
Software
Architecutre

Software Architecture

- ***Software Architecture*** is the structure or structures of the system, which comprise software elements, the externally visible properties of these components, and the relationship among them.
- Box-and-line drawings alone are not architecture, but a starting point.
- Architecture includes behaviour of components.

Role of Architecture

- An architecture is one of the first artefacts that represents decision on how requirements are to be achieved. As the manifestation of early design decisions, the architecture represents those design decisions that are hardest to change and hence deserve the most careful consideration.
- An architecture is the key artefact in achieving successful product line engineering, the disciplined development of a family of similar system with less effort, expense, and risk than developing each system independently.
- An architecture is usually the first design artefact to be examined when someone starts working on a system.
- Software architecture provides a framework of reference for maintenance and modification decisions.

Why is software architecture important?

- Software architecture provides a vehicle for communication
  - It is a frame of reference in which competing interests may be identified and negotiated
    - Negotiating requirements with users
    - Keeping customer informed of progress, cost etc.
    - Implementing management decisions and allocations.

- Software architecture manifests the earliest set of design decisions
  - It constraints the implementation and developers
    - Implementation must conform to architecture
    - Resource allocation decisions constrain implementations of individual components

Why is software architecture important?

- Manifestation of early design decisions
  - Software architecture dictates organisational structure for development & maintenance efforts, e.g.,
    - Division into teams
    - Units for budgeting, planning
    - Basis of Work Breakdown Structure
    - Organisation for documentation
    - Organisation for CM libraries
    - Basis of integration
    - Basis of test plans, testing
    - Basis of maintenance

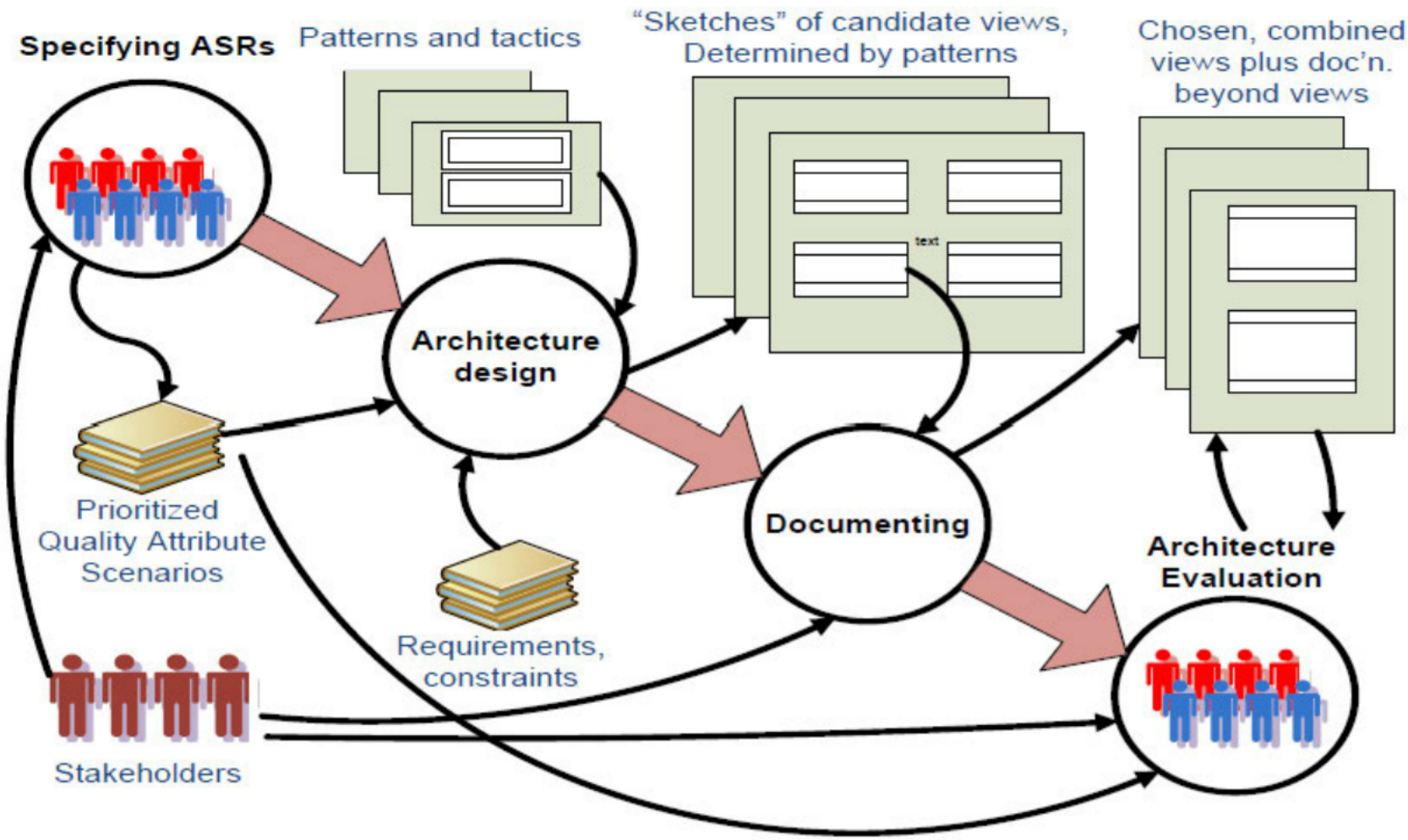
Why is software architecture important?

- Architecture facilitates/hinders achievement of quality attributes, e.g., modifiability, security, usability etc.
- Architecture influences qualities, but may not guarantee them as there are a number of other factors involved.
- An architecture invokes discussion about potential change (80% of effort for a system is post-deployment effort)
- Architecture categorise changes into three types:
  - Local: signal component modification
  - Non-local: several component modification.
  - Architectural: modification of the system's basic structure, communication, and coordination mechanism

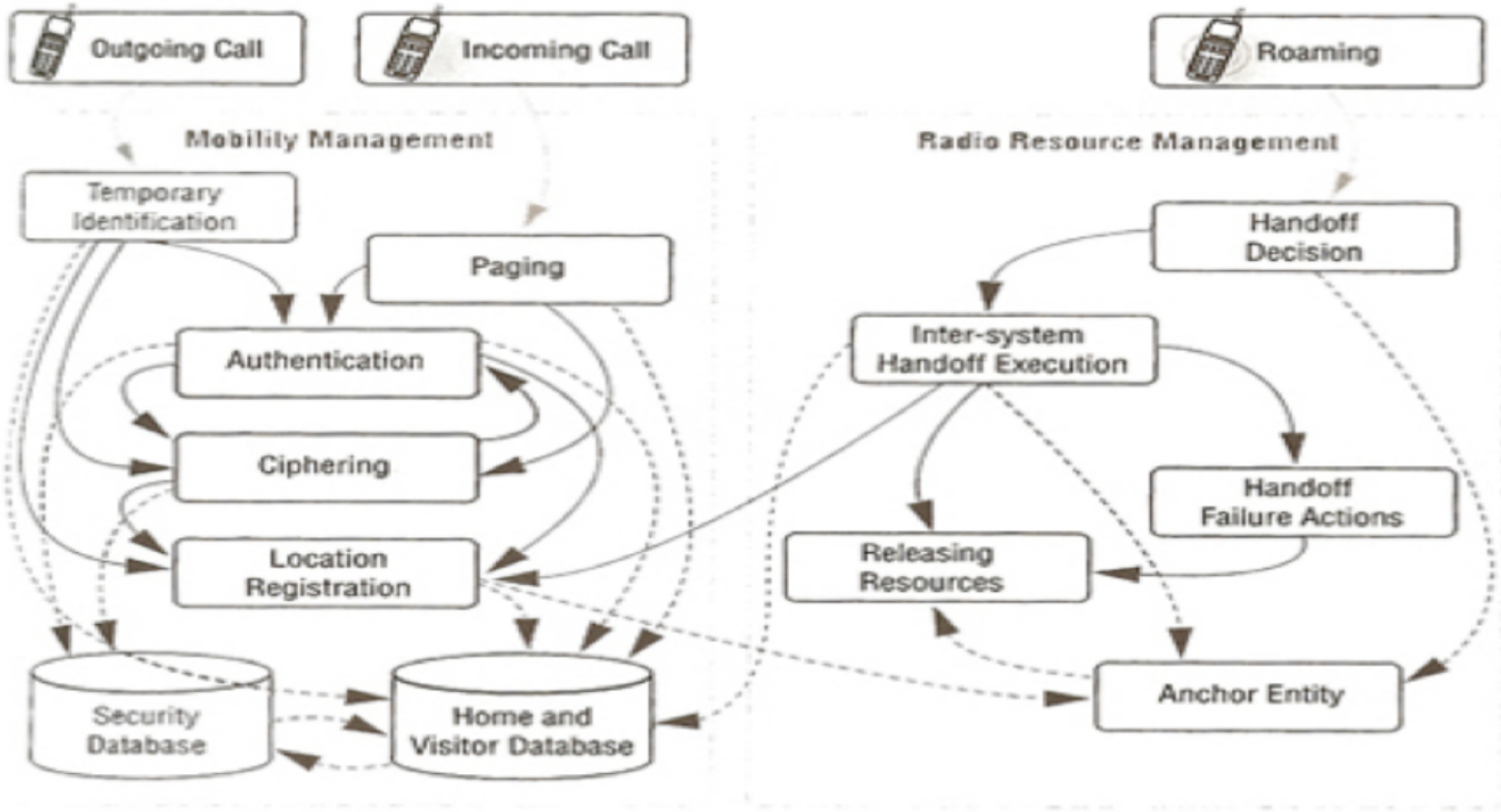
Why is software architecture important?

- Architecture is a transferable and reusable abstraction — one-to-many mapping (one architecture, many systems)
- Architecture is the basis for product commonality. A whole product line shares a single architecture
- Systems can be developed by integrating independently developed components via architecture (Component-Based Software Engineering – CBSE)

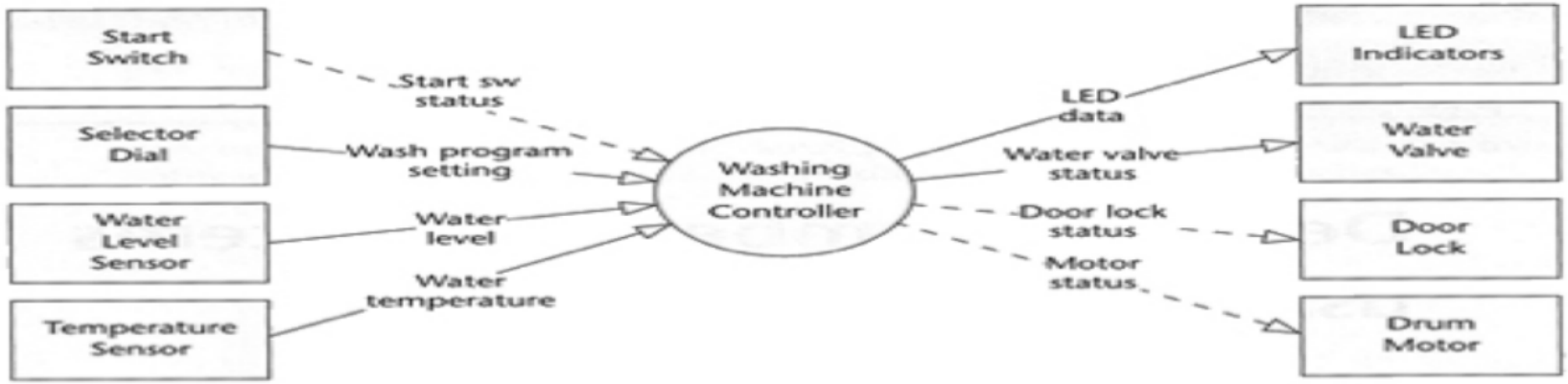
Software Architecture Process

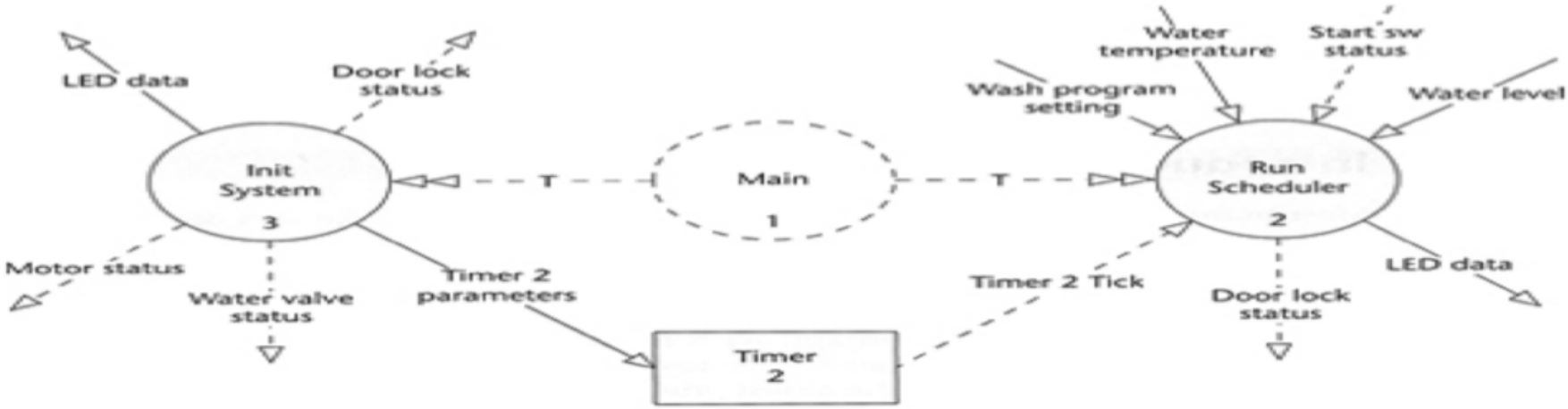


Mobile Phone System Architecture



Washing Machine Architecture



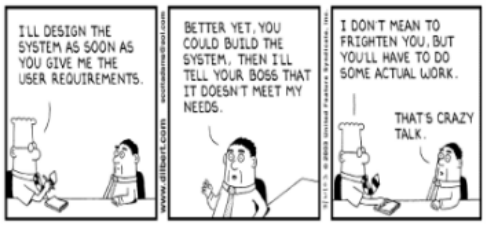


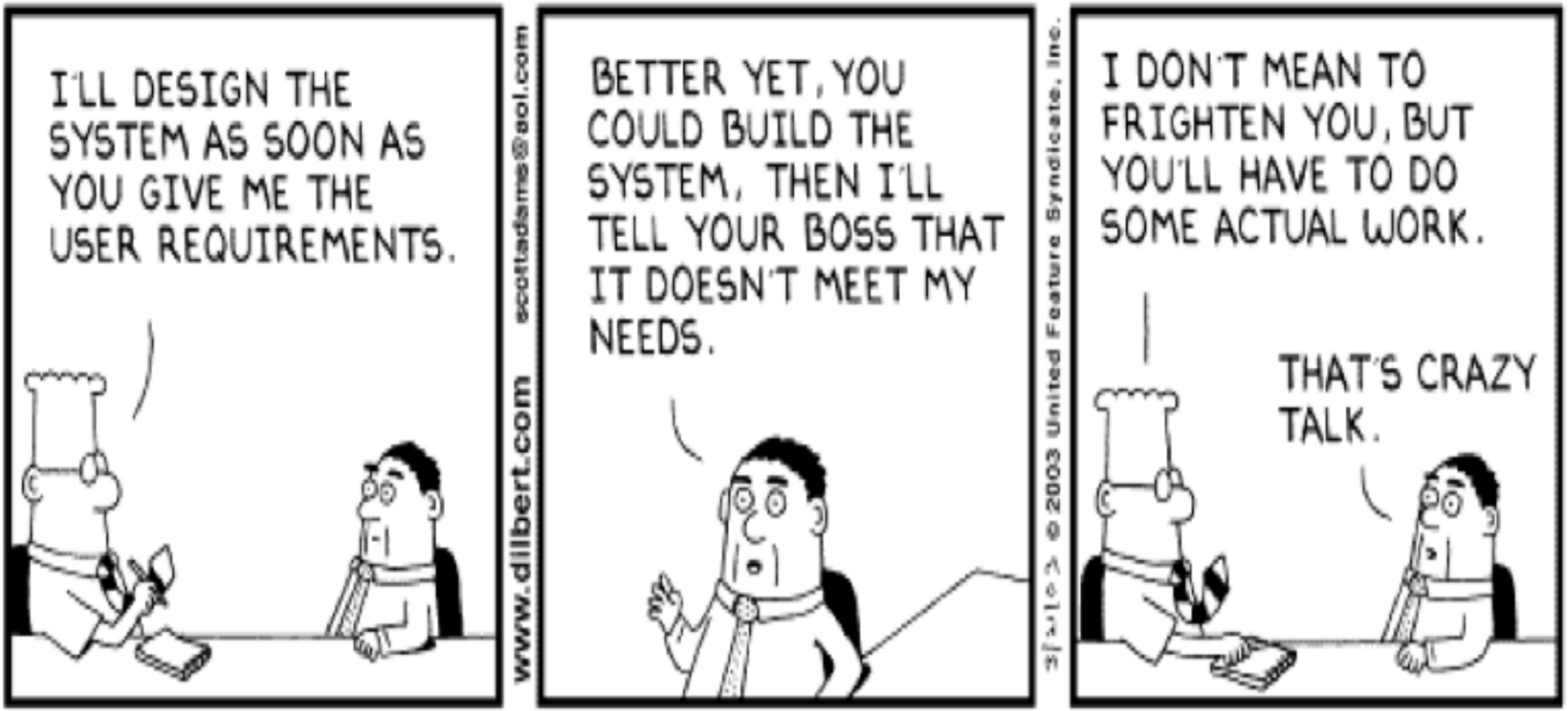
Discussion

- What is Difference between Science and Engineering?
- What is Difference between 'Software' and 'Hardware'?
- What is Difference between Architecture and Design?
- What is Difference between Architecture and Structure?
- Why Abstraction in Architecture?

