UNIT II
QUALITY ATTRIBUTE WORKSHOP

Quality Attribute Workshop – Documenting Quality Attributes – Six part scenarios – Case studies.

1.QUALITY ATTRIBUTE WORKSHOPS (QAWS)

- Quality attribute workshops (QAWs) provide a method for analyzing a system's architecture against a number of critical quality attributes, such as *availability, performance, security, interoperability, and modifiability*, that are derived from mission or business goals

- QAW is a systematic approach to elicit the needed requirements to ensure that all needed quality attributes are included in the final design.

- Quality Attribute Workshops are designed to gather stakeholders together, and get them to discuss the features of the system that they want.

- Helps the system architects to get maximum possible information about the quality

- Defining the structure of the system

- QAW is one way to discover document and prioritize a systems quality attributes early in its stage

- Scenarios are created for the attributes so that their purpose is better understood.
- This allows the developer to better understand what the attributes of the software need to be, and how they are supposed to be used.
- This also allows the stakeholders to better understand the system as a whole.
- The final goal is to produce documentation that includes as many as possible of the quality attributes specified by the stakeholders.

QAW has eight steps:

- QAW Presentation and Introductions
- Business/Mission Presentation
- Architectural Plan Presentation
- Identification of Architectural Drivers
- Scenario Brainstorming
- Scenario Consolidation
- Scenario Prioritization
- Scenario Refinement

2. **FUNCTIONALITY AND ARCHITECTURE**

- Functionality and quality attributes are orthogonal. (Used to describe two things that are independent of each other. One does not imply the other.)

- If functionality and quality attributes were not orthogonal, the choice of function would dictate the level of security or performance or availability or usability.

- Clearly though, it is possible to independently choose a desired level of each.

- Manipulating complex graphical images or sorting an enormous database might be naturally complex, making lightning-fast performance impossible.

- But what is possible is that, for any of these functions your choices as an architect will determine the relative level of quality.

- Some architectural choices will lead to higher performance; some will lead in the other direction.

- Important quality attribute were examined

2.1 WHAT IS FUNCTIONALITY?

- It is the ability of the system to do the work for which it was intended.

- A task requires that many or most of the system's elements work in a coordinated manner to complete the job, just as framers, electricians, plumbers, drywall hangers, painters, and finish carpenters all come together to cooperatively build a house.

- Therefore, if the elements have not been assigned the correct responsibilities or have not been capable with the correct facilities for coordinating with other elements the system will be unable to offer the required functionality.

- Functionality may be achieved through the use of any of a number of possible structures.

- Functionality requirement is decomposed into modules to make it understandable and to support a variety of other purposes.

- Software architecture constrains(limit) its allocation to structure when *other quality attributes are important.*

- *For example*, *systems are* frequently divided so that several people can cooperatively build them.

- The interest of functionality is how it interacts with, and constrains, those other qualities.

3. ARCHITECTURE AND QUALITY ATTRIBUTES

- Achieving quality attributes must be considered throughout design, implementation, and deployment.

- No quality attribute is entirely dependent on design, nor is it entirely dependent on implementation or deployment.

For example:

- Usability involves both architectural and non-architectural aspects.
- The non-architectural aspects include
  - making the user interface clear and easy to use.
  - Should you provide a radio button or a check box?
  - What screen layout is most understanding?
  - What typeface(font,style) is most clear?

- Although these details matter tremendously to the end user and influence usability, they are not architectural because they belong to the details of design.

- Whether a system provides the user with the ability to cancel operations, to undo operations, or to re-use data previously entered is architectural

- These requirements involve the cooperation of multiple elements.

- Modifiability is determined by how functionality is divided (architectural) and by coding techniques within a module (non-architectural).
- Thus, a system is modifiable if changes involve the fewest possible number of different elements.
- Performance involves both architectural and non-architectural dependencies.
- It depends partially on how much communication is necessary among components (architectural), partially on what functionality has been allocated to each component (architectural), partially on how shared resources are allocated (architectural), partially on the choice of algorithms to implement selected functionality (non-architectural), and partially on how these algorithms are coded (non-architectural).

This section is twofold:

- Architecture is critical to the realization of many qualities of interest in a system, and these qualities should be designed in and can be evaluated at the architectural level.

1.Architecture, by itself, is unable to achieve qualities. It provides the foundation for achieving quality, but this foundation will be to no avail if attention is not paid to the details.

2.Within complex systems, quality attributes can *never be achieved in isolation.*

- *The achievement of any one will have an effect*, sometimes positive and sometimes negative, on the achievement of others.
- For example, security and reliability often exist in a state of Mutual tension: The most secure system has the fewest points of failure—typically a security kernel.

- The most reliable system has the most points of failure—typically a set of redundant processes or processors where the failure of any one will not cause the system to fail.

- Another example of the tension between quality attributes is that almost every quality attribute negatively affects performance.

- Eg: Portability.

- The main technique for achieving portable software is to isolate system dependencies, which introduces overhead into the system's execution, typically as process or procedure boundaries, and this hurts performance.

Three Classes of Quality Attributes

1. Qualities of the system.
   - Focus on availability, modifiability, performance, security, testability, and usability.
2. Business qualities
   - Such as time to market that are affected by the architecture.
3. Qualities, such as conceptual integrity, that are about the architecture itself although they indirectly affect other qualities, such as modifiability

4. SYSTEM QUALITY ATTRIBUTES

- System quality attributes have been of interest to the software community at least since the 1970s.
- There are a variety of published taxonomies and definitions, and many of them have their own research and practitioner communities.
- From an architect's perspective, there are three problems with previous discussions of system quality attributes:
- The definitions provided for an attribute are not operational.
- It is meaningless to say that a system will be modifiable.

- Every system is modifiable with respect to one set of changes and not modifiable with respect to another.

- The other attributes are similar.

- A focus of discussion is often on which quality a particular aspect belongs to.

- Is a system failure an aspect of availability, an aspect of security, or an aspect of usability?

- All three attribute communities would argue ownership of a system failure.

- Each attribute community has developed its own vocabulary.

- The performance community has "events" arriving at a system
- The security community has "attacks" arriving at a system
- The availability community has "failures" of a system
- The usability community has "user input."
- All of these may actually refer to the same occurrence, but are described using different terms.

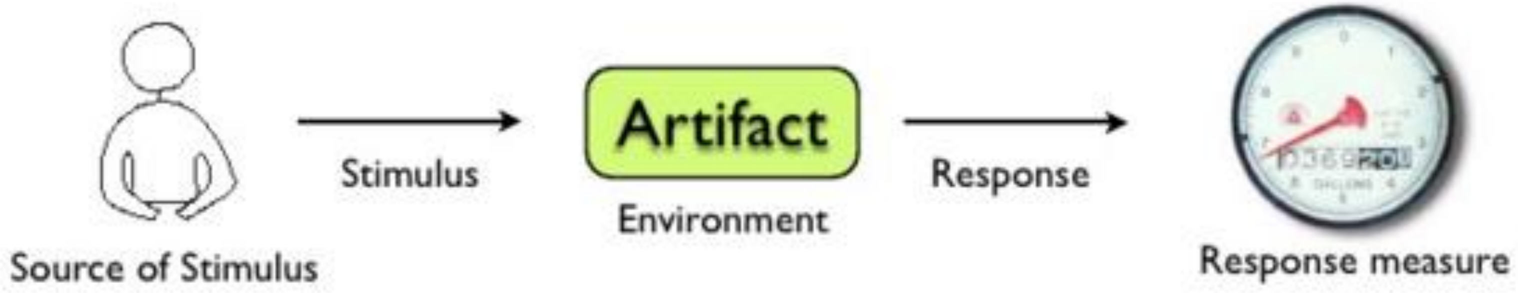
- A solution to the first two of these problems is to use *quality attribute*
- *Scenarios* as a means of characterizing(describing) quality attributes.
- A solution to the third problem is to provide a brief discussion of each attribute—concentrating on its underlying concern —to illustrate the concepts that are fundamental to that attribute community.

4.1 QUALITY ATTRIBUTE SCENARIOS

- A quality attribute scenario is a quality-attribute-specific requirement.

It consists of six parts.

- **Source of stimulus.** This is some entity (a human, a computer system, or any other actuator) that generated the stimulus. (WHO?)
- **Stimulus(Event).** The stimulus is a condition that needs to be considered when it arrives at a system. (WHAT?)
- **Environment.** The stimulus occurs within certain conditions. The system may be in an overload condition or may be running (WHEN?)
  - when the stimulus occurs, or some other condition may be true.
- **Artifact.** Some artifact is stimulated. This may be the whole system or some pieces of it. (WHERE?)
- **Response (Reaction).** The response is the activity undertaken after the arrival of the stimulus. (WHICH?)
- **Response measure.** When the response occurs, it should be measurable in some fashion so that the requirement can be tested.(HOW?)



Source of Stimulus	Developer
Stimulus	Wishes to change the UI
Artifact	Code
Environment	At design time
Response	Modification is made with no side effect
Response measure	In 3 hours

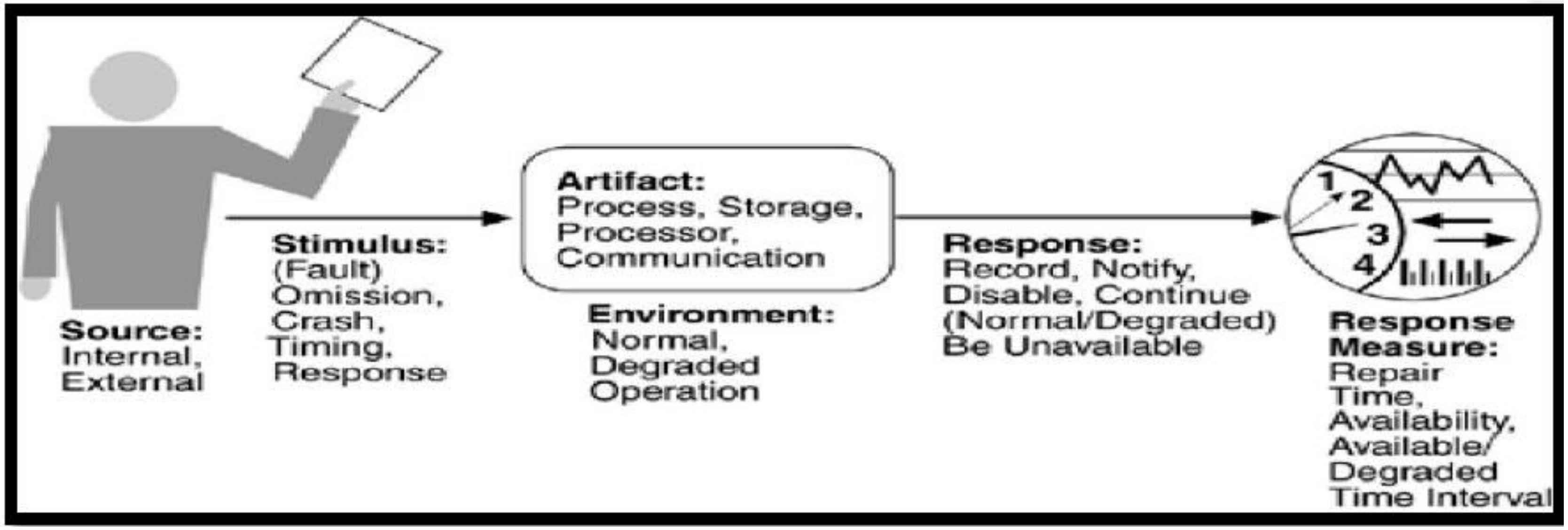
5. AVAILABILITY SCENARIO

- **Availability** is concerned with system failure and its associated consequences
- Failures are usually a result of system errors that are derived from faults in the system.
- It is typically defined as

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

- **Source of stimulus**.
  - Differentiate between internal and external indications of faults or failure since the desired system response may be different.
  - In the example, the unexpected message arrives from outside the system.

- ***Stimulus.*** A fault of one of the following classes occurs.
  - *Omission*-A component fails to respond to an input.
  - *Crash*-The component repeatedly suffers omission faults.
  - *Timing*-A component responds but the response is early or late.
  - *Response*- A component responds with an incorrect value.



- ***Artifact.*** This specifies the resource that is required to be highly available, such as a processor, communication channel, process, or storage.

- ***Environment***. The state of the system when the fault or failure occurs may also affect the desired system response.

- For example, if the system has already seen some faults and is operating in other than normal mode, it may be desirable to shut it down totally.

- However, if this is the first fault observed, some degradation of response time or function may be preferred. In our example, the system is operating normally.

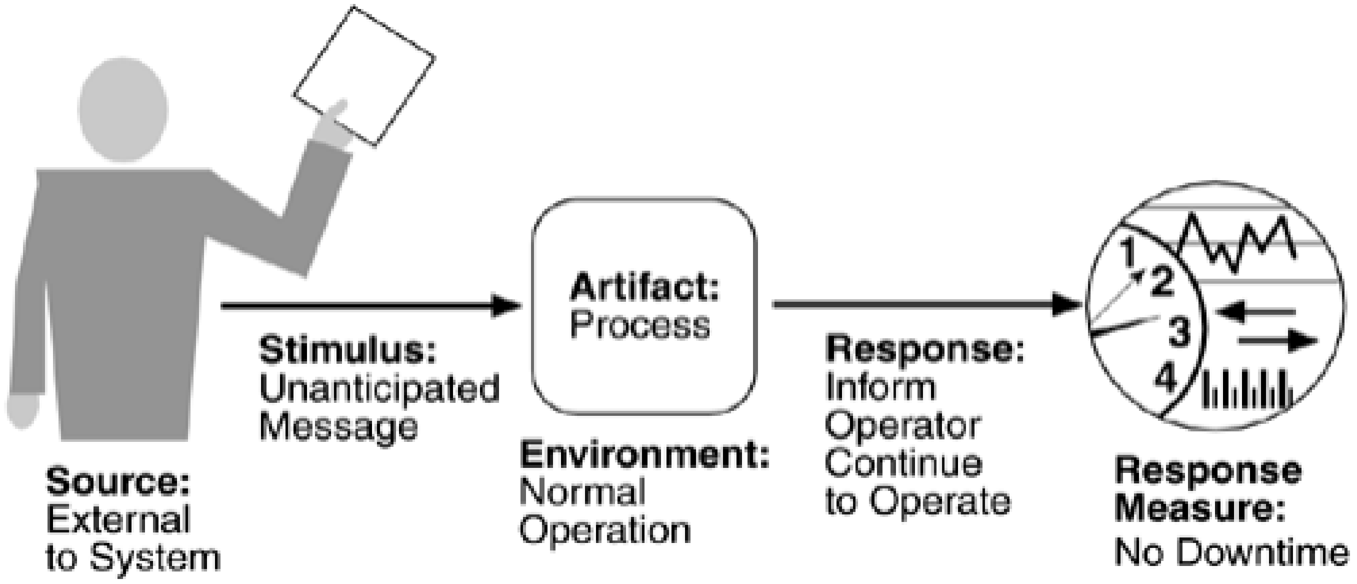
- ***Response.*** There are a number of possible reactions to a system failure. These include logging(classifying) the failure, notifying selected users or other systems, switching to a degraded mode with less capacity or less function, shutting down external systems, or becoming unavailable during repair.
- In the example, the system should notify the operator of the unexpected message and continue to operate normally.

