use cases),
  - anticipated changes to (growth scenarios), or
  - unanticipated stresses (exploratory) to **the system.**

Step 8 Brainstorm and Prioritize Scenarios

- Stakeholders generate scenarios using a brainstorming process.
- Each stakeholder is allocated a number of votes roughly equal to 30% of the scenarios
- Prioritized scenarios are compared with the utility tree and differences are reconciled.

Step 9 Map Scenarios onto Styles

Identify styles and components within styles impacted by each scenario.

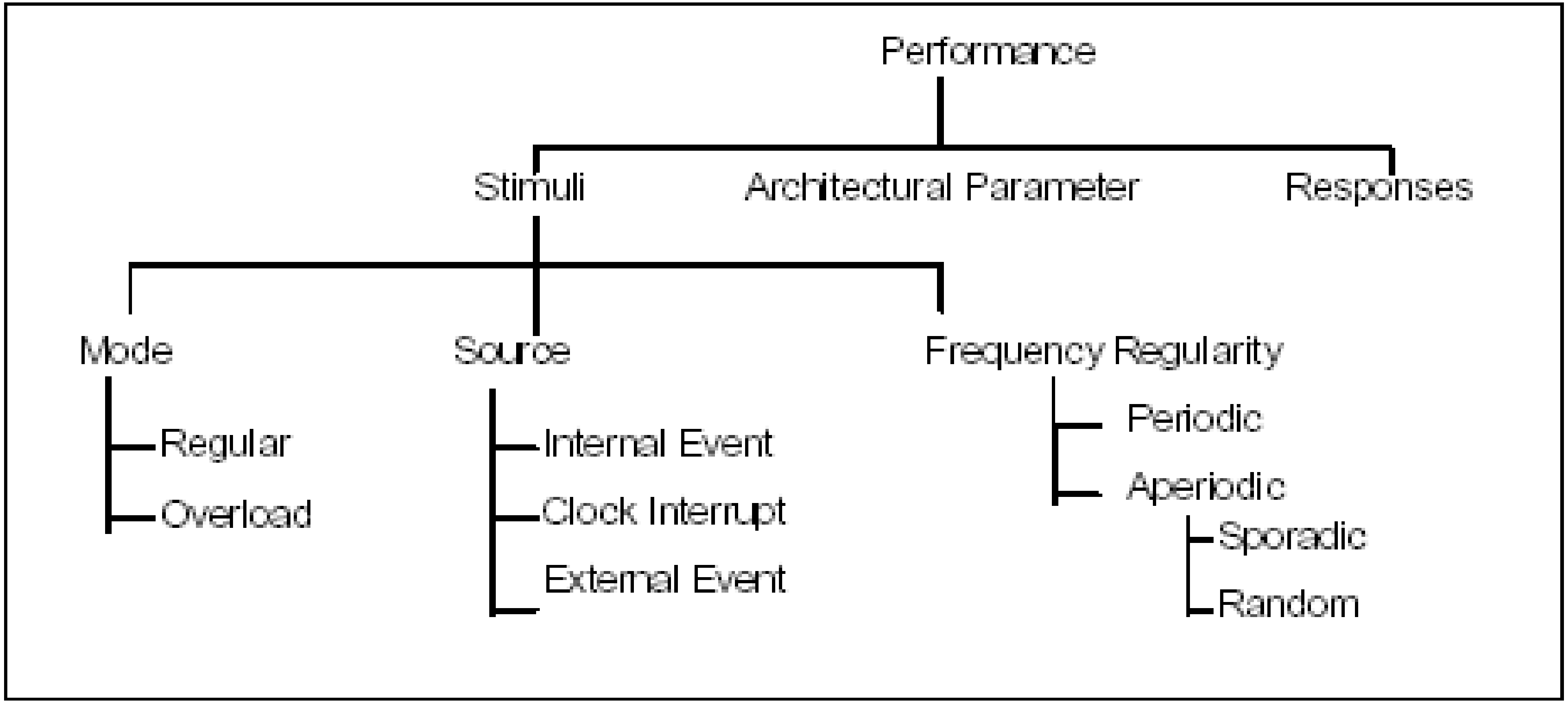
- Continue identifying risks and non-risks.
- Continue annotating architectural information.