- Functionality may be achieved th
number of possible structures.
- Functionality is largely independ
because it could exist as a single
without any internal structure.

Requirements




I'LL DESIGN THE SYSTEM AS SOON AS YOU GIVE ME THE USER REQUIREMENTS.
www.dilbert.com scottadams@aol.com
BETTER YET, YOU COULD BUILD THE SYSTEM, THEN I'LL TELL YOUR BOSS THAT IT DOESN'T MEET MY NEEDS.
3/3/03 © 2003 United Feature Syndicate, Inc.
I DON'T MEAN TO FRIGHTEN YOU, BUT YOU'LL HAVE TO DO SOME ACTUAL WORK.
THAT'S CRAZY TALK.

Functional Requirements

- Functional requirements state what the system must do and address how the system provides value to the stakeholders.
- Functional requirements means the behaviour of the system.
- Functionality is the ability of the system to do the work for which it was intended, e.g., enable students to enrol online.
- Functionality may be achieved through the use of any number of possible structures.
- Functionality is largely independent of structure, because it could exist as a single monolithic system without any internal structure.

- Functional requirements state what the system must do and address how the system provides value to the stakeholders.
- Functional requirements means the behaviour of the system.
- Functionality is the ability of the system to do the work for which it was intended, e.g., enable students to enrol online.
- Functionality may be achieved through the use of any number of possible structures.
- Functionality is largely independent of structure, because it could exist as a single monolithic system without any internal structure.

Quality Requirements

- Quality requirements are desirable characteristics of the overall system (aka. quality attributes) that system should provide on the top of its functional requirements.
- Quality requirements are qualifications of the functional requirements or of the overall product.
- Software architecture constrains the allocation (mapping) of the functionality onto various structures if quality attributes are important.

- Functionality
  development
- However, sys
  they lack des
  maintain, po
- Software arc
  of various qu
  security, usal
- That is why s
  most approp
  issues.
- No quality a
  design, nor is
  deployment.

Specif

- Precise defin
  necessary to
- Quality attri
  desired qual
- Scenarios ar
  structure. Tw
  - General
    scenarios
    attribute
  - Concrete
    scenarios
    attribute
    system. T
    scenarios

- General scer
  generating c
  independent
- Each scenar
  relevant to th
- To make the
  particular sy
  specific.
- Making a ge
  translating it
  system.

Modelin

- **Stimulus**: A conditio
  system.
- **Source** of Stimulus:
  generates the stimul
- **Response**: The activ
- Response **Measure**:
  some fashion so that
- **Environment**: A syst
  overloaded, running
- **Artifact**: The whole
  requirement applies.

- Quality requirements are desirable characteristics of the overall system (aka. quality attributes) that system should provide on the top of its functional requirements.
- Quality requirements are qualifications of the functional requirements or of the overall product.
- Software architecture constrains the allocation (mapping) of the functionality onto various structures if quality attributes are important.

Constraints

- A constraint is a design decision with ZERO degrees of freedom.
- Constraints are pre-specified design decisions that have been already made.
- Constraints are satisfied by accepting the design decision and reconciling it with other affected design decisions.

- A constraint is a design decision with ZERO degrees of freedom.
- Constraints are pre-specified design decisions that have been already made.
- Constraints are satisfied by accepting the design decision and reconciling it with other affected design decisions.

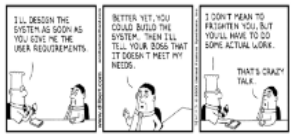
Functional Requirements

- Functional requirements state what the system must do and address how the system provides value to the stakeholders.
- Functional requirements means the behaviour of the system.
- Functionality is the ability of the system to do the work for which it was intended, e.g., enable students to enrol online.
- Functionality may be achieved through the use of any number of possible structures.
- Functionality is largely independent of structure, because it could exist as a single monolithic system without any internal structure.

Non-functional Requirements

- Non-functional requirements or architectural requirements are alternative terms used for quality attributes.
- It is not possible to get the functionality right and then try to accommodate non-functional requirements (FAD refactoring cycling).
- Non-functional requirements must be taken into account during any design decision.
- There are two broad categories of non-functional requirements:
  - Observable (External) during execution: How well a system satisfies its behavioural requirements? e.g., performance, security, availability, usability etc.
  - Not observable (Internal) during execution: How easily a system can be maintained, integrated, or tested? e.g., modifiability, portability, reusability, testability etc.

Requirements



Quality Requirements

- Quality requirements are desirable characteristics of the overall system (also quality attributes) that system should provide on the top of its functional requirements.
- Quality requirements are qualifications of the functional requirements or of the overall product.
- Software architecture constrains the allocation (mapping) of the functionality onto various structures if quality attributes are important.

Quality Design Decisions

- Architecture is a collection of design decisions.
- Seven categories of design decisions (may overlap):
  - Allocation of responsibilities
  - Coordination model
  - Data model
  - Management of resources
  - Mapping among architecture elements
  - Binding time decisions
  - Choice of technology

Constraints

Quality Requirements

- Quality requirements are desirable characteristics of the overall system (aka. quality attributes) that system should provide on the top of its functional requirements.
- Quality requirements are qualifications of the functional requirements or of the overall product.
- Software architecture constrains the allocation (mapping) of the functionality onto various structures if quality attributes are important.

- Functionality
  development
- However, sys
  they lack des
  maintain, po
- Software arc
  of various qu
  security, usal
- That is why s
  most approp
  issues.
- No quality a
  design, nor is
  deployment.

Specif

- Precise defin
  necessary to
- Quality attri
  desired qual
- Scenarios ar
  structure. Tw
  - General
    scenarios
    attribute
  - Concrete
    scenarios
    attribute
    system. T
    scenarios

- General scer
  generating c
  independent
- Each scenar
  relevant to th
- To make the
  particular sy
  specific.
- Making a ge
  translating it
  system.

Modelin

- **Stimulus**: A conditio
  system.
- **Source** of Stimulus:
  generates the stimul
- **Response**: The activ
- Response **Measure**:
  some fashion so that
- **Environment**: A syst
  overloaded, running
- **Artifact**: The whole
  requirement applies.

Non-functional Requirements

- Non-functional requirements or architectural requirements are alternative terms used for quality attributes.
- It is not possible to get the functionality right and then try to accommodate non-functional requirements (NO retro-fitting quality).
- Non-functional requirements must be taken into account during any design decision.
- There are two broad categories of non-functional requirements:
  - Observable (External) during execution: How well a system satisfies its behavioural requirements? e.g., performance, security, availability, usability etc.
  - Not observable (Internal) during execution: How easily a system can be maintained, integrated, or tested? e.g., modifiability, portability, reusability, testability etc.

Quality Attributes

- Quality isn't something that can be added to a software intensive system after development finishes.
- Quality concerns need to be addressed during ALL phases of the software development.
- Business goals determine qualities that a system must posses.
- Quality attributes are over and above of system's functionality, which is the basic statement of the system's capabilities, services, and behaviours.

Quality Attributes

- Functionality usually takes the front seat in the development plan.
- However, systems are usually redesigned because they lack desired level of quality, i.e. difficult to maintain, port, or scale.
- Software architecture constrains the achievement of various quality attributes, e.g., performance, security, usability etc.
- That is why software architecture is considered the most appropriate level of addressing the quality issues.
- No quality attribute is entirely dependent on design, nor is it dependent on implementation or deployment.

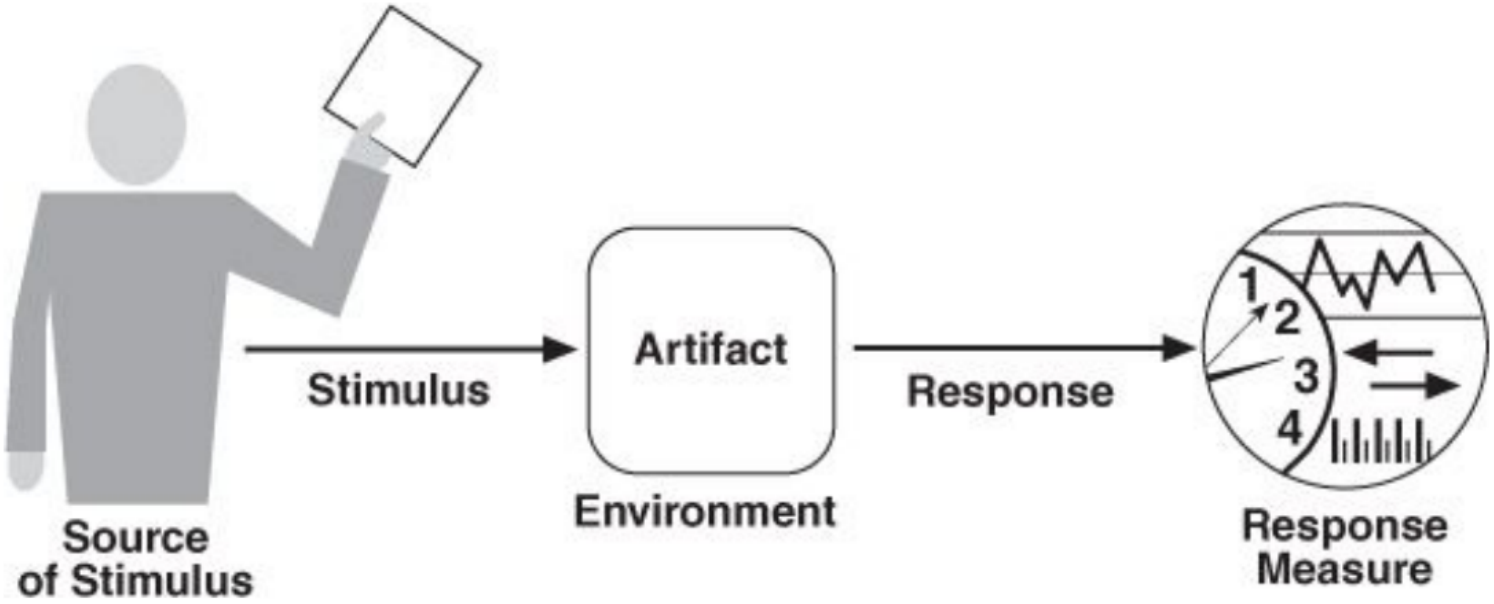
Specifying Quality Attributes

- Precise definition of a quality attribute is necessary to evaluate it at the architecture level.
- Quality attribute scenarios are used to define the desired quality attribute.
- Scenarios are simple descriptions with certain structure. Two main classes of scenarios are:
  - General scenarios are system independent scenarios to guide the specification of quality attribute requirements.
  - Concrete scenarios are system specific scenarios to guide the specification of quality attribute requirements for a particular system. They are instances of general scenarios.

General Scenarios

- General scenarios provide a framework for generating a large number of generic, system-independent, quality attribute specific scenarios.
- Each scenario is potentially but not necessarily relevant to the system we are concerned with.
- To make the general scenario useful for a particular system, we must make them system specific.
- Making a general scenario system specific means translating it into concrete terms for the particular system.

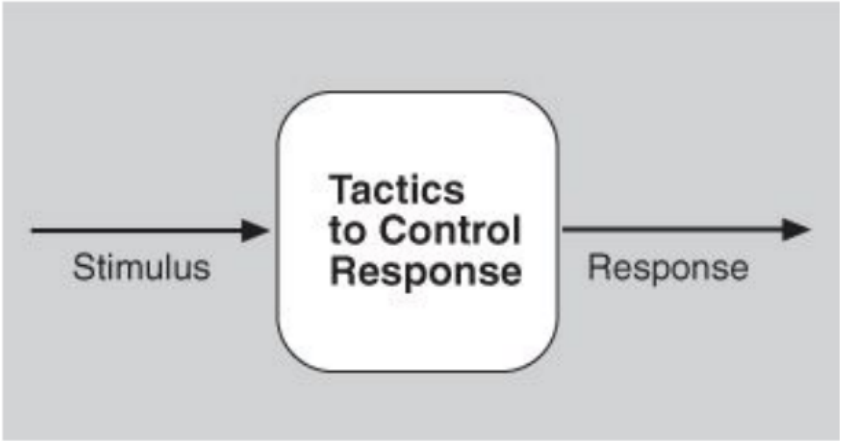
Modeling Quality Attribute Scenarios



- **Stimulus**: A condition that needs to be considered when it arrives at a system.
- **Source** of Stimulus: An entity (human, system, or any actuator) that generates the stimulus.
- **Response**: The activity undertaken after the arrival of the stimulus.
- Response **Measure**: The response to the stimulus should be measurable in some fashion so that the requirement can be testable.
- **Environment**: A system's condition when a stimulus occurs, e.g., overloaded, running etc.
- **Artifact**: The whole system or the portion of the system to which the requirement applies.

Tactics

- **Style** or **pattern** applies **tactics** to provide the promised benefit.
- A tactic is a design decision, .e.g., redundancy, that influences the control of a quality attribute response.
- A collection of tactics is called an architectural **strategy**.
- A system design consists of a collection of design decisions: some of these decisions help control the quality attribute response; others ensure achievement of system functionality.
- Like patterns, tactics may also be composed of other tactics, e.g., redundancy may be composed of redundancy of data, redundancy of computation- Designer chooses one or other depending upon requirements.
- Tactics can be used as hierarchy of tactics.



Quality Design Decisions

- Archtecture is a collection of design decisions.
- Seven categories of design decisions (may overlap):
  - Allocation of responsibilities
  - Coordination model
  - Data model
  - Management of resources
  - Mapping among architecture elements
  - Binding time decisions
  - Choice of technology

Adaptability
Availability
Configurability
Flexibility
Interoperability
Performance
Reliability
Responsiveness
Recoverability
Scalability
Stability
Security
Extensibility
Modularity
Portability
Reusability
Testability
Auditability
Maintainability
Manageability
Sustainability
Supportability
Usability

Quality Attributes & Tactics

Availability

mined by:

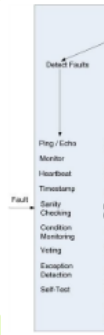
ure

art

performance levels
ter an application or

applications
of the required time

usiness hours
heduled downtime
% availability)
liability
ffer poor



Checklist for

Availability

- Key requirement for most IT applications
- Measured by the proportion of the required time it is useable, e.g.,
  - 100% available during business hours
  - No more than 2 hours scheduled downtime per week – 24x7x52 (100% availability)
- Related to an application's reliability
  - Unreliable applications suffer poor availability

Availability

- Period of loss of availability determined by:
  - Time to detect failure
  - Time to correct failure
  - Time to restart application
- Strategies for high availability:
  - Eliminate single points of failure
  - Replication and failover
  - Automatic detection and restart
- Recoverability (e.g., a database)
  - The capability to re-establish performance levels and recover affected data after an application or system failure.

Availability

- Availability can be calculated as the probability that it will provide the specified services within required bounds over a specified time interval.
  - MTBF (mean time between failures)
  - MTTR (mean time to repair)

$$\frac{MTBF}{(MTBF + MTTR)}$$

- Scheduled downtimes may not be considered when calculating availability.

Outage, Failure, Fault, and Error

- Availablity is about minimizing the service outage time by mitigating faults.

- A failure's cause is called a fault.
- A failure occurs when a system cannot deliver a service that is expected of that system.
- A failure is an observable characteristics of a system's state.

- A fault in any part of a system has a potential to cause a failure; a system can be repaired or recovered from a failure.

- Intermediate states between the occurrence of a fault and a failure are called errors.

Service-Level Agreement

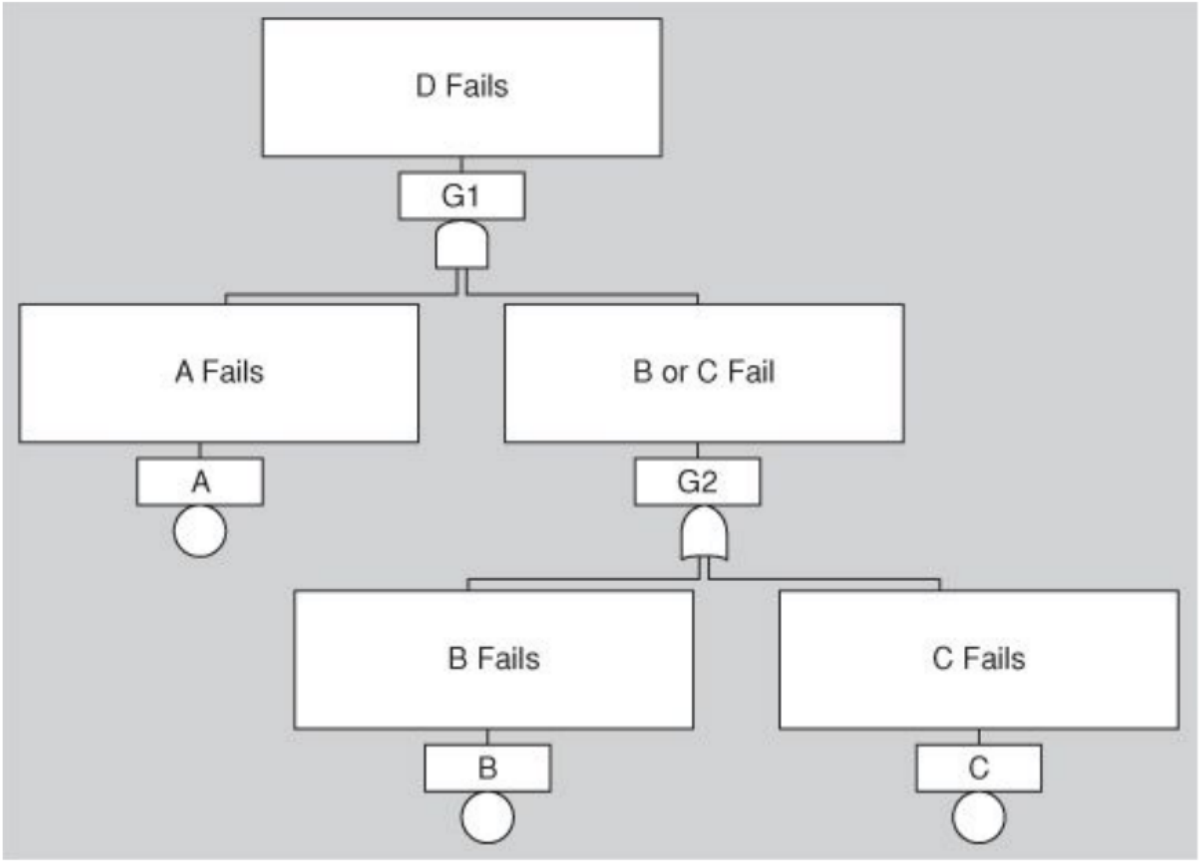
Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

- Amazon EC2's SLA:
  AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage of at least 99.95% during the Service Year. In the event Amazon EC2 does not meet the Annual Uptime Percentage commitment, you will be eligible to receive a Service Credit.