- ***Response measure.*** The response measure can specify an availability percentage, or it can specify a time to repair, times during which the system must be available, or the duration for which the system must be available

Scenario Portion	Possible Values
Source	Internal to system or external to system
Stimulus	Crash, omission, timing, no response, incorrect response
Artifact	System’s processors, communication channels, persistent storage
Environment	Normal operation; degraded (failsafe) mode
Response	Log the failure, notify users/operators, disable source of failure, continue (normal/degraded)
RespMeasure	Time interval available, availability%, repair time, unavailability time interval

Availability sample scenario

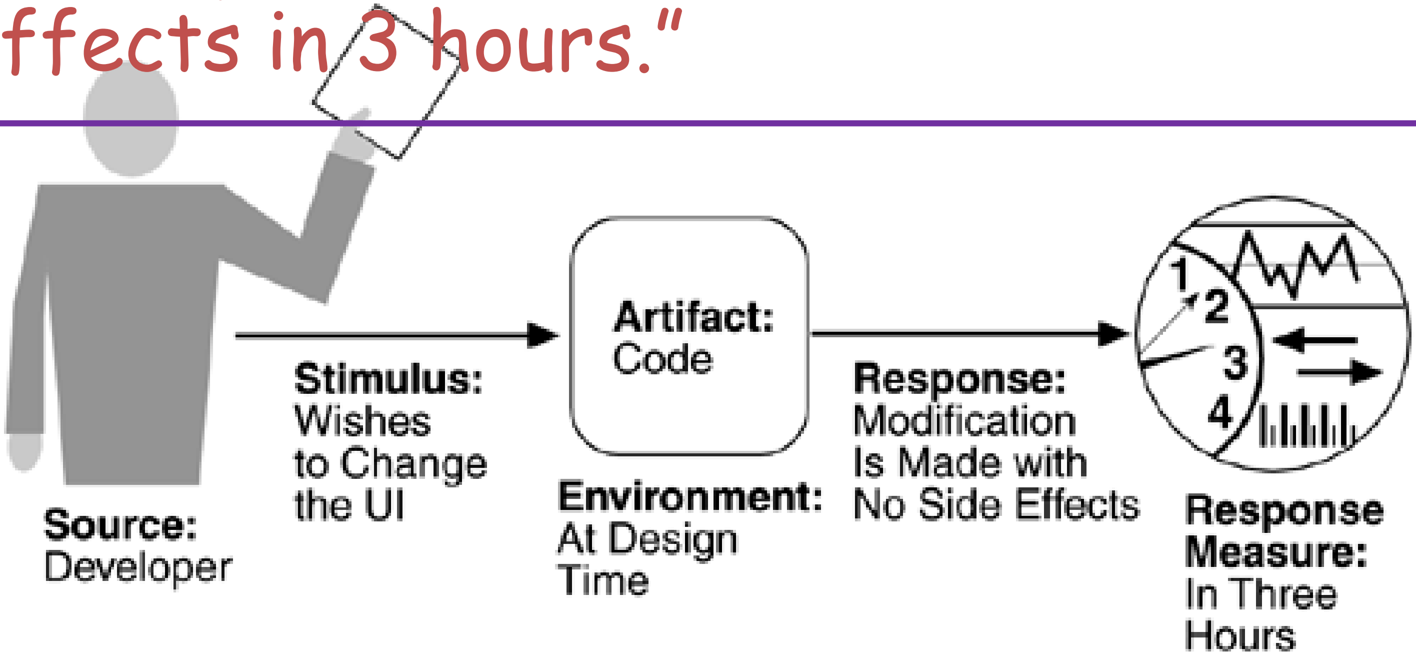
"An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and continues to operate with no downtime." Figure below shows the pieces of this derived scenario.



6.MODIFIABILITY GENERAL SCENARIO

- Modifiability is about the cost of change.
- It brings up two concerns.
  - *What can change (the artifact)?*
  - *When is the change made and who makes it (the environment)?*

"A developer wants to change the UI code at design time; the modification is made without side effects in 3 hours."



***Source of stimulus*.**

- This portion specifies who makes the changes—the developer, a system administrator, or an end user.
- There must be machinery(technology) in place to allow the system administrator or end user to modify a system.
- In the above Figure the **modification is to be made by the developer.**

Stimulus

- This portion specifies the changes to be made.
- A change can be the addition of a function, the modification of an existing function, or the deletion of a function.
- It can also be made to the qualities of the system—making it more responsive, increasing its availability, and so forth.

- The capacity of the system may also change.
- Increasing the number of simultaneous users is a frequent requirement.
- In our example, the stimulus is a request to make a **modification, which can be to the function, quality, or capacity.**

Artifact

- This portion specifies what is to be changed
- The functionality of a system, its platform, its user interface, its environment, or another system with which it interoperates.
- Modification is to the **user interface.**

Environment

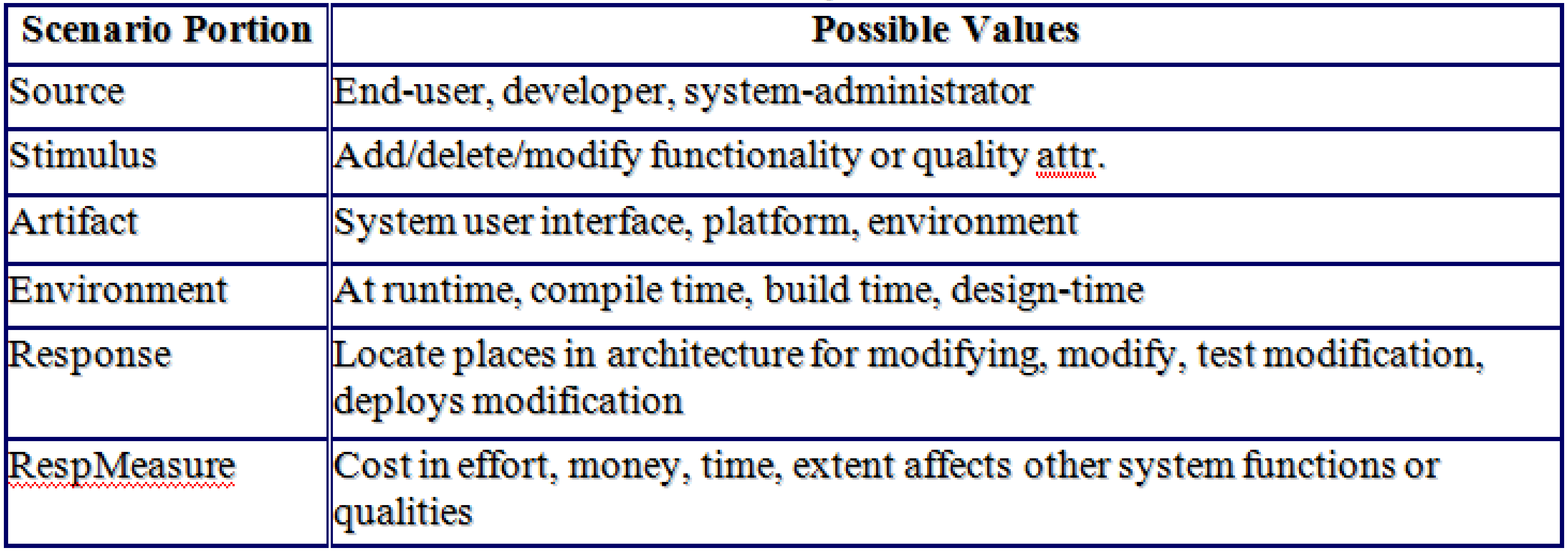
- This portion specifies when the change can be made—design time, compile time, build time, initiation time, or runtime.
- In this example **modification is to occur at design time.**

Response

- Whoever makes the change must understand how to make it, test it and deploy it.
- In this example, **the modification is made with no side effects.**

***Response measure*.**

- All of the possible responses take time and cost money
- Time and cost are the most desirable measures.
- In this example, **the time to perform the modification should be less than three hours.**



Scenario Portion	Possible Values
Source	End-user, developer, system-administrator
Stimulus	Add/delete/modify functionality or quality attr.
Artifact	System user interface, platform, environment
Environment	At runtime, compile time, build time, design-time
Response	Locate places in architecture for modifying, modify, test modification, deploys modification
RespMeasure	Cost in effort, money, time, extent affects other system functions or qualities

7.PERFORMANCE

- Performance is about timing.

- Events (interrupts, messages, requests from users, or the passage of time) occur, and the system must respond to them.

- There are a variety of characterizations of event arrival and the response

- Performance is concerned with how long it takes the system to respond when an event occurs.

- One of the things that make performance complicated is the number of event sources and arrival patterns.

- Events can arrive from user requests, from other systems, or from within the system.

- A Web-based financial services system gets events from its users (possibly numbering in the tens or hundreds of thousands).

- An engine control system gets its requests from the passage of time and must control both the firing of the ignition when a cylinder is in the correct position and the mixture of the fuel to maximize power and minimize pollution.

- For the Web-based financial system, the response might be the number of transactions that can be processed in a minute.

- For the engine control system, the response might be the variation in the firing time.

- An arrival pattern for events may be characterized as either periodic or stochastic.

- For example, a periodic event may arrive every 10 milliseconds.

- Periodic event arrival is most often seen in real-time systems.

- Stochastic arrival means that events arrive according to some probabilistic distribution.

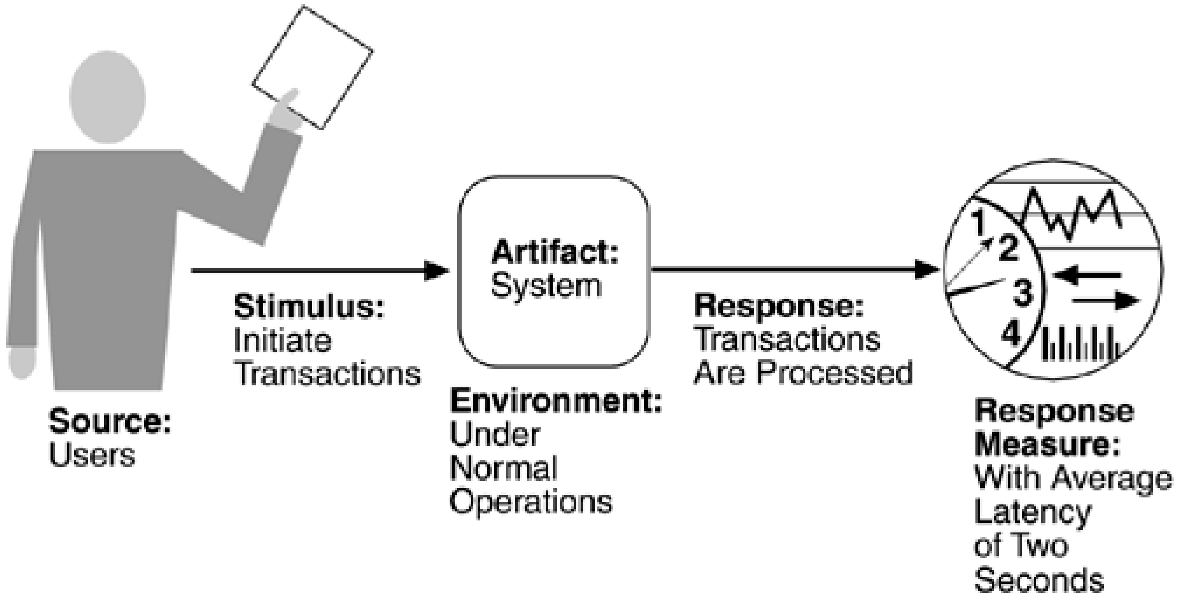
- Events can also arrive sporadically(irregular), that is, according to a pattern not capturable by either periodic or stochastic characterizations.

Performance General Scenarios

- "Users initiate 1,000 transactions per minute stochastically under normal operations, and these transactions are processed with an average latency of two seconds."

Source of stimulus

- The stimuli arrive either from external (possibly multiple) or internal sources. In this example, the source of the stimulus is a collection of users.


Source: Users
Stimulus: Initiate Transactions
Artifact: System
Environment: Under Normal Operations
Response: Transactions Are Processed
1
2
3
4
Response Measure: With Average Latency of Two Seconds

Stimulus.

- The stimuli are the event arrivals.
- The arrival pattern can be characterized as periodic, stochastic, or sporadic.
- In this example, the stimulus is the stochastic initiation of 1,000 transactions per minute.

Artifact

- The artifact is always the system's services, as it is in the example.

Environment.

- The system can be in various operational modes, such as normal, emergency, or overload.
- In this example, the system is in normal mode.

Response.

- The system must process the arriving events.
- This may cause a change in the system environment (e.g., from normal to overload mode).
- In this example, the transactions are processed.

Response measure.

- The response measures are the time it takes to process the arriving events
  - latency or a deadline by which the event must be processed
  - variation in this time (jitter)
  - Number of events that can be processed within a particular time interval (throughput)
  - characterization of the events that cannot be processed (miss rate, data loss)
- In this example, the transactions should be processed with an average latency of two seconds.

Portion of Scenario	Possible Values
Source	One of a number of independent sources, possibly from within system
Stimulus	Periodic events arrive; sporadic events arrive; stochastic events arrive
Artifact	System
Environment	Normal mode; overload mode
Response	Processes stimuli; changes level of service
Response Measure	Latency, deadline, throughput, jitter, miss rate, data loss

8. SECURITY

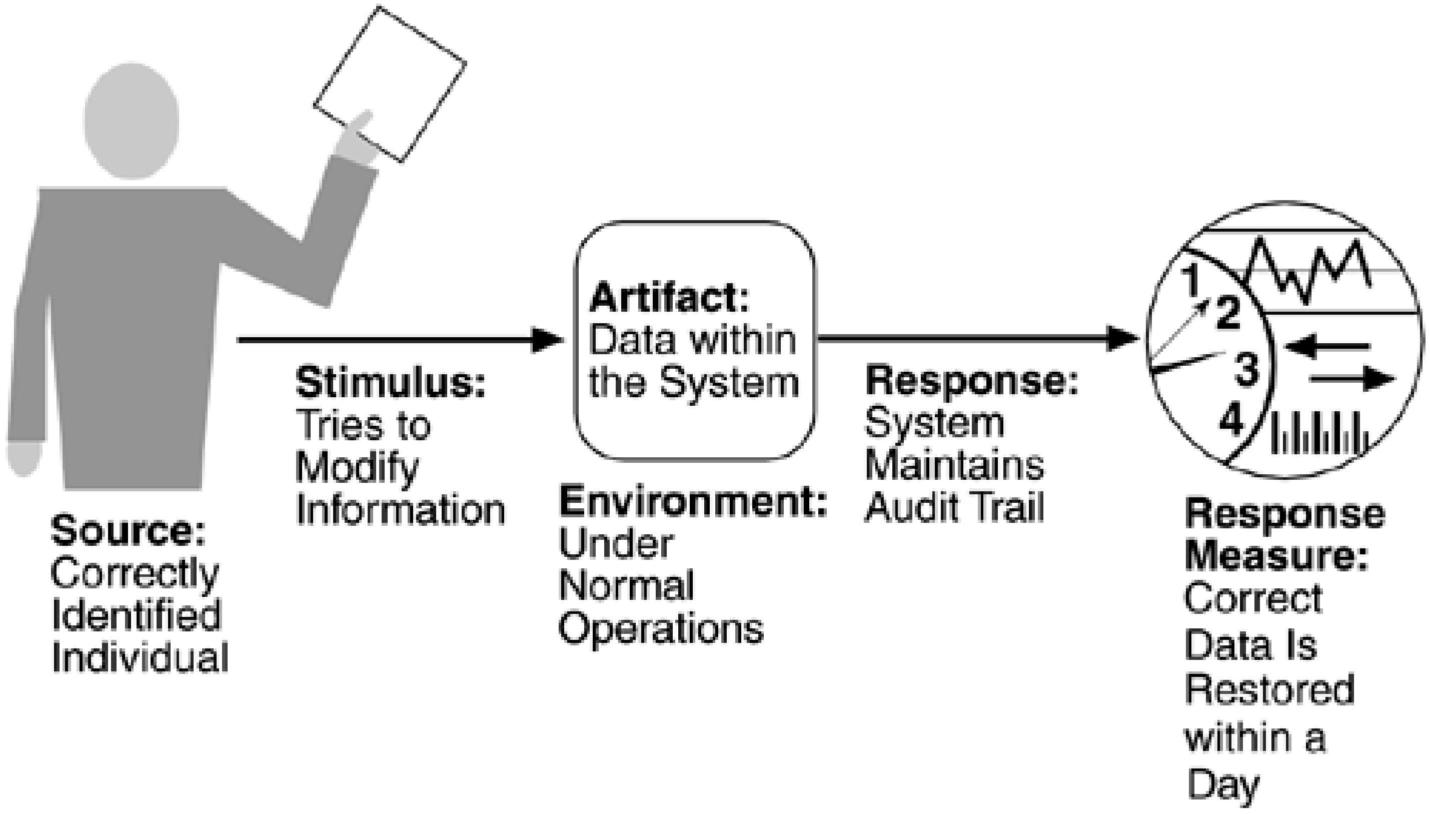
- Security is a measure of the system's ability to resist(oppose) unauthorized usage while still providing its services to legitimate(valid) users.

