Software Architecture and Design Overview I

Mark C. Paulk, Ph.D.

Mark.Paulk@utdallas.edu, Mark.Paulk@ieee.org
http://mark.paulk123.com/

Software Architecture Topics

➡ Introduction to Architecture

Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

What Is a Software Architecture?

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

- **some partition systems into implementation units (modules), which are static**
- **some are dynamic, focusing on the way the elements interact with each other at runtime to carry out the system’s functions (component-and-connector)**
- **some describe the mapping from software structures to the system’s organizational, developmental, installation, and execution environments (allocation)**

Every System Has an Architecture

Architecture-indifferent design

- **opens the door to complexity…**

Architecture-focused design

Architecture hoisting

- **design the architecture with the intent of guaranteeing a goal or property of the system**
- **you will either find**
  - **code that manages the goal or property**
  - **a deliberate structural constraint (often with reasoning or calculations) that ensures it**

Structures and Views

A view is a representation of a coherent set of architectural elements, as written by and read by system stakeholders.

- **It consists of a representation of a set of elements and the relations among them.**

A structure is the set of elements itself, as they exist in software or hardware.

- **module**
- **communication and coordination (C&C)**
- **allocation**

A view is a representation of a structure.

Early Design Decisions

The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.

- **Will the system run on one processor or be distributed across multiple processors?**
- **Will the software be layered? If so, how many layers will there be? What will each one do?**
- **Will components communicate synchronously or asynchronously? Will they interact by transferring control or data or both?**
- **Will the system depend on specific features of the operating system or hardware?**
- **Will the information that flows through the system be encrypted or not?**
- **What operating system will we use?**
- **What communication protocol will we choose?**

Managing Change

The decisions made in an architecture allow you to reason about and manage change as the system evolves.

Roughly 80% of a typical software system's total cost occurs after initial deployment.

Three categories of change: local, non-local, architectural

Architectural change affects the fundamental ways in which the elements interact with each other – will probably require changes all over the system.

Enabling Quality Attributes

An architecture will inhibit or enable a system's driving quality attributes.

High performance… modifiability… security… scalability… incremental subsets… reuseable…

Poor downstream design or implementation decisions can always undermine an adequate architectural design.

Partioning, Knowledge, and Abstractions

Developers…

partition a problem so that its parts are smaller and more tractable

apply knowledge of similar problems

use abstractions to help them reason

… combatting complexity and scale.

Foote and Yoder: the most common software architecture is a big ball of mud.

Architectural Patterns

Compositions of architectural elements

that solve particular problems

have been found useful over time and different domains

documented

disseminated

Software Architecture Topics

Introduction to Architecture

→ Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Quality Attribute

A measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders.

You can think of a quality attribute as measuring the "goodness" of a product along some dimension of interest to a stakeholder.

Functional Requirements

State what the system must do, and how it must behave or react to runtime stimuli.

Are satisfied by assigning an appropriate sequence of responsibilities throughout the design.

Assigning responsibilities to architectural elements is a fundamental architectural design decision.

Functionality

Functionality does not determine architecture.

- **If functionality were the only thing that mattered, you wouldn’t have to divide the system into pieces at all; a single monolithic blob with no internal structure would do just fine.**

Although functionality is independent of any particular structure, functionality is achieved by assigning responsibilities to architectural elements, resulting in one of the most basic of architectural structures.

The architect’s interest in functionality is in how it interacts with and constrains other qualities.

Constraints

A constraint is a design decision with zero degrees of freedom.

- **it’s a design decision that’s already been made**

Constraints are satisfied by accepting the design decision and reconciling it with other affected design decisions.

Restricting Choice

By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.

Engineering is about discipline, and discipline comes in part by *restricting* the vocabulary of alternatives to proven solutions.

Quality Attribute Requirements

Qualifications of the functional requirements or of the overall product.

- **A qualification of a functional requirement is an item such as how fast the function must be performed, or how resilient it must be to erroneous input.**
- **A qualification of the overall product is an item such as the time to deploy the product or a limitation on operational costs.**

Are satisfied by the various structures designed into the architecture, and the behaviors and interactions of the elements that populate those structures.

Categories of Quality Attributes

Those that describe some property of the system at runtime

- **availability, performance, usability, …**

Those that describe some property of the development of the system

- **modifiability, testability, …**

Almost every quality attribute negatively affects performance.

Specifying Quality Attribute Requirements

Source of stimulus

- **some entity (a human, a computer system, or any other actuator) that generated the stimulus**

Stimulus

- **a condition that requires a response when it arrives at a system**

Environment

- **the stimulus occurs under certain conditions**

Artifact

- **some artifact is stimulated: a collection of systems, the whole system, or some piece or pieces of it**

Response

- **the activity undertaken as the result of the arrival of the stimulus**

Response measure

- **when the response occurs, it should be measurable in some fashion so that the requirement can be tested**

Tactics

A tactic is a design decision that influences the achievement of a quality attribute response.

The focus of a tactic is on a single quality attribute response.

- **Within a tactic, there is no consideration of tradeoffs.**
- **Tradeoffs must be explicitly considered and controlled by the designer.**
- **In this respect, tactics differ from architectural patterns, where tradeoffs are built into the pattern.**

Categories of Design Decisions

Allocation of responsibilities

Coordination model

Data model

Management of resources

Mapping among architectural elements

Binding time decisions

Choice of technology

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Availability

A property of software that it is there and ready to carry out its task when you need it to be.

Builds upon the concept of reliability by adding the notion of recovery

“Availability refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval.”

Availability is about minimizing service outage time by mitigating faults.

Availability General Scenario -1

Source

- **Internal/external: people, hardware, software, physical infrastructure, physical environment**

Stimulus

- **Fault: omission, crash,incorrect timing, incorrect response**

Artifact

- **Processors, communication channels, persistent storage, processes**

Environment

- **Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation**

Availability General Scenario -2

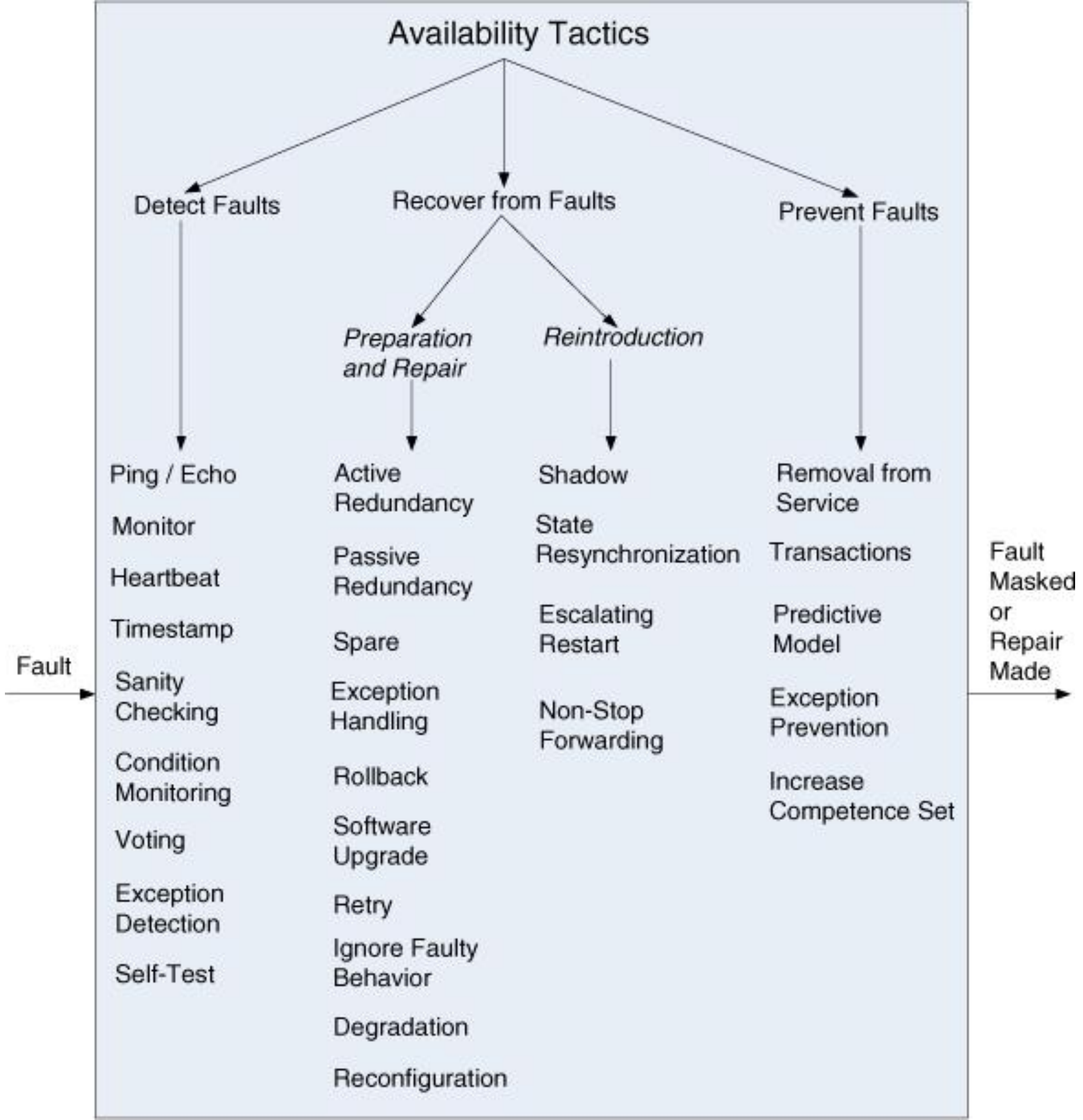
Response

- **Prevent the fault from becoming a failure**
- **Detect the fault**
  - **log the fault**
  - **notify appropriate entities (people or systems)**
- **Recover from the fault**
  - **disable source of events causing the fault**
  - **be temporarily unavailable while repair is being effected**
  - **fix or mask the fault/failure or contain the damage it causes**
  - **operate in a degraded mode while repair is being effected**

Availability General Scenario -3

Response Measure

- **Time or time interval when the system must be available**
- **Availability percentage**
- **Time to detect the fault**
- **Time to repair the fault**
- **Time or time interval in which system can be in degraded mode**
- **Proportion or rate of a certain class of faults that the system prevents, or handles without failing**


Availability Tactics
Detect Faults
Recover from Faults
Prevent Faults
Preparation and Repair
Reintroduction
Ping / Echo
Monitor
Heartbeat
Timestamp
Sanity Checking
Condition Monitoring
Voting
Exception Detection
Self-Test
Active Redundancy
Passive Redundancy
Spare
Exception Handling
Rollback
Software Upgrade
Retry
Ignore Faulty Behavior
Degradation
Reconfiguration
Shadow
State Resynchronization
Escalating Restart
Non-Stop Forwarding
Removal from Service
Transactions
Predictive Model
Exception Prevention
Increase Competence Set
Fault
Fault Masked or Repair Made

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Interoperability

The degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.

Syntactic interoperability – the ability to exchange data.

Semantic interoperability – the ability to correctly interpret the data being exchanged.