Planning for Failure

- Hazard analysis:
  - Catastrophic/Hazardous
  - Major/Minor
  - No effect

- Fault tree analysis:



Planning for Failure

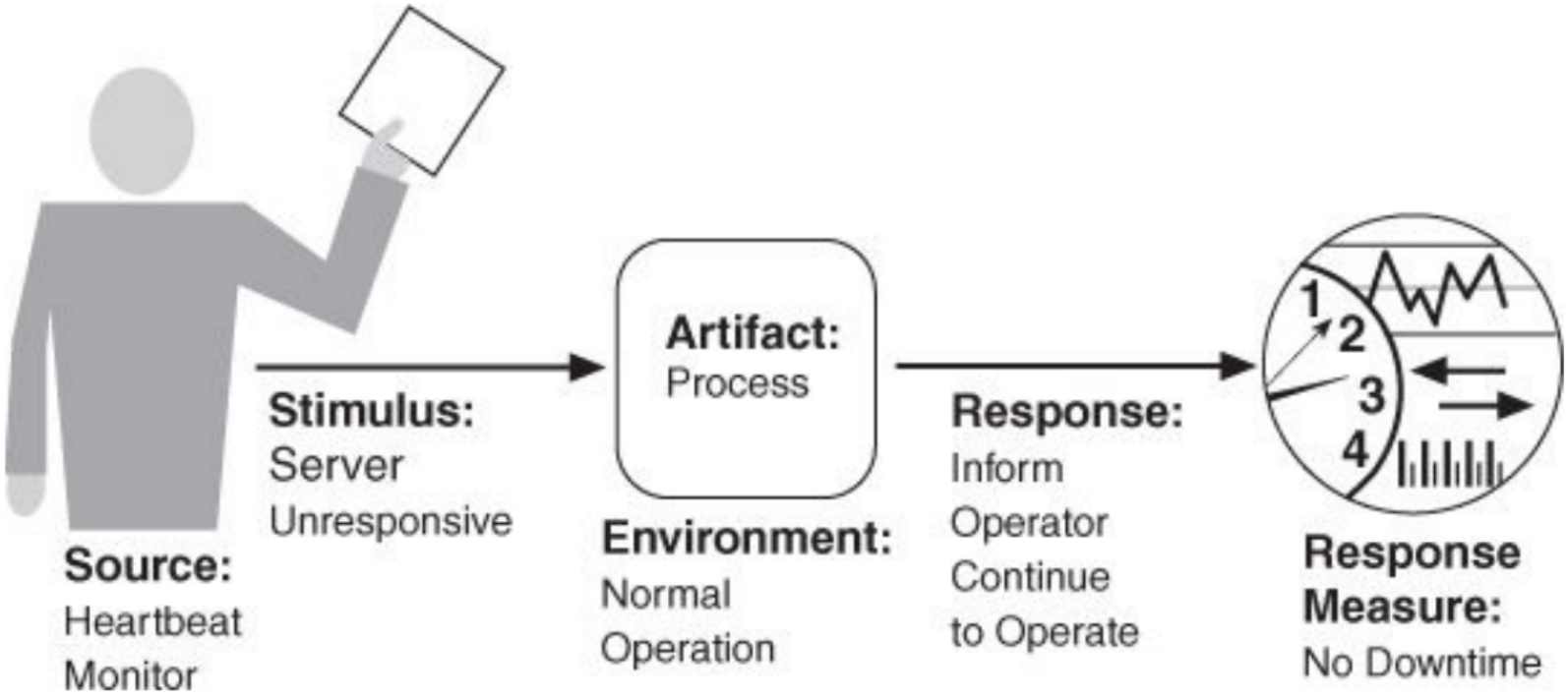
- Failure Mode, Effects, and Criticality Analysis (FMECA)
  - FMECA relies on the history of failure of similar systems in the past.

Component	Failure Probability	Failure Mode	% Failures by Mode	Effects	
				Critical	Noncritical
		Open	90		X
A	1×10^{-3}	Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	
B	1×10^{-3}	Open	90		X
		Short	5	X (5×10^{-5})	
		Other	5	X (5×10^{-5})	

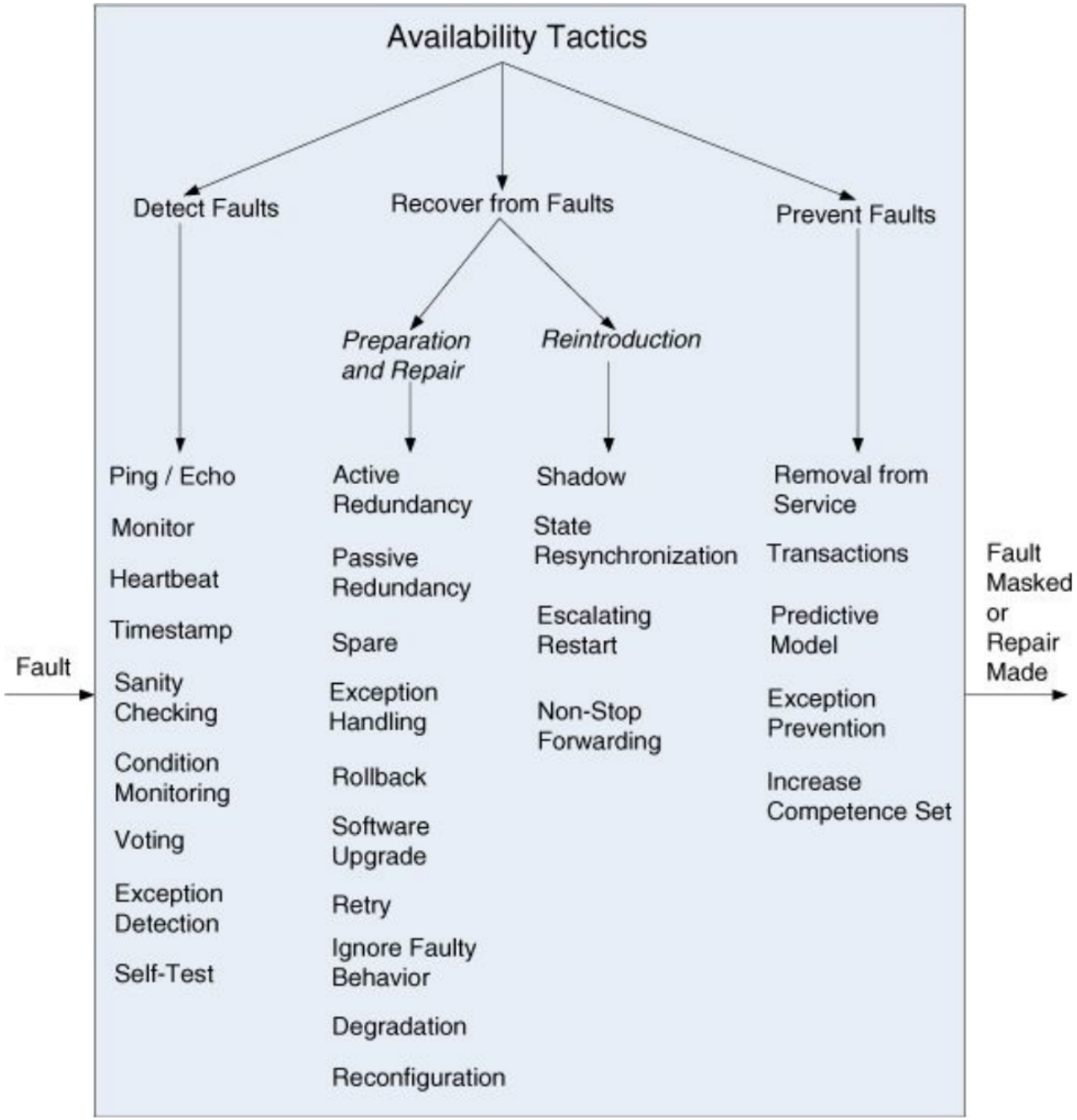
Availability General Scenario

Portion of Scenario	Possible Values
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
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
Response Measure	Time or time interval when the system must be available Availability percentage (e.g., 99.999%) Time to detect the fault Time to repair the fault Time or time interval in which system can be in degraded mode Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

Availability Sample Scenario



Availability Tactics



Fault Detection

- **Ping/Echo**
  - One component issues a ping and expects an echo from another component within a pre-defined time.
  - Ping/Echo can be used within a group of components responsible for one task.
- **Heartbeat** (dead man time)
  - One component emits a heartbeat message (can also carry data) periodically and another component listens for it.
  - If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
- **Exception**
  - One method for recognising faults is to encounter an exception.
  - The exception handler typically executes in the same process that introduces the exception.
- The ping and heartbeat tactics operate among distinct processes, and the exception tactic operates within a single process.

Fault Recovery

- **Voting**
  - Processes running on redundant processors each take equivalent input and compute a simple value that is sent to a voter.
  - If the voter detects deviant behaviour from a single process, it fails it.
- **Active redundancy**
  - All redundant components respond to events in parallel – there are all in the same state.
  - The response from only one component is used, and the rest are discarded.
  - When a failure occurs, the downtime is usually non-existent as backup is current and the only switching time is the recovery time.
- **Passive redundancy**
  - One component (primary) responds to events and informs the other components (secondary) of state updates they must make.
  - When a failure occurs, the system must first ensure that the backup state is sufficiently recent before resuming services.
- **Spare**
  - A standby spare computing platform is configured to replace many different failed components.

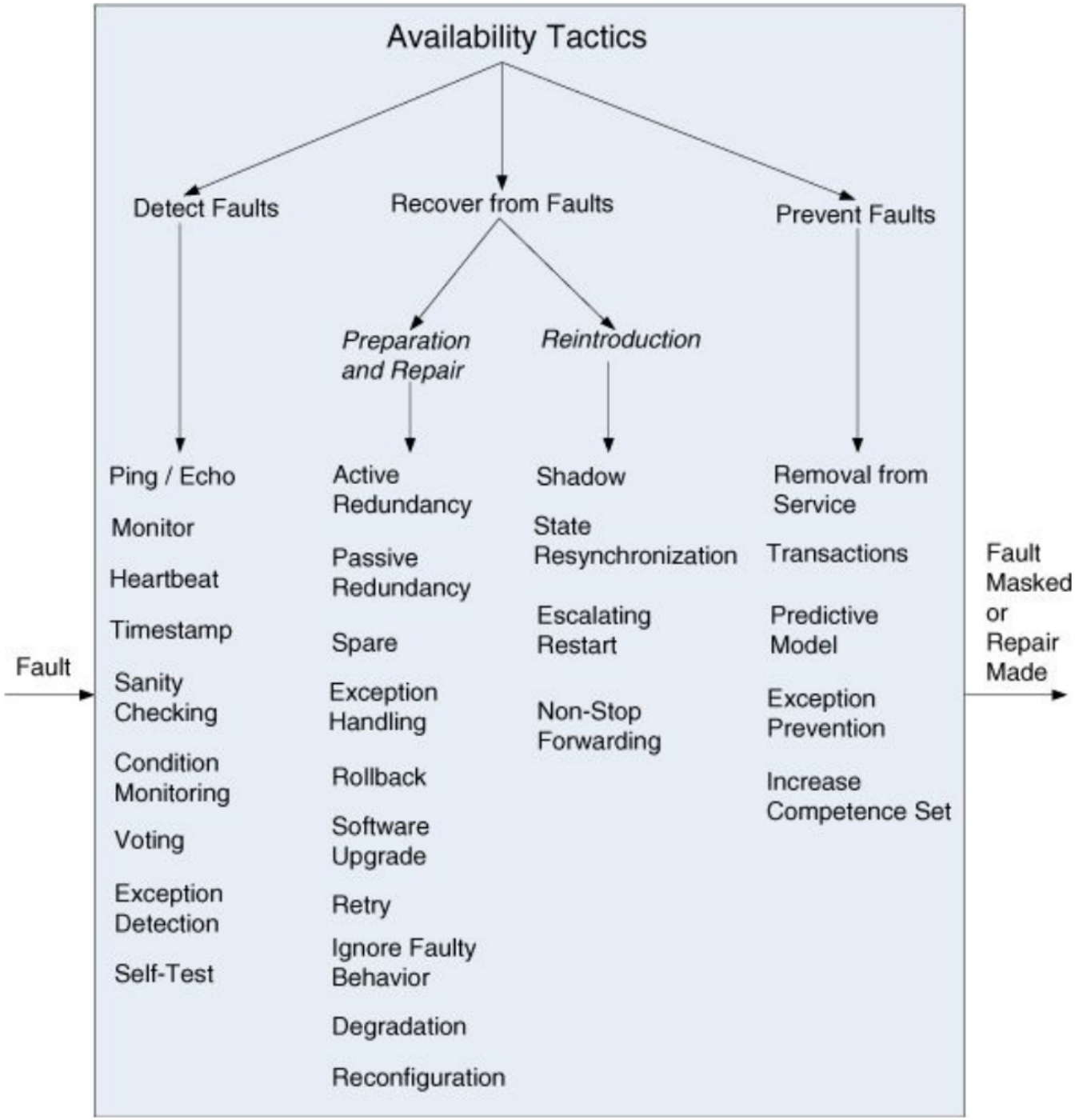
Fault Recovery

- **Shadow operation**
  - A previously failed component may be run in "shadow mode" for a short time to make sure that it mimics the behaviour of the working components before restoring it to service.
- **State re-synchronisation**
  - The passive and active redundancy tactics require the component being restored to have its state upgraded before its return to service.
- **Checkpoint/Rollback**
  - A checkpoint is recording of a consistent state created either periodically or in response to specific events.

Fault Recovery

- **Removal from service**
  - This tactic removes a component of the system from operation to undergo some activities to prevent anticipated failure.
- **Transaction**
  - A transaction is the bundling of several sequential steps such that the entire bundle can be undone at once.
- **Process monitor**
  - Once a fault in a process has been detected, a monitoring process can detect the non-performing process and create a new instance of it, initialised to some appropriate state as in the spare tactic.

Availability Tactics



Checklist for Availability Design & Analysis

Category	Checklist
Allocation of Responsibilities	Determine the system responsibilities that need to be highly available. Within those responsibilities, ensure that additional responsibilities have been allocated to detect an omission, crash, incorrect timing, or incorrect response. Additionally, ensure that there are responsibilities to do the following: ▪ Log the fault ▪ Notify appropriate entities (people or systems) ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable ▪ Fix or mask the fault/failure ▪ Operate in a degraded mode
Coordination Model	Determine the system responsibilities that need to be highly available. With respect to those responsibilities, do the following: ▪ Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response. Consider, for example, whether guaranteed delivery is necessary. Will the coordination work under conditions of degraded communication? ▪ Ensure that coordination mechanisms enable the logging of the fault, notification of appropriate entities, disabling of the source of the events causing the fault, fixing or masking the fault, or operating in a degraded mode. ▪ Ensure that the coordination model supports the replacement of the artifacts used (processors, communications channels, persistent storage, and processes). For example, does replacement of a server allow the system to continue to operate? ▪ Determine if the coordination will work under conditions of degraded communication, at startup/shutdown, in repair mode, or under overloaded operation. For example, how much lost information can the coordination model withstand and with what consequences?
Data Model	Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions, along with their operations or their properties, could cause a fault of omission, a crash, incorrect timing behavior, or an incorrect response. For those data abstractions, operations, and properties, ensure that they can be disabled, be temporarily unavailable, or be fixed or masked in the event of a fault. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce a fault: omission, crash, incorrect timing, or incorrect response. Ensure that the mapping (or remapping) of architectural elements is flexible enough to permit the recovery from the fault. This may involve a consideration of the following: ▪ Which processes on failed processors need to be reassigned at runtime ▪ Which processors, data stores, or communication channels can be activated or reassigned at runtime ▪ How data on failed processors or storage can be served by replacement units ▪ How quickly the system can be reinstalled based on the units of delivery provided ▪ How to (re)assign runtime elements to processors, communication channels, and data stores ▪ When employing tactics that depend on redundancy of functionality, the mapping from modules to redundant components is important. For example, it is possible to write one module that contains code appropriate for both the active component and backup components in a protection group.
Resource Management	Determine what critical resources are necessary to continue operating in the presence of a fault: omission, crash, incorrect timing, or incorrect response. Ensure there are sufficient remaining resources in the event of a fault to log the fault; notify appropriate entities (people or systems); disable the source of events causing the fault; be temporarily unavailable; fix or mask the fault/failure; operate normally, in startup, shutdown, repair mode, degraded operation, and overloaded operation. Determine the availability time for critical resources, what critical resources must be available during specified time intervals, time intervals during which the critical resources may be in a degraded mode, and repair time for critical resources. Ensure that the critical resources are available during these time intervals. For example, ensure that input queues are large enough to buffer anticipated messages if a server fails so that the messages are not permanently lost.