- An attempt to breach(violate) security is called an attack .

- It may be an unauthorized attempt to access data or services or to modify data, or it may be intended to deny services to legitimate users.

- Attacks may range from theft of money by electronic transfer to modification of sensitive data, from theft of credit card numbers to destruction of files on computer systems, or to denial-of-service attacks carried out by worms or viruses.

Security General Scenario


Source:
Correctly Identified Individual
Stimulus:
Tries to Modify Information
Artifact:
Data within the System
Environment:
Under Normal Operations
Response:
System Maintains Audit Trail
1
2
3
4
Response Measure:
Correct Data Is Restored within a Day

- Security can be characterized as a system providing non repudiation, confidentiality, integrity, assurance, availability, and auditing.

- **Nonrepudiation** is the property that a transaction (access to or modification of data or services) cannot be denied by any of the parties to it. This means you cannot deny that you ordered that item over the Internet if, in fact, you did.

- **Confidentiality** is the property that data or services are protected from unauthorized access. This means that a hacker cannot access your income tax returns on a government computer.

- **Integrity** is the property that data or services are being delivered as intended. This means that your grade has not been changed since your instructor assigned it.

- **Assurance** is the property that the parties to a transaction are who they declare to be. This means that, when a customer sends a credit card number to an Internet merchant, the merchant is who the customer thinks they are.

- **Availability** is the property that the system will be available for legitimate use. This means that a denial-of-service attack won‘t prevent your ordering *this book.*

- **Auditing** is the property that the system tracks activities within it at levels sufficient to reconstruct them. This means that, if you transfer money out of one account to another account, in Switzerland, the system will maintain a record of that transfer.

- **Source of stimulus:** The source of the attack may be either a human or another system.

- **Stimulus:** The stimulus is an attack to break security. This as an unauthorized person or system trying to display information, change and/or delete information, access services of the system, or reduce availability of system services. The stimulus is an attempt to modify data

- **Artifact:** The target of the attack can be either the services of the system or the data within it.

- **Environment:** The attack can come when the system is either online or offline.

- **Response:** Using services without authorization.

- **Response measure:** Measures of a system’s response include the difficulty of mounting(increasing) various attacks and the difficulty of recovering from and surviving attacks.

Security General Scenario.

Scenario Portion	Possible Values
Source	User/system who is legitimate/imposter/unknown with full/limited access
Stimulus	Attempt to display/modify data; access services
Artifact	System services, data
Environment	Normal operation; degraded (failsafe) mode
Response	Authenticate user; hide identity of user; grant/block access; encrypt data; detect excessive demand…
RespMeasure	Time /effort/resources to circumvent security measures with probability of success

9.TESTABILITY GENERAL SCENARIO

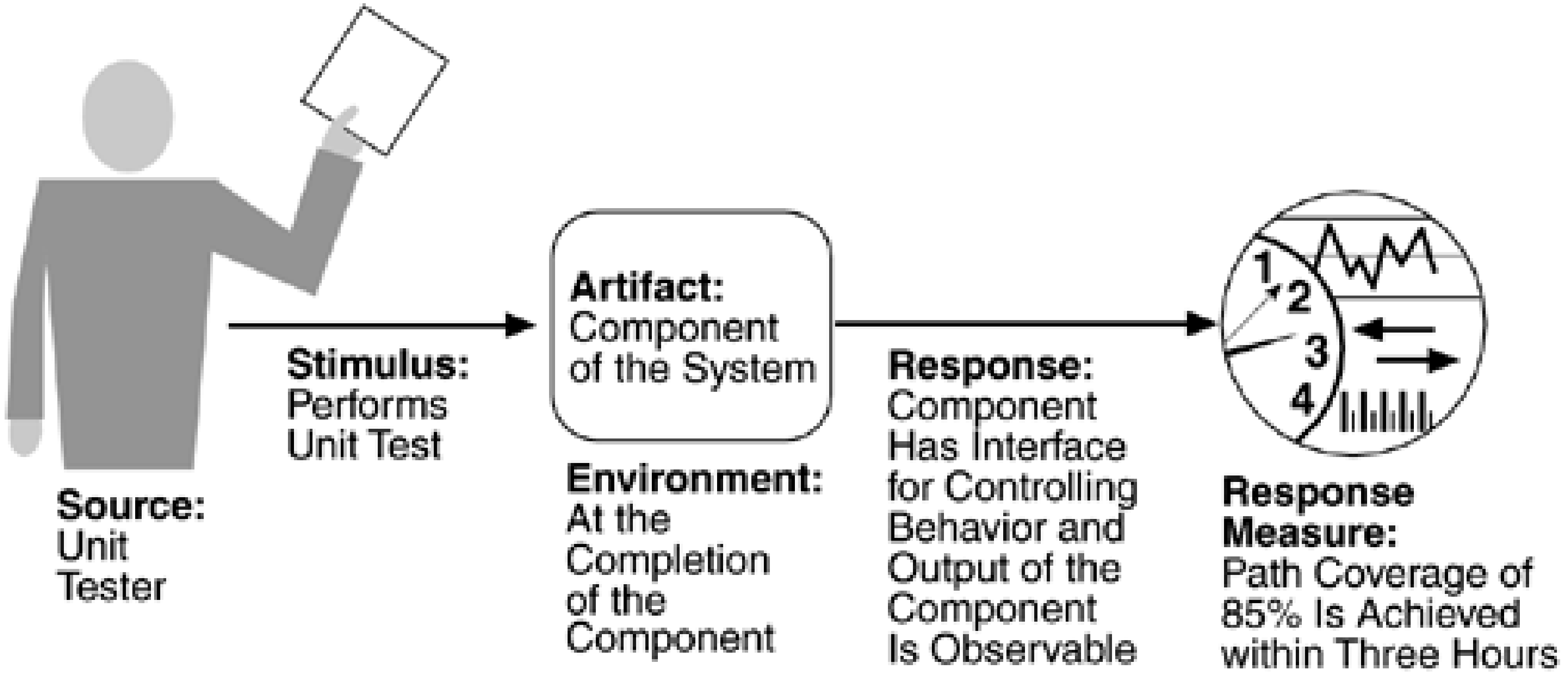
- This example is about concerning the performance of a unit test
- A unit tester performs a unit test on a completed system component that provides an interface for controlling its behavior and observing its output
- 85% path coverage is achieved within three hours.
- Software testability refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

- Testing is done by various developers, testers, verifiers, or users and is the last step of various parts of the software life cycle.

- Portions of the code, the design, or the complete system may be tested.

- The response measures for testability deal with
  - how effective the tests are in discovering faults
  - how long it takes to perform the tests to some desired level of coverage.

Sample testability scenario


Source: Unit Tester
Stimulus: Performs Unit Test
Artifact: Component of the System
Environment: At the Completion of the Component
Response: Component Has Interface for Controlling Behavior and Output of the Component Is Observable
1
2
3
4
Response Measure: Path Coverage of 85% Is Achieved within Three Hours

Source of stimulus.

- The testing is performed by unit testers, integration testers, system testers, or the client.
- A test of the design may be performed by other developers or by an external group.
- In our example, the testing is performed by a tester.

Stimulus.

- The stimulus for the testing is that a milestone(target) in the development process is met.
- This might be the completion of an analysis or design increment, the completion of a coding increment such as a class, the completed integration of a subsystem, or the completion of the whole system.
- In our example, the testing is triggered by the completion of a unit of code.

Artifact.

- A design, a piece of code, or the whole system is the artifact being tested.
- In our example, a unit of code is to be tested.

Environment.

- The test can happen at design time, at development time, at compile time, or at deployment time.
- In this the test occurs during development.

Response.

- Since testability is related to observability and controllability, the desired response is that the system can be controlled to perform the desired tests and that the response to each test can be observed.
- In our example, the unit can be controlled and its responses captured.

Response measure.

- Response measures are the percentage of statements that have been executed in some test, the length of the longest test chain (a measure of the difficulty of performing the tests), and estimates of the probability of finding additional faults.
- In this the measurement is percentage coverage of executable statements.

Testability General Scenario.

Scenario Portion	Possible Values
Source	Unit developer, increment integrator, system verifier, client acceptance tester, system user
Stimulus	Analysis, architecture, design, class, subsystem integration, system delivered
Artifact	Piece of design, piece of code, complete system
Environment	At design time, at development time, at compile time, at deployment time
Response	Provide access to state data values, observes results, compares
RespMeasure	% coverage; prob. of failure; time to perform tests; length of time to prepare test environment

10.USABILITY

- **Usability** is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.
- **It can be broken down into the following areas:**
- Learning system features. what can the system do to make the task of learning easier?
- Using a system efficiently. What can the system do to make the user more efficient in its operation?
- Minimizing the impact of errors. What can the system do so that a user error has minimal impact?
- Adapting the system to user needs. How can the user (or the system itself) adapt to make the user's task easier?
- Increasing confidence and satisfaction. What does the system do to give the user confidence that the correct action is being
- taken?

USABILITY GENERAL SCENARIOS

- A user, wanting to minimize the impact of an error, wishes to cancel a system operation at runtime; cancellation takes place in less than one second.

Source of stimulus.

– The end user is always the source of the stimulus.

Stimulus.

– The stimulus is that the end user wishes to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or feel comfortable with the system.

– In our example, the user wishes to cancel an operation, which is an example of minimizing the impact of errors.

Artifact.

- The artifact is always the system.

Environment.

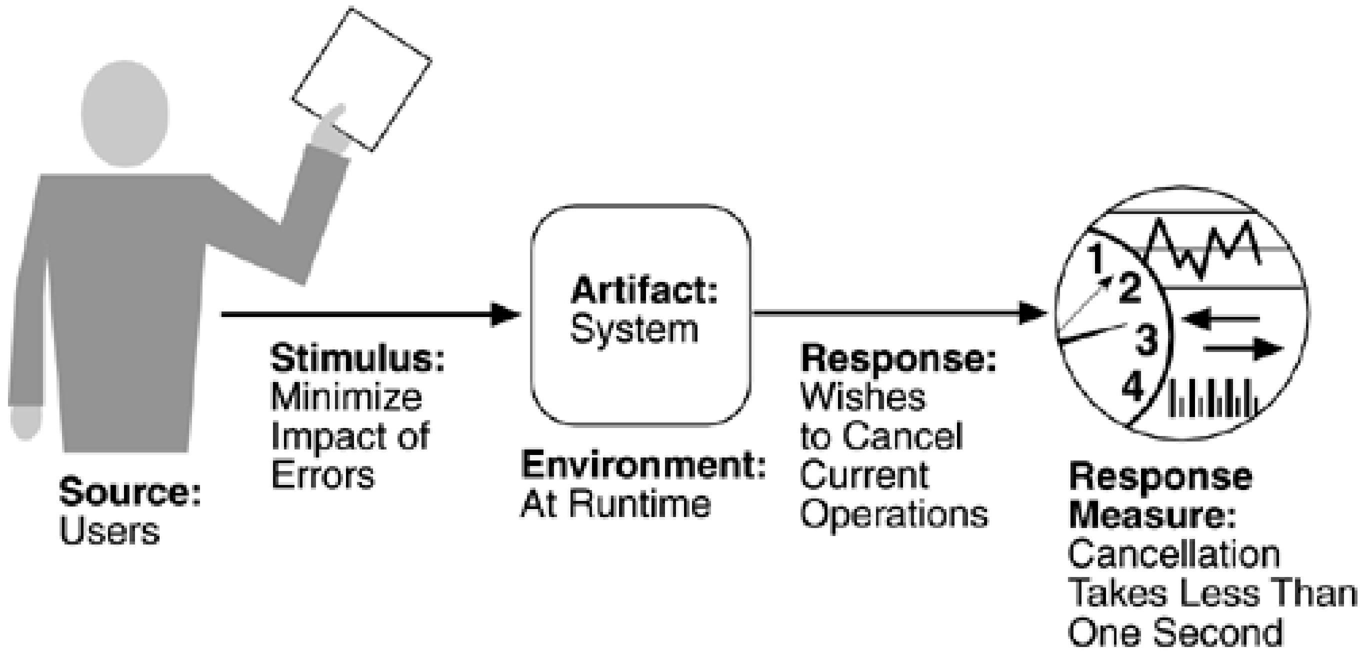
- The user actions with which usability is concerned always occur at runtime or at system configuration time.
- In this the cancellation occurs at runtime.

Response.

- The system should either provide the user with the features needed or predict the user's needs.
- In our example, the cancellation occurs as the user wishes and the system is restored to its prior state.

Response measure.

- The response is measured by task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations, or amount of time/data lost when an error occurs.
- In this the cancellation should occur in less than one second.

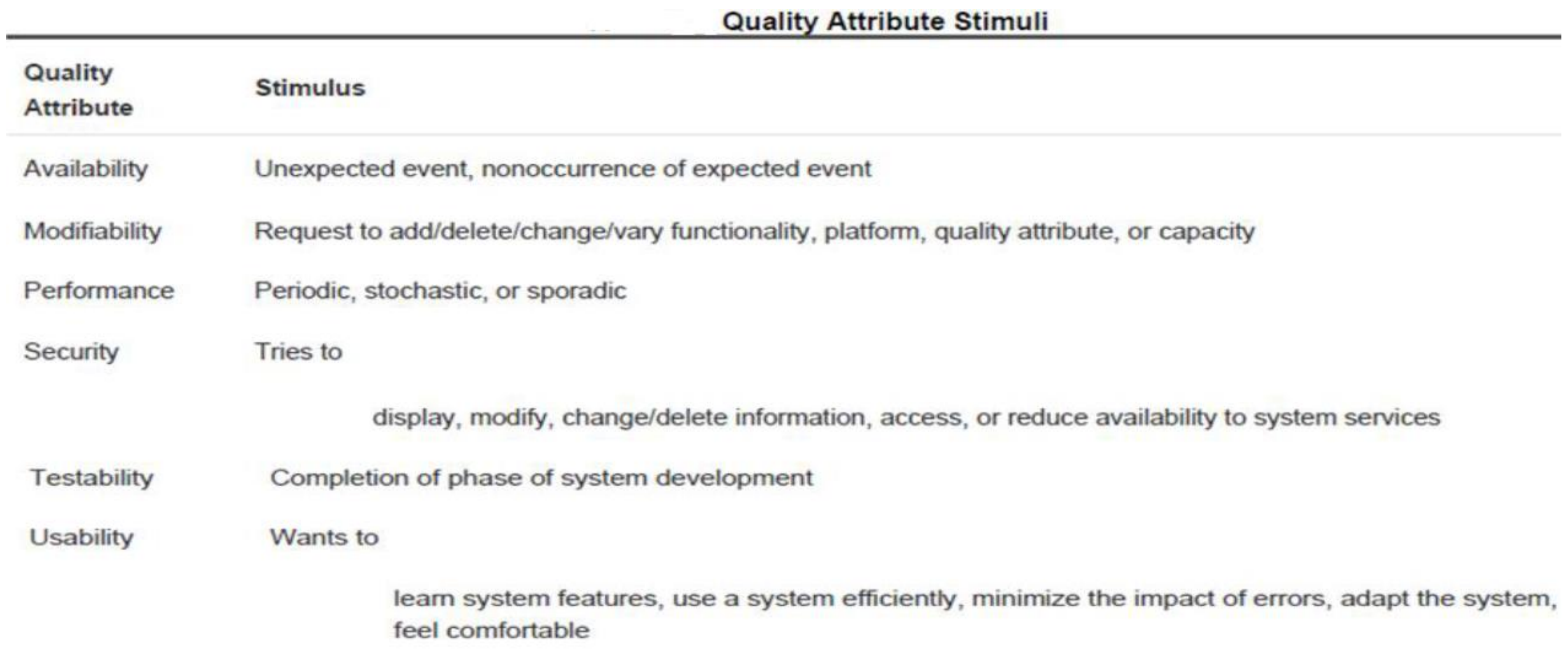

Source: Users
Stimulus: Minimize Impact of Errors
Artifact: System
Environment: At Runtime
Response: Wishes to Cancel Current Operations
Response Measure: Cancellation Takes Less Than One Second
1
2
3
4

Usability General Scenario.