Exchanging Information via Interfaces

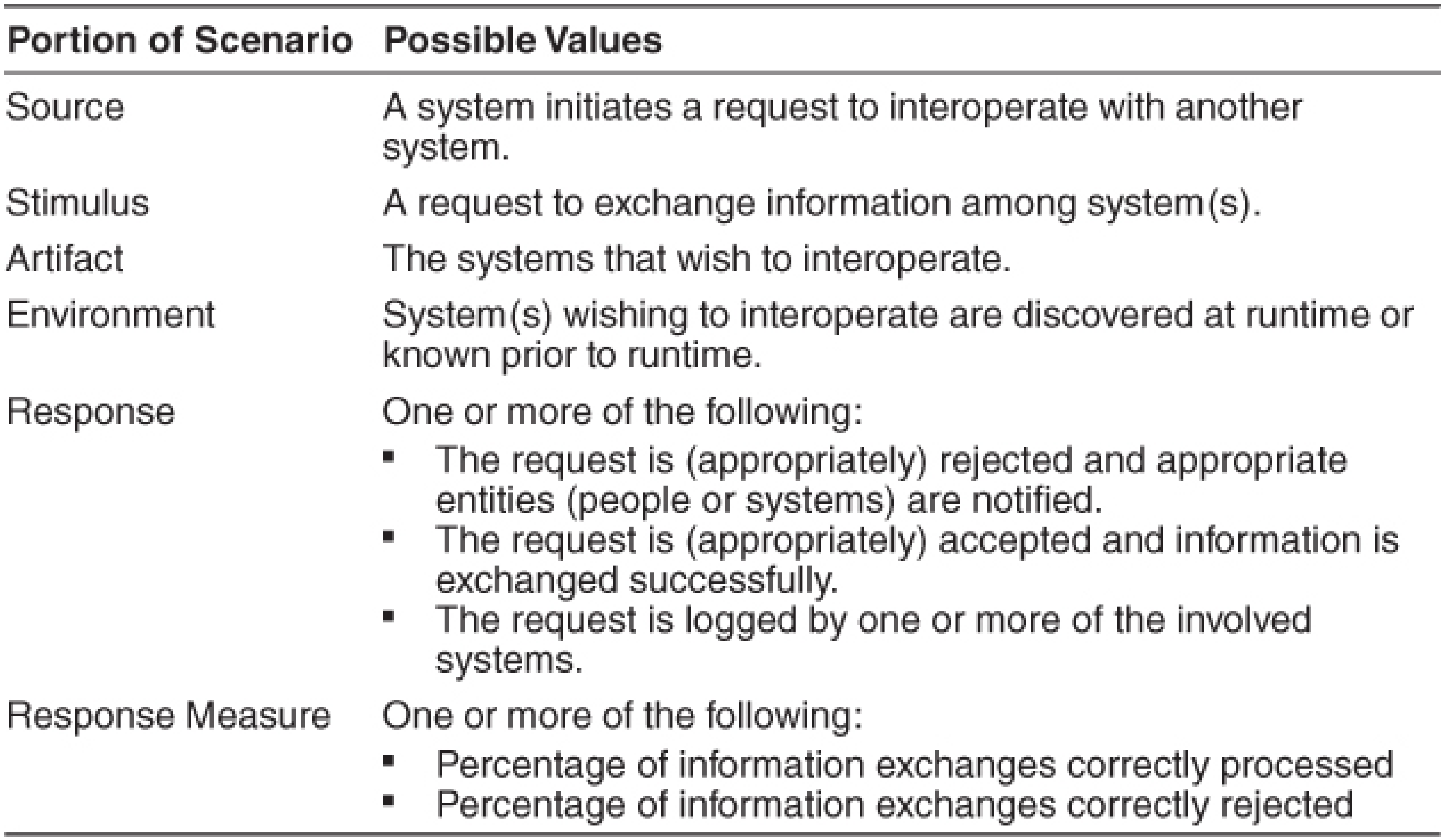
Information may be exchanged even though systems do not communicate directly with one another.

Expected behavior of the system

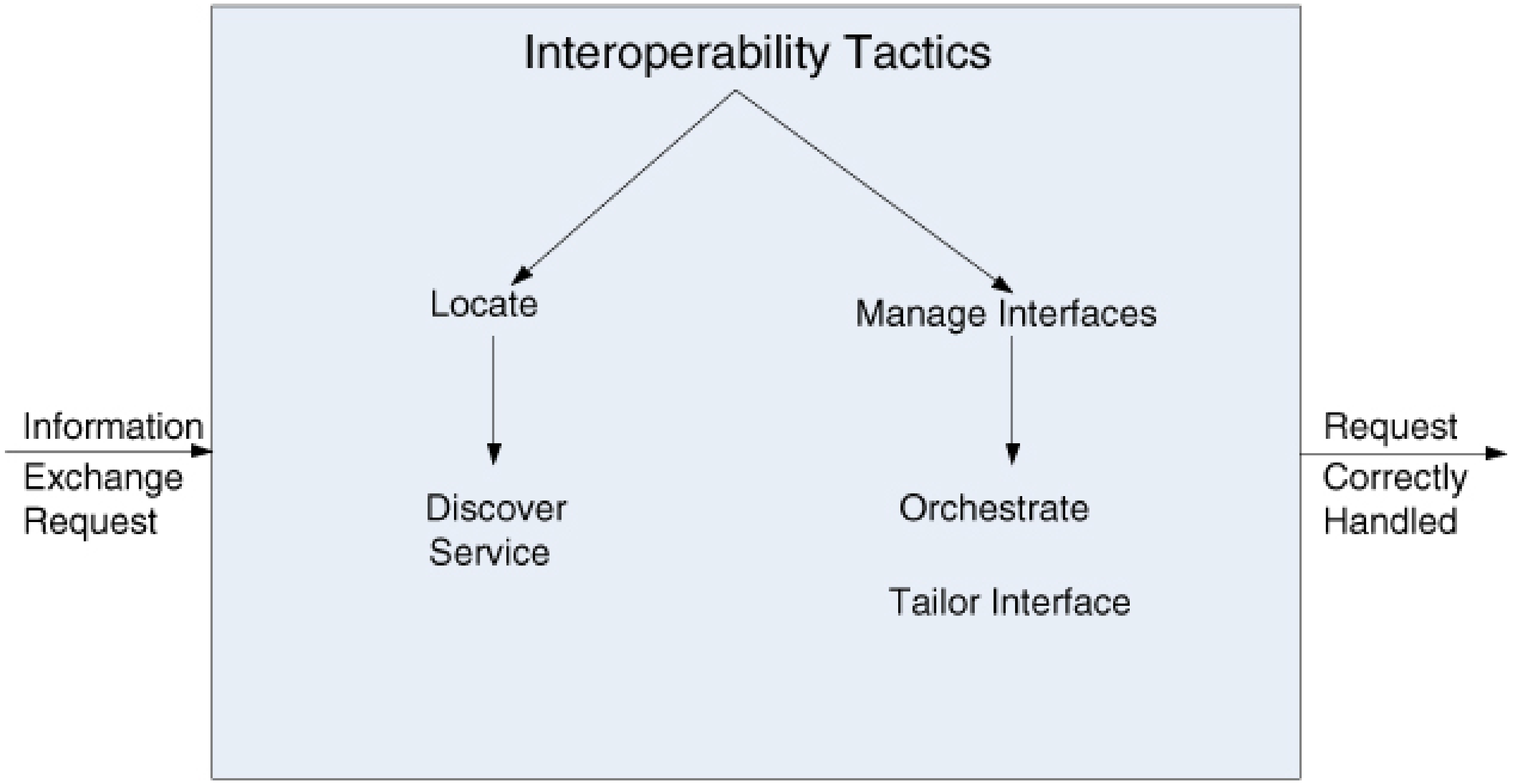
Expected behavior of "information exchange" partners

Interface → the set of assumptions that you can make safely about an entity.

Interoperability General Scenario



Portion of Scenario	Possible Values
Source	A system initiates a request to interoperate with another system.
Stimulus	A request to exchange information among system(s).
Artifact	The systems that wish to interoperate.
Environment	System(s) wishing to interoperate are discovered at runtime or known prior to runtime.
Response	One or more of the following: ▪ The request is (appropriately) rejected and appropriate entities (people or systems) are notified. ▪ The request is (appropriately) accepted and information is exchanged successfully. ▪ The request is logged by one or more of the involved systems.
Response Measure	One or more of the following: ▪ Percentage of information exchanges correctly processed ▪ Percentage of information exchanges correctly rejected


Interoperability Tactics
Locate
Discover Service
Manage Interfaces
Orchestrate
Tailor Interface
Information Exchange Request
Request Correctly Handled

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Modifiability

Modifiability is about change, and our interest in it centers on the cost and risk of making changes.

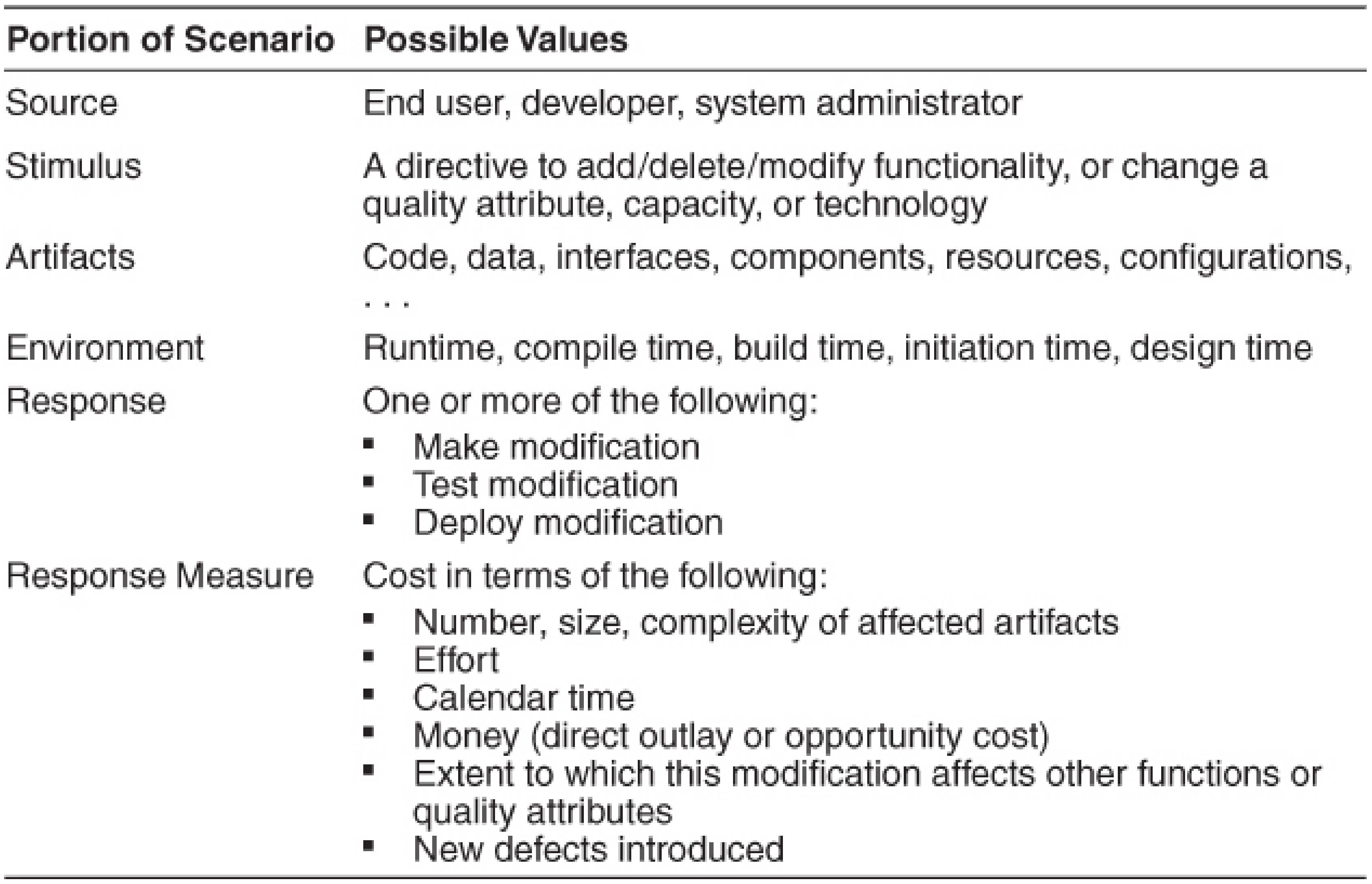
What can change?