Category	Checklist
Allocation of Responsibilities	Determine the system responsibilities that need to be highly available. Within those responsibilities, ensure that additional responsibilities have been allocated to detect an omission, crash, incorrect timing, or incorrect response. Additionally, ensure that there are responsibilities to do the following: ▪ Log the fault ▪ Notify appropriate entities (people or systems) ▪ Disable the source of events causing the fault ▪ Be temporarily unavailable ▪ Fix or mask the fault/failure ▪ Operate in a degraded mode
Coordination Model	Determine the system responsibilities that need to be highly available. With respect to those responsibilities, do the following: ▪ Ensure that coordination mechanisms can detect an omission, crash, incorrect timing, or incorrect response. Consider, for example, whether guaranteed delivery is necessary. Will the coordination work under conditions of degraded communication? ▪ Ensure that coordination mechanisms enable the logging of the fault, notification of appropriate entities, disabling of the source of the events causing the fault, fixing or masking the fault, or operating in a degraded mode. ▪ Ensure that the coordination model supports the replacement of the artifacts used (processors, communications channels, persistent storage, and processes). For example, does replacement of a server allow the system to continue to operate? ▪ Determine if the coordination will work under conditions of degraded communication, at startup/shutdown, in repair mode, or under overloaded operation. For example, how much lost information can the coordination model withstand and with what consequences?
Data Model	Determine which portions of the system need to be highly available. Within those portions, determine which data abstractions, along with their operations or their properties, could cause a fault of omission, a crash, incorrect timing behavior, or an incorrect response. For those data abstractions, operations, and properties, ensure that they can be disabled, be temporarily unavailable, or be fixed or masked in the event of a fault. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce

	ensure that they can be disabled, be temporarily unavailable, or be fixed or masked in the event of a fault. For example, ensure that write requests are cached if a server is temporarily unavailable and performed when the server is returned to service.
Mapping among Architectural Elements	Determine which artifacts (processors, communication channels, persistent storage, or processes) may produce a fault: omission, crash, incorrect timing, or incorrect response. Ensure that the mapping (or remapping) of architectural elements is flexible enough to permit the recovery from the fault. This may involve a consideration of the following: ▪ Which processes on failed processors need to be reassigned at runtime ▪ Which processors, data stores, or communication channels can be activated or reassigned at runtime ▪ How data on failed processors or storage can be served by replacement units ▪ How quickly the system can be reinstalled based on the units of delivery provided ▪ How to (re)assign runtime elements to processors, communication channels, and data stores ▪ When employing tactics that depend on redundancy of functionality, the mapping from modules to redundant components is important. For example, it is possible to write one module that contains code appropriate for both the active component and backup components in a protection group.
Resource Management	Determine what critical resources are necessary to continue operating in the presence of a fault: omission, crash, incorrect timing, or incorrect response. Ensure there are sufficient remaining resources in the event of a fault to log the fault; notify appropriate entities (people or systems); disable the source of events causing the fault; be temporarily unavailable; fix or mask the fault/failure; operate normally, in startup, shutdown, repair mode, degraded operation, and overloaded operation. Determine the availability time for critical resources, what critical resources must be available during specified time intervals, time intervals during which the critical resources may be in a degraded mode, and repair time for critical resources. Ensure that the critical resources are available during these time intervals. For example, ensure that input queues are large enough to buffer anticipated messages if a server fails so that the messages are not permanently lost.

Interoperability

teroperability

ts of interoperability:
nsumer of a service must discover the
and the interface of the service.
response:
o the requester with response.
onse on to another system.
response to any interested parties.

which two or more
ful information via

nteroperability)
(semantic

om, with what, and

revor heard that

Tactics for Interop

- Locate:
  - Discovery service: locate a servi
    known directory service.
    - multiple levels of indirection
- Manage interfaces:
  - Orchestrate: uses a control mec
    and manage and sequence the
    services.
  - Tailor interface: adds or remove
    interface.

Checklist fo

Interoperability

- **Interoperability** is about the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.
  - Ability to exchange data (syntactic interoperability)
  - Ability to correctly interpret the data (semantic interoperability)
- Interoperability needs to identify with whom, with what, and under what circumstances (the context).

- Interface:

> "Charlene said that Kim told her that Trevor heard that Heather wants to come to your party."

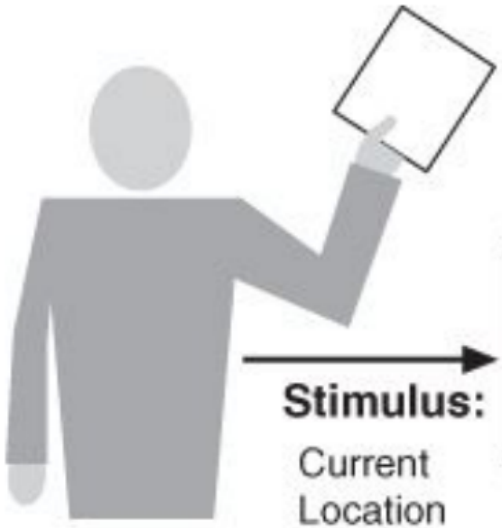
Interoperability

- Two important aspects of interoperability:
  - **Discovery**: the consumer of a service must discover the location, identity, and the interface of the service.
  - **Handling** of the response:
    - reports back to the requester with response.
    - sends its response on to another system.
    - broadcasts its response to any interested parties.

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 Sample Scenario



Source of Stimulus:
Our Vehicle Information System

Stimulus:
Current Location Sent

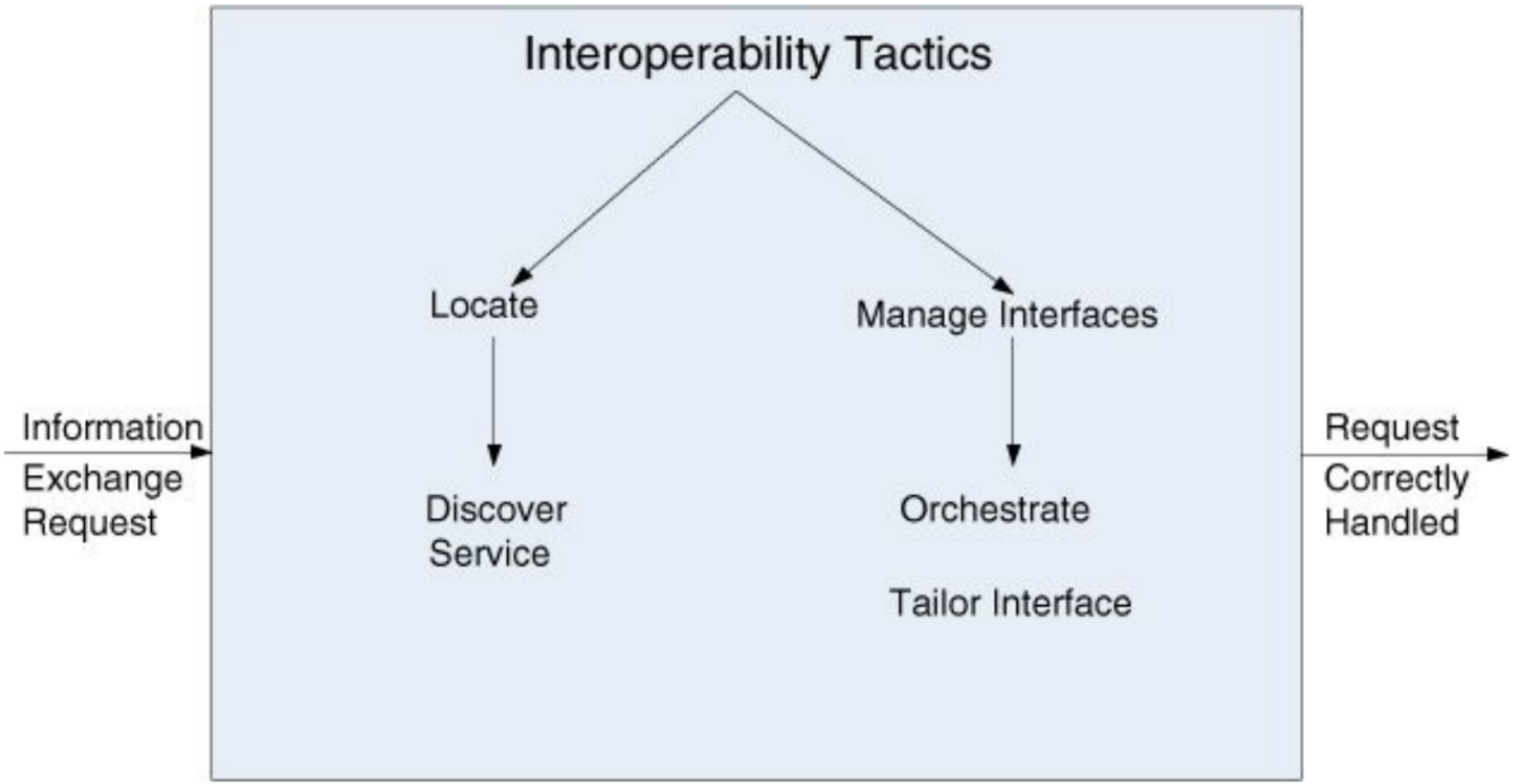
Artifact:
Traffic Monitoring System

Environment:
Systems known prior to run-time

Response:
Traffic Monitor Combines Current Location with Other Information, Overlays on Google Maps, and Broadcasts

Response Measure:
Our Information Included Correctly 99.9% of the Time

Interoperability Tactics



Tactics for Interoperability

- Locate:
  - Discovery service: locate a service through searching a known directory service.
    - multiple levels of indirection
- Manage interfaces:
  - Orchestrate: uses a control mechanism to coordinate and manage and sequence the invocation of particular services.
  - Tailor interface: adds or removes capabilities to an interface.

Checklist for Interoperability Design & Analysis

Category	Checklist
Allocation of Responsibilities	Determine which of your system responsibilities will need to interoperate with other systems. Ensure that responsibilities have been allocated to detect a request to interoperate with known or unknown external systems. Ensure that responsibilities have been allocated to carry out the following tasks: ▪ Accept the request ▪ Exchange information ▪ Reject the request ▪ Notify appropriate entities (people or systems) ▪ Log the request (for interoperability in an untrusted environment, logging for nonrepudiation is essential)
Coordination Model	Ensure that the coordination mechanisms can meet the critical quality attribute requirements. Considerations for performance include the following: ▪ Volume of traffic on the network both created by the systems under your control and generated by systems not under your control ▪ Timeliness of the messages being sent by your systems ▪ Currency of the messages being sent by your systems ▪ Jitter of the messages' arrival times ▪ Ensure that all of the systems under your control make assumptions about protocols and underlying networks that are consistent with the systems not under your control.
Data Model	Determine the syntax and semantics of the major data abstractions that may be exchanged among interoperating systems. Ensure that these major data abstractions are consistent with data from the interoperating systems. (If your system's data model is confidential and must not be made public, you may have to apply transformations to and from the data abstractions of systems with which yours interoperates.)
Mapping among Architectural Elements	For interoperability, the critical mapping is that of components to processors. Beyond the necessity of making sure that components that communicate externally are hosted on processors that can reach the network, the primary considerations deal with meeting the security, availability, and performance requirements for the communication. These will be dealt with in their respective chapters.
Resource Management	Ensure that interoperation with another system (accepting a request and/or rejecting a request) can never exhaust critical system resources (e.g., can a flood of such requests cause service to be denied to legitimate users?). Ensure that the resource load imposed by the communication requirements of interoperation is acceptable. Ensure that if interoperation requires that resources be shared among the participating systems, an adequate arbitration policy is in place.
Binding Time	Determine the systems that may interoperate, and when they become known to each other. For each system over which you have control: ▪ Ensure that it has a policy for dealing with binding to both known and unknown external systems. ▪ Ensure that it has mechanisms in place to reject unacceptable bindings and to log such requests. ▪ In the case of late binding, ensure that mechanisms will support the discovery of relevant new services or protocols, or the sending of information using chosen protocols.
Choice of Technology	For any of your chosen technologies, are they "visible" at the interface boundary of a system? If so, what interoperability effects do they have? Do they support, undercut, or have no effect on the interoperability scenarios that apply to your system? Ensure the effects they have are acceptable. Consider technologies that are designed to support interoperability, such as web services. Can they be used to satisfy the interoperability requirements for the systems under your control?

Category	Checklist
Allocation of Responsibilities	Determine which of your system responsibilities will need to interoperate with other systems. Ensure that responsibilities have been allocated to detect a request to interoperate with known or unknown external systems. Ensure that responsibilities have been allocated to carry out the following tasks: ▪ Accept the request ▪ Exchange information ▪ Reject the request ▪ Notify appropriate entities (people or systems) ▪ Log the request (for interoperability in an untrusted environment, logging for nonrepudiation is essential)
Coordination Model	Ensure that the coordination mechanisms can meet the critical quality attribute requirements. Considerations for performance include the following: ▪ Volume of traffic on the network both created by the systems under your control and generated by systems not under your control ▪ Timeliness of the messages being sent by your systems ▪ Currency of the messages being sent by your systems ▪ Jitter of the messages' arrival times ▪ Ensure that all of the systems under your control make assumptions about protocols and underlying networks that are consistent with the systems not under your control.
Data Model	Determine the syntax and semantics of the major data abstractions that may be exchanged among interoperating systems. Ensure that these major data abstractions are consistent with data from the interoperating systems. (If your system's data model is confidential and must not be made public, you may have to apply transformations to and from the data abstractions of systems with which yours interoperates.)
Mapping among	For interoperability, the critical mapping is that of components

	of systems with which yours interoperates.)
Mapping among Architectural Elements	For interoperability, the critical mapping is that of components to processors. Beyond the necessity of making sure that components that communicate externally are hosted on processors that can reach the network, the primary considerations deal with meeting the security, availability, and performance requirements for the communication. These will be dealt with in their respective chapters.
Resource Management	Ensure that interoperation with another system (accepting a request and/or rejecting a request) can never exhaust critical system resources (e.g., can a flood of such requests cause service to be denied to legitimate users?). Ensure that the resource load imposed by the communication requirements of interoperation is acceptable. Ensure that if interoperation requires that resources be shared among the participating systems, an adequate arbitration policy is in place.
Binding Time	Determine the systems that may interoperate, and when they become known to each other. For each system over which you have control: ▪ Ensure that it has a policy for dealing with binding to both known and unknown external systems. ▪ Ensure that it has mechanisms in place to reject unacceptable bindings and to log such requests. ▪ In the case of late binding, ensure that mechanisms will support the discovery of relevant new services or protocols, or the sending of information using chosen protocols.
Choice of Technology	For any of your chosen technologies, are they "visible" at the interface boundary of a system? If so, what interoperability effects do they have? Do they support, undercut, or have no effect on the interoperability scenarios that apply to your system? Ensure the effects they have are acceptable. Consider technologies that are designed to support interoperability, such as web services. Can they be used to satisfy the interoperability requirements for the systems under your control?

Modifiability

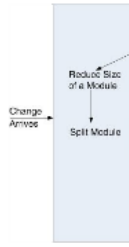
quality attributes
- New defects introduced

Modifi

Source:
Developer

Stimulu
Wishes
to Chan
the UI

M



Tactics for Modifiability

- **Split module**: If the module being modified includes a great deal of capabilities, the modification costs will likely be high.

Modifiability

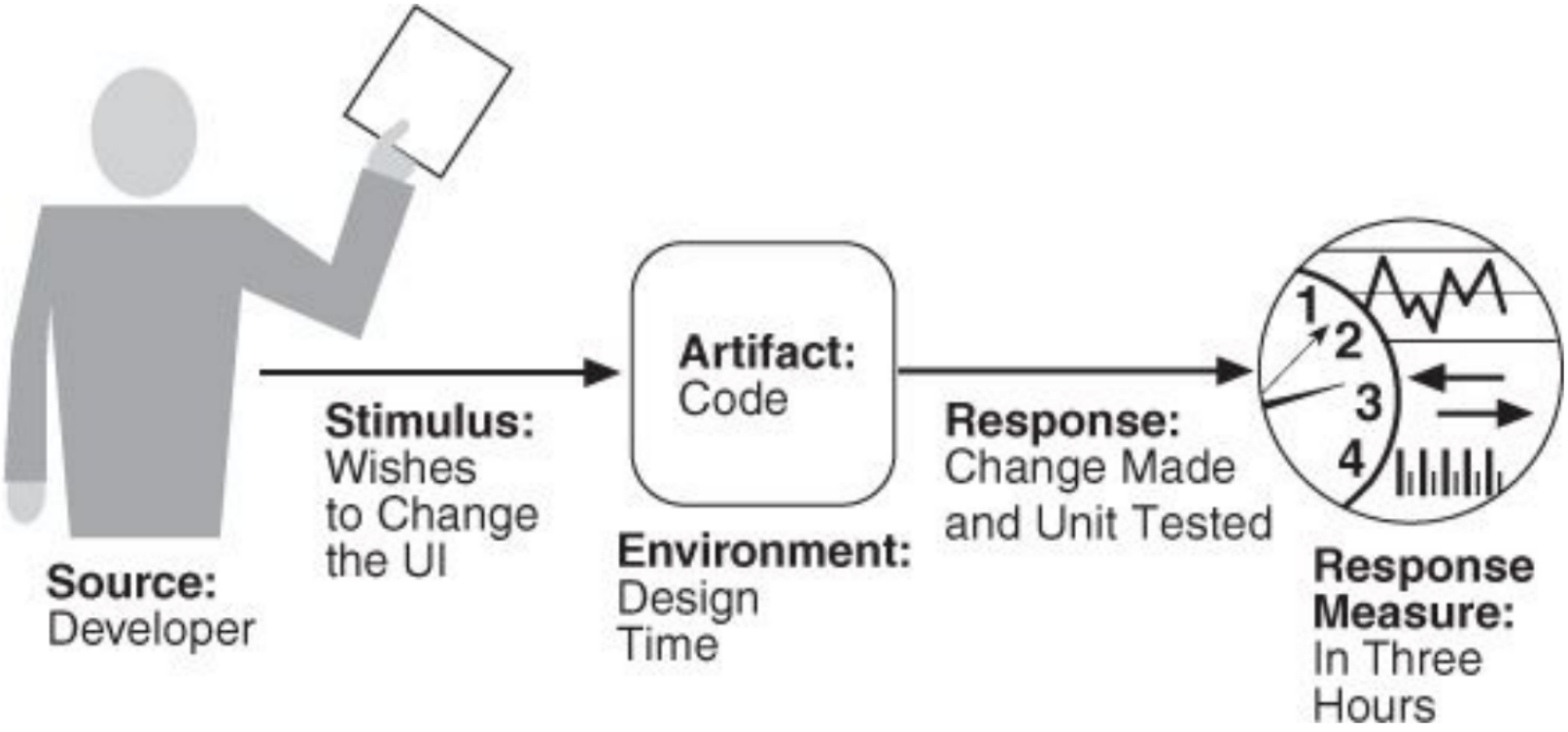
- Modifiability deal with change and the cost in time or money of making a change, including the extent to which this modifiability affects other functions or quality attributes.
- There is a cost of prepraring for change as well as a cost of making a change.
- Four questions to plan for modifiability
  - 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?
- If fewer changes than expected come in, then an expensive modification mechanism may not be warranted.