Scenario Portion	Possible Values
Source	End user
Stimulus	Wants to: learn system, use system, recover from errors, adapt system, feel comfortable
Artifact	System
Environment	At runtime, or configure time, install-time
Response	To learn system, To use system efficiently, To recover from errors
RespMeasure	Task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, amount of time/data lost

11. COMMUNICATING CONCEPTS USING GENERAL SCENARIOS

- One of the uses of general scenarios is to enable stakeholders to communicate.
- Table below gives the stimuli possible for each of the attributes and shows a number of different concepts.
- Some stimuli occur during runtime and others occur before.
- The problem for the architect is to understand which of these stimuli represent the same occurrence, which are aggregates of other stimuli, and which are independent.



Quality Attribute Stimuli

Quality Attribute	Stimulus
Availability	Unexpected event, nonoccurrence of expected event
Modifiability	Request to add/delete/change/vary functionality, platform, quality attribute, or capacity
Performance	Periodic, stochastic, or sporadic
Security	Tries to display, modify, change/delete information, access, or reduce availability to system services
Testability	Completion of phase of system development
Usability	Wants to learn system features, use a system efficiently, minimize the impact of errors, adapt the system, feel comfortable

OTHER SYSTEM QUALITY ATTRIBUTES

- SCALABILITY
- PORTABILITY

12. BUSINESS QUALITIES

Time to market.

- If there is competitive pressure or a short window of opportunity for a system or product, development time becomes important.
- This in turn leads to pressure to buy or otherwise re-use existing elements.

Cost and benefit.

- The development effort will naturally have a budget that must not be exceeded.
- Different architectures will yield different development costs.
- An architecture that relies on technology (or expertise with a technology) will be more expensive to realize than one that takes advantage of assets already in-house.
- An architecture that is highly flexible will typically be more costly to build than one that is rigid

Projected lifetime of the system.

- If the system is intended to have a long lifetime, modifiability, scalability, and portability become important.
- On the other hand, a modifiable, extensible product is more likely to survive longer in the marketplace, extending its lifetime.

Targeted market.

- For general-purpose software, the platforms on which a system runs as well as its feature set will determine the size of the potential market.
- Portability and functionality are key to market share.
- Other qualities, such as performance, reliability, and usability also play a role.

Rollout schedule.

- If a product is to be introduced as base functionality with many features released later, the flexibility and customizability of the architecture are important.
- System must be constructed with ease of expansion and contraction in mind.

Integration with legacy systems.

- If the new system has to integrate with existing systems, care must be taken to define appropriate integration mechanisms.

13. ARCHITECTURE QUALITIES

- Conceptual integrity is the underlying theme or vision that unifies the design of the system at all levels. The architecture should do similar things in similar ways.

- **Correctness and completeness** are essential for the architecture to allow for all of the system's requirements and runtime resource constraints to be met.

- **Buildability** allows the system to be completed by the available team in a timely manner and to be open to certain changes as development progresses.

14. TACTICS

- The achievement of these qualities relies on fundamental design decisions.
- A tactic is a design decision that influences the control of a quality attribute response
- Each tactic is a design option for the architect.
- For example, one of the tactics introduces redundancy to increase the availability of a system.
- This is one option the architect has to increase availability, but not the only one.
- Usually achieving high availability through redundancy implies a concomitant(associating) need for synchronization (to ensure that the redundant copy can be used if the original fails).

TWO IMMEDIATE RAMIFICATIONS OF THIS EXAMPLE.

Tactics can refine other tactics.

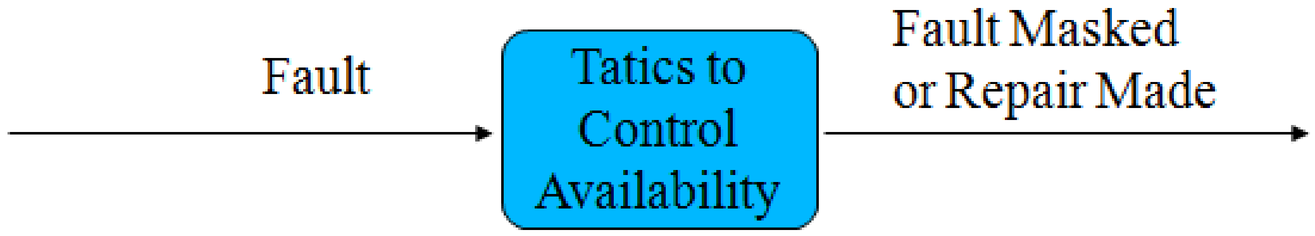
- We identified redundancy as a tactic.
- As such, it can be refined into redundancy of data (in a database system) or redundancy of computation (in an embedded control system).
- Both types are also tactics.

Patterns package tactics.

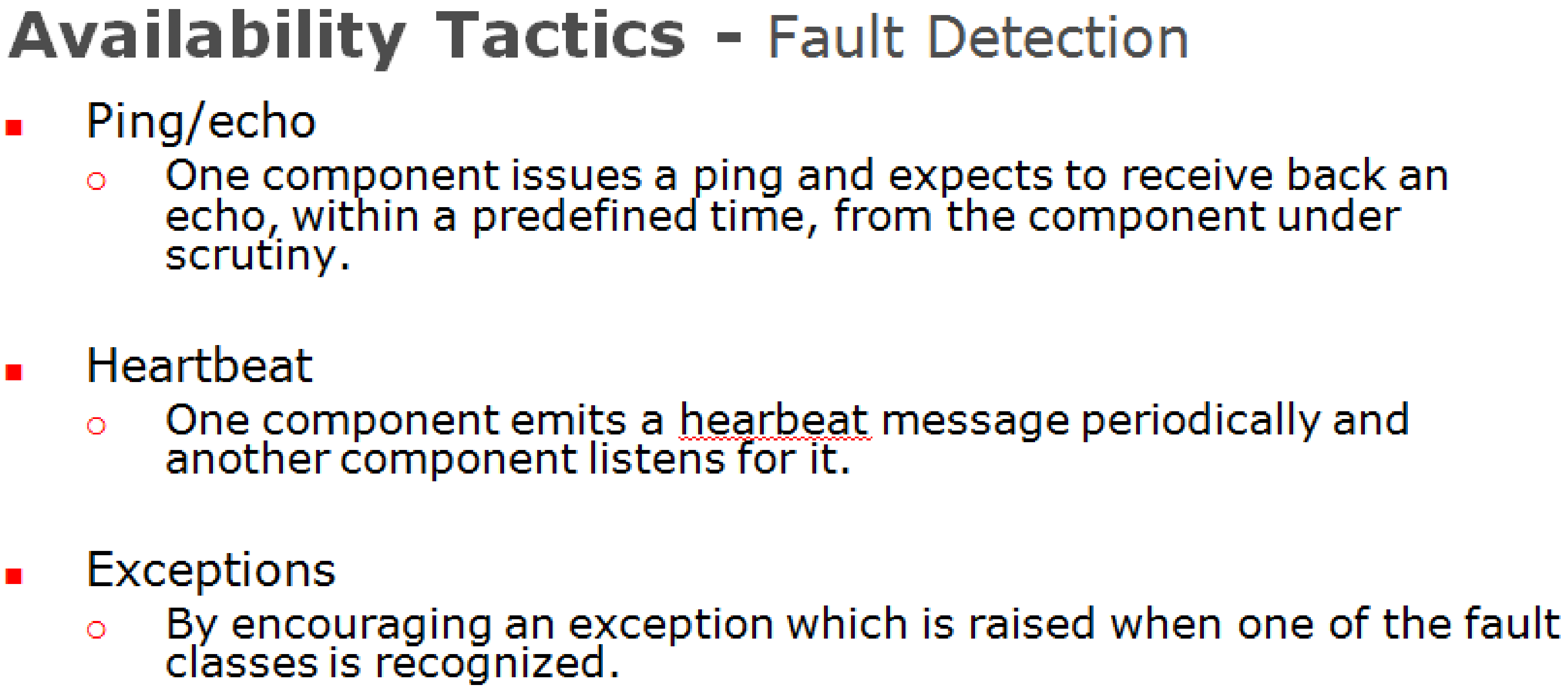
- A pattern that supports availability will likely use both a redundancy tactic and a synchronization tactic.
- It will also likely use more concrete versions of these tactics.

14.1 AVAILABILITY TACTICS

- A failure occurs when the system no longer delivers a service that is consistent(regular) with it specification
- Use tactics to keep the fault;
  - Fault detection;
  - Fault recovery;
  - Fault prevention;



Fault Masking is a Error condition hiding another error Condition.



Availability Tactics - Fault Detection

- Ping/echo
  - One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny.
- Heartbeat
  - One component emits a hearbeat message periodically and another component listens for it.
- Exceptions
  - By encouraging an exception which is raised when one of the fault classes is recognized.

Query (another computer on a network) to determine whether there is a connection to it.

Availability Tactics - Fault Recovery

- Voting
  - Process **running** on redundant processors each take equivalent input and compute a simple output value that is sent to a voter. If the voter detects deviant behaviour from a single processor, it fails it.

- Active Redundancy
  - All redundant component respond to events in parallel.
  - Often used in client/serve configuration.

- Passive Redundancy
  - One component responds to events and informs the other components of state updates they must make. When a fault occurs, the system must first ensure that the backup state is sufficiently fresh before resuming services.

Active redundancy in active components requires reconfiguration when failure occurs.

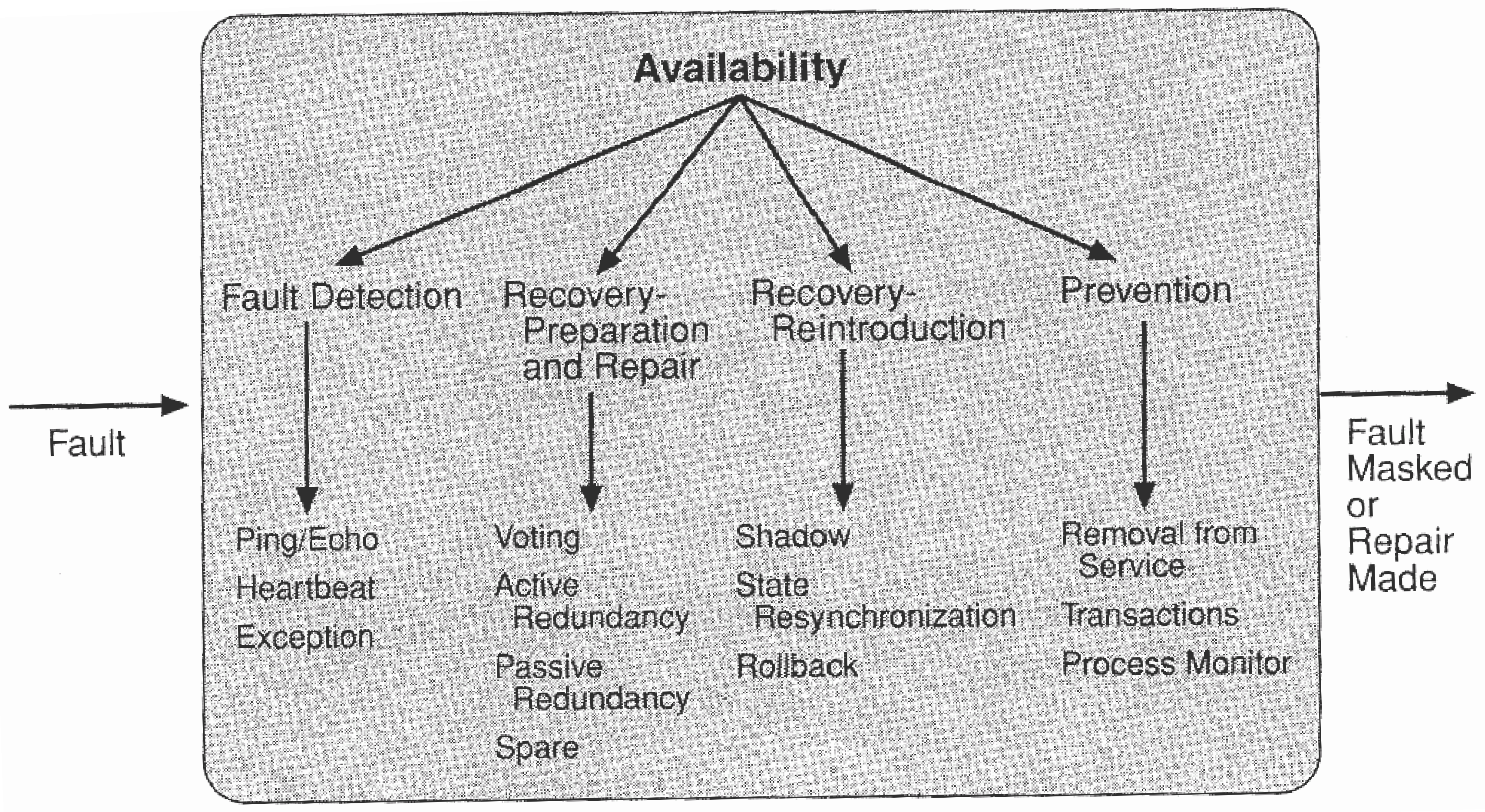
Passive redundancy A device that is doing something and a device that is idle and ready to take over the task of the other device in case it fails.

Availability Tactics - Fault Recovery

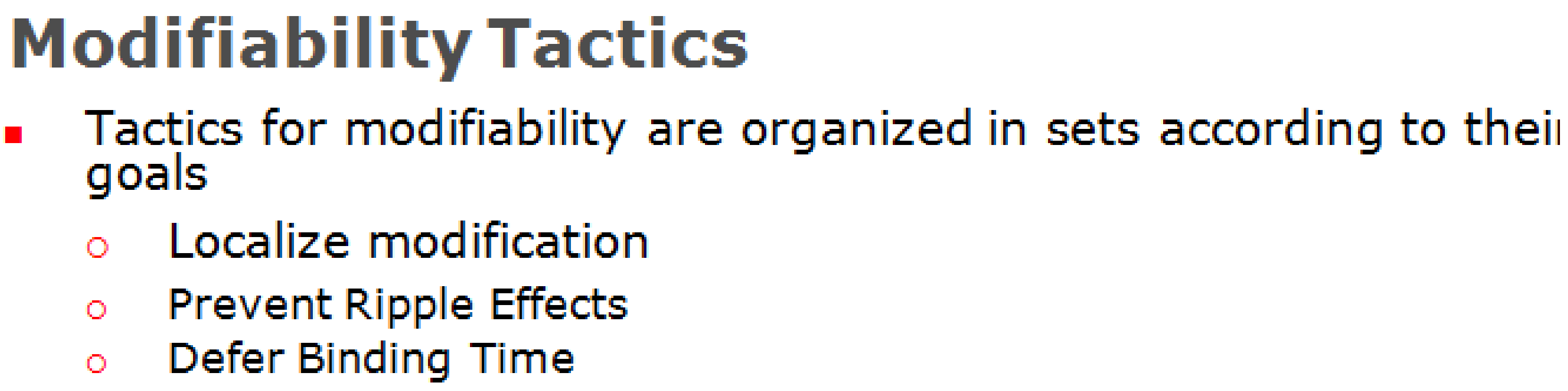
- Spare
  - A standby spare computing platform is configured to replace many different failed components. It state initialized when a failure occurs.
- Shadow Operation
  - A previously failed component may be run in “shadow mode” for a short time to make sure that it imitate the behaviour of the working components before restoring it to service.
- State resynchronization
  - The passive and active redundancy tactics require the component being restored to have its state upgraded before its return to service.
- Checkpoint/Rollback
  - It is a recording of a consistent state created either periodically or in response to specific events.