What is the likelihood of the change?

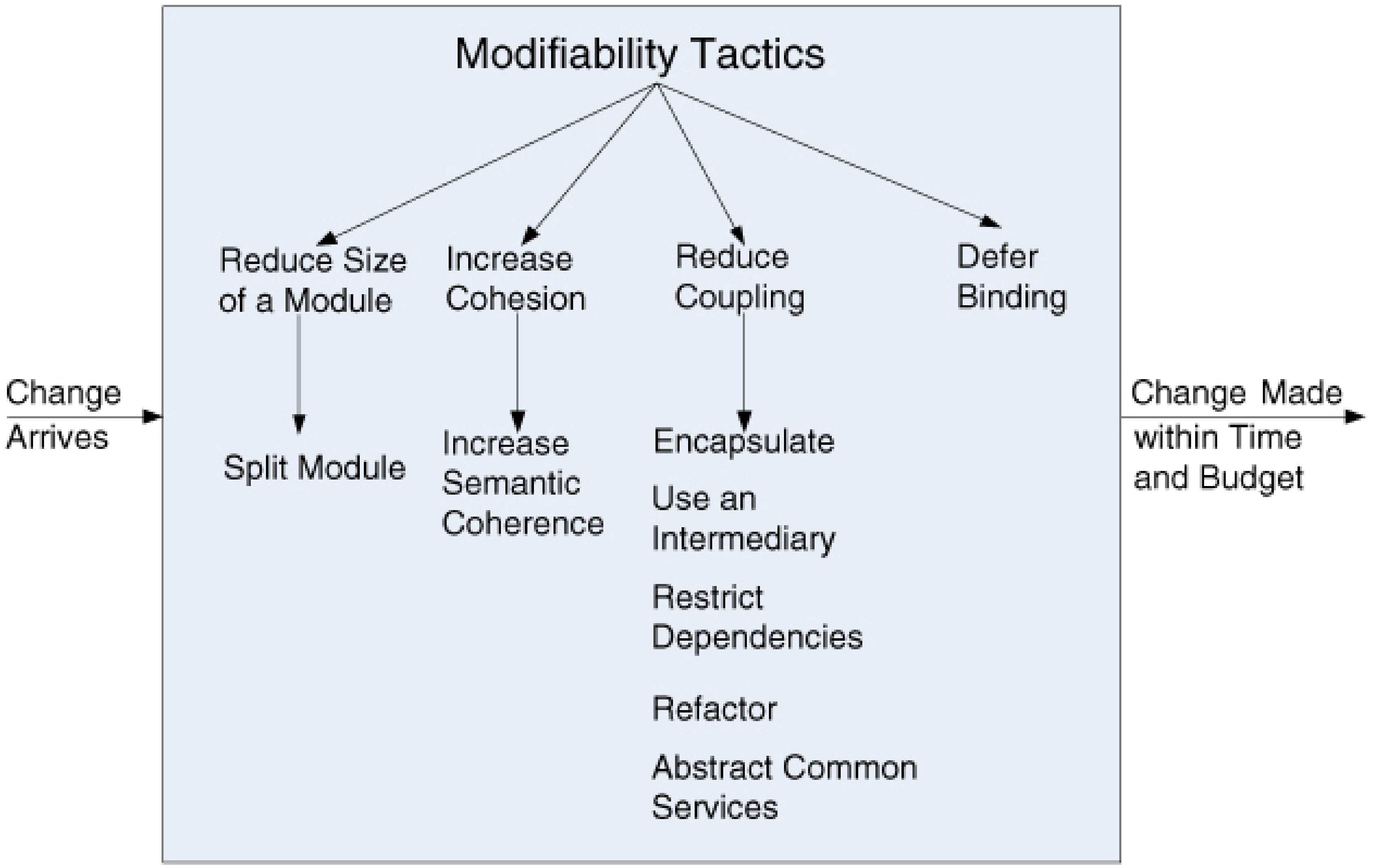
When is the change made and who makes it?

What is the cost of the change?

Modifiability General Scenario



Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced


Modifiability Tactics
Change Arrives
Reduce Size of a Module
Split Module
Increase Cohesion
Increase Semantic Coherence
Reduce Coupling
Encapsulate
Use an Intermediary
Restrict Dependencies
Refactor
Abstract Common Services
Defer Binding
Change Made within Time and Budget

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Performance

It’s about time and the software system’s ability to meet timing requirements.

When events occur, the system, or some element of the system, must respond to them in time.

- **real-time**
- **hard real-time**

Characterizing the events that can occur (and when they can occur) and the system or element’s time-based response to those events is the essence is discussing performance.

Measuring System Response

Latency

- **the time between the arrival of the stimulus and the system's response to it**

Deadlines

Throughput

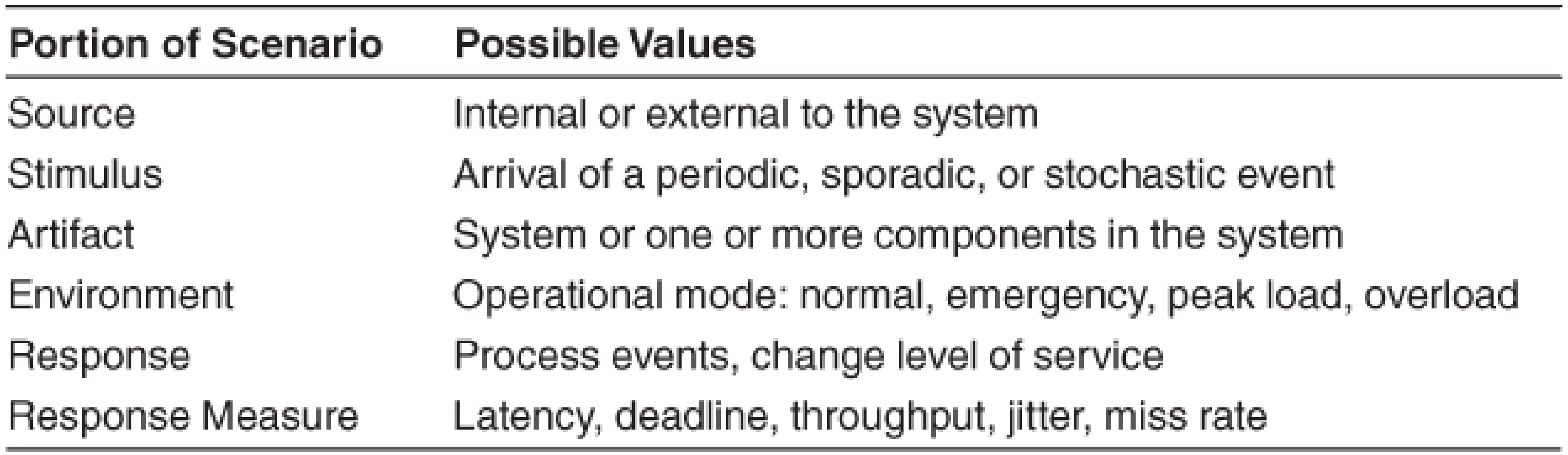
- **usually given as the number of transactions the system can process in a unit of time**

Jitter of the response

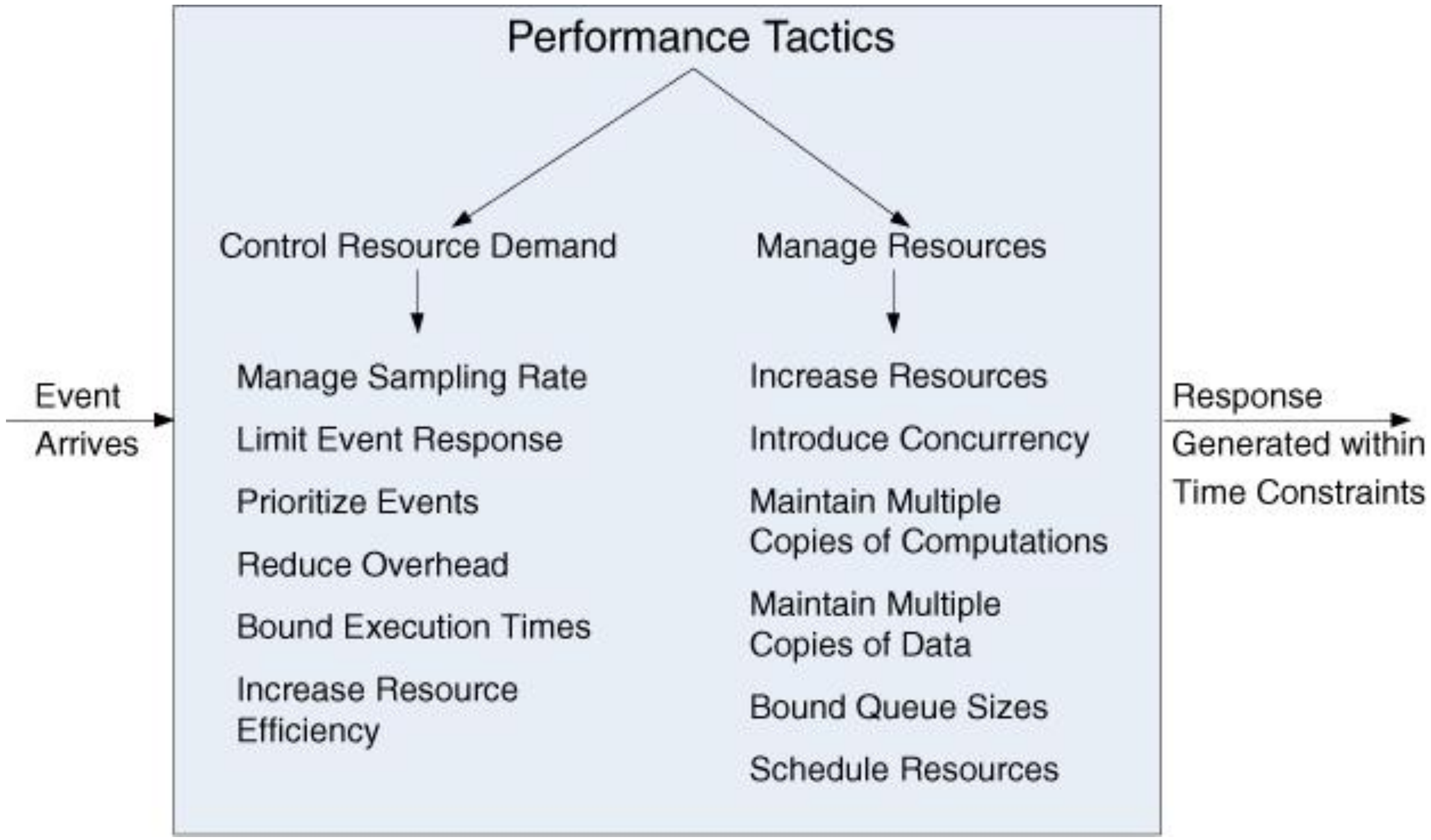
- **the allowable variation in latency**

The number of events not processed because the system was too busy to respond.

