Quality Attribute Scenarios and Tactics

Chapters 5-11 in Text

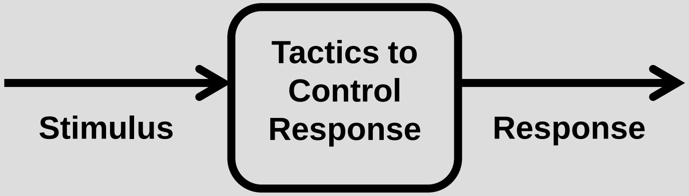
Some material in these slides is adapted from Software Architecture in Practice, 3rd edition by Bass, Clements and Kazman.

Quality Attributes – Master List

- **Operational categories**
  - **Availability**
  - **Interoperability**
  - Reliability
  - **Usability**
  - **Performance**
  - Deployability
  - Scalability
  - Monitorability
  - Mobility
  - Compatibility
  - **Security**
  - Safety
- **Developmental categories**
  - **Modifiability**
  - Variability
  - Supportability
  - **Testability**
  - Maintainability
  - Portability
  - Localizability
  - Development distributability
  - Buildability

Achieving Quality Attributes – Design Tactics

- A system design is a **collection of design decisions**
- Some respond to **quality attributes**, some to achieving **functionality**
- A **tactic** is a **design decision** to achieve a **QA response**
- **Tactics are a building block of architecture patterns** – more primitive/granular, proven design technique



Stimulus

Tactics to Control Response

Response

Categories of Design Decisions

- **Allocation** of **responsibilities** – system functions to modules
- **Coordination model** – module interaction
- **Data model** – operations, properties, organization
- **Resource management** – use of shared resources
- **Architecture element mapping** – logical to physical entities; i.e., threads, processes, processors
- **Binding time decisions** – variation of life cycle point of module “connection”
- **Technology choices**

Design Checklists

- **Design considerations** for each **QA** organized by **design decision category**
- For example, allocation of system responsibilities for performance:
  - What responsibilities will involve **heavy loading or time critical response**?
  - What are the processing requirements, will there be **bottlenecks**?
  - How will **threads of control** be handled across process and processor boundaries?
  - What are the responsibilities for managing **shared resources?**