Step 10 Present Out-Brief/Write Report

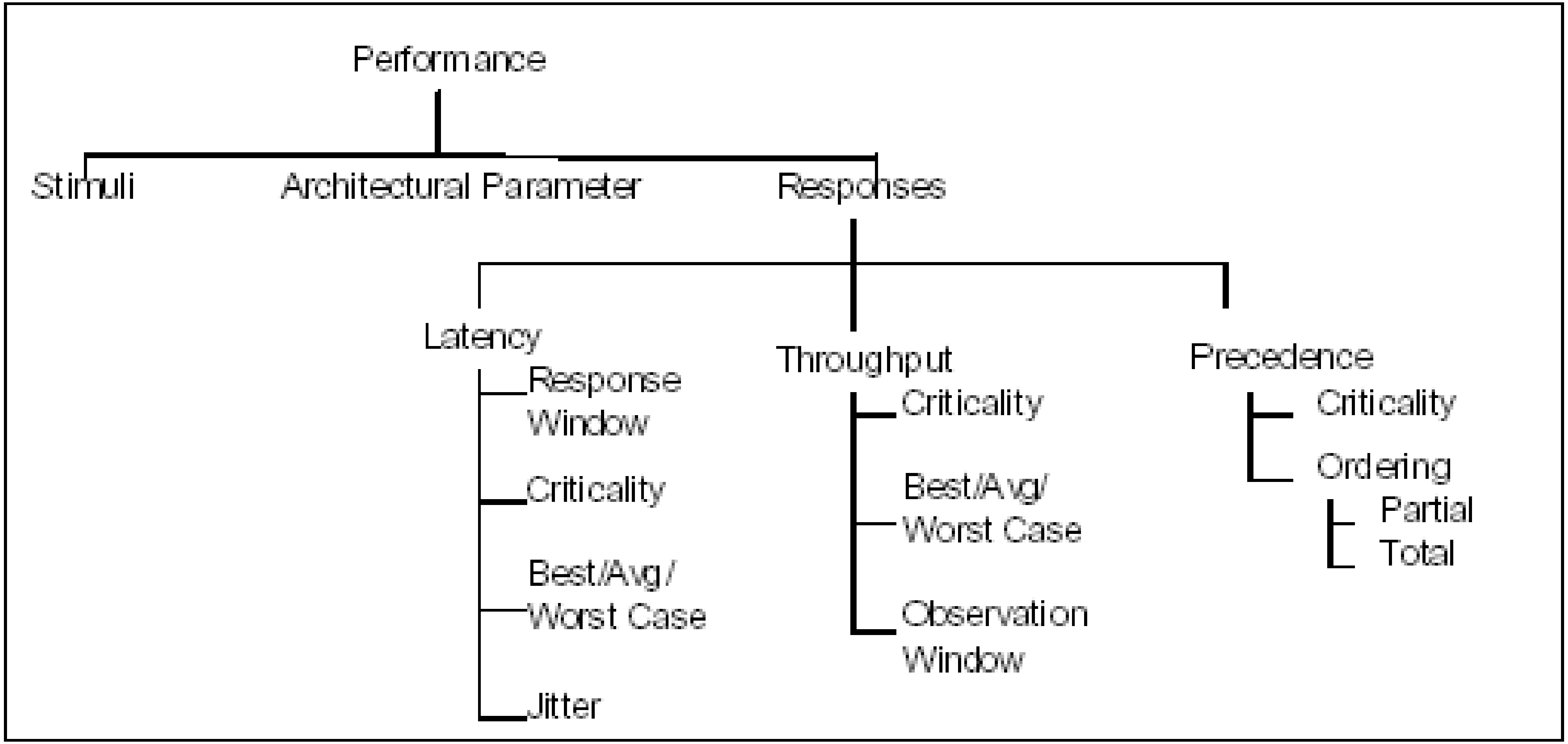
- **Recapitulate steps of ATAM**
- **Present ATAM outputs**
  - styles
  - scenarios
  - questions
  - utility tree
  - risks
  - non-risks
- **Offer recommendations**