Performance General Scenario



Portion of Scenario	Possible Values
Source	Internal or external to the system
Stimulus	Arrival of a periodic, sporadic, or stochastic event
Artifact	System or one or more components in the system
Environment	Operational mode: normal, emergency, peak load, overload
Response	Process events, change level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate


Performance Tactics
Control Resource Demand
Manage Resources
Manage Sampling Rate
Limit Event Response
Prioritize Events
Reduce Overhead
Bound Execution Times
Increase Resource Efficiency
Increase Resources
Introduce Concurrency
Maintain Multiple Copies of Computations
Maintain Multiple Copies of Data
Bound Queue Sizes
Schedule Resources
Event Arrives
Response Generated within Time Constraints

Performance Tactics on the Road

Manage event rate.

- **Lights on highway entrance ramps let cars onto the highway only at set intervals, and cars must wait (queue) on the ramp for their turn.**

Prioritize events.

- **Ambulances and police, with their lights and sirens going, have higher priority than ordinary citizens; some highways have high-occupancy vehicle (HOV) lanes, giving priority to vehicles with two or more occupants.**

Maintain multiple copies.

- **Add traffic lanes to existing roads, or build parallel routes.**

Some User Tricks on the Road

Increase resources.

- **Buy a Ferrari. All other things being equal, the fastest car with a competent driver on an open road will get you to your destination more quickly.**

Increase efficiency.

- **Find a new route that is quicker and/or shorter than your current route.**

Reduce computational overhead.

- **You can drive closer to the car in front of you, or you can load more people into the same vehicle (that is, carpooling).**

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- → **Security**
- **Testability**
- **Usability**

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Security

A measure of the system’s ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.

An action taken against a computer system with the intention of doing harm is called an attack.

- **an unauthorized attempt to access data or services**
- **an unauthorized attempt to modify data**
- **intended to deny services to legitimate users**

A Simple Approach to Security

Confidentiality

- **data or services are protected from unauthorized access**
  - **a hacker cannot access your income tax returns on a government computer**

Integrity

- **data or services are not subject to unauthorized manipulation**
  - **your grade has not been changed since your instructor assigned it**

Availability

- **the system will be available for legitimate use**
  - **a denial-of-service attack won’t prevent you from ordering book from an online bookstore**

Other Characteristics: Authentication

Authentication verifies the identities of the parties to a transaction and checks if they are truly who they claim to be.

- **when you get an email purporting to come from a bank, authentication guarantees that it actually comes from the bank**

Other Characteristics: Nonrepudiation & Authorization

<u>Nonrepudiation</u> guarantees that the sender of a message cannot later deny having sent the message, and that the recipient cannot deny having received the message.

- **you cannot deny ordering something from the Internet, or the merchant cannot disclaim getting your order**

<u>Authorization</u> grants a user the privileges to perform a task.

- **an online banking system authorizes a legitimate user to access his account.**

Security General Scenario -1

Source

- **Human or another system which may have been previously certified (either correctly or incorrectly) or may be currently unknown. A human attacker may be from outside the organization or from inside the organization.**

Stimulus

- **Unauthorized attempt is made to display data, change or delete data, access system services, change the system’s behavior, or reduce availability.**

Security General Scenario -2

Artifact

- **System services, data within the system, a component or resources of the system, data produced or consumed by the system.**

Environment

- **The system is either online or offline; either connected to or disconnected from a network; either behind a firewall or open to a network; fully operational, partially operational, or not operational.**

Security General Scenario -3

Response

Transactions are carried out in fashion such that

- **Data or services are protected from unauthorized access.**
- **Data or services are not being manipulated without authorization.**
- **Parties to a transaction are identified with assurance.**
- **The parties to the transaction cannot repudiate their involvements.**
- **The data resources and system services will be available for legitimate use**

Security General Scenario -4

Response

The system tracks activities within it by

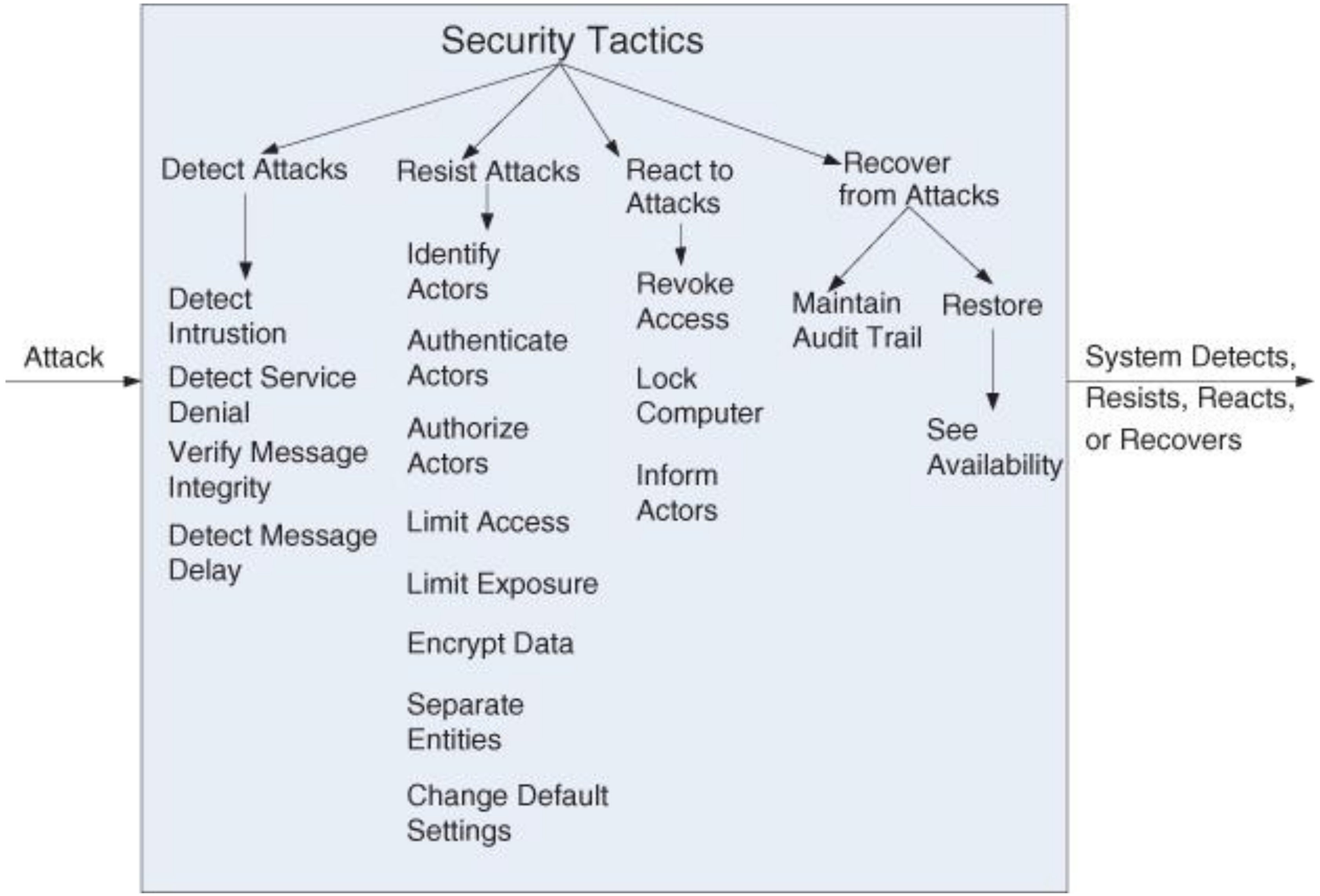
- **Recording access or modification**
- **Recording attempts to access data, resources, or services**
- **Notifying appropriate entities (people or systems) when an apparent attack is occurring**

Security General Scenario -5

Response Measure

One or more of the following:

- **How much of the system is compromised when a particular component or data value is compromised?**
- **How much time passed before an attack was detected?**
- **How many attacks were resisted?**
- **How long does it take to recover from a successful attack?**
- **How much data is vulnerable to a particular attack?**


Security Tactics
Attack
Detect Attacks
Detect Intrustion
Detect Service Denial
Verify Message Integrity
Detect Message Delay
Resist Attacks
Identify Actors
Authenticate Actors
Authorize Actors
Limit Access
Limit Exposure
Encrypt Data
Separate Entities
Change Default Settings
React to Attacks
Revoke Access
Lock Computer
Inform Actors
Recover from Attacks
Maintain Audit Trail
Restore
See Availability
System Detects, Resists, Reacts, or Recovers