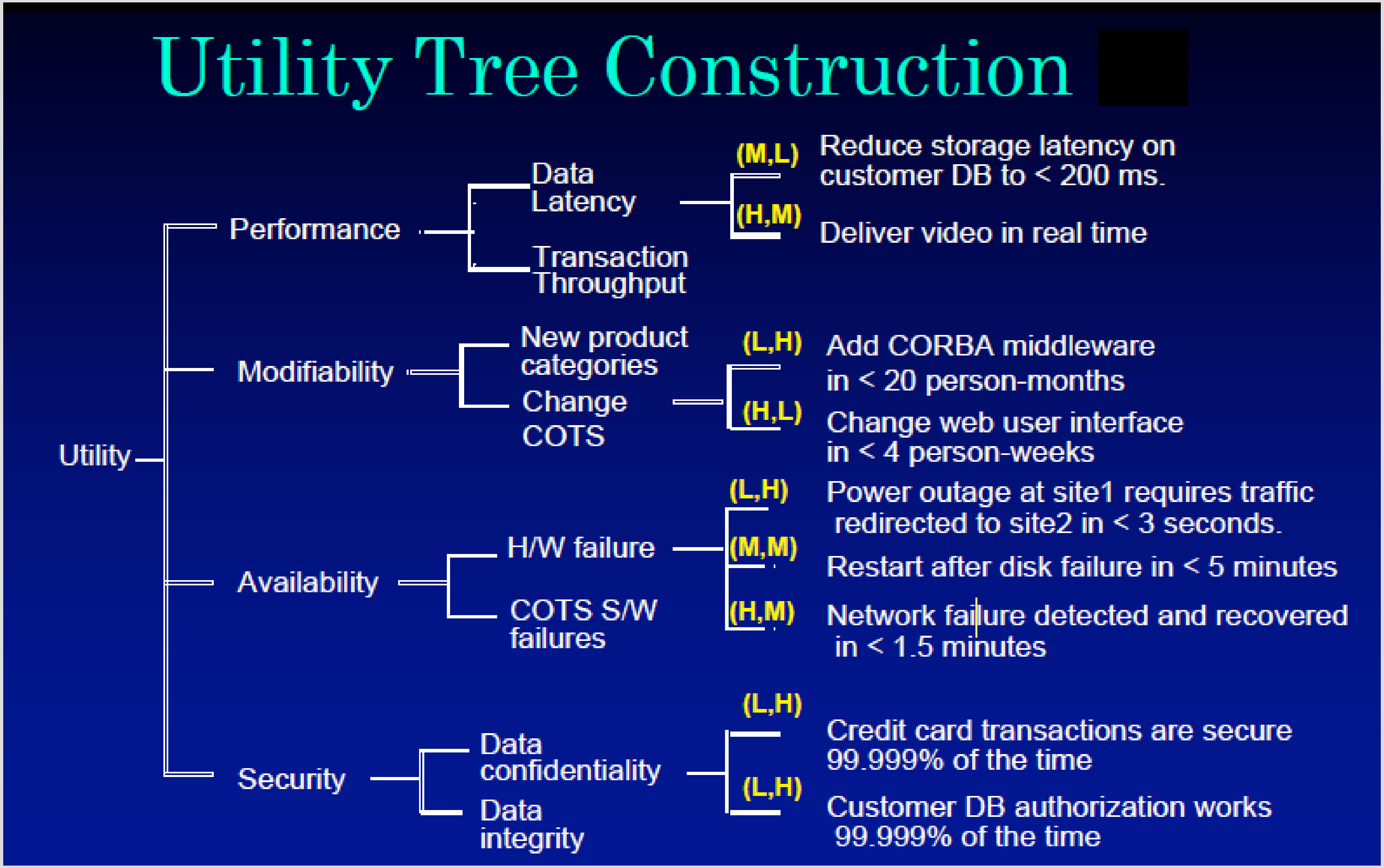
QA Utility Tree

Capture all QA’s (ASRs) in one place

QA	Attribute Refinement	ASR scenario
	Response time	**Scenario … (Priority)**
Performance	Throughput	**Scenario … (Priority)**
Security	Privacy	**Scenario … (Priority)**
	Integrity	**Scenario … (Priority)**
Availability	Downtime	**Scenario … (Priority)**
	…	
Modifiability	**...**	

QA Utility Tree(cont)

- “**Utility**” to express the overall “**goodness**” of the system
- QA utility tree construction:
  - Most important **QA goals** are high level **nodes** (typically performance, modifiability, security, and availability)
  - **Scenarios** are the **leaves**
  - Output: a characterization and prioritization of specific quality attribute requirements.
  - High/Medium/Low **importance** for the success of the system
  - High/Medium/Low **difficulty** to achieve (architect’s assessment)



Utility Tree Construction

[redacted]

- Utility
  - Performance
    - Data Latency
      - (M,L) Reduce storage latency on customer DB to < 200 ms.
      - (H,M) Deliver video in real time
    - Transaction Throughput
  - Modifiability
    - New product categories
    - Change COTS
      - (L,H) Add CORBA middleware in < 20 person-months
      - (H,L) Change web user interface in < 4 person-weeks
  - Availability
    - H/W failure
      - (L,H) Power outage at site1 requires traffic redirected to site2 in < 3 seconds.
      - (M,M) Restart after disk failure in < 5 minutes
      - (H,M) Network failure detected and recovered in < 1.5 minutes
    - COTS S/W failures
  - Security
    - Data confidentiality
      - (L,H) Credit card transactions are secure 99.999% of the time
      - (L,H) Customer DB authorization works 99.999% of the time
    - Data integrity

System Quality Attributes

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

Note: design tactics across QA's may conflict requiring design tradeoffs

Availability

- A measure of the impact of failures and faults
- Mean time to failure, repair
- Downtime

Probability system is operational when needed:

(exclude scheduled downtime)

$$\alpha = \frac{\text{mean time to failure}}{\text{mean time to failure + mean time to repair}}$$

Availability Table

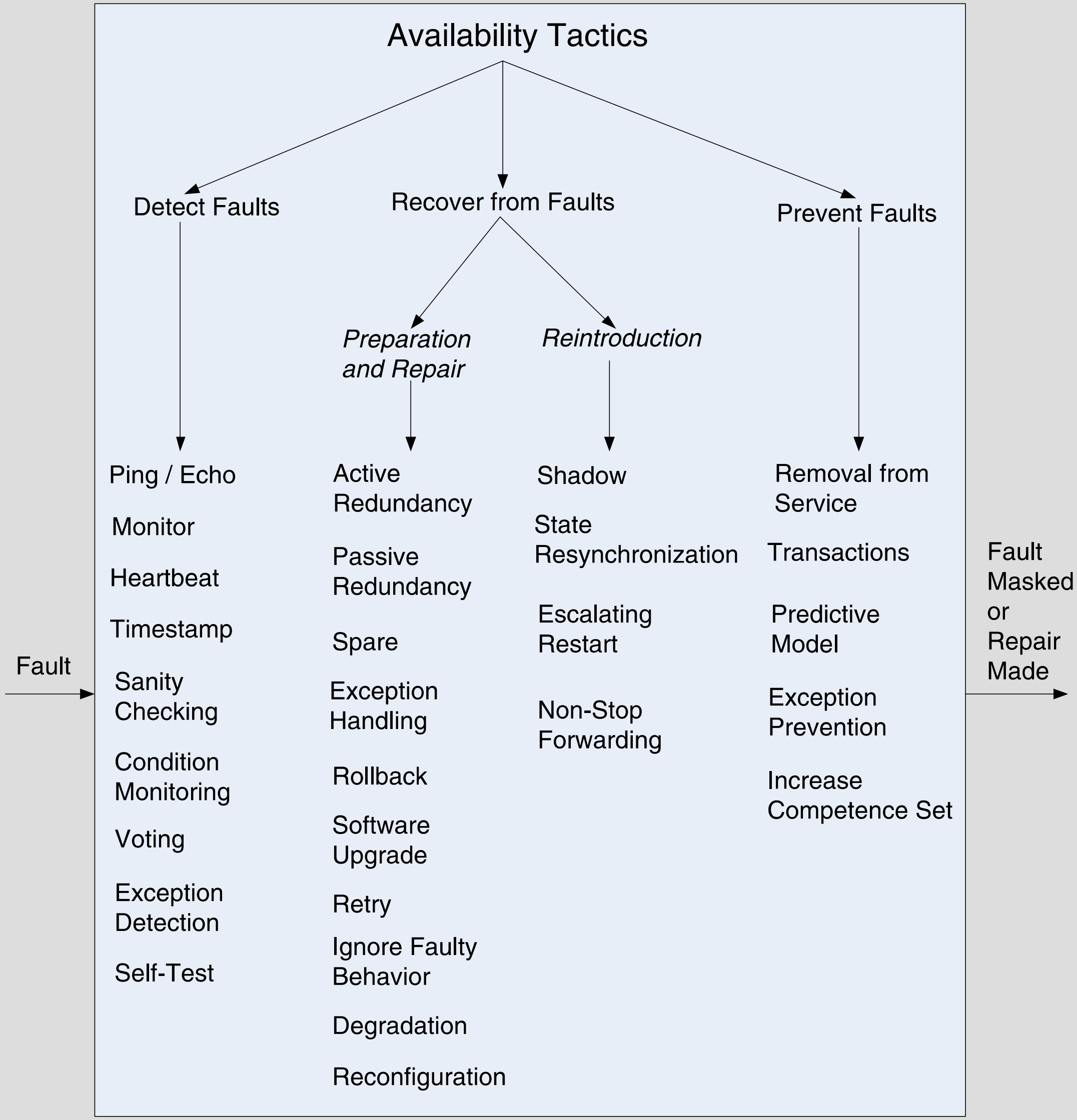
- Source: internal, external
- Stimulus: fault: omission, crash, timing, response
- Artifact: processors, channels, storage, processes
- Environment: normal, degraded
- Response: logging, notification, switching to backup, restart, shutdown
- Measure: availability, repair time, required uptime