Attribute Characteristics

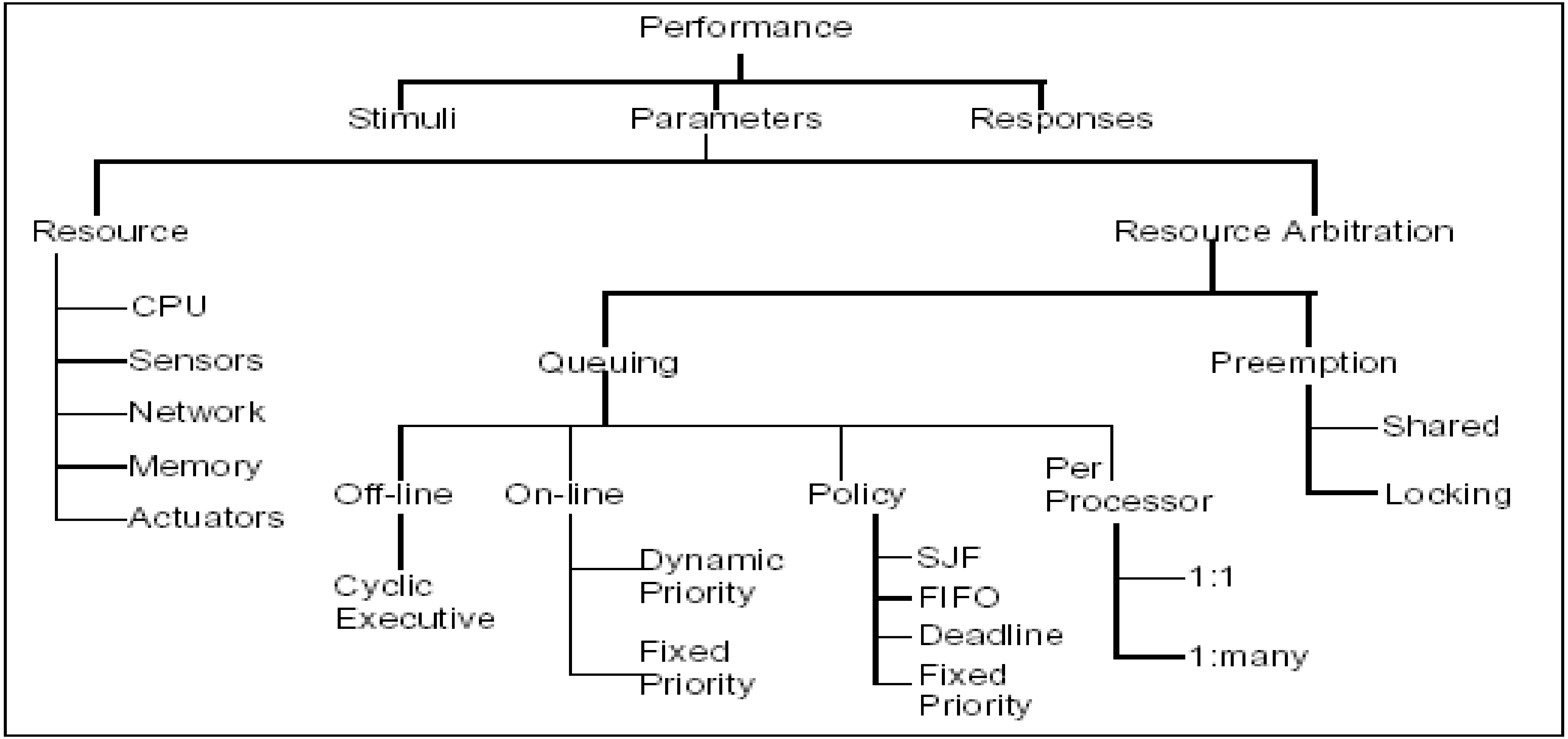
A.1 Performance



Attribute Characteristics

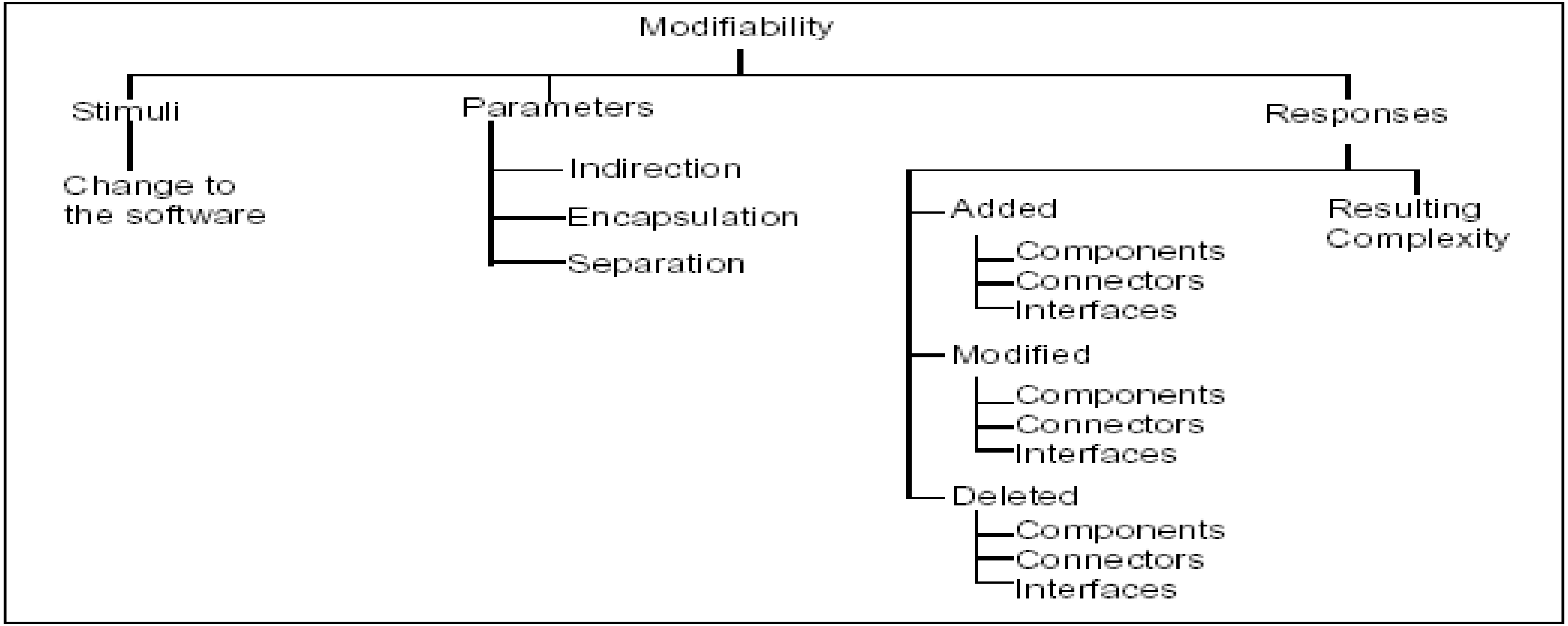


Attribute Characteristics



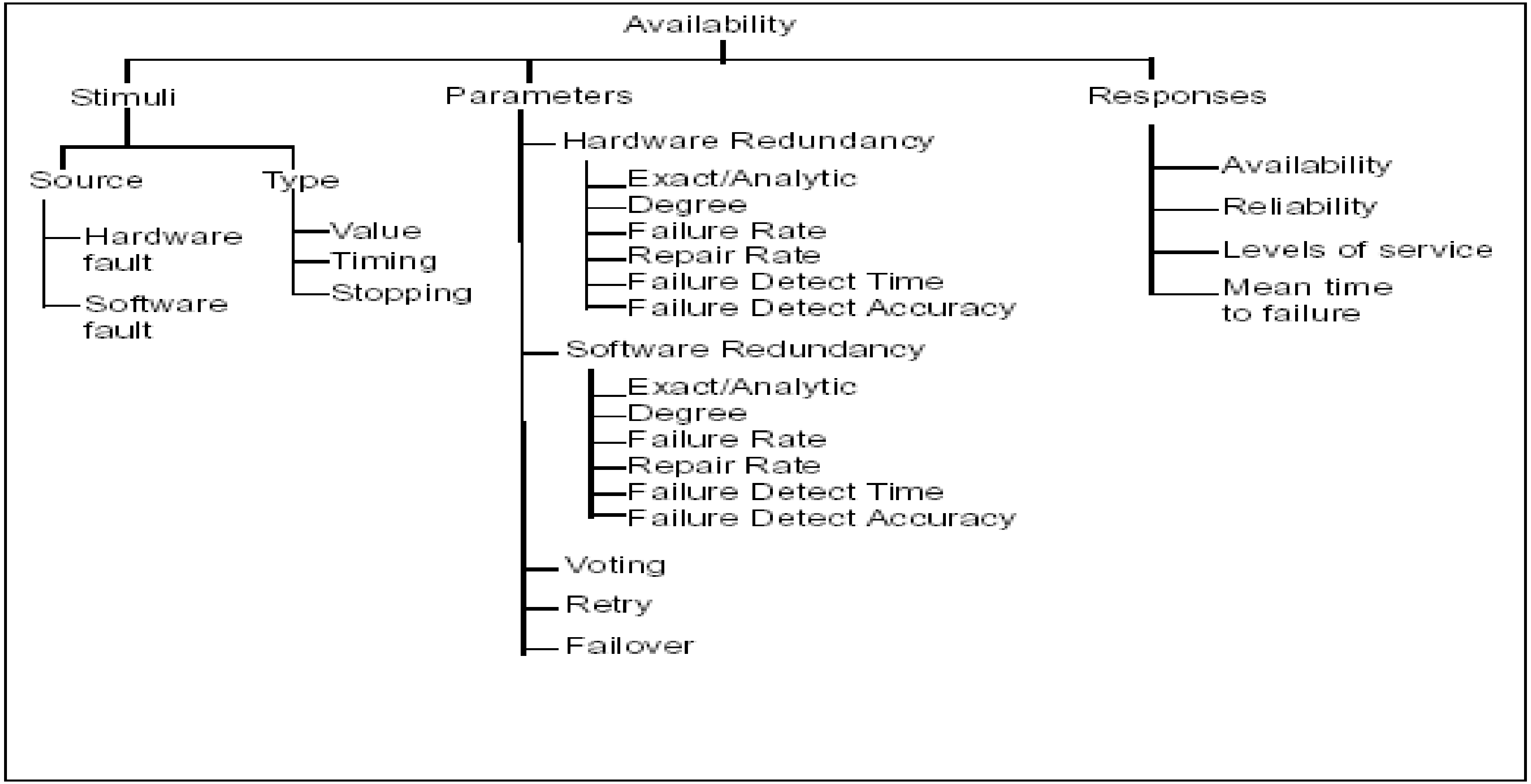
Attribute Characteristics

A.2 Modifiability



Attribute Characteristics

A.3 Availability



- Availability
  - Stimuli
    - Source
      - Hardware fault
      - Software fault
    - Type
      - Value
      - Timing
      - Stopping
  - Parameters
    - Hardware Redundancy
      - Exact/Analytic
      - Degree
      - Failure Rate
      - Repair Rate
      - Failure Detect Time
      - Failure Detect Accuracy
    - Software Redundancy
      - Exact/Analytic
      - Degree
      - Failure Rate
      - Repair Rate
      - Failure Detect Time
      - Failure Detect Accuracy
    - Voting
    - Retry
    - Failover
  - Responses
    - Availability
    - Reliability
    - Levels of service
    - Mean time to failure

Summary

- **ATAM is a method for evaluating an architecture with respect to multiple quality attributes.**
- **It is an effective risk mitigation strategy to avoid the disastrous consequences of a poor architecture. ATAM:**
  - can be done early
  - requires stakeholder participation
- **The key to the method is looking for trends, not in making precise analyses.**
- **ATAM relies crucially on:**
  - Clearly-articulated quality attribute requirements
  - Active stakeholder participation
  - Active participation by the architect
  - Familiarity with architecture styles and quality attribute models

Summary of Benefits

- Gets all stakeholders in same room again
- Forces articulation of specific quality goals (NFR’s)
- Sets early design goals
- Results in prioritization of conflicting goals
- Improves the quality of the architecture documentation
- Uncovers opportunities for cross project reuse
- More explicit presentation of the architecture