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

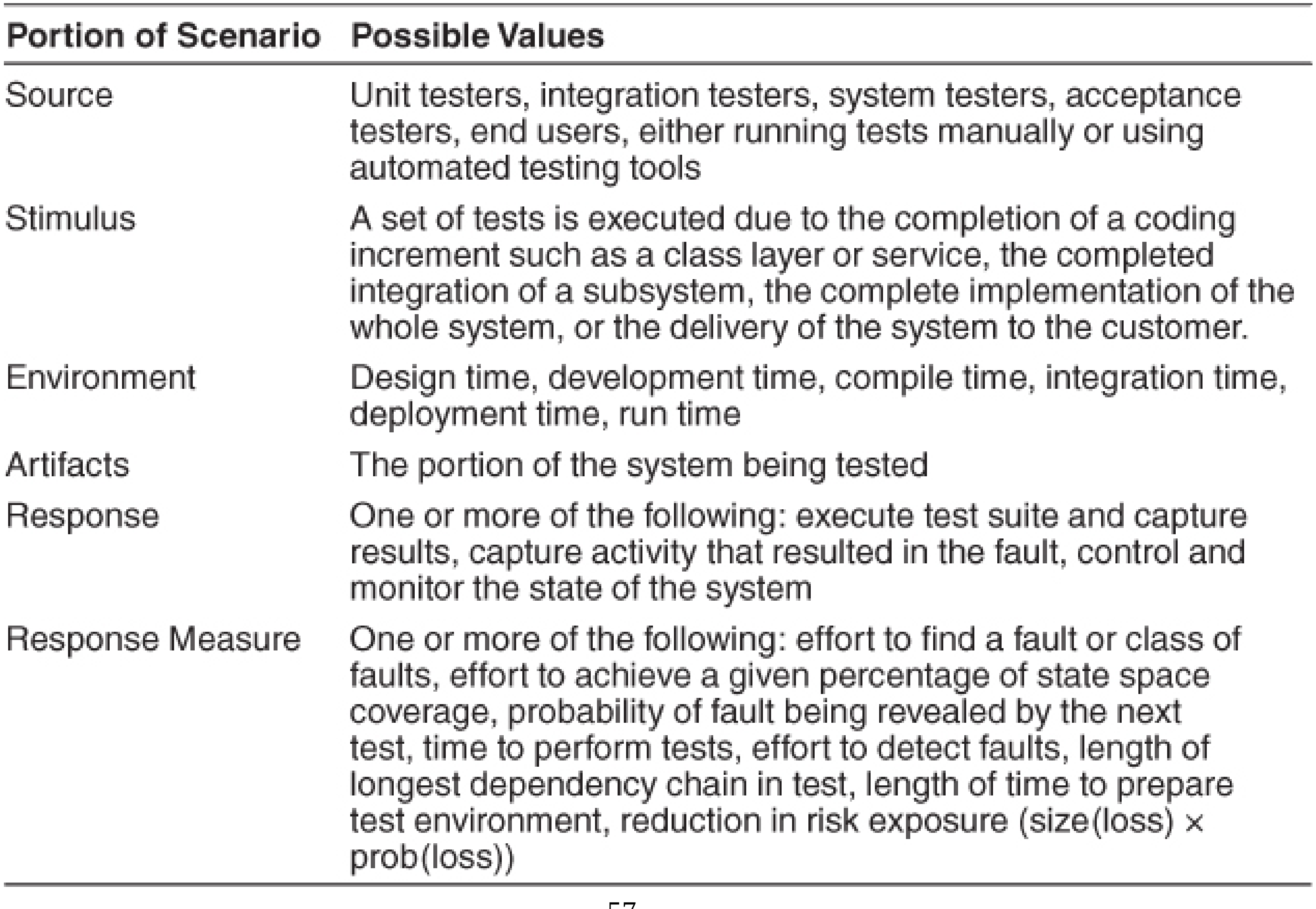
Testability

Refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

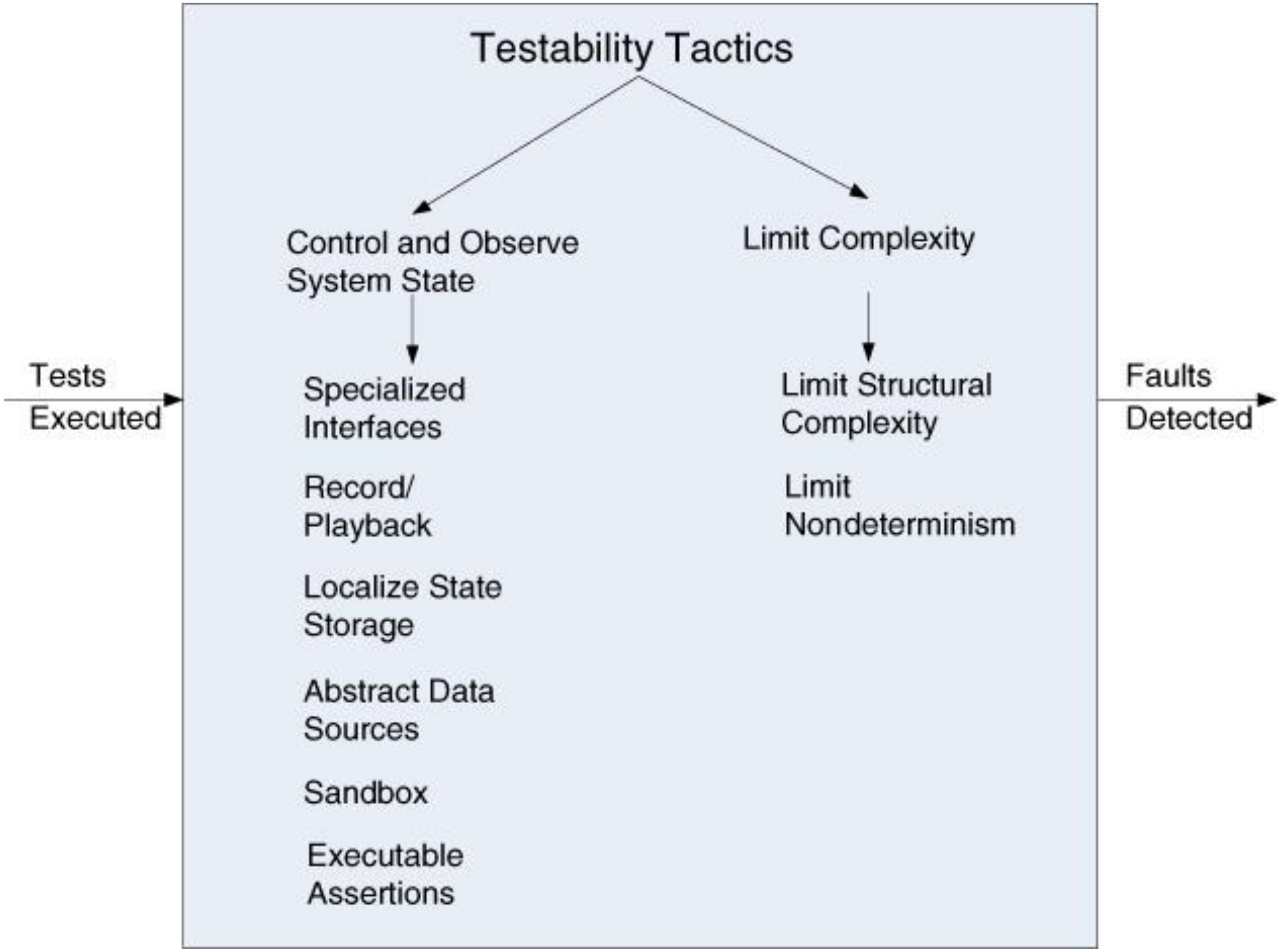
Refers to the probability, assuming that the software has at least one fault, that it will fail on its next test execution.

Industry estimates indicate that between 30 and 50 percent (or in some cases, even more) of the cost of developing well-engineered systems is taken up by testing.

Testability General Scenario



Portion of Scenario	Possible Values
Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools
Stimulus	A set of tests is executed due to the completion of a coding increment such as a class layer or service, the completed integration of a subsystem, the complete implementation of the whole system, or the delivery of the system to the customer.
Environment	Design time, development time, compile time, integration time, deployment time, run time
Artifacts	The portion of the system being tested
Response	One or more of the following: execute test suite and capture results, capture activity that resulted in the fault, control and monitor the state of the system
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage, probability of fault being revealed by the next test, time to perform tests, effort to detect faults, length of longest dependency chain in test, length of time to prepare test environment, reduction in risk exposure (size(loss) × prob(loss))


Testability Tactics
Tests Executed
Control and Observe System State
Specialized Interfaces
Record/ Playback
Localize State Storage
Abstract Data Sources
Sandbox
Executable Assertions
Limit Complexity
Limit Structural Complexity
Limit Nondeterminism
Faults Detected