Availability Scenario Example

Availability of the crossing gate controller:

Scenario: Main processor fails to receive an acknowledgement from gate processor.

- Source: external to system
- Stimulus: timing
- Artifact: communication channel
- Environment: normal operation
- Response: log failure and notify operator via alarm
- Measure: no downtime



Availability Tactics

Fault

Detect Faults

- Ping / Echo
- Monitor
- Heartbeat
- Timestamp
- Sanity Checking
- Condition Monitoring
- Voting
- Exception Detection
- Self-Test

Recover from Faults

Preparation and Repair

- Active Redundancy
- Passive Redundancy
- Spare
- Exception Handling
- Rollback
- Software Upgrade
- Retry
- Ignore Faulty Behavior
- Degradation
- Reconfiguration

Reintroduction

- Shadow
- State Resynchronization
- Escalating Restart
- Non-Stop Forwarding

Prevent Faults

- Removal from Service
- Transactions
- Predictive Model
- Exception Prevention
- Increase Competence Set

Fault Masked or Repair Made

System Quality Attributes

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

Interoperability

- The degree to which two or more systems can **usefully exchange** meaningful information in a particular context
- Exchange data – syntactic interoperability
- Interpret exchanged data – semantic interoperability
- To provide a service
- To integrate existing systems – system of systems (SoS)
- May need to discover the service at runtime or earlier
- Some request/response scenario