Availability Tactics - Fault Prevention

- Removel from service
    - Remove a component of the system from operation to undergo some activities to prevent anticipated failures.
- Transactions
    - It is used to prevent any data from being affected if one step in a process fails and also to prevent collisions among simultaneous threads accessing the same data.
- Process monitor
    - Once a fault has been detected, a monitoring process can delete the nonperforming process and create a new instance of it.

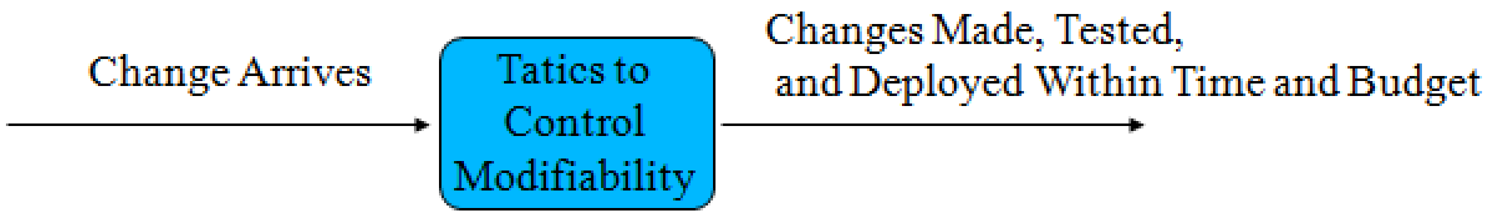

Availability
Fault Detection
Recovery-Preparation and Repair
Recovery-Reintroduction
Prevention
Fault
Fault Masked or Repair Made
Ping/Echo
Heartbeat
Exception
Voting
Active Redundancy
Passive Redundancy
Spare
Shadow
State Resynchronization
Rollback
Removal from Service
Transactions
Process Monitor

14.2 Modifiability Tactics



Modifiability Tactics

- Tactics for modifiability are organized in sets according to thei goals
  - Localize modification
  - Prevent Ripple Effects
  - Defer Binding Time


Change Arrives
Tatics to Control Modifiability
Changes Made, Tested, and Deployed Within Time and Budget

Localize Modifications: reduce the number of modules that are affected by a change.

When a modification is made by the developer, there is usually a testing and distribution process that determines the time lag between the making of the change and the availability of that change to the end user.

LOCALIZE MODIFICATION

- **Maintain Semantic Coherence(logic):** this refers to the relationships among responsibilities in a module. The goal is to ensure that all of these responsibilities work together without excessive dependence on other modules.

- **Anticipate(predict) expected changes:** considering the set of predicted changes provides a way to evaluate a particular assignment of responsibilities.

- **Generalize(simplify) the module:** Making a module more general allows it to compute a broader range of functions based on input.

- **Limit possible options:** Modifications, especially within a product line may be far ranging and hence affect many modules. Restricting the possible options will reduce the effect of these modifications.

Prevent Ripple Effects

- **Prevent Ripple Effects:** limit modifications to the localized modules. A ripple effect from a modification is the necessity of making changes to modules not directly affected by it.

- **Hide information:** this is the decomposition of the responsibilities for an entity into smaller pieces and choosing which information to make private and which to make public.

- **Maintain Existing interface:** If module B depends on the name and signature of interface of module A, maintaining this interface and its syntax allows module B to remain unchanged.

- **Restrict communication paths:** restrict the modules with which a given module shares data. That is, reduce the number of modules that consume data produced by the given module and the number of modules that produce data consumed by it.

- **Use and intermediary:** if module B has any type dependency on module A other than semantic, it is possible to insert an intermediary between module A and module B that manages activities associated with the dependency.

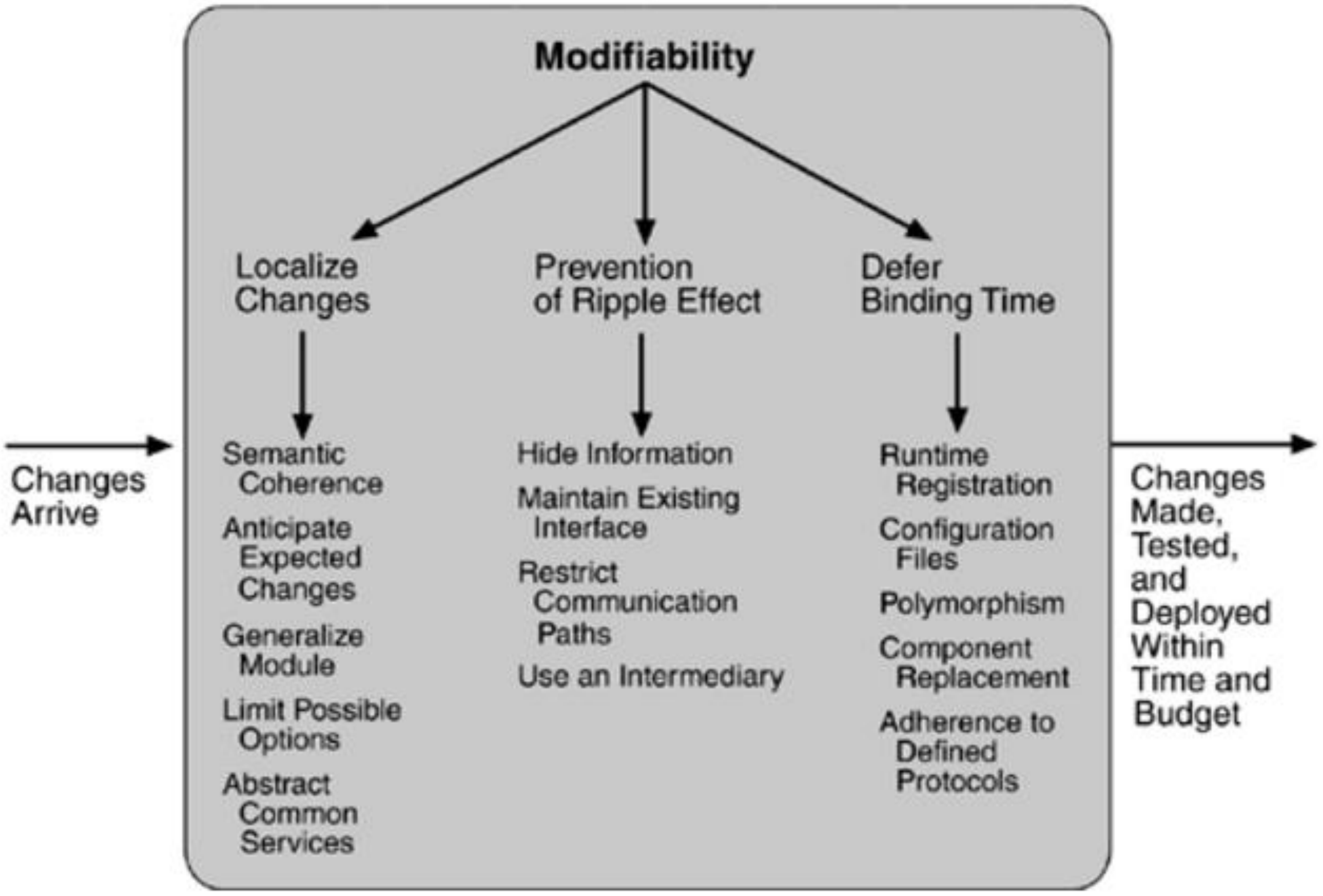
Defer Binding Time

- **Defer Binding Time:** When a modification is made by the developer, there is usually a testing and distribution process that determines the time lag between the making of the change and the availability of that change to the end user.

- Binding at runtime means that the system has been prepared for that binding and all of the testing and distribution steps have been completed.

- Deferring(rescheduling) binding time also supports allowing the end user or system administrator to make setting or provide input that affects behavior.
  - **Runtime registration:** Publish/subscribe for example.
  - **Configuration files:** are intended to set parameters at startup.
  - **Polymorphism:** allows late binding of method calls
  - **Component replacement:** allows load time binding.
  - **Adherence to defined protocols:** allows runtime binding of independent processes.

SUMMARY

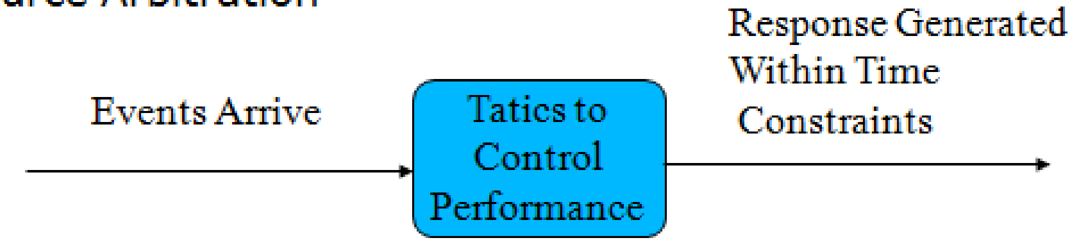


Modifiability
Localize Changes
Prevention of Ripple Effect
Defer Binding Time
Semantic Coherence
Anticipate Expected Changes
Generalize Module
Limit Possible Options
Abstract Common Services
Hide Information
Maintain Existing Interface
Restrict Communication Paths
Use an Intermediary
Runtime Registration
Configuration Files
Polymorphism
Component Replacement
Adherence to Defined Protocols
Changes Arrive
Changes Made, Tested, and Deployed Within Time and Budget

14.3 Performance Tactics

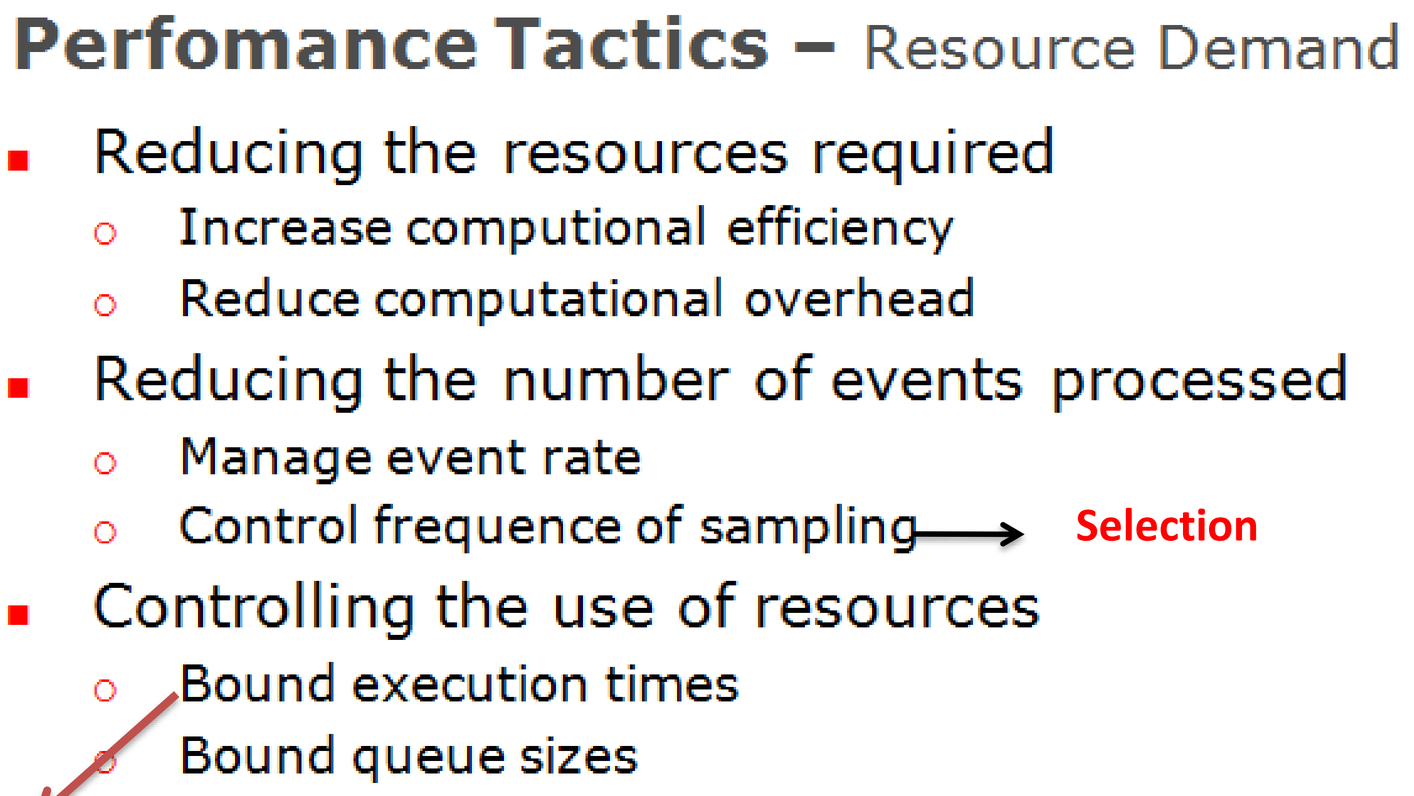
Perfomance Tactics

- The goal is generate a response to an event arriving at the system within some time constraint
- Contributors to response time:
  - Resource consumption and Blocked time
- Tactics:
  - Resource Demand
  - Resource Management
  - Resource Arbitration


Events Arrive
Tatics to Control Performance
Response Generated Within Time Constraints

Latency is the time between the arrival of an event and the generation of a response to it.

Resource Demand



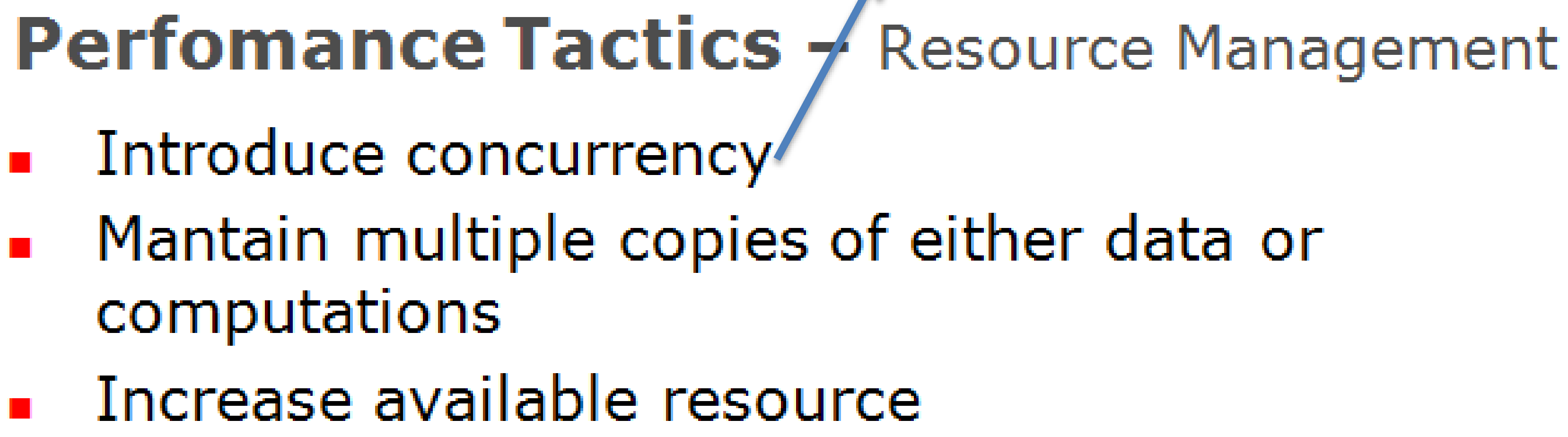
Perfomance Tactics – Resource Demand

- Reducing the resources required
  - Increase computional efficiency
  - Reduce computational overhead
- Reducing the number of events processed
  - Manage event rate
  - Control frequence of sampling → Selection
- Controlling the use of resources
  - Bound execution times
  - Bound queue sizes

Planning

Resource Management

Systems can do more than one thing at a time.