Software Architecture Topics

Introduction to Architecture

Quality Attributes

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

Other Quality Attributes

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

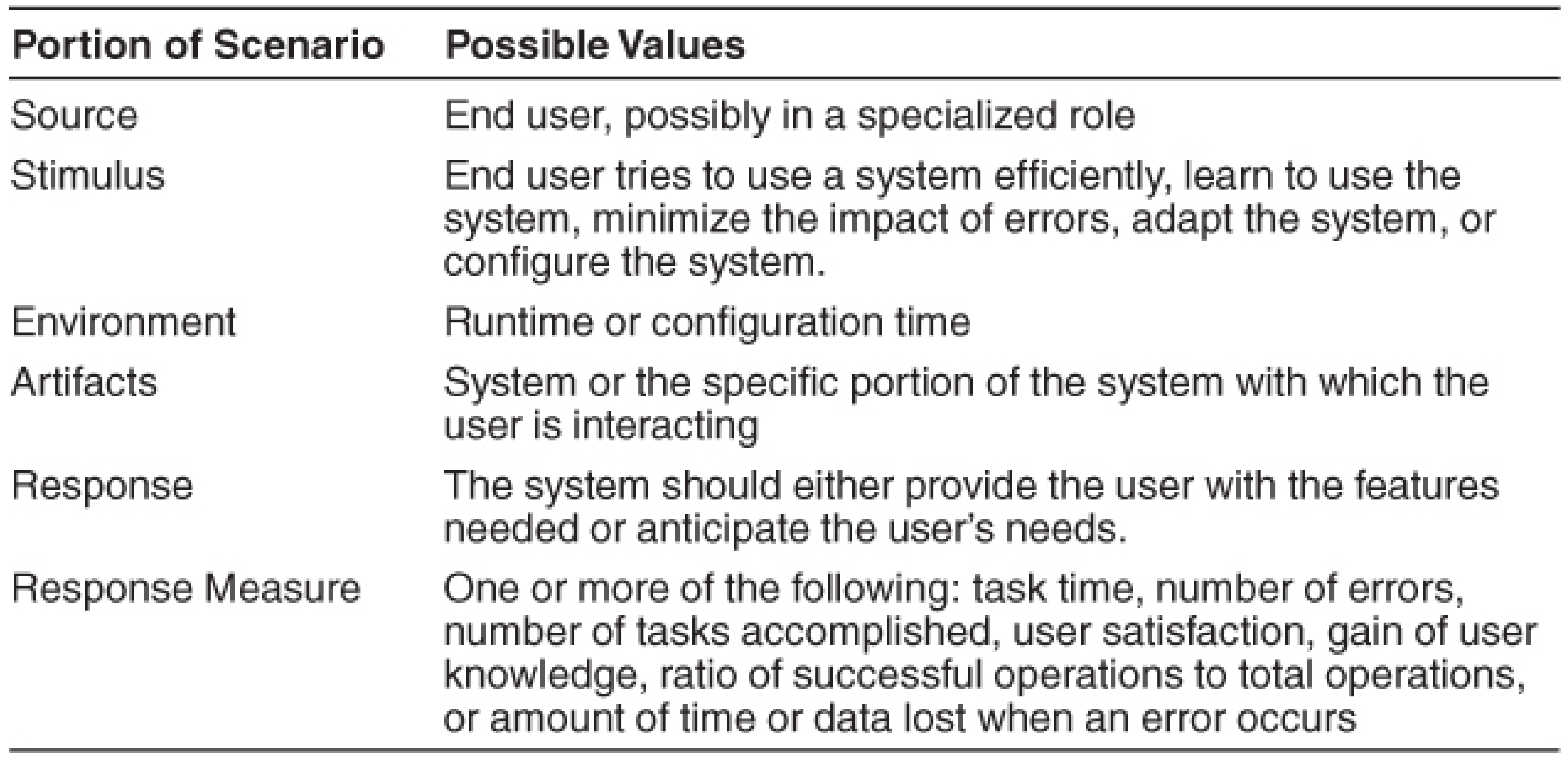
Usability

Concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

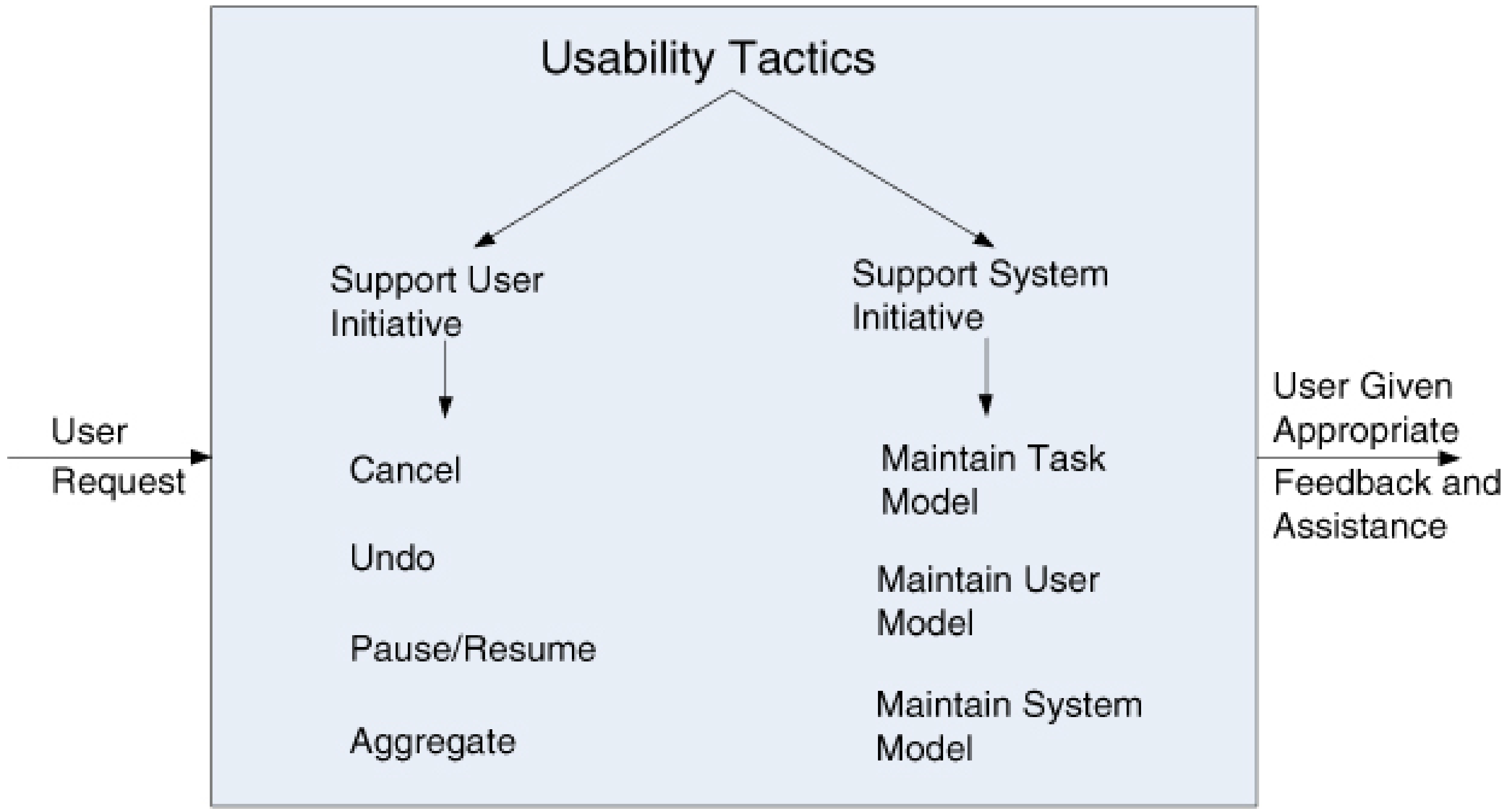
Usability comprises:

- **Learning system features.**
- **Using a system efficiently.**
- **Minimizing the impact of errors.**
- **Adapting the system to user needs.**
- **Increasing confidence and satisfaction.**

Usability General Scenario



Portion of Scenario	Possible Values
Source	End user, possibly in a specialized role
Stimulus	End user tries to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or configure the system.
Environment	Runtime or configuration time
Artifacts	System or the specific portion of the system with which the user is interacting
Response	The system should either provide the user with the features needed or anticipate the user's needs.
Response Measure	One or more of the following: task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time or data lost when an error occurs


Usability Tactics
Support User Initiative
Support System Initiative
User Request
Cancel
Undo
Pause/Resume
Aggregate
Maintain Task Model
Maintain User Model
Maintain System Model
User Given Appropriate Feedback and Assistance

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- **Availability**
- **Interoperability**
- **Modifiability**
- **Performance**
- **Security**
- **Testability**
- **Usability**

➡ **Other Quality Attributes**

Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Variability, Portability, and Development Distributability

Variability

- **support the production of a set of variants that differ from each other in a preplanned fashion**

Portability

- **refers to the ease with which software that was built to run on one platform can be changed to run on a different platform**

Development distributability

- **the quality of designing the software to support distributed software development**

Scalability

Horizontal scalability (scaling out) refers to adding more resources to logical units, such as adding another server to a cluster of servers.

- **In cloud environments, elasticity is a property that enables a customer to add or remove virtual machines from the resource pool.**

Vertical scalability (scaling up) refers to adding more resources to a physical unit, such as adding more memory to a single computer.

Deployability and Mobility

Deployability

- **concerned with how an executable arrives at a host platform and how it is subsequently invoked**

Mobility

- **deals with the problems of movement and affordances of a platform**
  - **e.g., size, type of display, type of input devices, availability and volume of bandwidth, and battery life)**
  - **issues include battery management, reconnecting after a period of disconnection, and the number of different user interfaces necessary to support multiple platforms**

Monitorability and Safety

Monitorability

- **deals with the ability of the operations staff to monitor the system while it is executing**

Safety

- **about the software's ability to avoid entering states that cause or lead to damage, injury, or loss of life to actors in the software's environment, and to recover and limit the damage when it does enter into bad states**
  - **concerned with the prevention of and recovery from hazardous failures**
  - **the architectural concerns with safety are almost identical to those for availability**

Conceptual Integrity of the Architecture

Refers to consistency in the design of the architecture

- **contributes to the understandability of the architecture**
- **leads to fewer errors of confusion**
- **demands that the same thing is done in the same way through the architecture**

Quality In Use

Effectiveness

- **building the system correctly (the system performs according to its requirements) and building the correct system (the system performs in the manner the user wishes)**
  - **Effectiveness is a measure of whether the system is correct.**

Efficiency

- **the effort and time required to develop a system**

Freedom from risk

- **degree to which a product or system affects economic status, human life, health, or the environment**

Marketability

The perception of an architecture can be more important than the qualities the architecture brings.

Many organizations have felt they had to build cloud-based systems (or some other technology du jour) whether or not that was the correct technical choice.

- **also structured XXX**
- **also object oriented XXX**
- **also …**

ISO/IEC 25010 SQuaRE

Functional suitability

- **functional completeness**
- **functional correctness**
- **functional appropriateness**

Performance efficiency

- **time behavior**
- **resource utilization**
- **capacity**

Compatibility

- **coexistence**
- **interoperability**

Usability

- **appropriateness recognizability**
- **learnability**
- **operability**
- **user error prediction**
- **user interface aesthetics**
- **accessibility**

ISO/IEC 25010 SQuaRE -1

Reliability

- **maturity**
- **availability**
- **fault tolerance**
- **recoverability**

Security

- **confidentiality**
- **integrity**
- **nonrepudiation**
- **accountability**
- **authenticity**

Maintainability

- **modularity**
- **reusability**
- **analyzability**
- **modifiability**
- **testability**

Portability

- **adaptability**
- **installability**
- **replaceability**

Dealing with a New Quality Attribute

Capture scenarios

Assemble design approaches

Model the new quality attribute

Assemble a set of tactics

Construct design checklists

Software Architecture Topics

Introduction to Architecture

Quality Attributes
- Availability
- Interoperability
- Modifiability
- Performance
- Security
- Testability
- Usability

Other Quality Attributes

→ Patterns and Tactics

Architecture in Agile Projects

Designing an Architecture

Documenting Software Architectures

Architecture and Business

Architecture and Software Product Lines

The Brave New World

Architectural Pattern

Is a package of design decisions that is found repeatedly in practice

Has known properties that permit reuse

Describes a class of architectures

Inspired by Christopher Alexander's A Pattern Language: Towns, Buildings, Construction (1977).

The Gang of Four (GoF): E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, 1994.

Discovering Patterns

Patterns are by definition found in practice

- **one does not invent them**
- **one discovers them**

Patterns spontaneously emerge in reaction to environmental conditions

- **as long as conditions change, new patterns will emerge**

Tactics vs Patterns

Tactics typically use just a single structure or computational mechanism.

- **meant to address a single architectural force (quality attribute)**
- **the “building blocks” of design**
- **comparable to atoms**

Patterns typically combine multiple design decisions into a package.

- **are constructed from several different tactics**
- **patterns package tactics**
- **comparable to molecules**

Patterns Establish A Relationship

A context

- **A recurring, common situation in the world that gives rise to a problem.**

A problem

- **outlines the problem and its variants**
- **describes any complementary or opposing forces**
- **includes quality attributes that must be met**

A solution

- **appropriately abstracted**
- **describes the architectural structures that solve the problem**
- **how to balance the forces at work**

The Solution for a Pattern

A set of element types

- **data repositories, processes, objects, …**

A set of interaction mechanisms or connectors

- **method calls, events, message bus, …**

A topological layout of the components

A set of semantic constraints covering topology, element behavior, and interaction mechanisms

Complex systems exhibit multiple patterns at once.

Patterns Catalog Overview

Module patterns

- **layered**

Component-and-connector patterns

- **broker**
- **model-view-controller**
- **pipe-and-filter**
- **client-server**
- **peer-to-peer**
- **service-oriented architecture**
- **publish-subscribe**
- **shared-data**

Allocation patterns

- **map-reduce**
- **multi-tier**

Layered Pattern -1

Context

- **all complex systems experience the need to develop and evolve portions of the system independently**
- **need a clear and well-documented separation of concerns**

Problem

- **The software needs to be segmented in such a way that the modules can be developed and evolved separately with little interaction among the parts, supporting portability, modifiability, and reuse.**

Layered Pattern -2

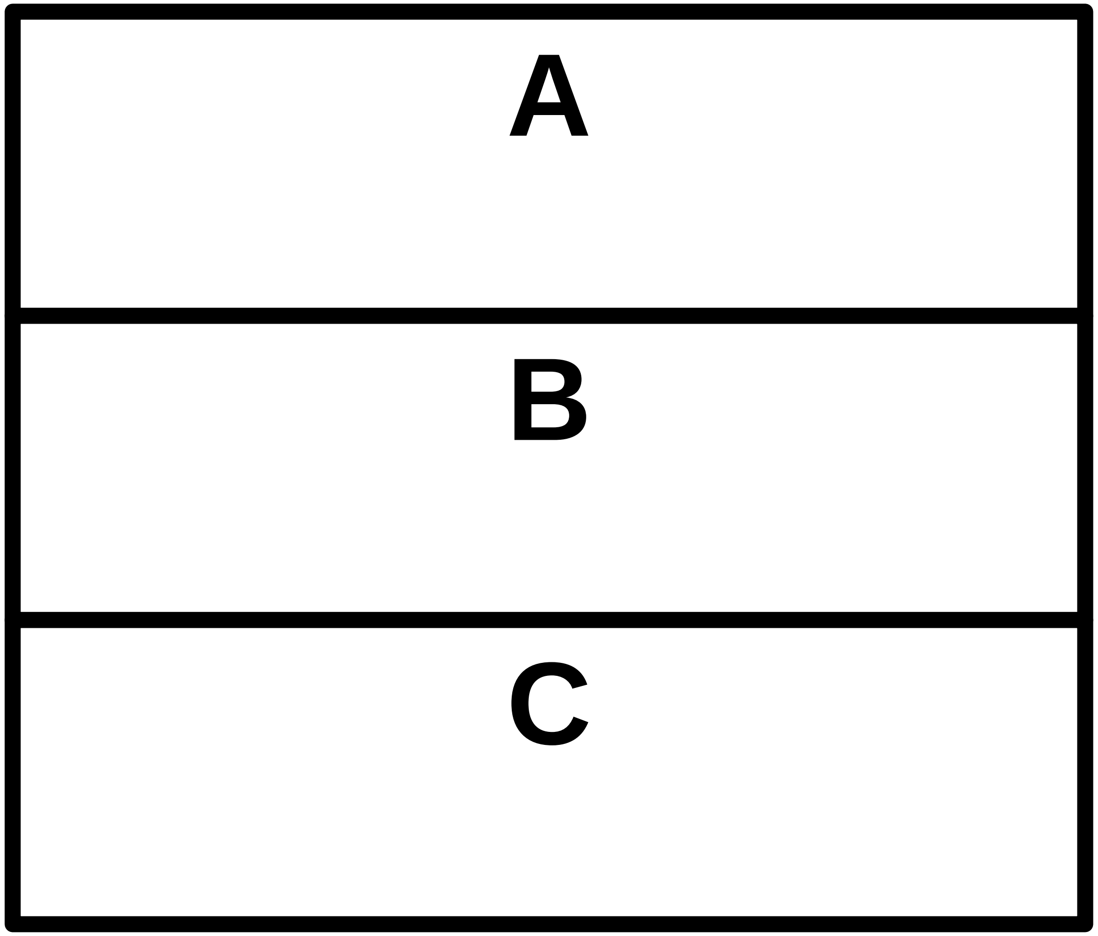
Solution

- **divide the software into units called layers**
- **each layer is a grouping of modules that offers a cohesive set of services**
- **completely partition a set of software**
- **a public interface**
- **relations between layers must be unidirectional**
  - **if (A,B) is in this relation, we say that the implementation of layer A is allowed to use any of the public facilities provided by layer B**

Pattern Almanac 2000 by L. Rising lists over 100 patterns that are variants of, or related to, Layers.