$N \times$ Cost of making the change without the mechanism $\leq$ Cost of installing the mechanism + ($N \times$ Cost of making the change using the mechanism).

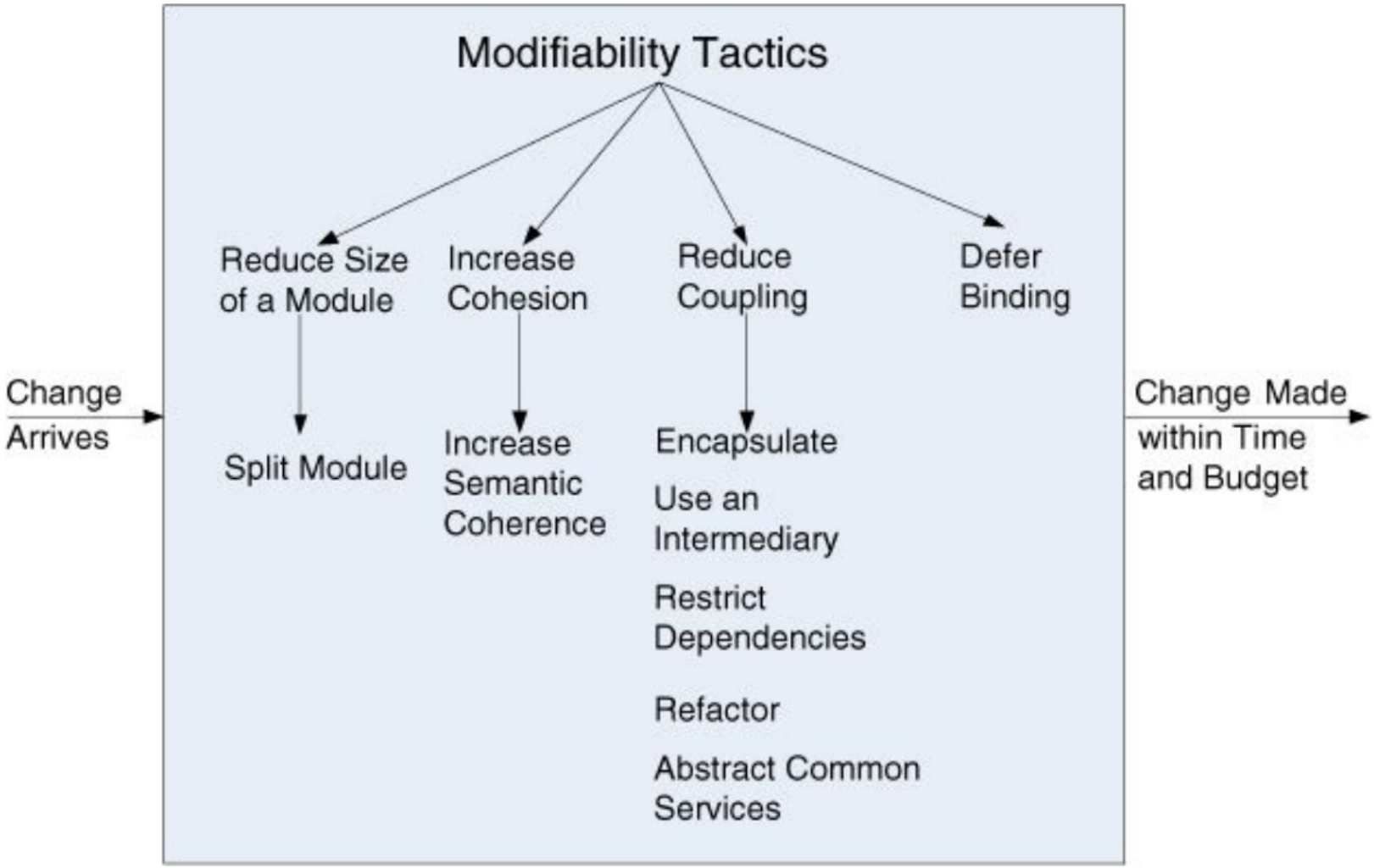
Modifiability General Scenario

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 Sample Scenario



Modifiability Tactics



Tactics for Modifiability

- **Split module**: If the module being modified includes a great deal of capabilities, the modification costs will likely be high.
- **Increase semantic coherence**: If the responsibilities A and B in a module do not serve the same purpose, they should be placed in different modules by creating a new module or moving a responsibility to an existing module.
- **Encapsulation** introduces an explicit interface to a module, and reduces the probability that a change to one module propagates to other modules.
- Use an **intermediary** breaks a dependency.
- **Refactor** when two modules are affected by the same change.
- **Defer binding**: Binds the value of some parameters at a different phase in the life cycle than the one in which they are initially defined.

Checklist for Modifiability Design and Analysis

Category	Checklist
Allocation of Responsibilities	Determine which changes or categories of changes are likely to occur through consideration of changes in technical, legal, social, business, and customer forces. For each potential change or category of changes: ▪ Determine the responsibilities that would need to be added, modified, or deleted to make the change. ▪ Determine what responsibilities are impacted by the change. ▪ Determine an allocation of responsibilities to modules that places, as much as possible, responsibilities that will be changed (or impacted by the change) together in the same module, and places responsibilities that will be changed at different times in separate modules.
Coordination Model	Determine which functionality or quality attribute can change at runtime and how this affects coordination; for example, will the information being communicated change at runtime, or will the communication protocol change at runtime? If so, ensure that such changes affect a small number set of modules. Determine which devices, protocols, and communication paths used for coordination are likely to change. For those devices, protocols, and communication paths, ensure that the impact of changes will be limited to a small set of modules. For those elements for which modifiability is a concern, use a coordination model that reduces coupling such as publish-subscribe, defers bindings such as enterprise service bus, or restricts dependencies such as broadcast.
Data Model	Determine which changes (or categories of changes) to the data abstractions, their operations, or their properties are likely to occur. Also determine which changes or categories of changes to these data abstractions will involve their creation, initialization, persistence, manipulation, translation, or destruction. For each change or category of change, determine if the changes will be made by an end user, a system administrator, or a developer. For those changes to be made by an end user or system administrator, ensure that the necessary attributes are visible to that user and that the user has the correct privileges to modify the data, its operations, or its properties. For each potential change or category of change: ▪ Determine which data abstractions would need to be added, modified, or deleted to make the change. ▪ Determine whether there would be any changes to the creation, initialization, persistence, manipulation, translation, or destruction of these data abstractions. ▪ Determine which other data abstractions are impacted by the change. For these additional data abstractions, determine whether the impact would be on the operations, their properties, their creation, initialization, persistence, manipulation, translation, or destruction. ▪ Ensure an allocation of data abstractions that minimizes the number and severity of modifications to the abstractions by the potential changes. Design your data model so that items allocated to each element of the data model are likely to change together.
Mapping among Architectural Elements	Determine if it is desirable to change the way in which functionality is mapped to computational elements (e.g., processes, threads, processors) at runtime, compile time, design time, or build time. Determine the extent of modifications necessary to accommodate the addition, deletion, or modification of a function or a quality attribute. This might involve a determination of the following, for example: ▪ Execution dependencies ▪ Assignment of data to databases ▪ Assignment of runtime elements to processes, threads, or processors Ensure that such changes are performed with mechanisms that utilize deferred binding of mapping decisions.
Resource Management	Determine how the addition, deletion, or modification of a responsibility or quality attribute will affect resource usage. This involves, for example: ▪ Determining what changes might introduce new resources or remove old ones or affect existing resource usage ▪ Determining what resource limits will change and how Ensure that the resources after the modification are sufficient to meet the system requirements. Encapsulate all resource managers and ensure that the policies implemented by those resource managers are themselves encapsulated and bindings are deferred to the extent possible.
Binding Time	For each change or category of change: ▪ Determine the latest time at which the change will need to be made. ▪ Choose a defer-binding mechanism (see Section 7.2) that delivers the appropriate capability at the time chosen. ▪ Determine the cost of introducing the mechanism and the cost of making changes using the chosen mechanism. Use the equation on page 118 to assess your choice of mechanism. ▪ Do not introduce so many binding choices that change is impeded because the dependencies among the choices are complex and unknown.
Choice of Technology	Determine what modifications are made easier or harder by your technology choices. ▪ Will your technology choices help to make, test, and deploy modifications? ▪ How easy is it to modify your choice of technologies (in case some of these technologies change or become obsolete)? Choose your technologies to support the most likely modifications. For example, an enterprise service bus makes it easier to change how elements are connected but may introduce vendor lock-in.

Category	Checklist
Allocation of Responsibilities	Determine which changes or categories of changes are likely to occur through consideration of changes in technical, legal, social, business, and customer forces. For each potential change or category of changes: ▪ Determine the responsibilities that would need to be added, modified, or deleted to make the change. ▪ Determine what responsibilities are impacted by the change. ▪ Determine an allocation of responsibilities to modules that places, as much as possible, responsibilities that will be changed (or impacted by the change) together in the same module, and places responsibilities that will be changed at different times in separate modules.
Coordination Model	Determine which functionality or quality attribute can change at runtime and how this affects coordination; for example, will the information being communicated change at runtime, or will the communication protocol change at runtime? If so, ensure that such changes affect a small number set of modules. Determine which devices, protocols, and communication paths used for coordination are likely to change. For those devices, protocols, and communication paths, ensure that the impact of changes will be limited to a small set of modules. For those elements for which modifiability is a concern, use a coordination model that reduces coupling such as publish-subscribe, defers bindings such as enterprise service bus, or restricts dependencies such as broadcast.
Data Model	Determine which changes (or categories of changes) to the data abstractions, their operations, or their properties are likely to occur. Also determine which changes or categories of changes to these data abstractions will involve their creation, initialization, persistence, manipulation, translation, or destruction. For each change or category of change, determine if the changes will be made by an end user, a system administrator, or a developer. For those changes to be made by an end user or system administrator, ensure that the necessary attributes are visible to that user and that the user has the correct privileges to modify the data, its operations, or its properties. For each potential change or category of change: ▪ Determine which data abstractions would need to be added, modified, or deleted to make the change. ▪ Determine whether there would be any changes to the creation, initialization, persistence, manipulation, translation, or destruction of these data abstractions. ▪ Determine which other data abstractions are impacted by the change. For these additional data abstractions, determine whether the impact would be on the operations, their properties, their creation, initialization, persistence, manipulation, translation, or destruction. ▪ Ensure an allocation of data abstractions that minimizes the

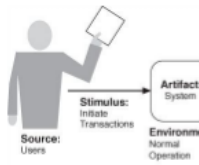
	manipulation, translation, or destruction. ▪ Ensure an allocation of data abstractions that minimizes the number and severity of modifications to the abstractions by the potential changes. Design your data model so that items allocated to each element of the data model are likely to change together.
Mapping among Architectural Elements	Determine if it is desirable to change the way in which functionality is mapped to computational elements (e.g., processes, threads, processors) at runtime, compile time, design time, or build time. Determine the extent of modifications necessary to accommodate the addition, deletion, or modification of a function or a quality attribute. This might involve a determination of the following, for example: ▪ Execution dependencies ▪ Assignment of data to databases ▪ Assignment of runtime elements to processes, threads, or processors Ensure that such changes are performed with mechanisms that utilize deferred binding of mapping decisions.
Resource Management	Determine how the addition, deletion, or modification of a responsibility or quality attribute will affect resource usage. This involves, for example: ▪ Determining what changes might introduce new resources or remove old ones or affect existing resource usage ▪ Determining what resource limits will change and how Ensure that the resources after the modification are sufficient to meet the system requirements. Encapsulate all resource managers and ensure that the policies implemented by those resource managers are themselves encapsulated and bindings are deferred to the extent possible.
Binding Time	For each change or category of change: ▪ Determine the latest time at which the change will need to be made. ▪ Choose a defer-binding mechanism (see Section 7.2) that delivers the appropriate capability at the time chosen. ▪ Determine the cost of introducing the mechanism and the cost of making changes using the chosen mechanism. Use the equation on page 118 to assess your choice of mechanism. ▪ Do not introduce so many binding choices that change is impeded because the dependencies among the choices are complex and unknown.
Choice of Technology	Determine what modifications are made easier or harder by your technology choices. ▪ Will your technology choices help to make, test, and deploy modifications? ▪ How easy is it to modify your choice of technologies (in case some of these technologies change or become obsolete)? Choose your technologies to support the most likely modifications. For example, an enterprise service bus makes it easier to change how elements are connected but may introduce vendor lock-in.

Performance

Perfo

Portion of Scenario
Source
Stimulus
Artifact
Environment
Response
Response Measure

Performance Sc



Checklist for Performance Design and Analysis

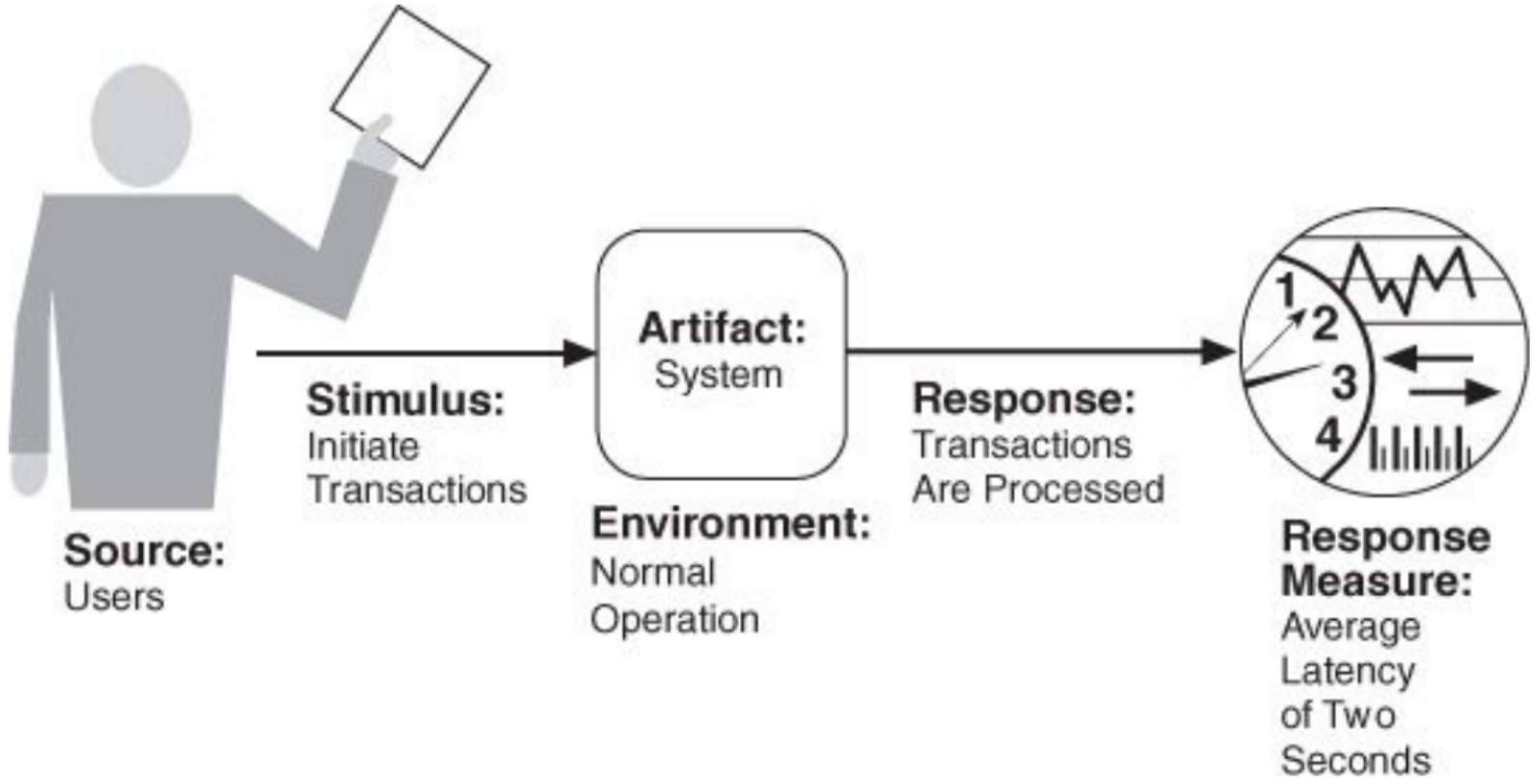
Performance

- **Performance** is about time and the software system's ability to meet timing requiements.
- All systems have performance requirements, even if they are not explicitly expressed.
- Two basic contributors to the response time:
  - **processing time** (when the system is working to response)
  - **blocked time** (when the system is unable to response)