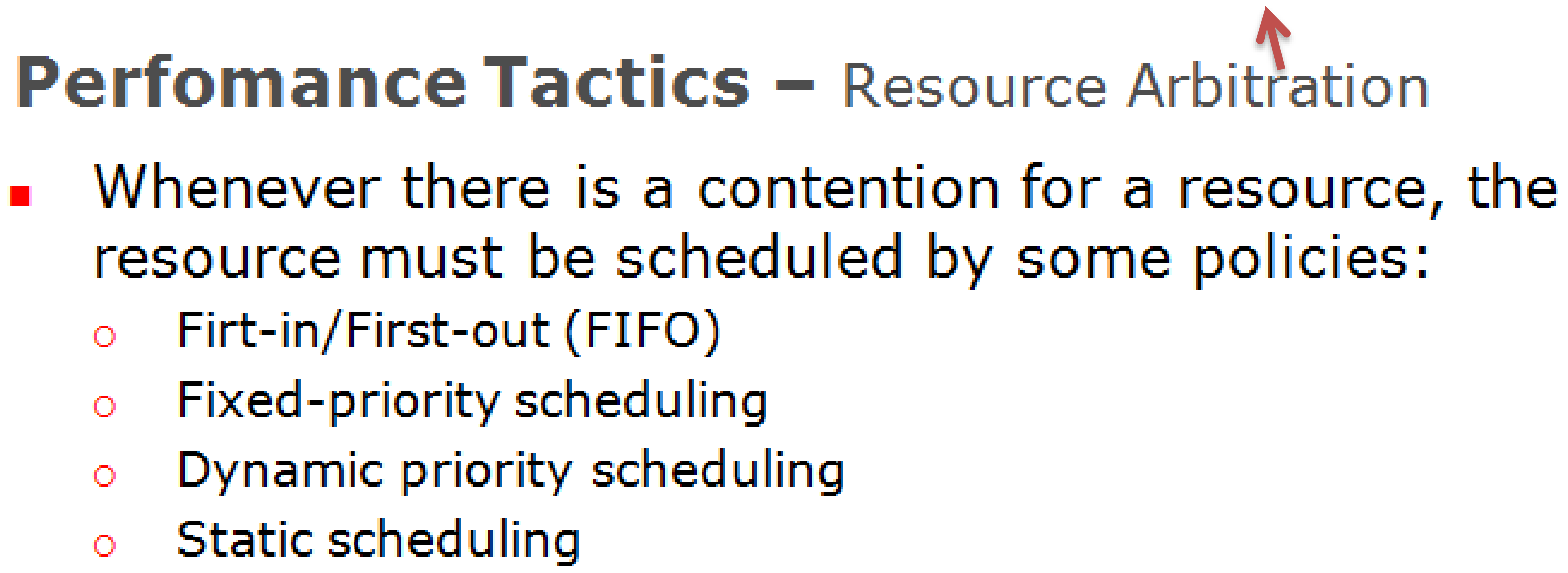


Perfomance Tactics – Resource Management

- Introduce concurrency
- Mantain multiple copies of either data or computations
- Increase available resource

Resource Arbitration

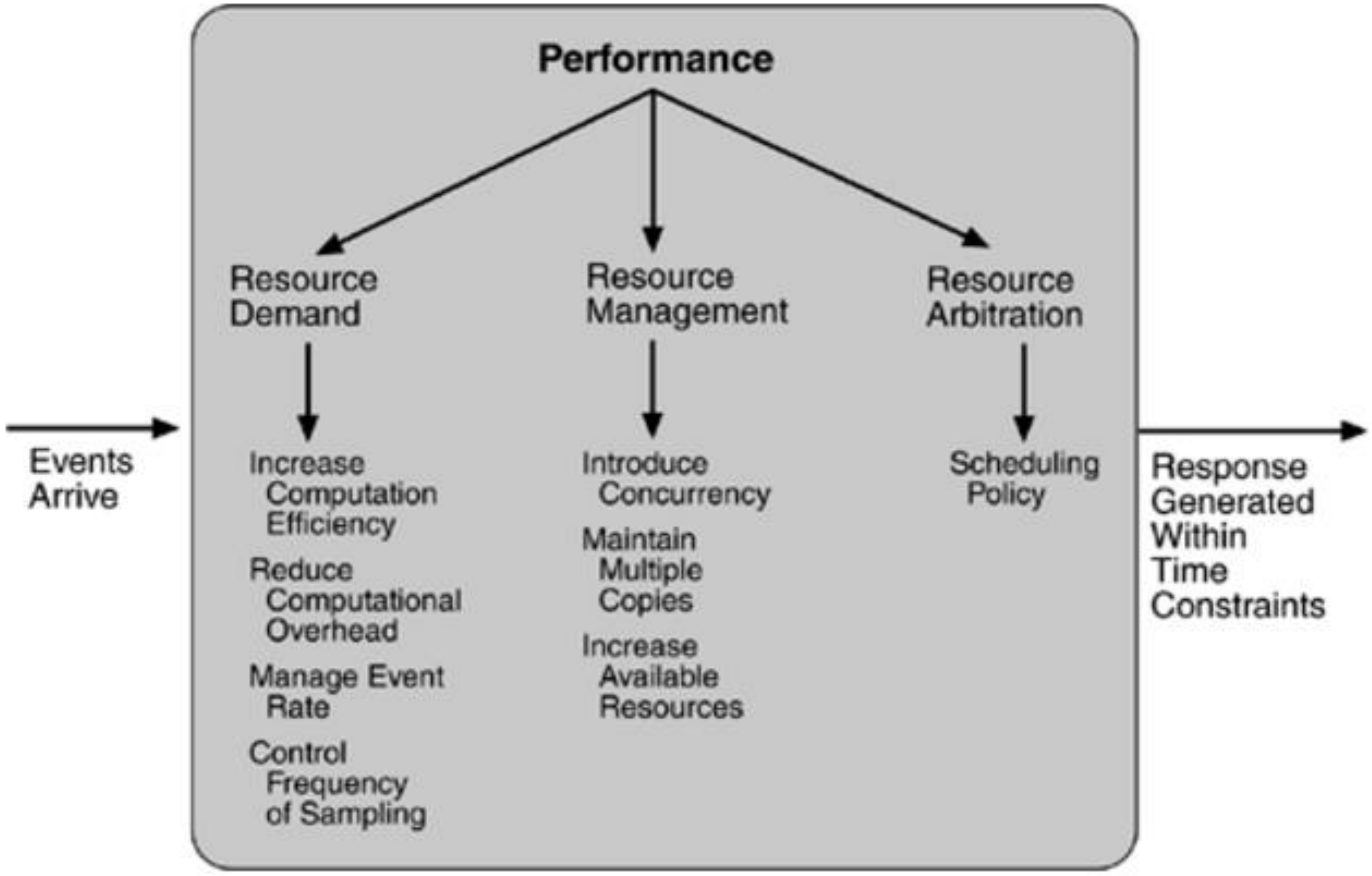
A set of rules for allocating machine resources, such as memory or peripheral devices, to more than one user or program.



Perfomance Tactics – Resource Arbitration

- Whenever there is a contention for a resource, the resource must be scheduled by some policies:
  - Firt-in/First-out (FIFO)
  - Fixed-priority scheduling
  - Dynamic priority scheduling
  - Static scheduling

SUMMARY

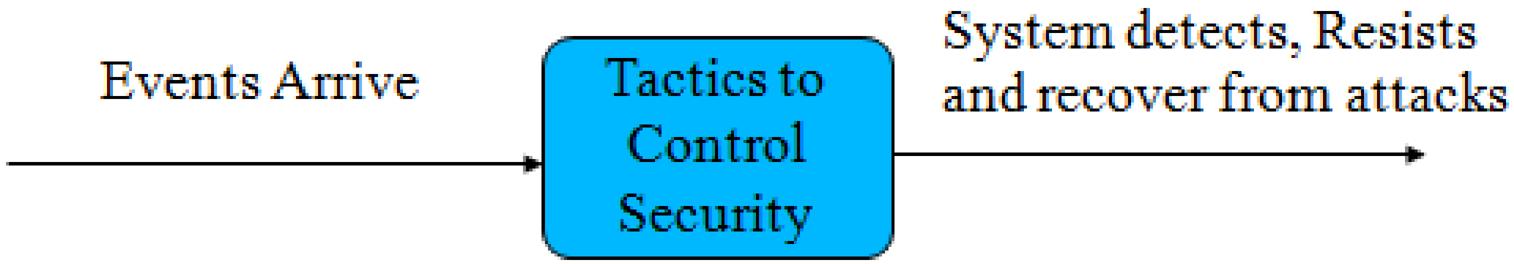


Performance
Resource Demand
Resource Management
Resource Arbitration
Events Arrive
Increase Computation Efficiency
Reduce Computational Overhead
Manage Event Rate
Control Frequency of Sampling
Introduce Concurrency
Maintain Multiple Copies
Increase Available Resources
Scheduling Policy
Response Generated Within Time Constraints

14.4 Security Tactics

Tactics can be divided into

- Resisting attacks
- Detecting attacks
- Recovering attacks


Events Arrive
Tactics to Control Security
System detects, Resists and recover from attacks

Resisting Attacks

- Authenticate users
  - Authentication is ensuring that a user or remote computer is actually who it maintain to be.
  - Passwords, one-time passwords, digital certificates, and biometric identification
- Authorize users
  - An authenticated user has the rights to access and modify either data or services.
  - Classes of users can be defined by user groups, by user roles, or by lists of individuals.

- Maintain data confidentiality
  - Data should be protected from unauthorized access.
  - Confidentiality is usually achieved by applying some form of encryption to data and to communication links.
- Maintain integrity
  - Data should be delivered as intended.
  - It can have redundant information encoded in it, such as checksums or hash results, which can be encrypted either along with or independently from the original data.

- Limit exposure
  - The architect can design the allocation of services to hosts so that limited services are available on each host.
- Limit access – firewall
  - Firewalls restrict access based on message source or destination port.
  - Messages from unknown sources may be a form of an attack.
  - One configuration used in this case is the so-called demilitarized zone (DMZ).
  - A DMZ is used when access must be provided to Internet services but not to a private network.
  - It sits between the Internet and a firewall in front of the internal network.

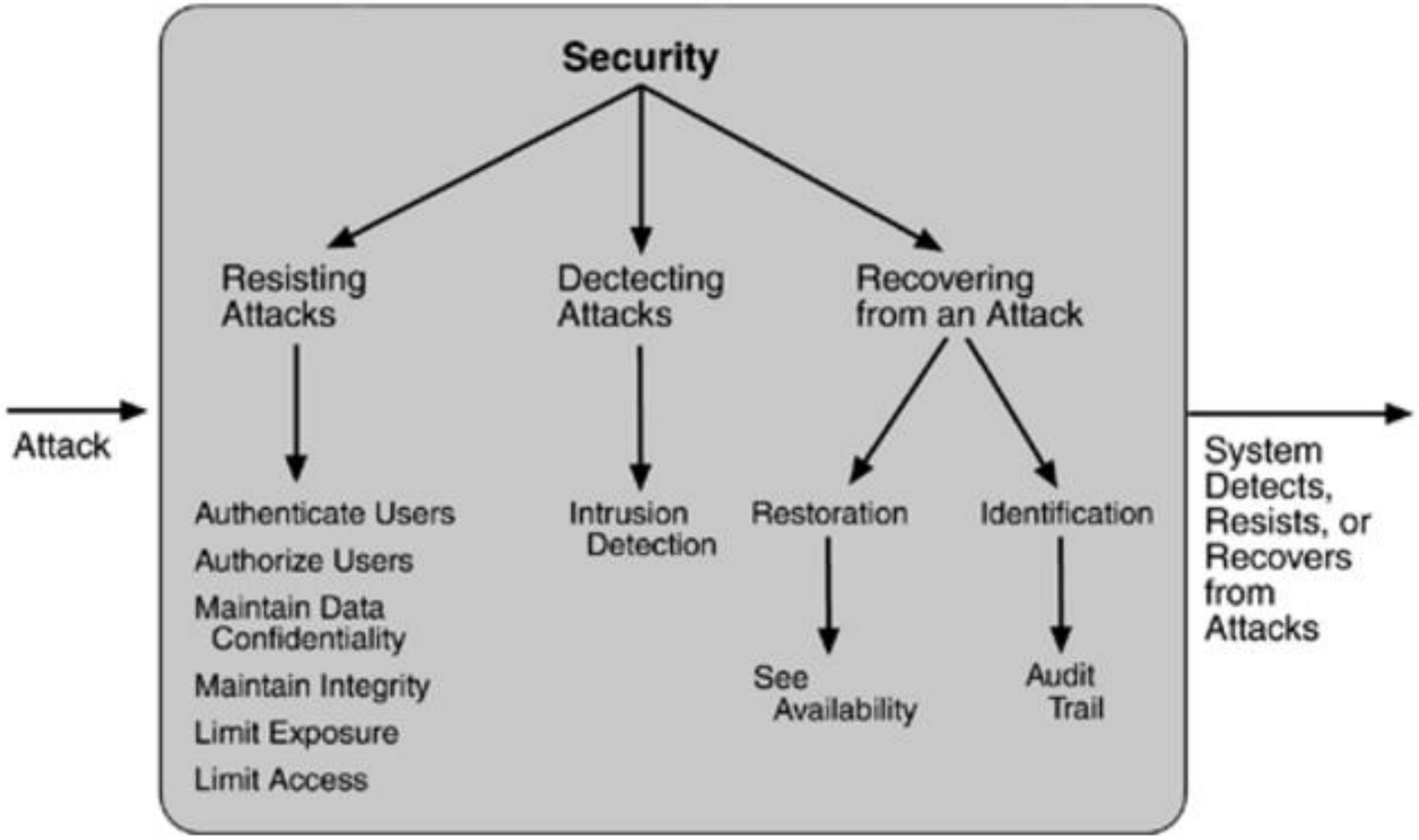
Detecting Attacks

- Use of intrusion detectors
  - e.g. Sensor to detect attacks
- By comparing network traffic patterns to a database
- By comparing historic patterns of known attacks

Recovering from Attacks

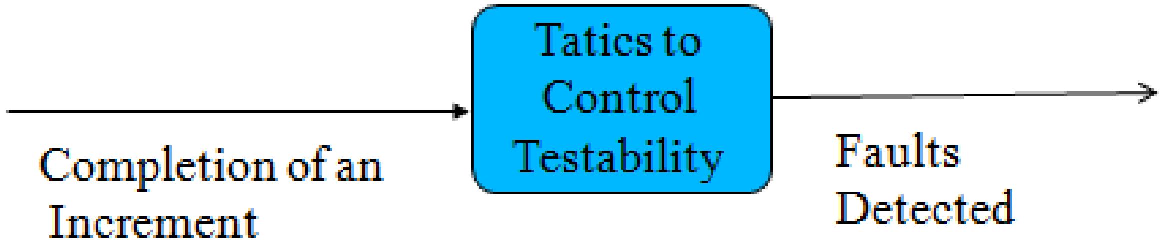
- **Restoring State**
  - Concerned with recovering a consistent state from an inconsistent state.
  - By maintaining the redundant copies of system data
  - Special attention is paid to maintaining redundant copies of system administrative data such as passwords, access control lists, domain name services, and user profile data.
- **Attacker Identification**
  - By maintaining an audit trail or a copy of each transaction applied to the data

SUMMARY


Security
Resisting Attacks
Dectecting Attacks
Recovering from an Attack
Attack
Authenticate Users
Authorize Users
Maintain Data Confidentiality
Maintain Integrity
Limit Exposure
Limit Access
Intrusion Detection
Restoration
Identification
See Availability
Audit Trail
System Detects, Resists, or Recovers from Attacks

14.5 TESTABILITY TACTICS

- To allow for easier testing when an increment of software development is completed
- Testing consumes high percentage of system development cost
- Executing the test procedures requires *some software to provide input to the software being tested and to capture the output*. This is called a test harness.