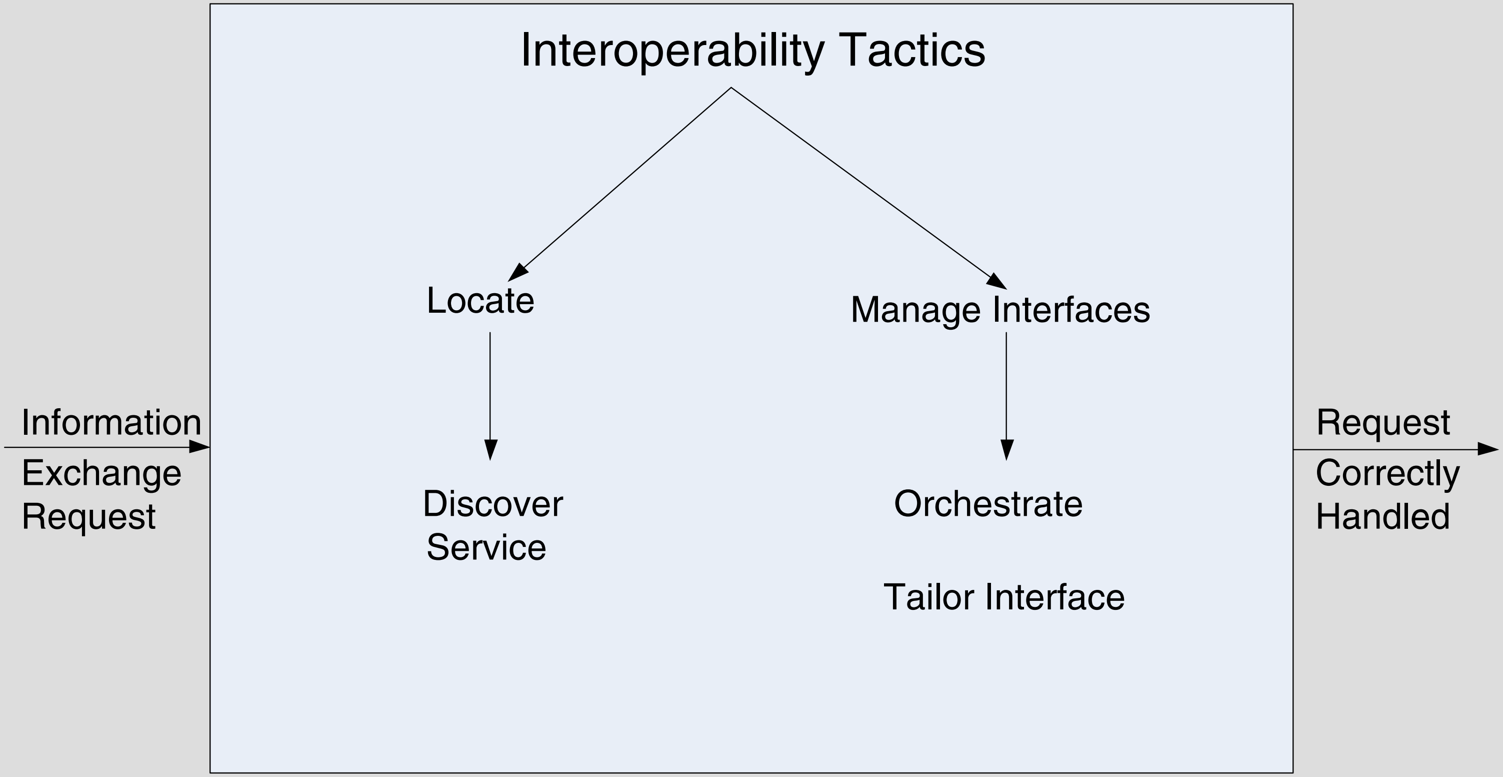
Interoperability General Scenario

- Source: a 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 run time or known prior to run time
- 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 successfully exchanged and understood
  - 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 Concrete Scenario

- Our vehicle information system sends our current location to the traffic monitoring system which combines our location with other information, overlays on a Google Map, and broadcasts it.

- Source: vehicle information system
- Stimulus: current location sent
- Artifact: traffic monitoring system
- Environment: systems known prior to runtime
- Response: traffic monitor combines current location with other data, overlays on Google Maps and broadcasts
- Response measure: Our information included correctly 99.9% of time


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

System Quality Attributes

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

Performance

- **Event arrival** patterns and load
  - Periodic – fixed frequency
  - Stochastic – probability distribution
  - Sporadic – random
- **Event servicing**
  - Latency - Time between the arrival of stimulus and the system’s response to it
  - Jitter - Variation in latency
  - Throughput - Number of transactions the system can process in a second
  - Events and data not processed

