Software Architecture

Quality Attributes

Quality Attribute Specification

- Architects are often told:
  - “My application must be fast/secure/scale”
  - Far too imprecise
- Quality attributes (QAs) must be made precise/measurable for a given system design, e.g.
  - *“It must be possible to scale the deployment from an initial 100 geographically dispersed user desktops to 10,000 without an increase in effort/cost for installation and configuration.”*

What are Quality Attributes

- Often know as –ilities
  - Performance
  - Security
  - Availability
  - Scalability
  - Usability
  - Reliability
  - Portability
  - Modifiability
  - Maintainability

Architecture and Quality Attributes

- Achieving quality attributes must be considered throughout design, implementation, and deployment.
- For example:
  - Usability involves both architectural and nonarchitectural aspects.
    - Making the user interface easy to use is nonarchitectural
    - providing the user with undo operations is architectural.

Architecture and Quality Attributes

- For example:
  - Modifiability
    - how functionality is divided (architectural)
    - by coding techniques within a module (nonarchitectural).
  - Performance
    - how much communication is necessary among components and how shared resources are allocated (architectural)
    - the choice of algorithms and how they are coded (nonarchitectural)

Architectural conflicts

- Within complex systems, the achievement of quality attributes affect each other.
- Sometimes the effect is positive and sometimes negative.
- For example,
  - Using large-grain components improves performance but reduces maintainability.
  - Introducing redundant data improves availability but makes security more difficult.
  - Localizing safety-related features usually means more communication so degraded performance.

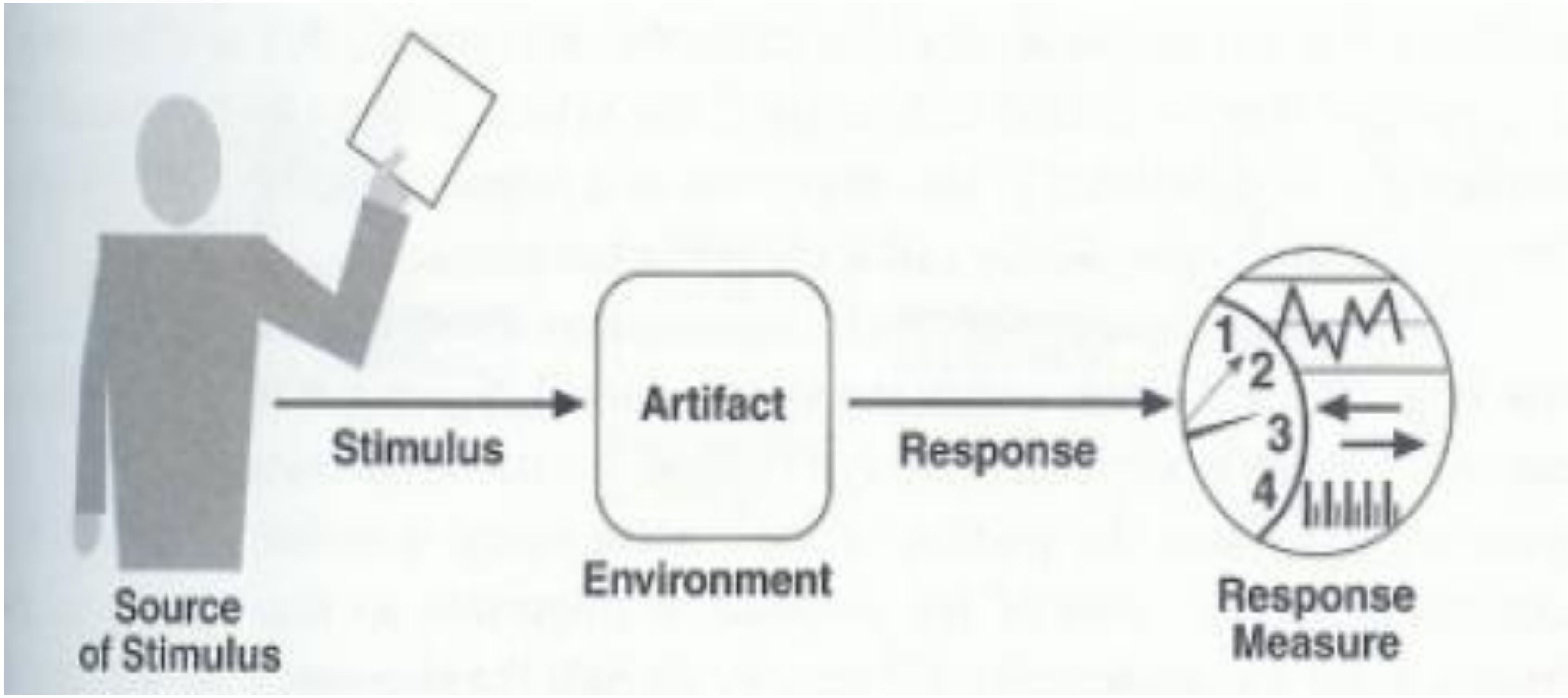
Quality Attribute Scenarios

- A quality attribute scenario is a quality-attribute-specific requirement. It consists of six parts.
  - Source of stimulus – the entity that generated the stimulus
  - Stimulus – a condition that needs to be considered when it arrives at a system
  - Environment – the particular conditions in which the stimulus occurs