Tactics:

Input/output

Internal Monitoring

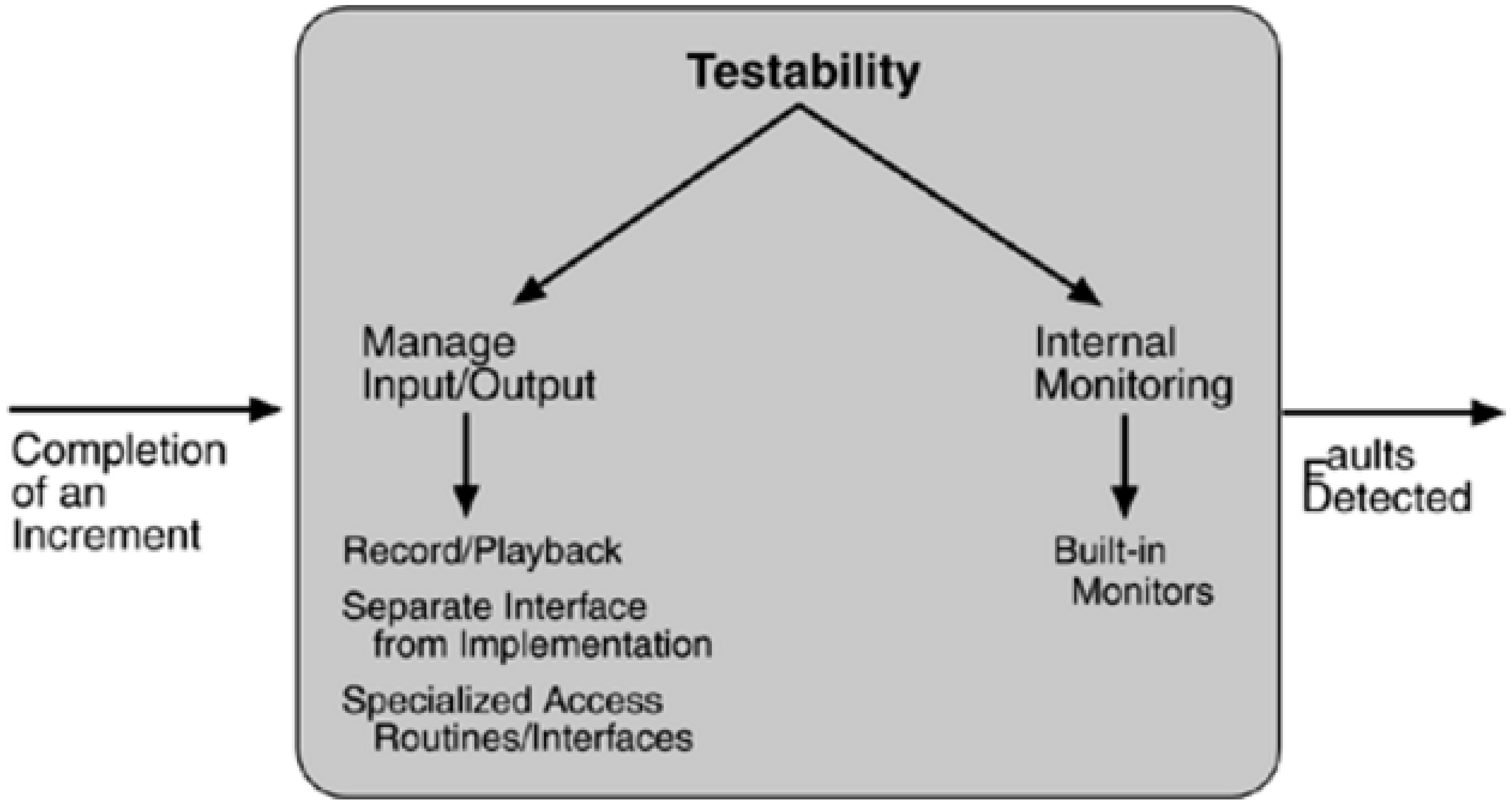
INPUT/OUTPUT

- Record/playback
  - By capturing information crossing a interface and using it as input into test harness.
- Separate interface from implementation
  - Allows substitution of implementations for various testing purposes.
- Specialize access routes/interfaces
  - Having specialized testing interfaces
  - Allows capturing or specification of variable value for a component through a test harness.
  - Having a hierarchy of test interfaces in the architecture means that test cases can be applied at any level in the architecture

INTERNAL MONITORING

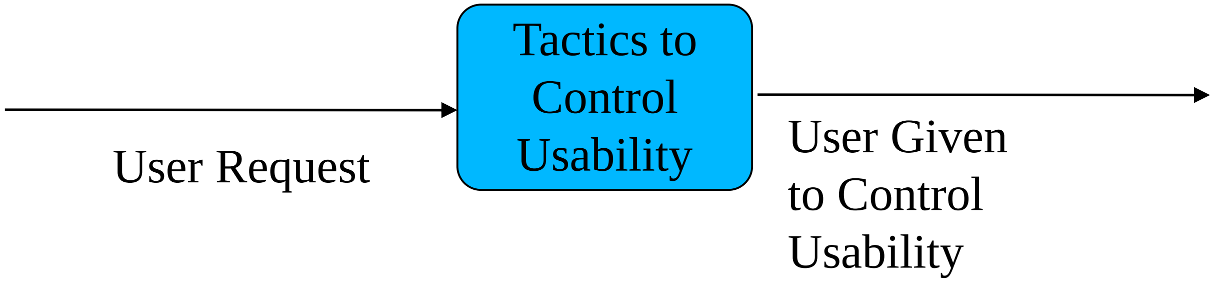
- Built-in Monitors
  - The component can maintain state, performance load, capacity, security and other information accessible through an interface.
  - A common technique is to record events when monitoring states have been activated.

SUMMARY


Testability
Manage Input/Output
Internal Monitoring
Completion of an Increment
Faults Detected
Record/Playback
Separate Interface from Implementation
Specialized Access Routines/Interfaces
Built-in Monitors

14.6 USABILITY TACTICS

- Usability is concerned with how easy it is for the user to accomplish a desired task
- Two types of tactics support usability, each intended for two categories of "users."
- The first category, runtime, includes those that support the user during system execution.
- The second category is based on the iterative nature of user interface design and supports the interface developer at design time.



RUNTIME TACTICS

- Once a system is executing, usability is enhanced by giving the user feedback
- Providing the user with the ability to issue usability-based commands
- **Example:** ***cancel, undo, aggregate, and show multiple views***
- Researchers in human-computer interaction have used the terms "user initiative," "system initiative," and "mixed initiative“
- These terms describe which of the human-computer pair takes the initiative in performing certain actions and how the interaction proceeds.

Example:

- When canceling a command the user issues a cancel "user initiative" and the system responds.
- During the cancel, the system may put up a progress indicator "system initiative".
- Thus, cancel demonstrates "mixed initiative."
- When the system takes the initiative, it must rely on some information-a model-about the user, the task being undertaken by the user, or the system state itself.
- Each model requires various types of input to accomplish its initiative.

Maintain a model of the task.

- In this case, the model maintained is that of the task.
- The task model is used to determine context so the system can have some idea of what the user is attempting and provide various kinds of assistance.

Example

- Knowing that sentences usually start with capital letters would allow an application to correct a lower-case letter in that position.

Maintain a model of the user.

- In this case, the model maintained is of the user.
- It determines the user's knowledge of the system, the user's behavior in terms of expected response time, and other aspects specific to a user or a class of users.

Example

- Maintaining a user model allows the system to pace(speed) scrolling so that pages do not fly past faster than they can be read.

Maintain a model of the system.

- In this case, the model maintained is that of the system.
- It determines the expected system behavior so that appropriate feedback can be given to the user.
- The system model predicts items such as the time needed to complete current activity.

DESIGN-TIME TACTICS

- User interfaces are typically revised frequently during the testing process.

- Usability engineer will give the developers revisions to the current user interface design and the developers will implement them.

- This leads to a tactic that is a refinement(modification) of the modifiability tactic of semantic coherence

- Separate the user interface from the rest of the application.

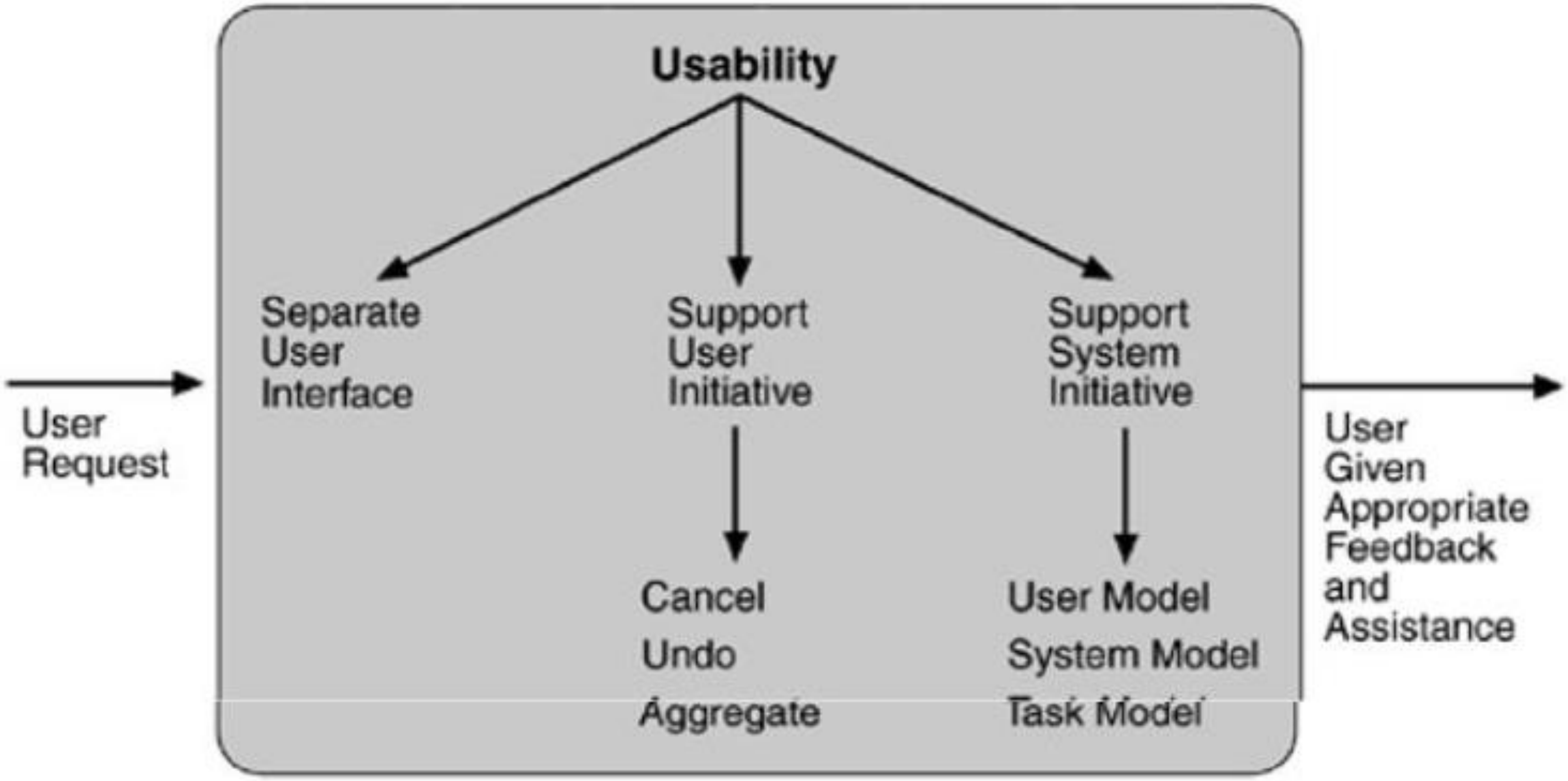
- Localizing expected changes is the rationale for semantic coherence.

- Since the user interface is expected to change frequently both during the development and after deployment

- Maintaining the user interface code separately will localize changes to it.

- The software architecture patterns developed to implement this tactic and to support the modification of the user interface are:
  - Model-View-Controller
  - Presentation-Abstraction-Control
  - Seeheim
  - Arch/Slinky

SUMMARY

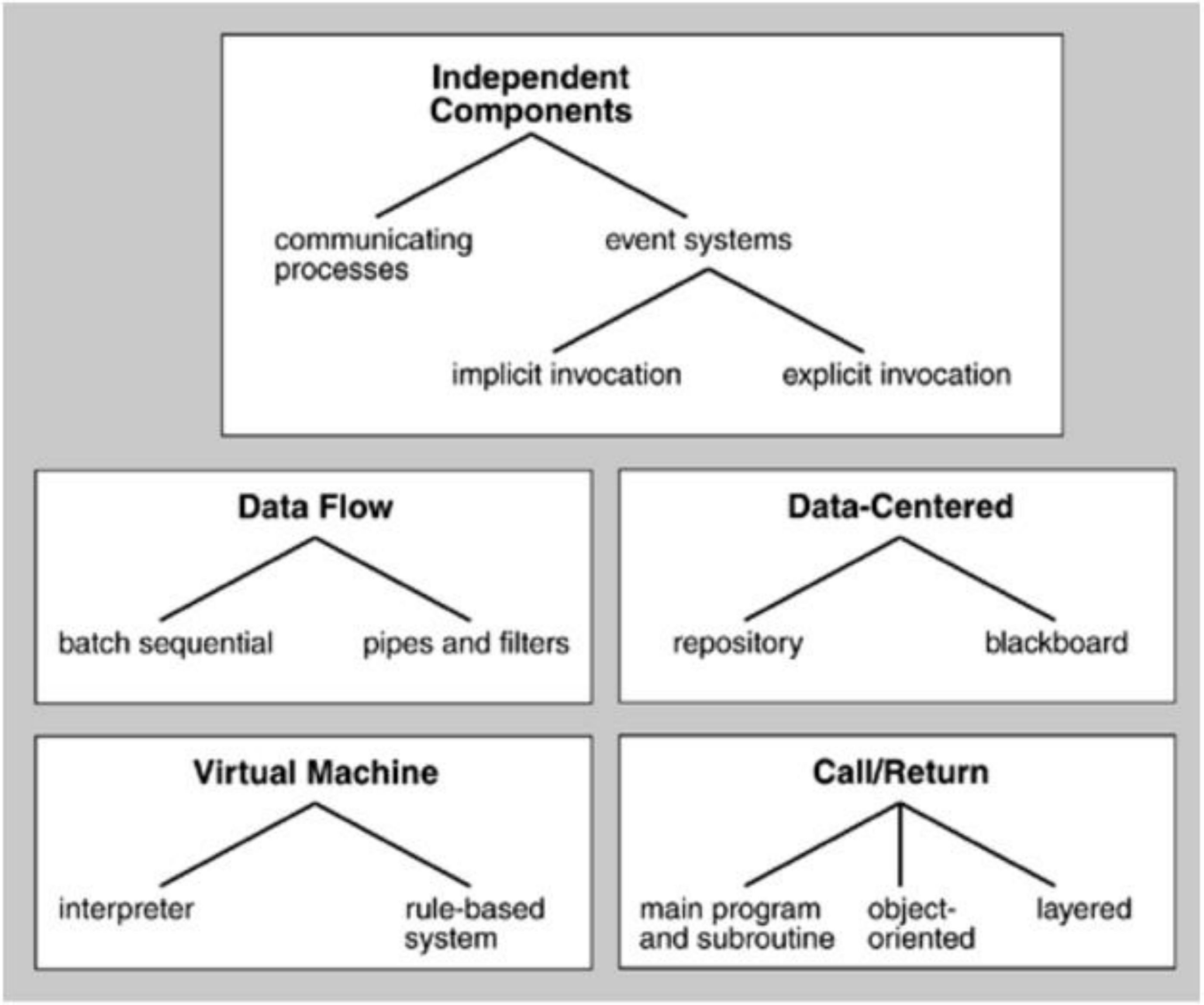

Usability
Separate User Interface
Support User Initiative
Support System Initiative
Cancel
Undo
Aggregate
User Model
System Model
Task Model
User Request
User Given Appropriate Feedback and Assistance

15. ARCHITECTURAL PATTERNS AND STYLES

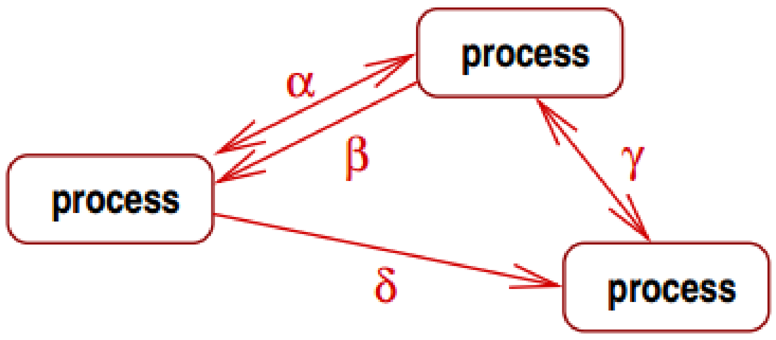
An architectural pattern is determined by:

- A set of element types (such as a data storehouse or a component that computes a mathematical function).
- A topological layout of the elements indicating their interrelation-ships.
- A set of semantic constraints (e.g., filters in a pipe-and-filter style are pure data transducers—they incrementally transform their input stream into an output stream, but do not control either upstream or downstream elements).
- A set of interaction mechanisms (e.g., subroutine call, event-subscriber, blackboard) that determine how the elements coordinate through the allowed topology.