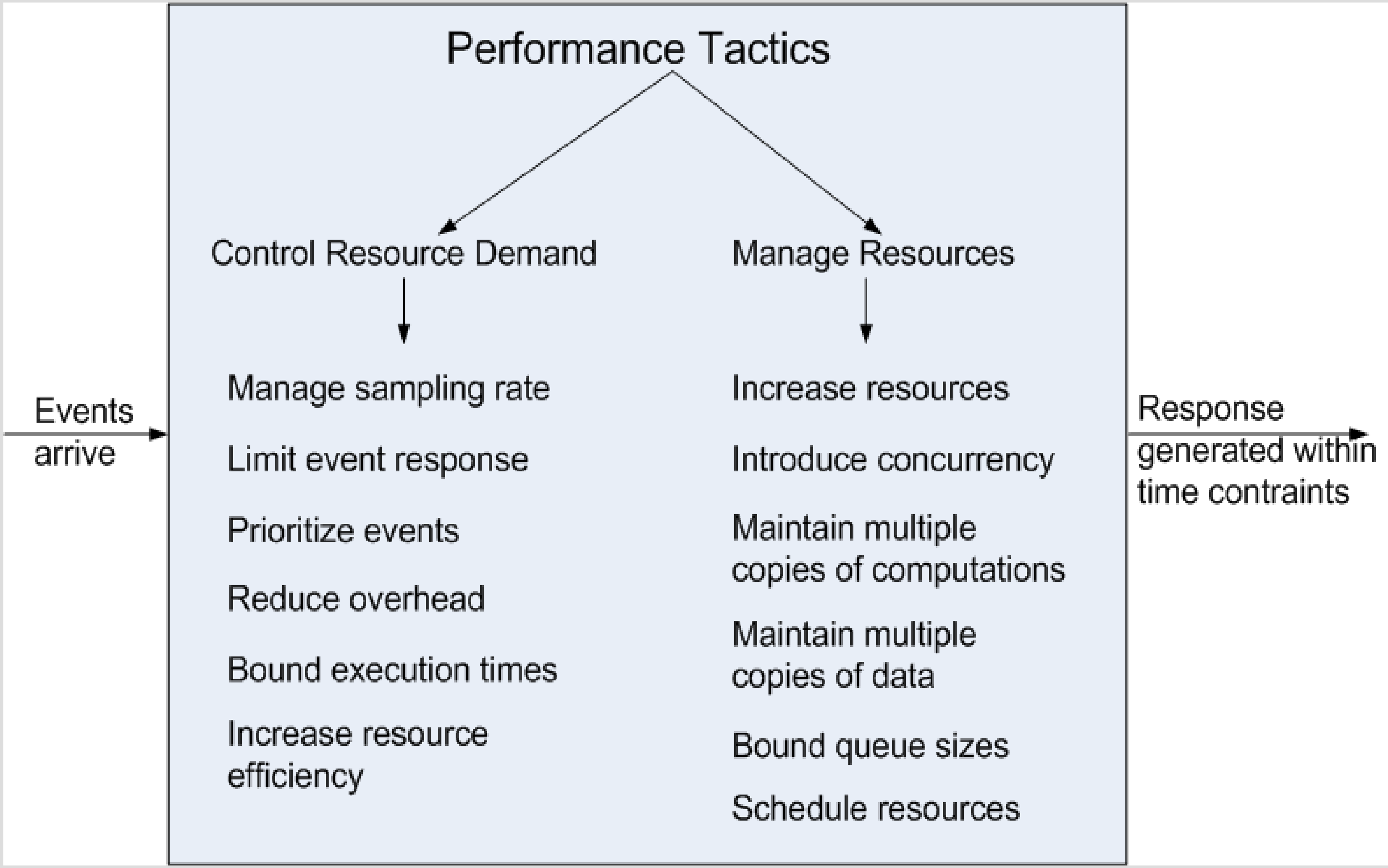
Performance Table

- Source: external, internal
- Stimulus: event arrival pattern
- Artifact: system services
- Environment: normal, overload
- Response: change operation mode?
- Measure: latency, deadline, throughput, jitter, miss rate, data loss

Performance Scenario Example

Performance of the crossing gate controller:

- Scenario: Main processor commands gate to lower when train approaches.
- Source: external - arriving train
- Stimulus: sporadic
- Artifact: system
- Environment: normal mode
- Response: remain in normal mode
- Measure: send signal to lower gate within 1 millisecond



Performance Tactics

Events arrive

Control Resource Demand

- Manage sampling rate
- Limit event response
- Prioritize events
- Reduce overhead
- Bound execution times
- Increase resource efficiency

Manage Resources

- Increase resources
- Introduce concurrency
- Maintain multiple copies of computations
- Maintain multiple copies of data
- Bound queue sizes
- Schedule resources

Response generated within time contraints

System Quality Attributes

- Availability
- **I**nteroperability
- Performance
- **Security**
- Modifiability
- Testability
- Usability

Security

- **Non-repudiation** – cannot deny existence of executed transaction
- **Confidentiality** – privacy, no unauthorized access
- **Integrity** – information and services delivered as intended and expected
- **Authentication** – parties are who they say they are
- **Availability** – no denial of service
- **Authorization** – grant users privileges to perform tasks

Security Table

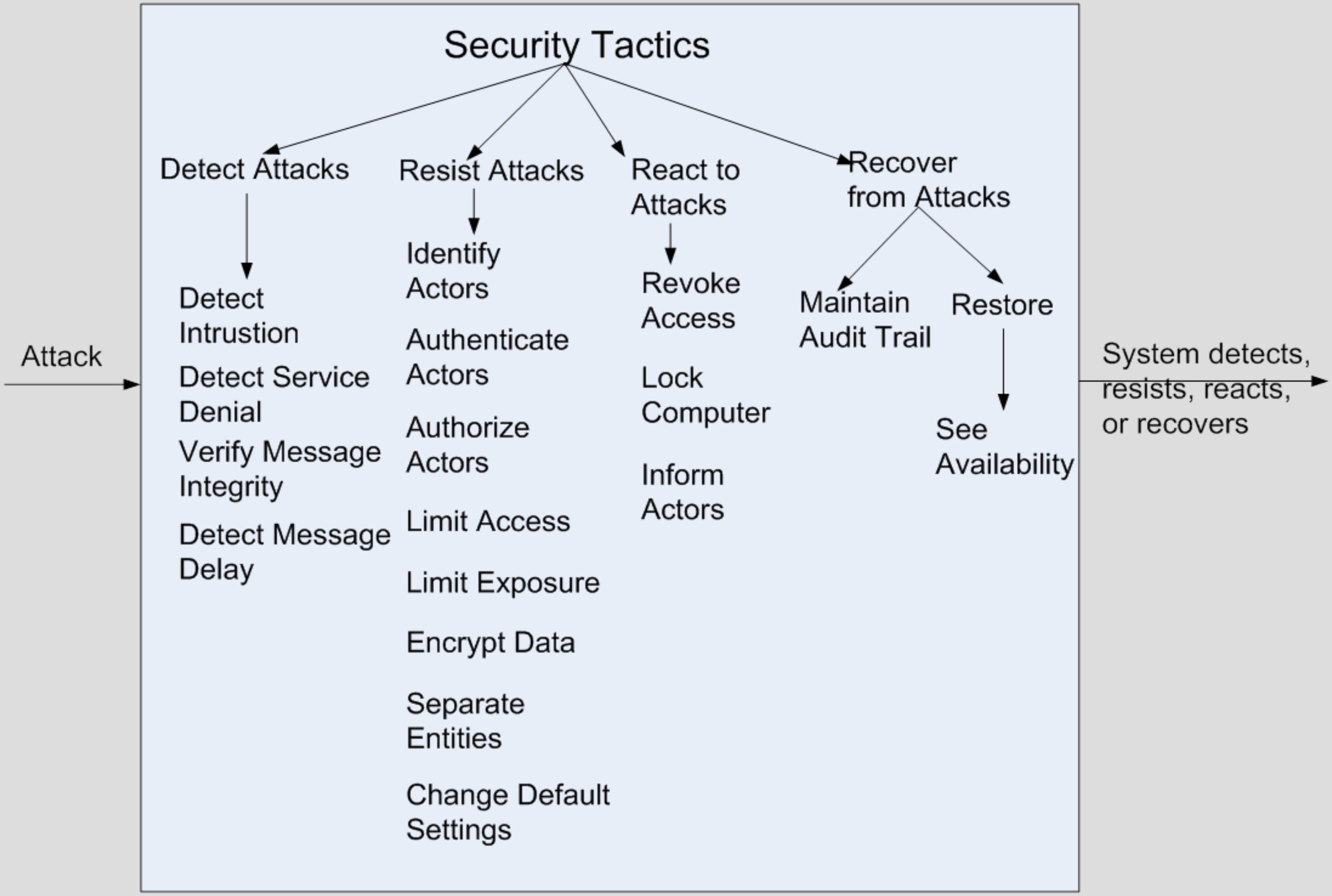
- Source: user/system, known/unknown
- Stimulus: attack to display info, change info, access services and info, deny services
- Artifact: services, data
- Environment: online/offline, connected or disconnected
- Response: authentication, authorization, encryption, logging, demand monitoring
- Measure: time, probability of detection, recovery

Security Scenario Example

Security of the crossing gate controller:

Scenario: Hackers are prevented from disabling system.