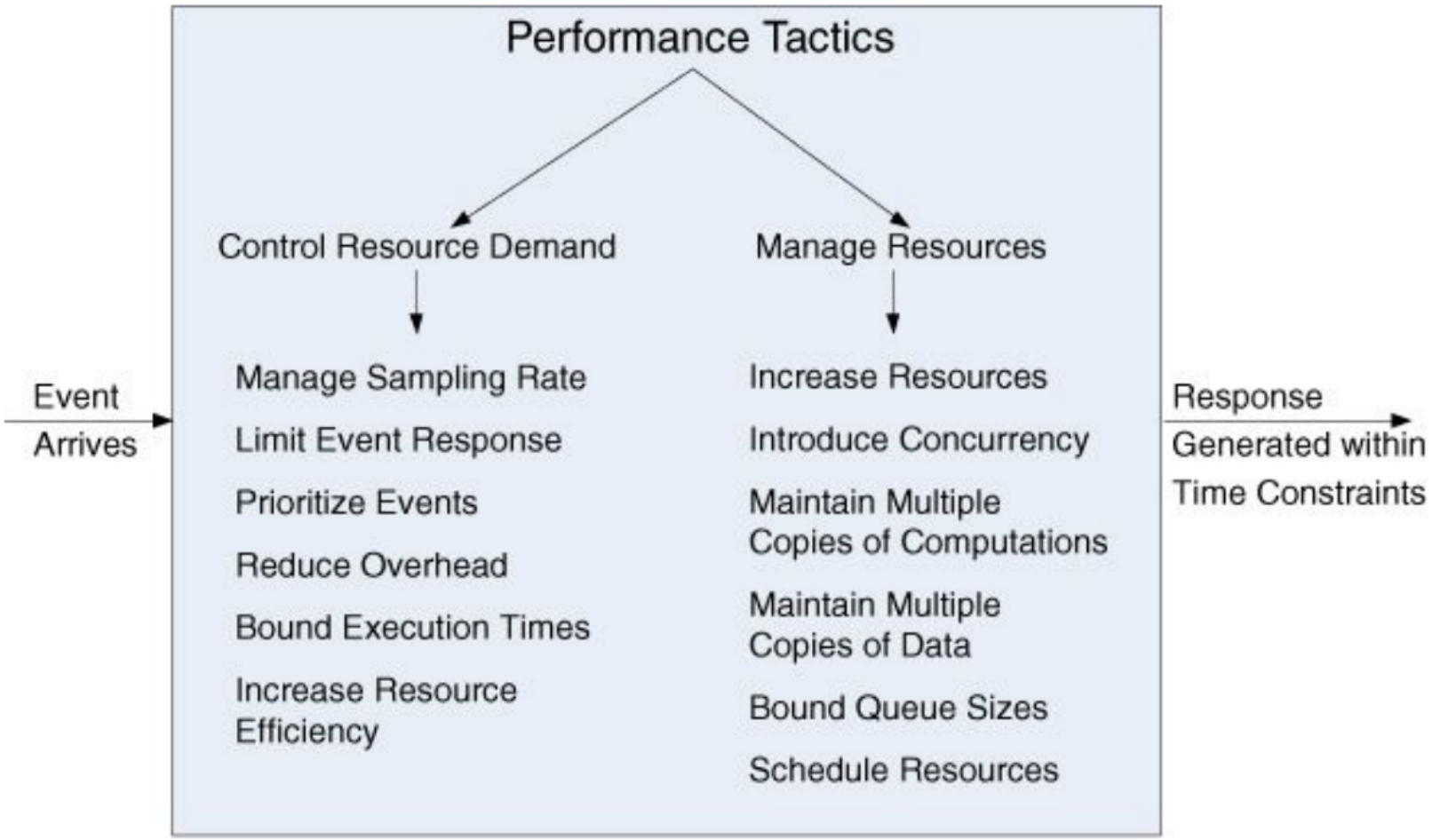
Performance General Scenario

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 Sample Scenario



Tactics for Performance



Tactics for Performance

- On the **demand** side
  - Manage sampling rate (reducing sampling frequency)
  - Limit event response: When discrete events arrive at the system too rapidly to be processed, the events must be queued until they can be processed.
  - Prioritize events if not all events are equally important.
  - Reduce overhead by using intermediaries to increase the resources in processing an event stream
- On the **resource** side
  - Increase resources (faster processor, additional memory, faster network...)
  - Introduce concurrency if requests can be processed in parallel.
  - Maintain multiple copies of computations: Use load balancer to assign new work to one of the available duplicate servers.
  - Maintain multiple copies of data:
    - caching
    - data replication

Checklist for Performance Design and Analysis

Category	Checklist
Allocation of Responsibilities	Determine the system's responsibilities that will involve heavy loading, have time-critical response requirements, are heavily used, or impact portions of the system where heavy loads or time-critical events occur. For those responsibilities, identify the processing requirements of each responsibility, and determine whether they may cause bottlenecks. Also, identify additional responsibilities to recognize and process requests appropriately, including ▪ Responsibilities that result from a thread of control crossing process or processor boundaries ▪ Responsibilities to manage the threads of control—allocation and deallocation of threads, maintaining thread pools, and so forth ▪ Responsibilities for scheduling shared resources or managing performance-related artifacts such as queues, buffers, and caches For the responsibilities and resources you identified, ensure that the required performance response can be met (perhaps by building a performance model to help in the evaluation).
Coordination Model	Determine the elements of the system that must coordinate with each other—directly or indirectly—and choose communication and coordination mechanisms that do the following: ▪ Support any introduced concurrency (for example, is it thread safe?), event prioritization, or scheduling strategy ▪ Ensure that the required performance response can be delivered ▪ Can capture periodic, stochastic, or sporadic event arrivals, as needed ▪ Have the appropriate properties of the communication mechanisms; for example, stateful, stateless, synchronous, asynchronous, guaranteed delivery, throughput, or latency
Data Model	Determine those portions of the data model that will be heavily loaded, have time-critical response requirements, are heavily used, or impact portions of the system where heavy loads or time-critical events occur. For those data abstractions, determine the following: ▪ Whether maintaining multiple copies of key data would benefit performance ▪ Whether partitioning data would benefit performance ▪ Whether reducing the processing requirements for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is possible ▪ Whether adding resources to reduce bottlenecks for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is feasible
Mapping among Architectural Elements	Where heavy network loading will occur, determine whether co-locating some components will reduce loading and improve overall efficiency. Ensure that components with heavy computation requirements are assigned to processors with the most processing capacity. Determine where introducing concurrency (that is, allocating a piece of functionality to two or more copies of a component running simultaneously) is feasible and has a significant positive effect on performance. Determine whether the choice of threads of control and their associated responsibilities introduces bottlenecks.
Resource Management	Determine which resources in your system are critical for performance. For these resources, ensure that they will be monitored and managed under normal and overloaded system operation. For example: ▪ System elements that need to be aware of, and manage, time and other performance-critical resources ▪ Process/thread models ▪ Prioritization of resources and access to resources ▪ Scheduling and locking strategies ▪ Deploying additional resources on demand to meet increased loads
Binding Time	For each element that will be bound after compile time, determine the following: ▪ Time necessary to complete the binding ▪ Additional overhead introduced by using the late binding mechanism Ensure that these values do not pose unacceptable performance penalties on the system.
Choice of Technology	Will your choice of technology let you set and meet hard, real-time deadlines? Do you know its characteristics under load and its limits? Does your choice of technology give you the ability to set the following: ▪ Scheduling policy ▪ Priorities ▪ Policies for reducing demand ▪ Allocation of portions of the technology to processors ▪ Other performance-related parameters Does your choice of technology introduce excessive overhead for heavily used operations?

Category	Checklist
Allocation of Responsibilities	Determine the system's responsibilities that will involve heavy loading, have time-critical response requirements, are heavily used, or impact portions of the system where heavy loads or time-critical events occur. For those responsibilities, identify the processing requirements of each responsibility, and determine whether they may cause bottlenecks. Also, identify additional responsibilities to recognize and process requests appropriately, including ▪ Responsibilities that result from a thread of control crossing process or processor boundaries ▪ Responsibilities to manage the threads of control—allocation and deallocation of threads, maintaining thread pools, and so forth ▪ Responsibilities for scheduling shared resources or managing performance-related artifacts such as queues, buffers, and caches For the responsibilities and resources you identified, ensure that the required performance response can be met (perhaps by building a performance model to help in the evaluation).
Coordination Model	Determine the elements of the system that must coordinate with each other—directly or indirectly—and choose communication and coordination mechanisms that do the following: ▪ Support any introduced concurrency (for example, is it thread safe?), event prioritization, or scheduling strategy ▪ Ensure that the required performance response can be delivered ▪ Can capture periodic, stochastic, or sporadic event arrivals, as needed ▪ Have the appropriate properties of the communication mechanisms; for example, stateful, stateless, synchronous, asynchronous, guaranteed delivery, throughput, or latency
Data Model	Determine those portions of the data model that will be heavily loaded, have time-critical response requirements, are heavily used, or impact portions of the system where heavy loads or time-critical events occur. For those data abstractions, determine the following: ▪ Whether maintaining multiple copies of key data would benefit performance ▪ Whether partitioning data would benefit performance ▪ Whether reducing the processing requirements for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is possible

	or destruction of the enumerated data abstractions is possible ▪ Whether adding resources to reduce bottlenecks for the creation, initialization, persistence, manipulation, translation, or destruction of the enumerated data abstractions is feasible
Mapping among Architectural Elements	Where heavy network loading will occur, determine whether co-locating some components will reduce loading and improve overall efficiency. Ensure that components with heavy computation requirements are assigned to processors with the most processing capacity. Determine where introducing concurrency (that is, allocating a piece of functionality to two or more copies of a component running simultaneously) is feasible and has a significant positive effect on performance. Determine whether the choice of threads of control and their associated responsibilities introduces bottlenecks.
Resource Management	Determine which resources in your system are critical for performance. For these resources, ensure that they will be monitored and managed under normal and overloaded system operation. For example: ▪ System elements that need to be aware of, and manage, time and other performance-critical resources ▪ Process/thread models ▪ Prioritization of resources and access to resources ▪ Scheduling and locking strategies ▪ Deploying additional resources on demand to meet increased loads
Binding Time	For each element that will be bound after compile time, determine the following: ▪ Time necessary to complete the binding ▪ Additional overhead introduced by using the late binding mechanism Ensure that these values do not pose unacceptable performance penalties on the system.
Choice of Technology	Will your choice of technology let you set and meet hard, real-time deadlines? Do you know its characteristics under load and its limits? Does your choice of technology give you the ability to set the following: ▪ Scheduling policy ▪ Priorities ▪ Policies for reducing demand ▪ Allocation of portions of the technology to processors ▪ Other performance-related parameters Does your choice of technology introduce excessive overhead for heavily used operations?

Security

nalysis

- **Security** measu
  information fron
  access to people
- Three characteri
  - **Confidentia**
    unauthorize
  - **Integrity**: D
    unauthorize
  - **Availability**:
    use.

for Security

aring network traffic or service
system to a set of signatures or

sing checksums or hash values.
any external input to the system.
or what they purport to be.
e the rights to access and modify

g resouces.
zing the attack surface of a system.

resources when an attack is

Security General Sc

Portion of Scenario	Possible Values
Source	Human or another system which may have b identified (either correctly or incorrectly), or m Unknown. A human attacker may be from ou from inside the organization.
Stimulus	Unauthorized attempt is made to display data data, access system services, change the sy reduce availability.
Artifact	System services, data within the system, a c of the system, data produced or consumed b
Environment	The system is either online or offline; either o classified nested from a network, either behind a network; fully connected, behind a firewall or
Response	Transactions are carried out in a fashion such • Data or services are protected from unau • Data or services are not being misused. • Parties to a transaction are identified with • The parties to the transaction cannot repu transaction. • The data, resources, and system services legitimate use. The system tracks activities within it by • Recording access or modification. • Recording attempts to access data, resou • Notifying appropriate entities (people or s apparent attack is occurring.
Response Measure	One or more of the following: • How much of a system is compromised w component or data value is compromised • How much time passed before an attack • How many actions were taken. • How long does it take to recover from a su

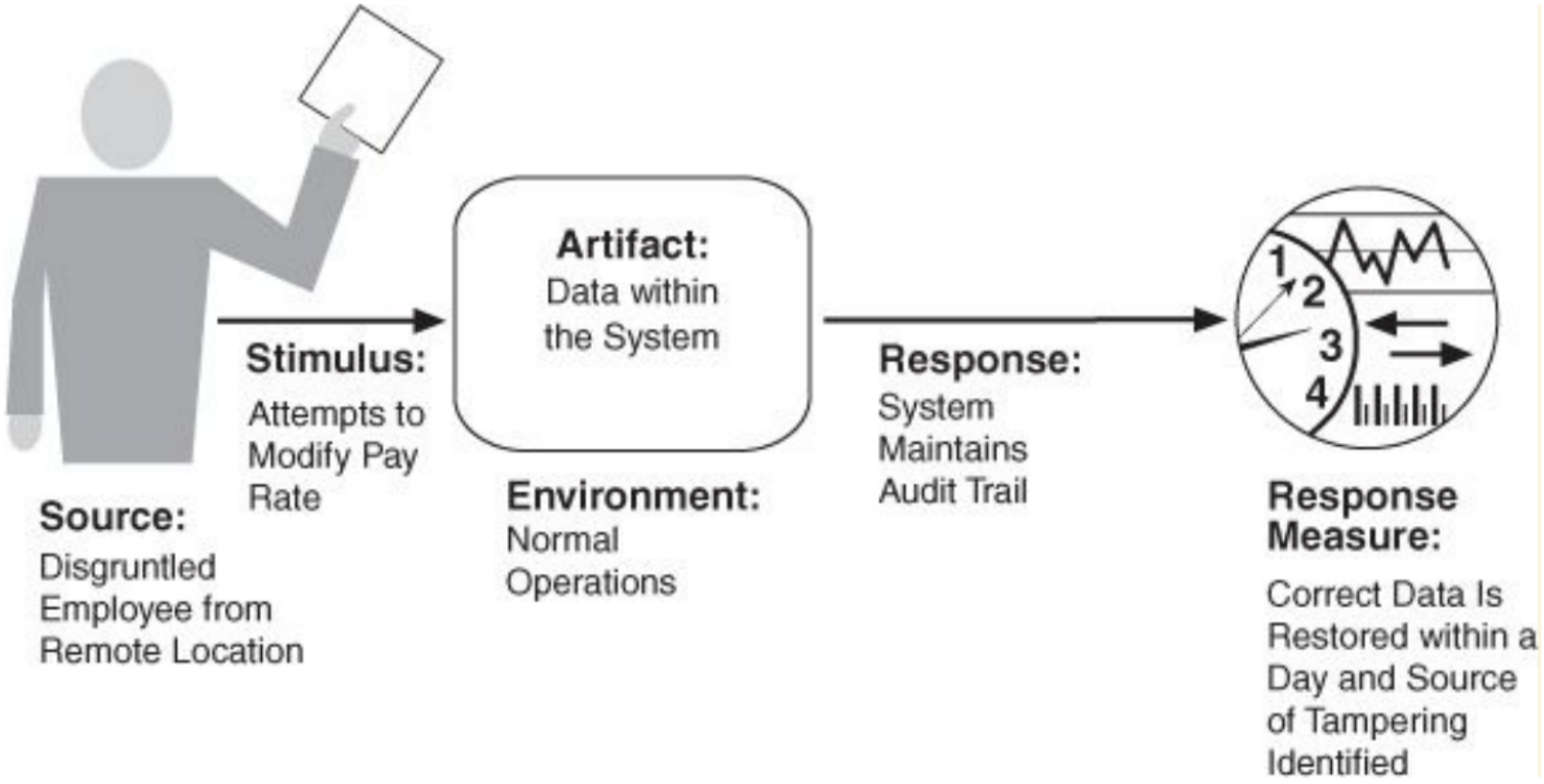
Security

- **Security** measures system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.
- Three characteristics of security: (CIA)
  - **Confidentiality**: Data and services are pretected from unauthorized access.
  - **Integrity**: Data and services are not subject to unauthorized manipulation.
  - **Availability**: The system will be available for legitimate use.