Stack of Boxes Notation

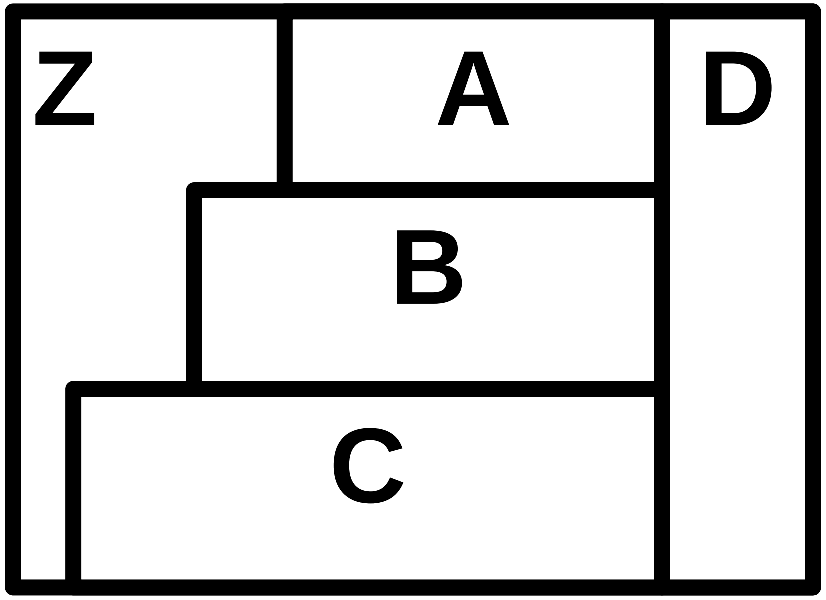
Allowed-to-use relation reads from top down



Finer Points of Layers
Bridging

It is impossible to look at a stack of boxes and tell whether layer bridging is allowed or not.

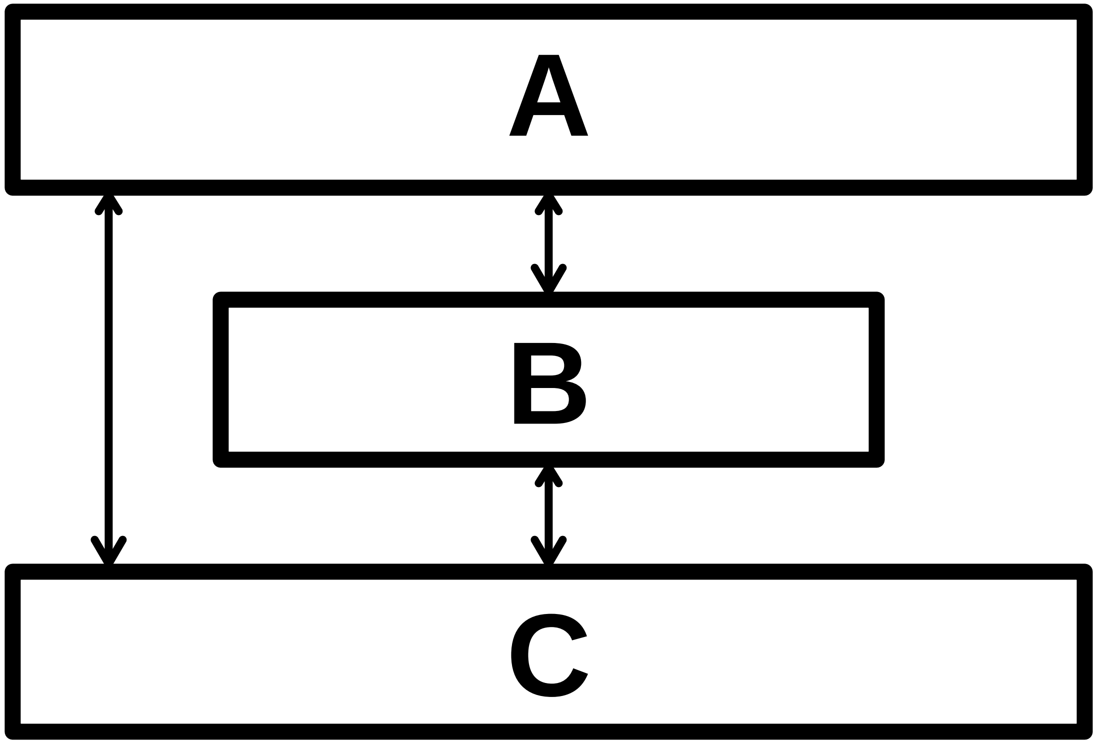
- **add stairsteps or vertical layers to your notation**
- **adding a key is essential!**



Finer Points of Layers
Arbitrary Allowed-to-Use

Any old set of boxes stacked on top of each other does not constitute a layered architecture.

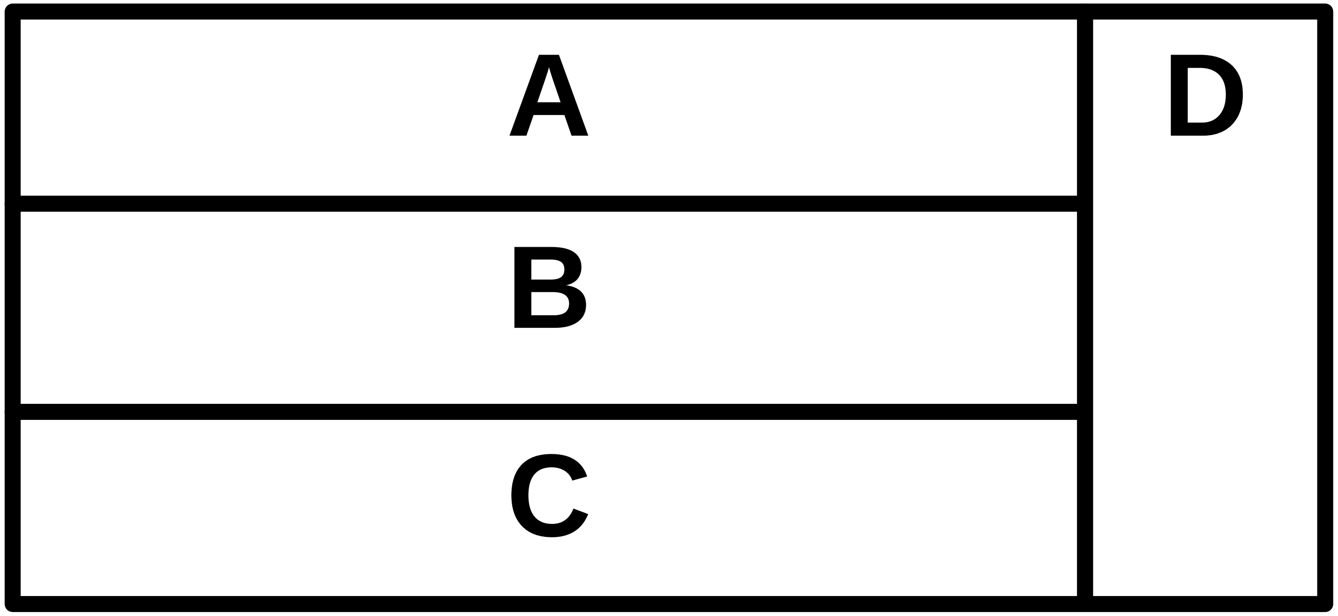
- **avoid arrows (allowed to use)**



Finer Points of Layers
Sidecars

“Sidecars” may contain common utilities (sometimes imported).

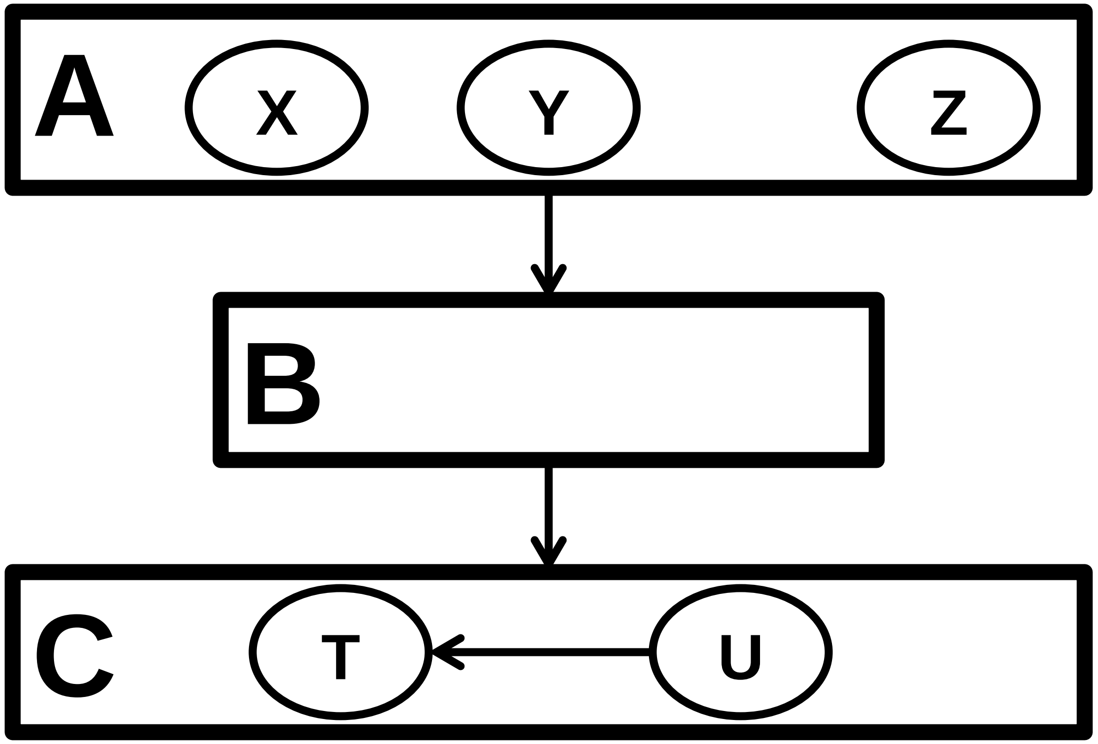
- **without a key, are you sure?**
- **implies bridging is not allowed…**



Finer Points of Layers
Segments

Segments may denote a finer-grained decomposition of the modules.

- **specify what usage rules are in effect among the segments**



Finer Points of Layers
No Using Above

The most important point about layering is that a layer isn’t allowed to *use* any layer above it.