- Source: unauthorized user
- Stimulus: tries to disable system
- Artifact: system service
- Environment: online
- Response: blocks access
- Measure: service is available within 1 minute


Security Tactics
Attack
Detect Attacks
Detect Intrusion
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

System Quality Attributes

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

Modifiability

- What can change?
- When is it changed?
- Who changes it?

Modifiability Table

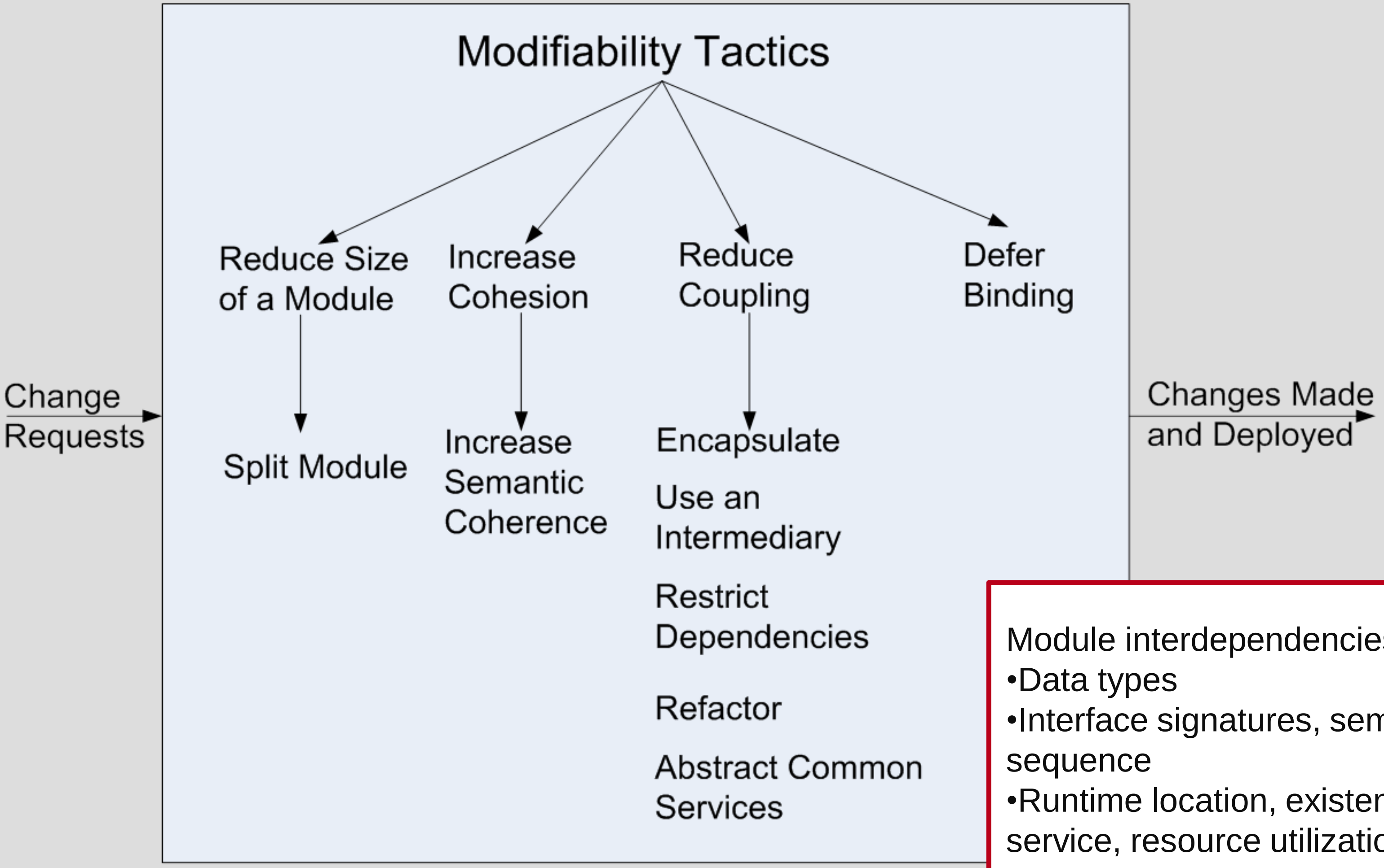
- Source: developer, system administrator, user
- Stimulus: add/delete/modify function or quality
- Artifact: UI, platform, environment, external system
- Environment: design, compile, build, run time
- Response: make change, test it, deploy it
- Measure: effort, time, cost, risk

Modifiability Scenario Example

Modifiability of a restaurant locator App:

Scenario: User may change behavior of system

- Source: end user
- Stimulus: wishes to change locale of search
- Artifact: list of available country locales
- Environment: runtime
- Response: user finds option to download new locale database; system downloads and installs it successfully
- Measure: download and installation occurs automatically



Modifiability Tactics

Change Requests

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

Changes Made and Deployed

Module interdependencies:
- Data types
- Interface signatures, semantics, control sequence
- Runtime location, existence, quality of service, resource utilization

System Quality Attributes

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

Testability

- The ease with which software can be made to demonstrate faults through testing
- Assuming software has one fault, the probability of fault discovery on *next* test execution
- Need to control components internal state and inputs
- Need to observe components output to detect failures

Testing activities can consume up to 40% of a project

Testability Table

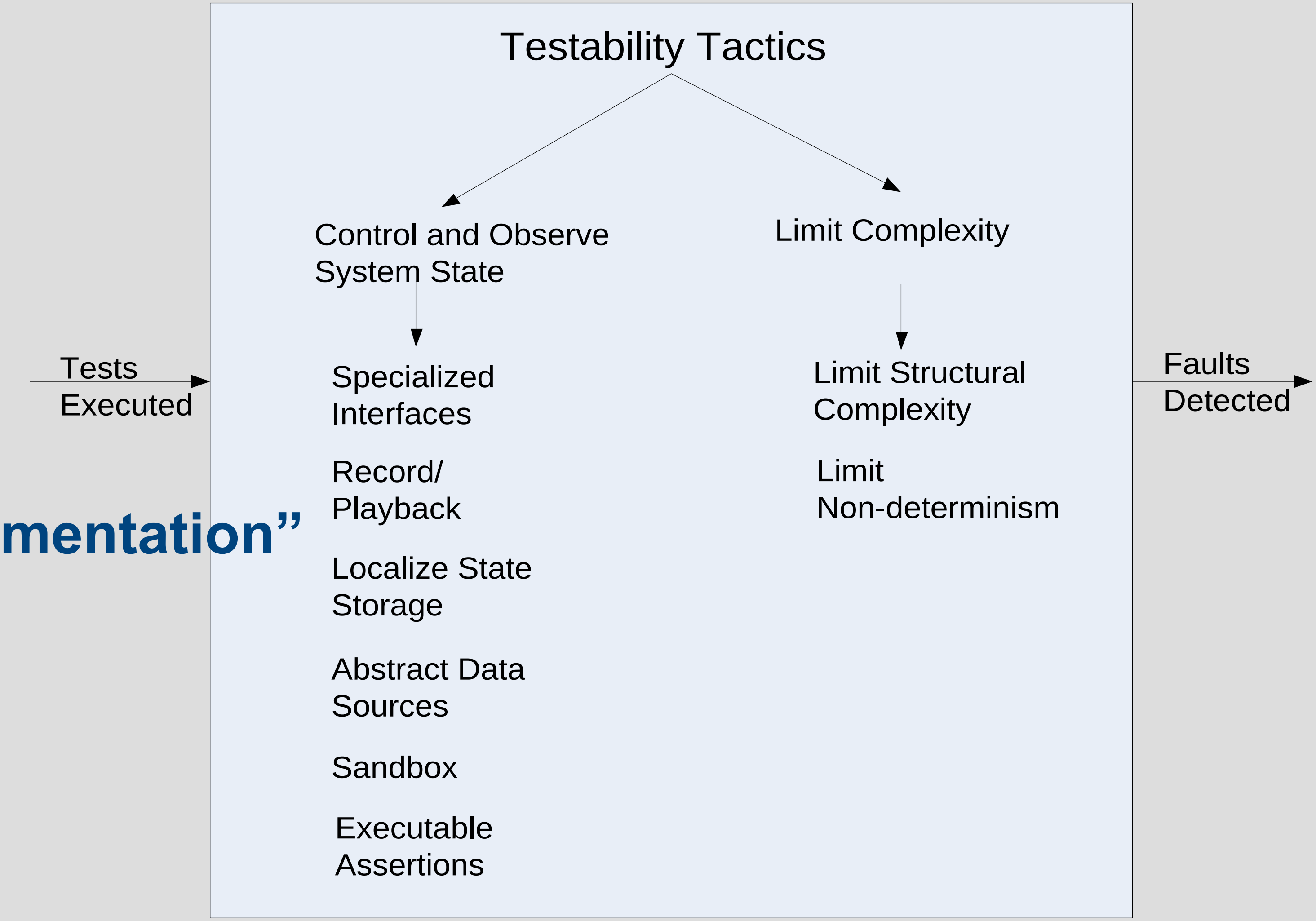
- Source: developer, tester, user
- Stimulus: project milestone completed
- Artifact: design, code component, system
- Environment: design, development, compile, deployment, or run time
- Response: can be controlled to perform the desired test and results observed
- Measure: coverage, probability of finding additional faults given a fault, time to test

Specific Testability Scenario example

Testability of a photo editor application:

Scenario: New versions of system can be completely tested relatively quickly.