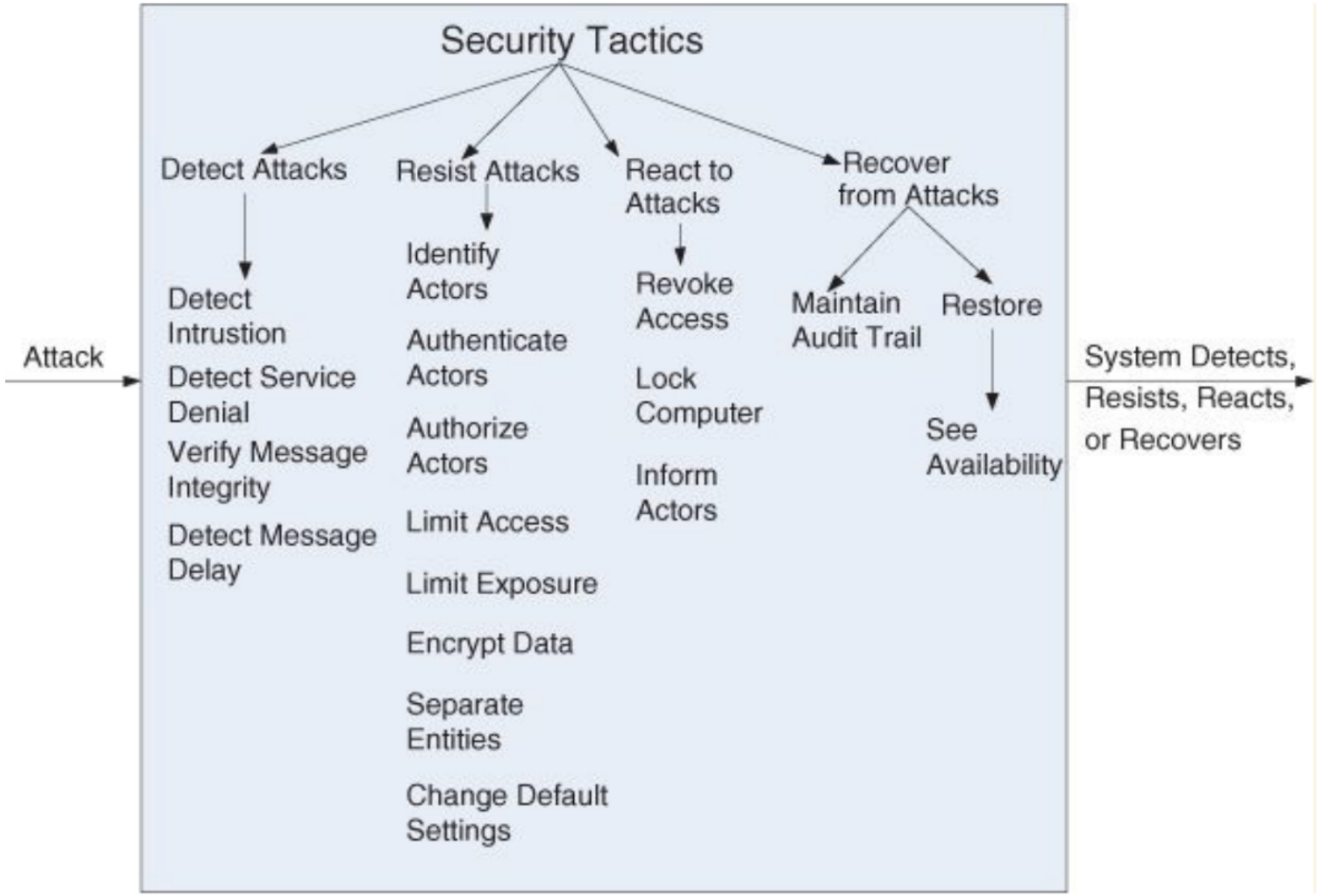
Security General Scenario

Portion of Scenario	Possible Values
Source	Human or another system which may have been previously identified (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.
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.
Response	Transactions are carried out in a 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. 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
Response Measure	One or more of the following: ▪ How much of a 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 Sample Scenario



Security Tactics



Tactics for Security

- Detect intrusion by comparing network traffic or service request patterns within a system to a set of signatures or known patterns.
- Detect service denial.
- Verify message integrity using checksums or hash values.
- Identify actors - source of any external input to the system.
- Authenticate actors who or what they purport to be.
- Authorize actors who have the rights to access and modify either data or services.
- Limit access to computing resouces.
- Limit exposure by minimizing the attack surface of a system.
- Encrypt data.
- Revoke access to sensitive resources when an attack is underway.

Checklist for Security Design and Analysis

Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities need to be secure. For each of these responsibilities, ensure that additional responsibilities have been allocated to do the following: ▪ Identify the actor ▪ Authenticate the actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services and inform appropriate personnel and restrict access ▪ Recover from an attack ▪ Verify checksums and hash values
Coordination Model	Determine mechanisms required to communicate and coordinate with other systems or individuals. For these communications, ensure that mechanisms for authenticating and authorizing the actor or system, and encrypting data for transmission across the connection, are in place. Ensure also that mechanisms exist for monitoring and recognizing unexpectedly high demands for resources or services as well as mechanisms for restricting or terminating the connection.
Data Model	Determine the sensitivity of different data fields. For each data abstraction: ▪ Ensure that data of different sensitivity is separated. ▪ Ensure that data of different sensitivity has different access rights and that access rights are checked prior to access. ▪ Ensure that access to sensitive data is logged and that the log file is suitably protected. ▪ Ensure that data is suitably encrypted and that keys are separated from the encrypted data. ▪ Ensure that data can be restored if it is inappropriately modified.
Mapping among Architectural Elements	Determine how alternative mappings of architectural elements that are under consideration may change how an individual or system may read, write, or modify data; access system services or resources; or reduce availability to system services or resources. Determine how alternative mappings may affect the recording of access to data, services or resources and the recognition of unexpectedly high demands for resources. For each such mapping, ensure that there are responsibilities to do the following: ▪ Identify an actor ▪ Authenticate an actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services, inform appropriate personnel, and restrict access ▪ Recover from an attack
Resource Management	Determine the system resources required to identify and monitor a system or an individual who is internal or external, authorized or not authorized, with access to specific resources or all resources. Determine the resources required to authenticate the actor, grant or deny access to data or resources, notify appropriate entities (people or systems), record attempts to access data or resources, encrypt data, recognize inexplicably high demand for resources, inform users or systems, and restrict access. For these resources consider whether an external entity can access a critical resource or exhaust a critical resource; how to monitor the resource; how to manage resource utilization; how to log resource utilization; and ensure that there are sufficient resources to perform the necessary security operations. Ensure that a contaminated element can be prevented from contaminating other elements. Ensure that shared resources are not used for passing sensitive data from an actor with access rights to that data to an actor without access rights to that data.
Binding Time	Determine cases where an instance of a late-bound component may be untrusted. For such cases ensure that late-bound components can be qualified; that is, if ownership certificates for late-bound components are required, there are appropriate mechanisms to manage and validate them; that access to late-bound data and services can be managed; that access by late-bound components to data and services can be blocked; that mechanisms to record the access, modification, and attempts to access data or services by late-bound components are in place; and that system data is encrypted where the keys are intentionally withheld for late-bound components
Choice of Technology	Determine what technologies are available to help user authentication, data access rights, resource protection, and data encryption. Ensure that your chosen technologies support the tactics relevant for your security needs.

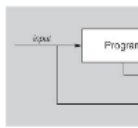
Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities need to be secure. For each of these responsibilities, ensure that additional responsibilities have been allocated to do the following: ▪ Identify the actor ▪ Authenticate the actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services and inform appropriate personnel and restrict access ▪ Recover from an attack ▪ Verify checksums and hash values
Coordination Model	Determine mechanisms required to communicate and coordinate with other systems or individuals. For these communications, ensure that mechanisms for authenticating and authorizing the actor or system, and encrypting data for transmission across the connection, are in place. Ensure also that mechanisms exist for monitoring and recognizing unexpectedly high demands for resources or services as well as mechanisms for restricting or terminating the connection.
Data Model	Determine the sensitivity of different data fields. For each data abstraction: ▪ Ensure that data of different sensitivity is separated. ▪ Ensure that data of different sensitivity has different access rights and that access rights are checked prior to access. ▪ Ensure that access to sensitive data is logged and that the log file is suitably protected. ▪ Ensure that data is suitably encrypted and that keys are separated from the encrypted data. ▪ Ensure that data can be restored if it is inappropriately modified.
Mapping among Architectural Elements	Determine how alternative mappings of architectural elements that are under consideration may change how an individual or system may read, write, or modify data; access system services or resources; or reduce availability to system services or resources. Determine how alternative mappings may affect the recording of access to data, services or resources and the recognition of unexpectedly high demands for resources. For each such mapping, ensure that there are responsibilities to do the following:

	▪ Identify an actor ▪ Authenticate an actor ▪ Authorize actors ▪ Grant or deny access to data or services ▪ Record attempts to access or modify data or services ▪ Encrypt data ▪ Recognize reduced availability for resources or services, inform appropriate personnel, and restrict access ▪ Recover from an attack
Resource Management	Determine the system resources required to identify and monitor a system or an individual who is internal or external, authorized or not authorized, with access to specific resources or all resources. Determine the resources required to authenticate the actor, grant or deny access to data or resources, notify appropriate entities (people or systems), record attempts to access data or resources, encrypt data, recognize inexplicably high demand for resources, inform users or systems, and restrict access. For these resources consider whether an external entity can access a critical resource or exhaust a critical resource; how to monitor the resource; how to manage resource utilization; how to log resource utilization; and ensure that there are sufficient resources to perform the necessary security operations. Ensure that a contaminated element can be prevented from contaminating other elements. Ensure that shared resources are not used for passing sensitive data from an actor with access rights to that data to an actor without access rights to that data.
Binding Time	Determine cases where an instance of a late-bound component may be untrusted. For such cases ensure that late-bound components can be qualified; that is, if ownership certificates for late-bound components are required, there are appropriate mechanisms to manage and validate them; that access to late-bound data and services can be managed; that access by late-bound components to data and services can be blocked; that mechanisms to record the access, modification, and attempts to access data or services by late-bound components are in place; and that system data is encrypted where the keys are intentionally withheld for late-bound components
Choice of Technology	Determine what technologies are available to help user authentication, data access rights, resource protection, and data encryption. Ensure that your chosen technologies support the tactics relevant for your security needs.

Testability

Analysis

- **Testability** refers t
  demonstrate its fa
  testing.
- For a system to be
  control each comp
  outputs.



ctics for Testability

ve system state: Maintain some sort of state
esters to assign a value to that state information,
nformation accessible to testers on demand.
erfaces allow you to control or capture values for

ck the state that caused a fault and re-create the

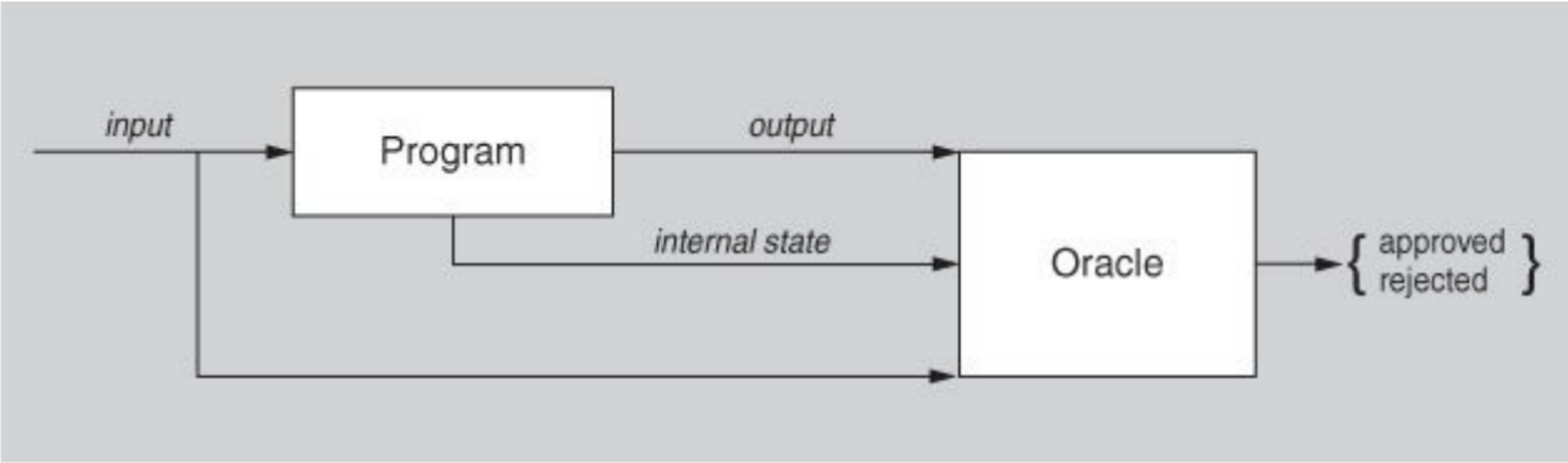
lates an instance of the system from the real
experimentation to undo its consequences.
Complex software is harder to test, because its
ce is very large and more difficult to re-create
large state space.
l complexity avoiding, reducing or resolving
etween components; isolating and
dependencies on external environment.
mber of classes from which a class is derived.
epths of the inheritance tree and the number of
a class.
norphism and dynamic calls.
minism - limiting **behavioral complexity**.

Testability General Sc

Portion of Scenario	Possible Values
Source	Unit testers, integration testers, sys testers, and users, either running t automated testing tools
Stimulus	A set of tests is executed due to th increment such as a class layer or integration of a subsystem, the com whole system, or the delivery of the
Environment	Design time, development time, co deployment time, run time
Artifacts	The portion of the system being tes
Response	One or more of the following: exec results, capture activity that result monitor the state of the system
Response Measure	One or more of the following: effort faults, effort to achieve a given per coverage, probability of fault being test, time to perform tests, effort to longest dependency chain in test, test environment, reduction in risk prob(loss))

Testability

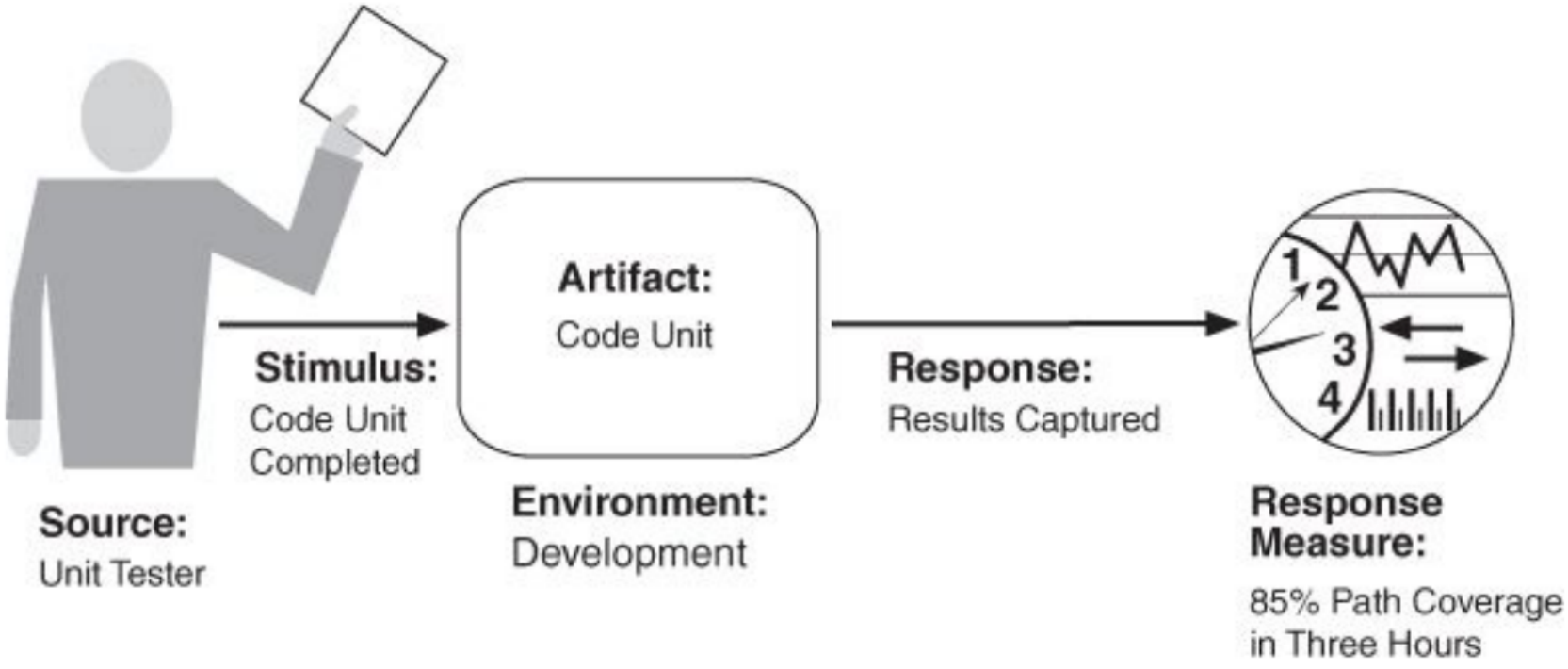
- **Testability** refers to ease with which software can be made to demonstrate its faults through (typically execution-based) testing.
- For a system to be properly testable, it must be possible to control each component's inputs and then to observe its outputs.



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 Sample Scenario



Testability Tactics

Tactics for Testability

- **Control and observe system state**: Maintain some sort of state information, allow testers to assign a value to that state information, and/or make that information accessible to testers on demand.
  - Specialized interfaces allow you to control or capture values for a component.
  - Record/playback the state that caused a fault and re-create the fault.
  - Sandboxing isolates an instance of the system from the real world to enable experimentation to undo its consequences.
- **Limit complexity**: Complex software is harder to test, because its operating state space is very large and more difficult to re-create an exact state in a large state space.
  - Limit **structural complexity** avoiding, reducing or resolving dependencies between components; isolating and encapsulating dependencies on external environment.
    - Limit the number of classes from which a class is derived.
    - Limit the depth of the inheritance tree and the number of children of a class.
    - Limit polymorphism and dynamic calls.
  - Limit nondeterminism - limiting **behavioral complexity**.
    - Nondeterminism systems are harder to test.