A module “uses” another module when it depends on the answer it gets back.

- **a layer is allowed to make upward calls, as long as it isn’t expecting an answer**
- **this is how the common error-handling scheme of callbacks works**

Advantages of the Layered Pattern

Support design based on increasing levels of abstraction

Support enhancement

- **affect at most two other layers**

Support reuse

- **standard interfaces with multiple implementations**

Weaknesses of the Layered Pattern

Adds up-front cost and complexity

Contribute a performance penalty

Bridging may prevent meeting portability and modifiability goals

Example of the Layered Pattern

ISO Open System Interconnection (OSI)

- **physical**
- **data link**
- **network**
- **transport**
- **session**
- **presentation**
- **application**

Patterns Catalog Overview

Module patterns

- **layered**

Component-and-connector patterns

- **broker**
- **model-view-controller**
- **pipe-and-filter**
- **client-server**
- **peer-to-peer**
- **service-oriented architecture**
- **publish-subscribe**
- **shared-data**

Allocation patterns

- **map-reduce**
- **multi-tier**

Pipe-and-Filter Pattern -1

Context

- **many systems transform streams of discrete data items,**
- **many types of transformations occur repeatedly in practice**
- **create these as independent, reusable parts**

Problem

- **divide into reusable, loosely coupled components with simple, generic interaction mechanisms**
- **can execute in parallel**

Pipe-and-Filter Pattern -2

Solution

- **successive transformations of streams of data**

  Data
  - **arrives at a filter's input port(s)**
  - **is transformed**
  - **is passed via its output port(s) through a pipe to the next filter**

- **a single filter can consume data from, or produce data to, one or more ports**

Advantages of the Pipe-and-Filter Pattern

Pipes buffer data during communication.

- **filters can execute asynchronously and concurrently**

Overall computation can be treated as the functional composition of the computations of the filters, making it easier to reason about end-to-end behavior.

Independent processing at each step supports reuse and parallelization.

Weaknesses of the Pipe-and-Filter Pattern

Typically not a good choice for an interactive system

- **disallows cycles, which are important for user feedback**

Large numbers of independent filters can add substantial amounts of overhead

- **each filter runs as its own thread or process**

May not be appropriate for long-running computations

- **need checkpoint/restore functionality**

Examples of the Pipe-and-Filter Pattern

Unix!

Data transformation systems for sensor data

Relationships Between Tactics and Patterns

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

Most patterns are constructed from several different tactics.

- **these tactics might all serve a common purpose**
- **they are often chosen to promote different quality attributes**

A pattern is a general solution.

A documented pattern is underspecified with respect to applying it in a specific situation.

	Modifiability									
	Increase Cohesion		Reduce Coupling					Defer Binding Time		
Pattern	Increase Semantic Coherence	Abstract Common Services	Encapsulate	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Startup-Time Binding	Use Runtime Binding
Layered	X	X	X		X	X	X			
Pipes and Filters	X		X		X	X			X	
Blackboard	X	X			X	X	X	X		X
Broker	X	X	X		X	X	X	X		
Model View Controller	X		X			X				X
Presentation Abstraction Control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							

Two Perspectives

To make a pattern work…

Inherent quality attribute tradeoffs that the pattern makes

- **Patterns exist to achieve certain quality attributes, and we need to compare the ones they promote (and the ones they diminish) with our needs.**

Other quality attributes that the pattern isn’t directly concerned with

- **but it affects…**
- **which are important in our application**

Using Tactics Together

Decided to employ ping/echo to detect failed components →

Security

- **How to prevent a ping flood attack?**

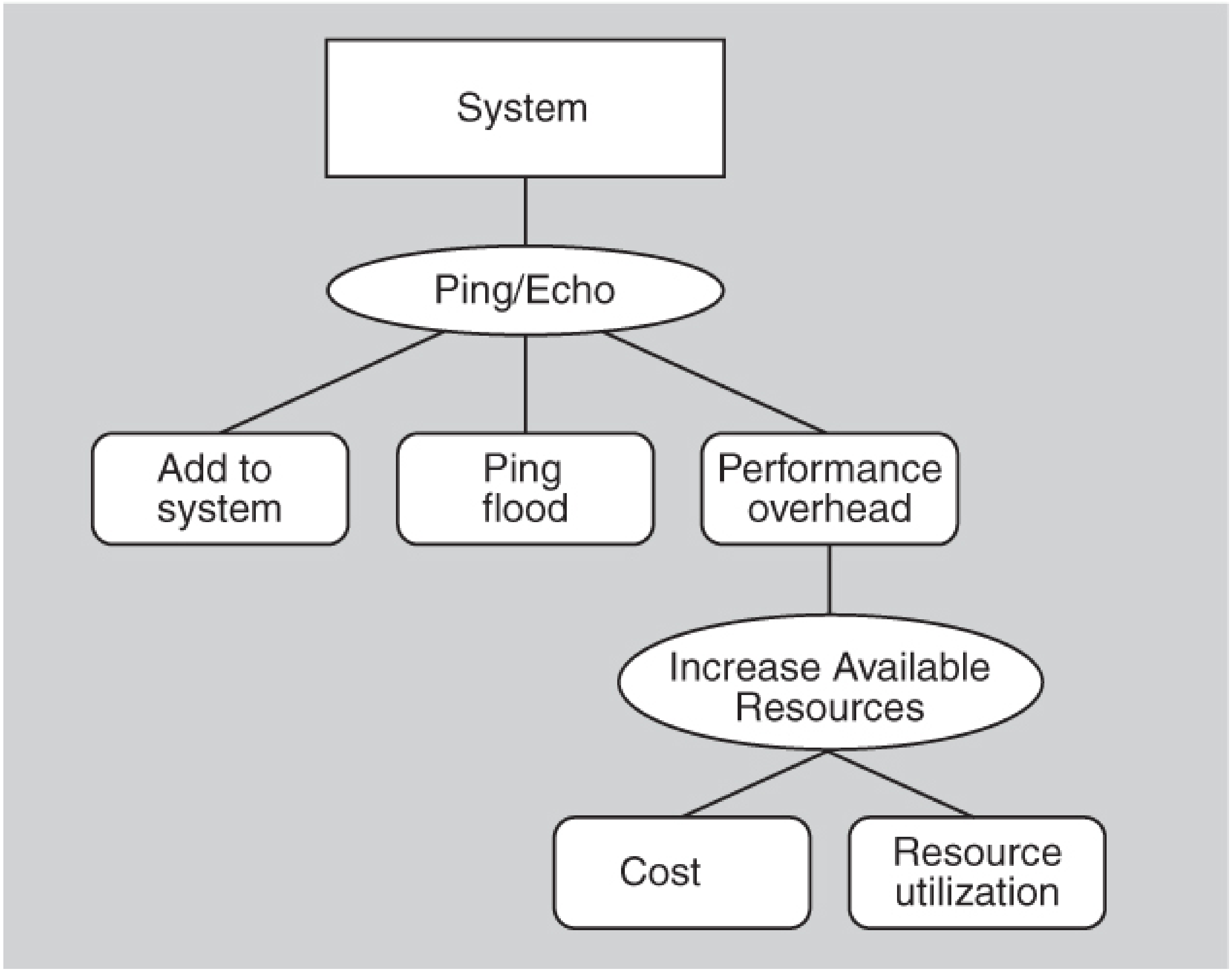
Performance

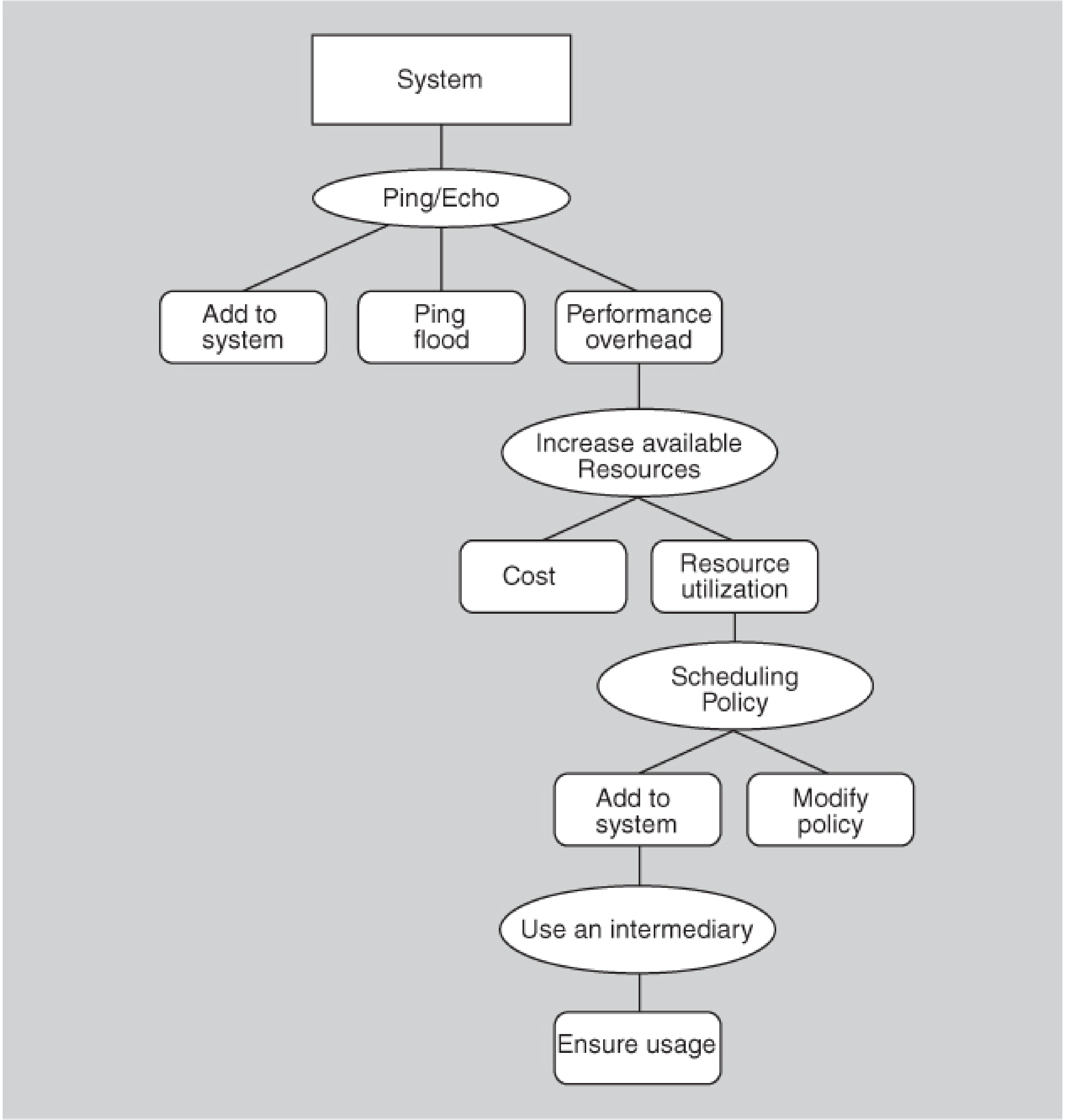
- **How to ensure that the performance overhead of ping/echo is small?**

Modifiability

- **How to add ping/echo to the existing architecture?**

Focus on Performance Tradeoff




System
Ping/Echo
Add to system
Ping flood
Performance overhead
Increase available Resources
Cost
Resource utilization
Scheduling Policy
Add to system
Modify policy
Use an intermediary
Ensure usage

Questions and Answers