- Source: system tester
- Stimulus: integration completed
- Artifact: whole system
- Environment: development time
- Response: all functionality can be controlled and observed
- Measure: entire regression test suite completed in less than 24 hours



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 Non-determinism

Faults Detected

“Instrumentation”

System Quality Attributes

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

Usability

- Ease of learning system features – **learnability**
- Ease of remembering – **memorability**
- Using a system **efficiently**
- Minimizing the impact of errors – **understandability**
- Increasing confidence and **satisfaction**

Usability Table

- Source: end user
- Stimulus: wish to learn/use/minimize errors/adapt/feel comfortable
- Artifact: system
- Environment: configuration or runtime
- Response: provide ability or anticipate (support good UI design principles)
- Measure: task time, number of errors, user satisfaction, efficiency, time to learn

Usability Scenario example

Usability of a restaurant locator App:

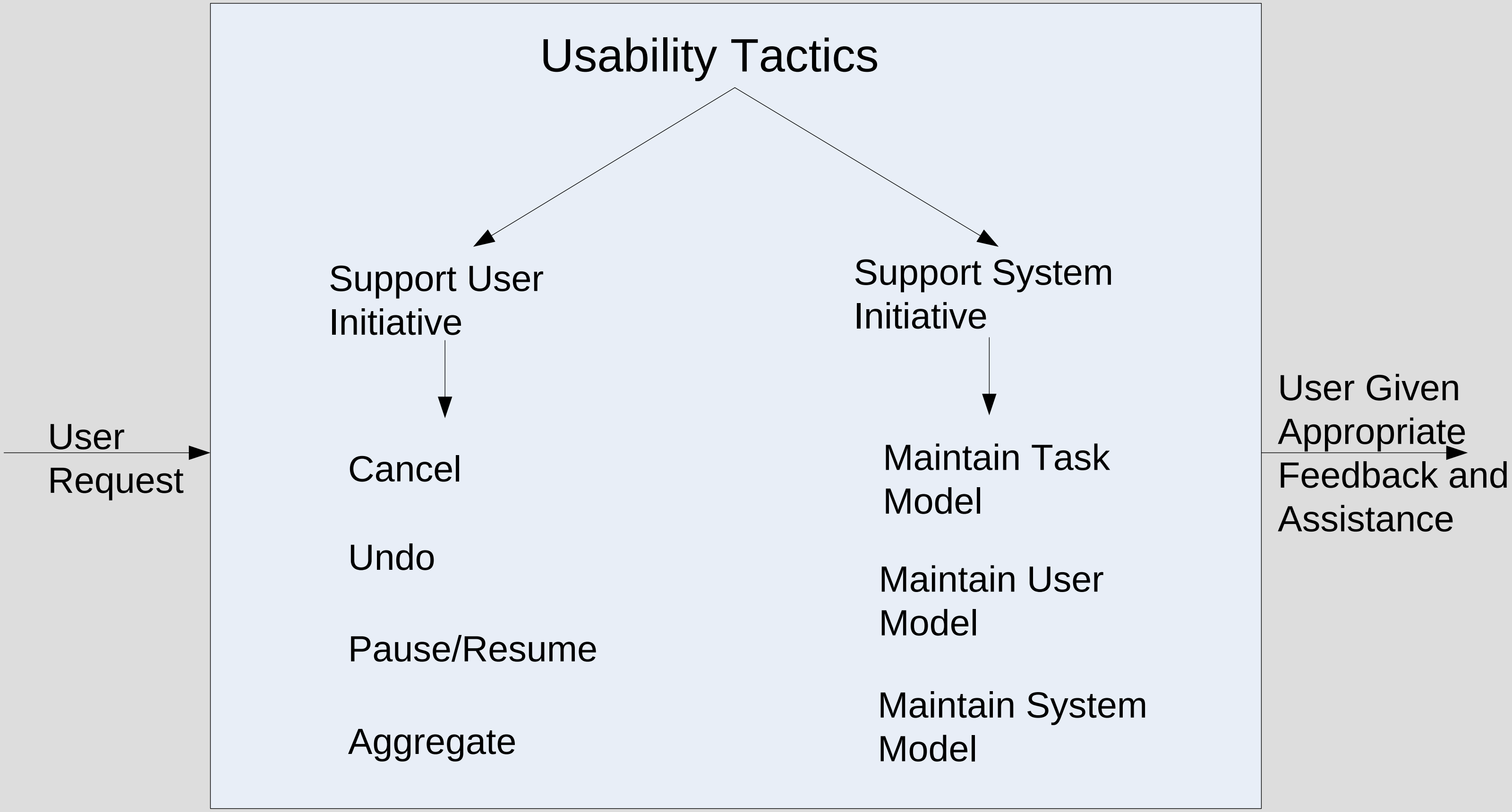
Scenario: User may undo actions easily.

- Source: end user
- Stimulus: minimize impact of errors
- Artifact: system
- Environment: runtime
- Response: wishes to undo a filter
- Measure: previous state restored within one second

Other Examples of Architecturally Significant Usability Scenarios

- Aggregating data
- Canceling commands
- Using applications concurrently
- Maintaining device independence
- Recovering from failure
- Reusing information
- Supporting international use
- Navigating within a single view
- Working at the user’s pace
- Predicting task duration
- Comprehensive search support

Can you explain how these have architectural implications?



Usability Tactics

User Request

Support User Initiative

Cancel

Undo

Pause/Resume

Aggregate

Support System Initiative

Maintain Task Model

Maintain User Model

Maintain System Model

User Given Appropriate Feedback and Assistance

QA Analysis Exercise

	Avail	Security	Perf	Inter	Mod	Test	Use
Enterprise inventory control							
Smart phone map app							
IDE (e.g. Eclipse)							
Operating system							
Medical records DB							
Video game							
Social network site							
Plane auto pilot							

Assign a QA priority of 1-5 for each system (1 lowest)