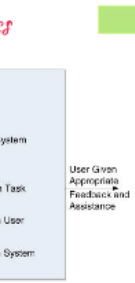
Checklist for Testability Design and Analysis

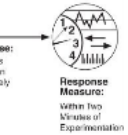
Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities are most critical and hence need to be most thoroughly tested. Ensure that additional system responsibilities have been allocated to do the following: ▪ Execute test suite and capture results (external test or self-test) ▪ Capture (log) the activity that resulted in a fault *or* that resulted in unexpected (perhaps emergent) behavior that was not necessarily a fault ▪ Control and observe relevant system state for testing Make sure the allocation of functionality provides high cohesion, low coupling, strong separation of concerns, and low structural complexity.
Coordination Model	Ensure the system's coordination and communication mechanisms: ▪ Support the execution of a test suite and capture the results within a system or between systems ▪ Support capturing activity that resulted in a fault within a system or between systems ▪ Support injection and monitoring of state into the communication channels for use in testing, within a system or between systems ▪ Do not introduce needless nondeterminism
Data Model	Determine the major data abstractions that must be tested to ensure the correct operation of the system. ▪ Ensure that it is possible to capture the values of instances of these data abstractions ▪ Ensure that the values of instances of these data abstractions can be set when state is injected into the system, so that system state leading to a fault may be re-created ▪ Ensure that the creation, initialization, persistence, manipulation, translation, and destruction of instances of these data abstractions can be exercised and captured
Mapping among Architectural Elements	Determine how to test the possible mappings of architectural elements (especially mappings of processes to processors, threads to processes, and modules to components) so that the desired test response is achieved and potential race conditions identified. In addition, determine whether it is possible to test for illegal mappings of architectural elements.
Resource Management	Ensure there are sufficient resources available to execute a test suite and capture the results. Ensure that your test environment is representative of (or better yet, identical to) the environment in which the system will run. Ensure that the system provides the means to do the following: ▪ Test resource limits ▪ Capture detailed resource usage for analysis in the event of a failure ▪ Inject new resource limits into the system for the purposes of testing ▪ Provide virtualized resources for testing
Binding Time	Ensure that components that are bound later than compile time can be tested in the late-bound context. Ensure that late bindings can be captured in the event of a failure, so that you can re-create the system's state leading to the failure. Ensure that the full range of binding possibilities can be tested.
Choice of Technology	Determine what technologies are available to help achieve the testability scenarios that apply to your architecture. Are technologies available to help with regression testing, fault injection, recording and playback, and so on? Determine how testable the technologies are that you have chosen (or are considering choosing in the future) and ensure that your chosen technologies support the level of testing appropriate for your system. For example, if your chosen technologies do not make it possible to inject state, it may be difficult to re-create fault scenarios.

Category	Checklist
Allocation of Responsibilities	Determine which system responsibilities are most critical and hence need to be most thoroughly tested. Ensure that additional system responsibilities have been allocated to do the following: ▪ Execute test suite and capture results (external test or self-test) ▪ Capture (log) the activity that resulted in a fault *or* that resulted in unexpected (perhaps emergent) behavior that was not necessarily a fault ▪ Control and observe relevant system state for testing Make sure the allocation of functionality provides high cohesion, low coupling, strong separation of concerns, and low structural complexity.
Coordination Model	Ensure the system's coordination and communication mechanisms: ▪ Support the execution of a test suite and capture the results within a system or between systems ▪ Support capturing activity that resulted in a fault within a system or between systems ▪ Support injection and monitoring of state into the communication channels for use in testing, within a system or between systems ▪ Do not introduce needless nondeterminism
Data Model	Determine the major data abstractions that must be tested to ensure the correct operation of the system. ▪ Ensure that it is possible to capture the values of instances of these data abstractions ▪ Ensure that the values of instances of these data abstractions can be set when state is injected into the system, so that system state leading to a fault may be re-created ▪ Ensure that the creation, initialization, persistence, manipulation, translation, and destruction of instances of these data abstractions can be exercised and captured
Mapping among	Determine how to test the possible mappings of

	of these data abstractions can be exercised and captured
Mapping among Architectural Elements	Determine how to test the possible mappings of architectural elements (especially mappings of processes to processors, threads to processes, and modules to components) so that the desired test response is achieved and potential race conditions identified. In addition, determine whether it is possible to test for illegal mappings of architectural elements.
Resource Management	Ensure there are sufficient resources available to execute a test suite and capture the results. Ensure that your test environment is representative of (or better yet, identical to) the environment in which the system will run. Ensure that the system provides the means to do the following: ▪ Test resource limits ▪ Capture detailed resource usage for analysis in the event of a failure ▪ Inject new resource limits into the system for the purposes of testing ▪ Provide virtualized resources for testing
Binding Time	Ensure that components that are bound later than compile time can be tested in the late-bound context. Ensure that late bindings can be captured in the event of a failure, so that you can re-create the system's state leading to the failure. Ensure that the full range of binding possibilities can be tested.
Choice of Technology	Determine what technologies are available to help achieve the testability scenarios that apply to your architecture. Are technologies available to help with regression testing, fault injection, recording and playback, and so on? Determine how testable the technologies are that you have chosen (or are considering choosing in the future) and ensure that your chosen technologies support the level of testing appropriate for your system. For example, if your chosen technologies do not make it possible to inject state, it may be difficult to re-create fault scenarios.

Usability





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

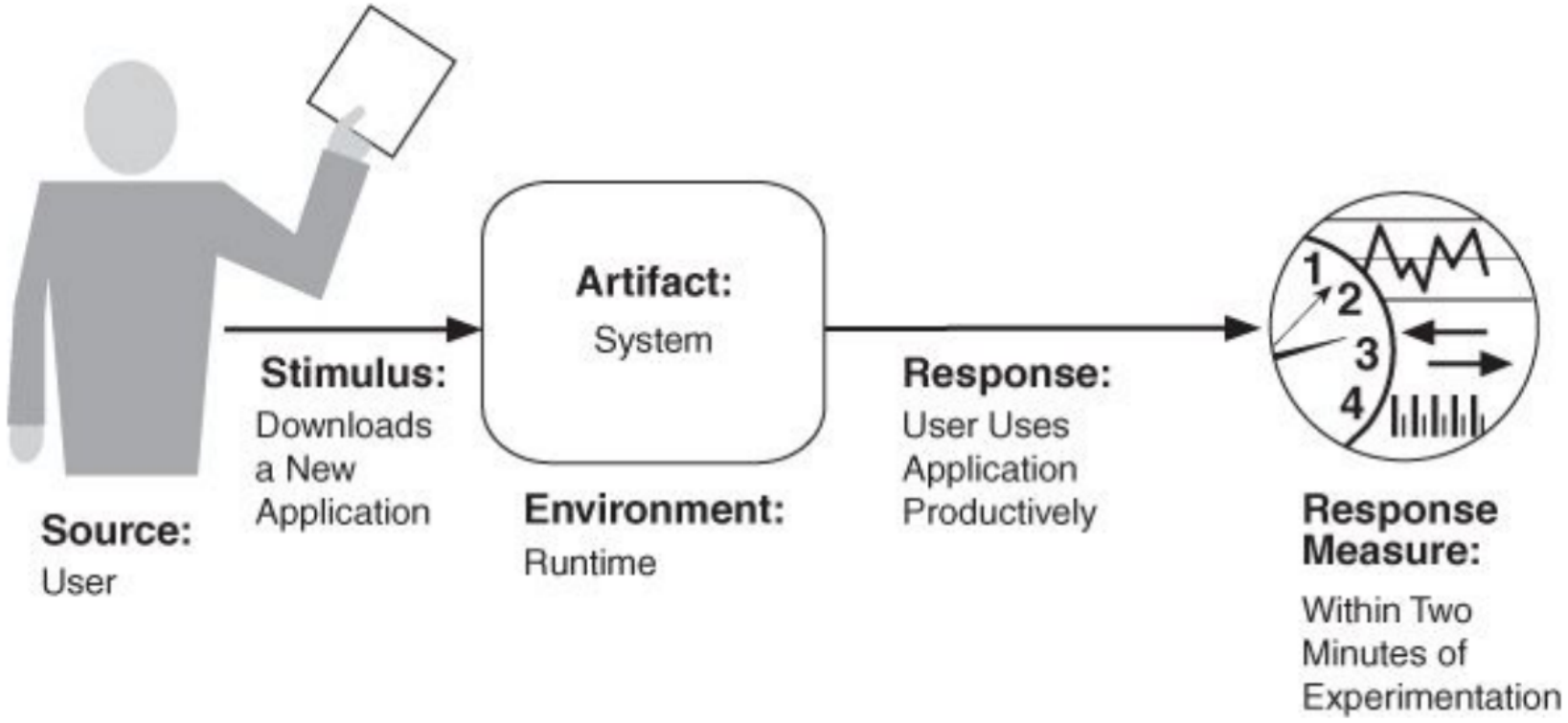
Usability

- **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.
- Usability comprises the following aspects:
  - Learning system features
  - Using a system efficiently
  - Minimizing the impact of errors
  - Adapting the system to user's needs
  - Increasing confidence and satisfaction

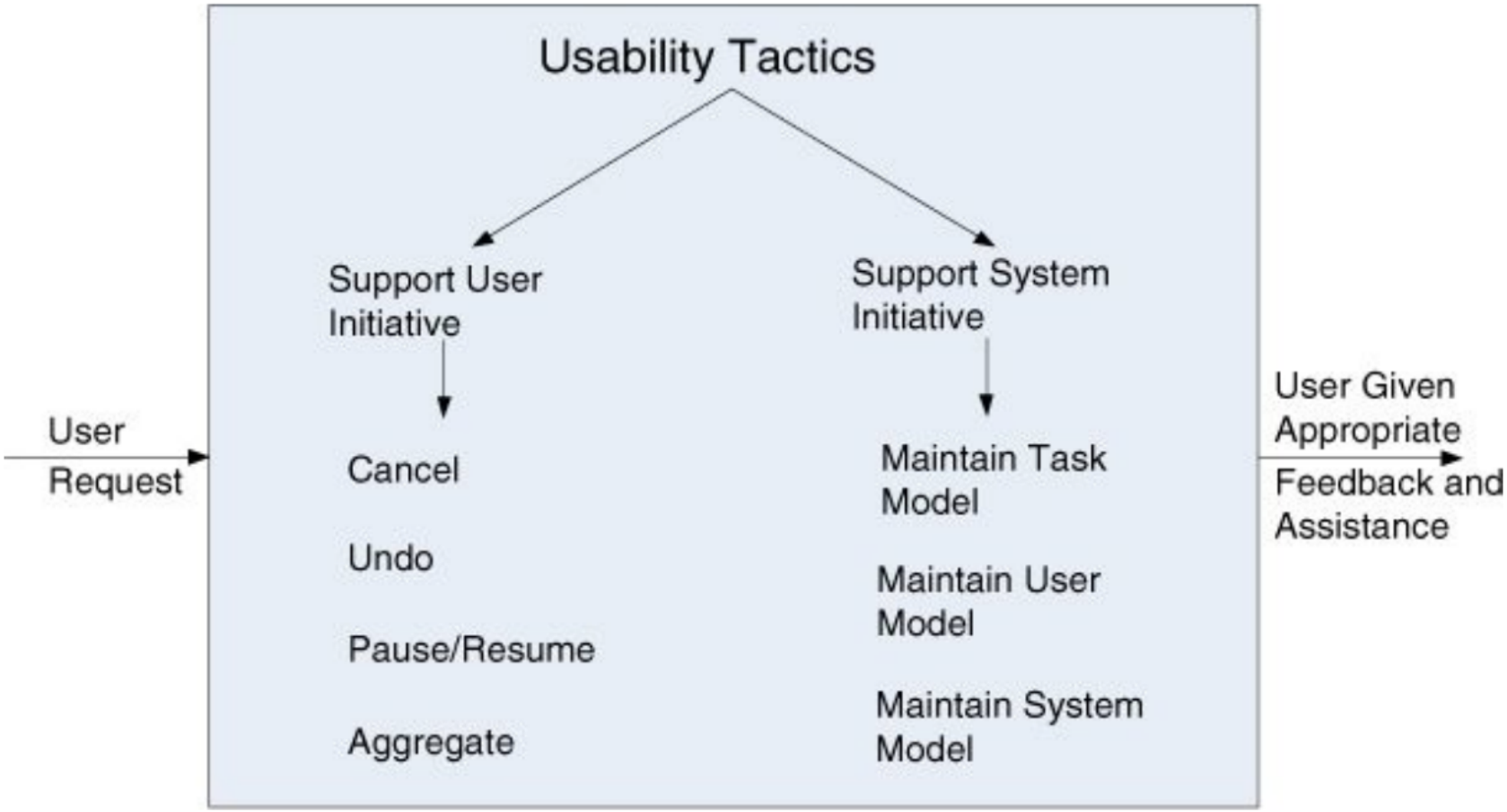
Usability General Scenario

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 Sample Scenario



Usability Tactics



Tactics for Usability

- Support **user initiative**: support the user in either correcting errors or being more efficient.
  - Cancel.
  - Undo: System must maintain a sufficient amount of system state so that an earlier state may be restored.
  - Pause/resume when a user has initiated a long-running operation.
  - Aggregate the lower-level objects into a single group, so that the operation may be applied to the group.
- Support **system initiaive**: Identify the models the system uses to predict either its own behavior or the user's intention.
  - Maintain task model: Determine context so the system can have some idea of what the user is attemping and provide assistance.
  - Maintain user model: Represent the user's knowledge of system.
  - Maintain system model: Determine expected system behavior so that appropriate feedback can be given to the user.

Checklist for Usability Design and Analysis

Category	Checklist
Allocation of Responsibilities	Ensure that additional system responsibilities have been allocated, as needed, to assist the user in the following: ▪ Learning how to use the system ▪ Efficiently achieving the task at hand ▪ Adapting and configuring the system ▪ Recovering from user and system errors
Coordination Model	Determine whether the properties of system elements' coordination—timeliness, currency, completeness, correctness, consistency—affect how a user learns to use the system, achieves goals or completes tasks, adapts and configures the system, recovers from user and system errors, and gains increased confidence and satisfaction. For example, can the system respond to mouse events and give semantic feedback in real time? Can long-running events be canceled in a reasonable amount of time?
Data Model	Determine the major data abstractions that are involved with user-perceivable behavior. Ensure these major data abstractions, their operations, and their properties have been designed to assist the user in achieving the task at hand, adapting and configuring the system, recovering from user and system errors, learning how to use the system, and increasing satisfaction and user confidence. For example, the data abstractions should be designed to support *undo* and *cancel* operations: the transaction granularity should not be so great that canceling or undoing an operation takes an excessively long time.
Mapping among Architectural Elements	Determine what mapping among architectural elements is visible to the end user (for example, the extent to which the end user is aware of which services are local and which are remote). For those that are visible, determine how this affects the ways in which, or the ease with which, the user will learn how to use the system, achieve the task at hand, adapt and configure the system, recover from user and system errors, and increase confidence and satisfaction.
Resource Management	Determine how the user can adapt and configure the system's use of resources. Ensure that resource limitations under all user-controlled configurations will not make users less likely to achieve their tasks. For example, attempt to avoid configurations that would result in excessively long response times. Ensure that the level of resources will not affect the users' ability to learn how to use the system, or decrease their level of confidence and satisfaction with the system.
Binding Time	Determine which binding time decisions should be under user control and ensure that users can make decisions that aid in usability. For example, if the user can choose, at runtime, the system's configuration, or its communication protocols, or its functionality via plug-ins, you need to ensure that such choices do not adversely affect the user's ability to learn system features, use the system efficiently, minimize the impact of errors, further adapt and configure the system, or increase confidence and satisfaction.
Choice of Technology	Ensure the chosen technologies help to achieve the usability scenarios that apply to your system. For example, do these technologies aid in the creation of online help, the production of training materials, and the collection of user feedback? How usable are any of your chosen technologies? Ensure the chosen technologies do not adversely affect the usability of the system (in terms of learning system features, using the system efficiently, minimizing the impact of errors, adapting/configuring the system, and increasing confidence and satisfaction).

Category	Checklist
Allocation of Responsibilities	Ensure that additional system responsibilities have been allocated, as needed, to assist the user in the following: ▪ Learning how to use the system ▪ Efficiently achieving the task at hand ▪ Adapting and configuring the system ▪ Recovering from user and system errors
Coordination Model	Determine whether the properties of system elements' coordination—timeliness, currency, completeness, correctness, consistency—affect how a user learns to use the system, achieves goals or completes tasks, adapts and configures the system, recovers from user and system errors, and gains increased confidence and satisfaction. For example, can the system respond to mouse events and give semantic feedback in real time? Can long-running events be canceled in a reasonable amount of time?
Data Model	Determine the major data abstractions that are involved with user-perceivable behavior. Ensure these major data abstractions, their operations, and their properties have been designed to assist the user in achieving the task at hand, adapting and configuring the system, recovering from user and system errors, learning how to use the system, and increasing satisfaction and user confidence. For example, the data abstractions should be designed to support *undo* and *cancel* operations: the transaction granularity should not be so great that canceling or undoing an operation takes an excessively long time.
Mapping among Architectural Elements	Determine what mapping among architectural elements is visible to the end user (for example, the extent to which the end user is aware of which services are local and which are remote). For those that are visible, determine how this affects the ways in which, or the ease with which, the user

	are remote). For those that are visible, determine how this affects the ways in which, or the ease with which, the user will learn how to use the system, achieve the task at hand, adapt and configure the system, recover from user and system errors, and increase confidence and satisfaction.
Resource Management	Determine how the user can adapt and configure the system's use of resources. Ensure that resource limitations under all user-controlled configurations will not make users less likely to achieve their tasks. For example, attempt to avoid configurations that would result in excessively long response times. Ensure that the level of resources will not affect the users' ability to learn how to use the system, or decrease their level of confidence and satisfaction with the system.
Binding Time	Determine which binding time decisions should be under user control and ensure that users can make decisions that aid in usability. For example, if the user can choose, at runtime, the system's configuration, or its communication protocols, or its functionality via plug-ins, you need to ensure that such choices do not adversely affect the user's ability to learn system features, use the system efficiently, minimize the impact of errors, further adapt and configure the system, or increase confidence and satisfaction.
Choice of Technology	Ensure the chosen technologies help to achieve the usability scenarios that apply to your system. For example, do these technologies aid in the creation of online help, the production of training materials, and the collection of user feedback? How usable are any of your chosen technologies? Ensure the chosen technologies do not adversely affect the usability of the system (in terms of learning system features, using the system efficiently, minimizing the impact of errors, adapting/configuring the system, and increasing confidence and satisfaction).


Deployability
Safety
X-ability


Mobility
Deployability
Safety
Scalability
X-ability
Development Distributability
Portability
Variability


Quality Attributes & Tactics
Availability
Interoperability
Modifiability
Performance
Security
Testability
Usability
X-ability
Mobility
Safety
Development Distributability
Variability
Portability
Deployability
Scalability