SMALL CATALOG OF ARCHITECTURAL PATTERNS ORGANIZED BY IS-A RELATIONS



Independent components: communicating processes

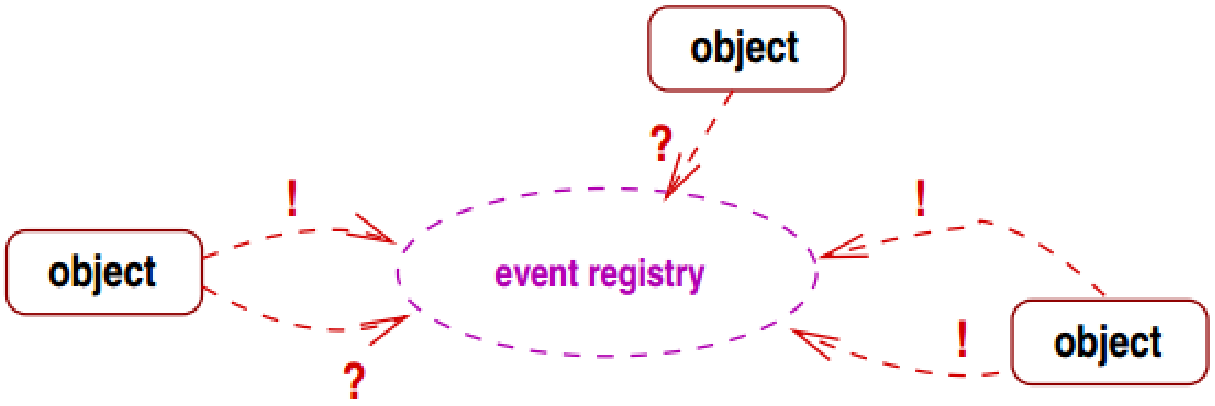


	Communicating processes
Components:	processes (“tasks”)
Connectors:	ports or buffers or RPC
Constraints:	processes execute in parallel and send messages (synchronously or asynchronously)

Example: client-server and peer-to-peer architectures

Advantages: easy to add and remove processes. *Disadvantages:* difficult to reason about control flow.

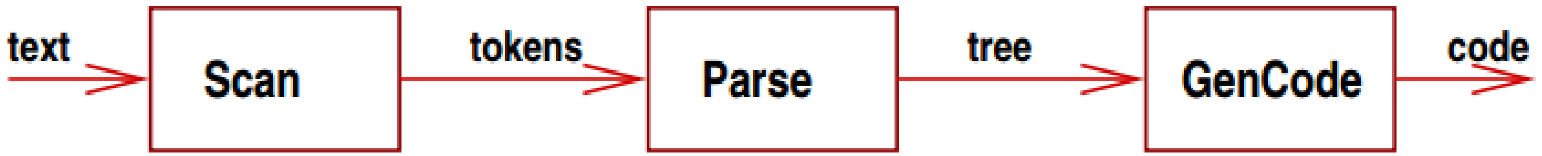
Independent components: event systems



	Event systems
Components:	objects or processes (“threads”)
Connectors:	event broadcast and notification (implicit invocation)
Constraints:	components “register” to receive event notification; components signal events, environment notifies registered “listeners”

Examples: GUI-based systems, debuggers, syntax-directed editors, database consistency checkers

Data-flow systems: **Batch-sequential and Pipe-and-filter**

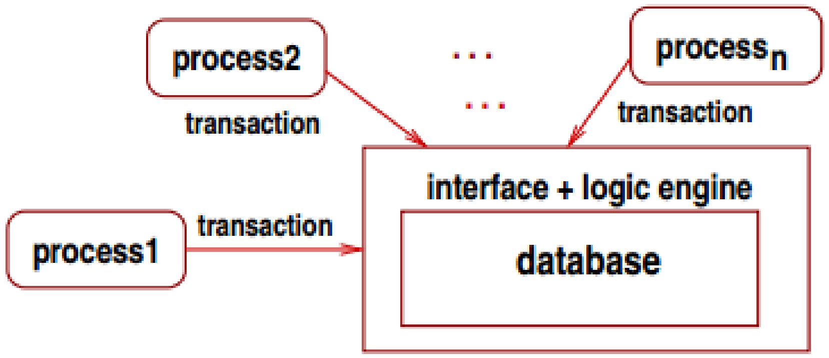


	Batch sequential	*Pipe and filter*
Components:	whole program	filter (function)
Connectors:	conventional input-output	pipe (data flow)
Constraints:	components execute to completion, consuming entire input, producing entire output	data arrives in increments to filters

Examples: Unix shells, signal processing, multi-pass compilers

Advantages: easy to unplug and replace filters; interactions between components easy to analyze. *Disadvantages:* interactivity with end-user severely limited; performs as quickly as slowest component.

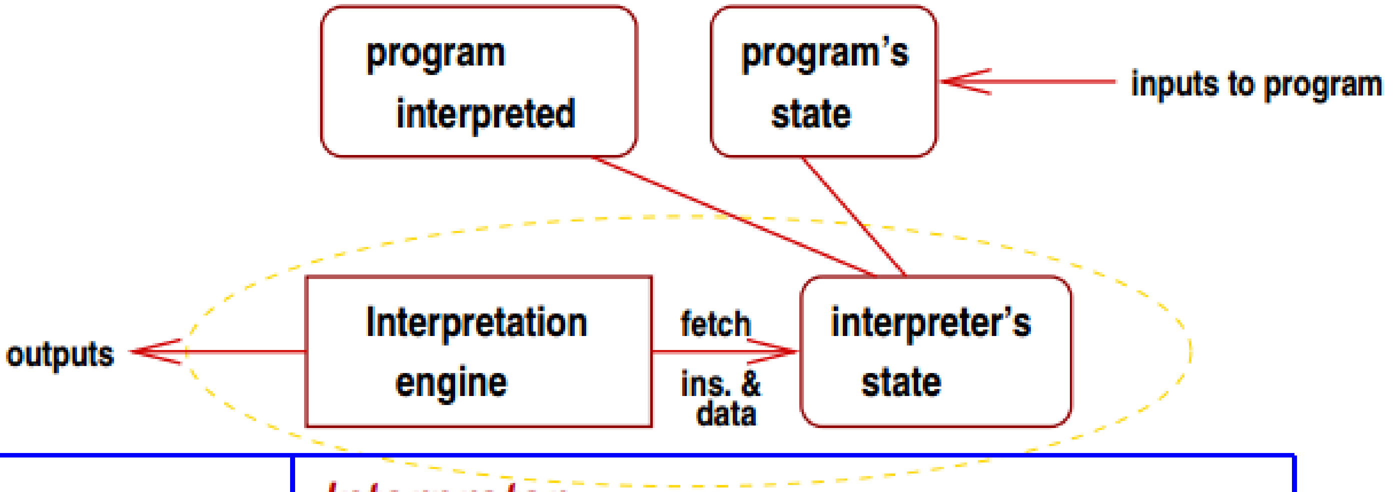
Repositories: databases and blackboards



	Database	*Blackboard*
Components:	processes and database	knowledge sources and blackboard
Connectors:	queries and updates	notifications and updates
Constraints:	transactions (queries and updates) drive computation	knowledge sources respond when enabled by the state of the blackboard. Problem is solved by cooperative computation on blackboard.

Examples: speech and pattern recognition (blackboard); syntax editors and compilers (parse tree and symbol table are repositories)

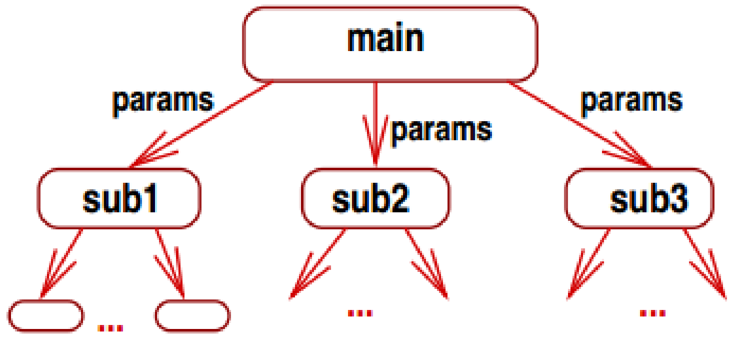
Virtual machine: interpreter

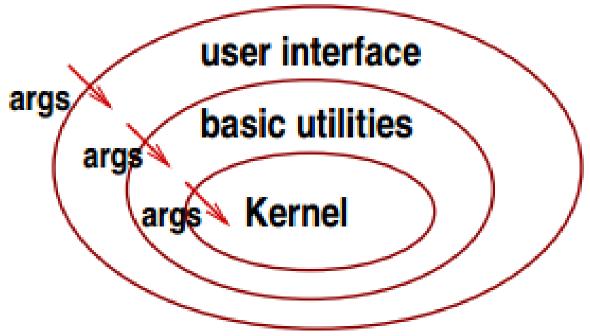


	Interpreter
Components:	"memories" and state-machine engine
Connectors:	fetch and store operations
Constraints:	engine's "execution cycle" controls the simulation of program's execution

Examples: high-level programming-language interpreters, byte-code machines, virtual machines

Call-and-return systems: subroutine and layered





	Subroutine	*Layered*
Components:	subroutines (“servers”)	functions (“servers”)
Connectors:	parameter passing	protocols
Constraints:	hierarchical execution and encapsulation	functions within a layer invoke (API of) others at next lower layer

Examples: modular, object-oriented, N-tier systems (subroutine); communication protocols, operating systems (layered)

RELATIONSHIP OF TACTICS TO ARCHITECTURAL PATTERNS

The pattern consists of six elements:

- a proxy, which provides an interface that allows clients to invoke publicly accessible methods on an active object;
- a method request, which defines an interface for executing the methods of an active object;
- an activation list, which maintains a buffer of pending method requests;
- a scheduler, which decides what method requests to execute next;
- a servant, which defines the behavior and state modeled as an active object;
- a future, which allows the client to obtain the result of the method invocation.

The tactics involves the following:

- Information hiding (modifiability). Each element chooses the responsibilities it will achieve and hides their achievement behind an interface.
- Intermediary (modifiability). The proxy acts as an intermediary that will buffer changes to the method invocation.
- Binding time (modifiability). The active object pattern assumes that requests for the object arrive at the object at runtime. The binding of the client to the proxy, however, is left open in terms of binding time.
- Scheduling policy (performance). The scheduler implements some scheduling policy.

16.CASE STUDY

- The KWIC (Key Word in Context) index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters.

- Any line may be “circularly shifted” by repeatedly removing the first word and appending it at the end of the line.

- The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order

Architectural solutions for KWIC

- Main program and subroutines, with shared data
- Abstract Data Types
- Event-based system
- Pipe-and-filter

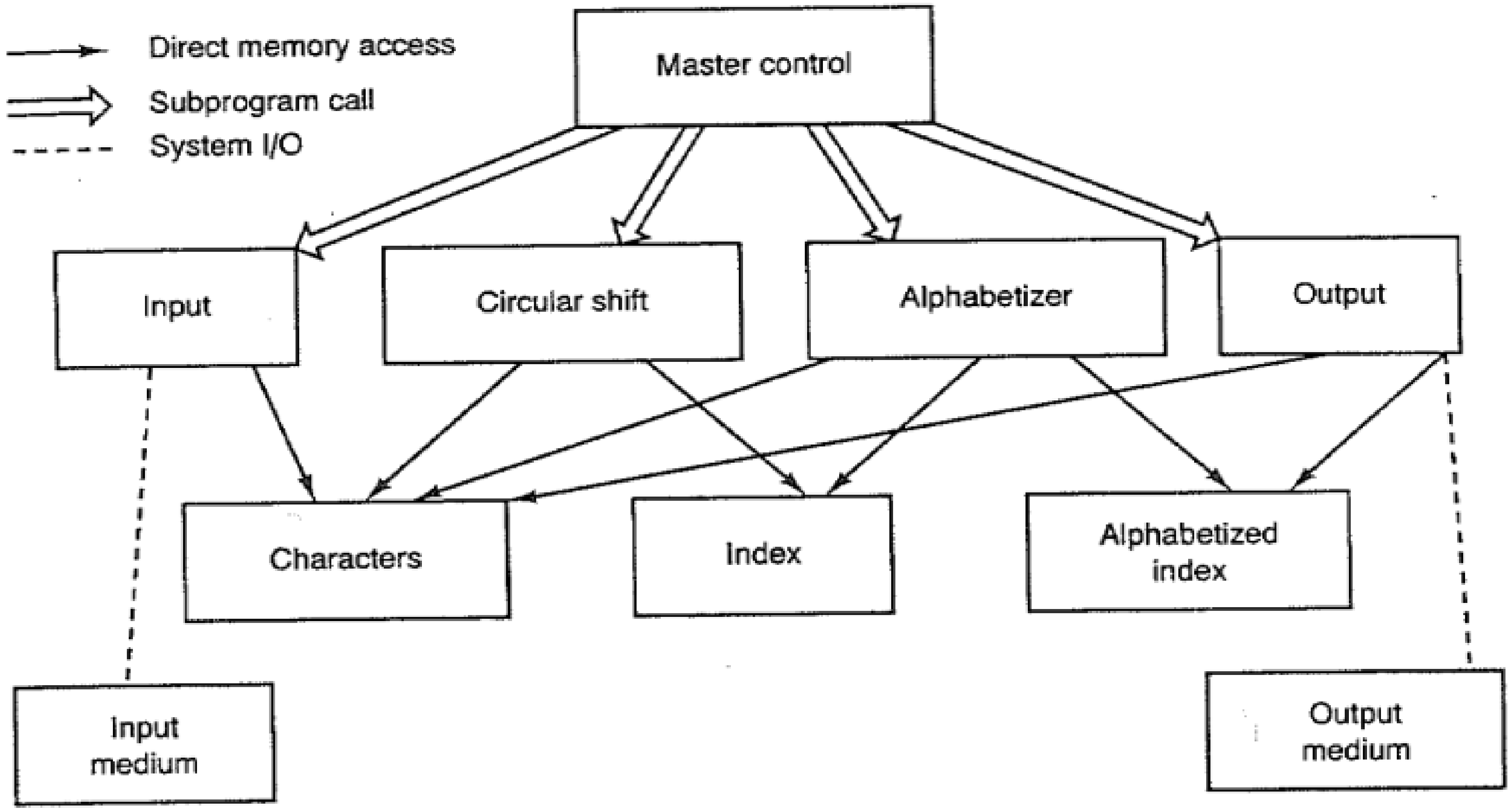
KWIC design issues

- Changes in the processing algorithm
- Line shifting on each line after it is read, on all lines after they are read or on demand
- Changes in the data representation
- Enhancement to system function
- Performance
- Reuse

Main Program/Subroutine with shared data

- Problem decomposed according to 4 basic functions
  - Input, shift, alphabetize, output
- Components coordinated by main program that sequences through them
- Data in shared storage
- Communication: unconstrained read-write protocol
- Coordinator ensures sequential access to data

KWIC Shared data Solution



SHARED DATA-PRO AND CONS

Advantages

- Data can be represented efficiently
- Intuitive appeal

Disadvantages

- Modifiability
  - Change in data format affects all components
  - Change in overall processing algorithm
  - Enhancements to system function
  - Reuse not easy to do