Quality Attribute Scenarios

- The six parts of a quality attribute scenario.
  - Artifact – the system or the pieces of it that are stimulated
  - Response – the activity undertaken after the arrival of the stimulus
  - Response measure – when the response occurs, it should be measurable in some fashion so that the requirement can be tested.

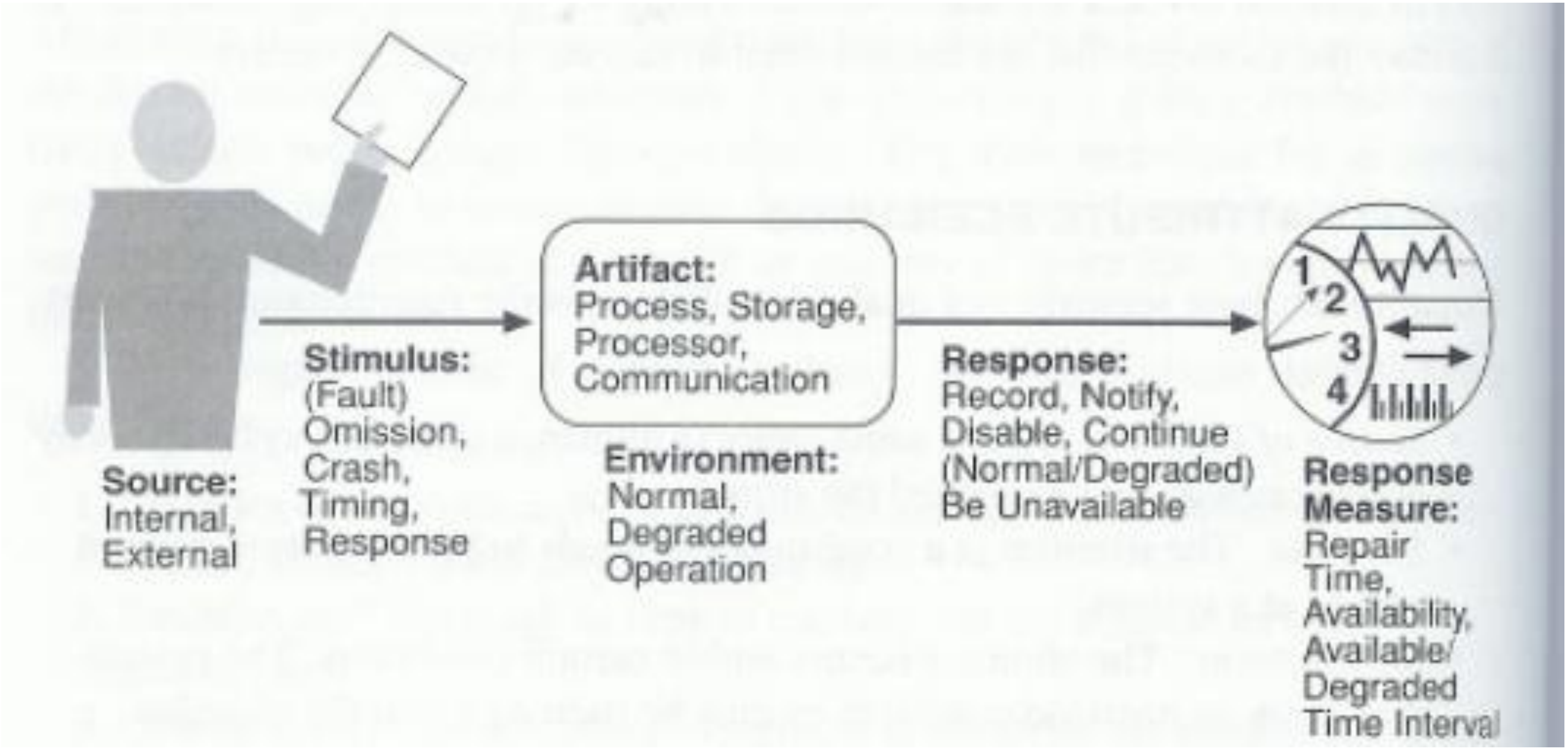
Quality Attribute Parts



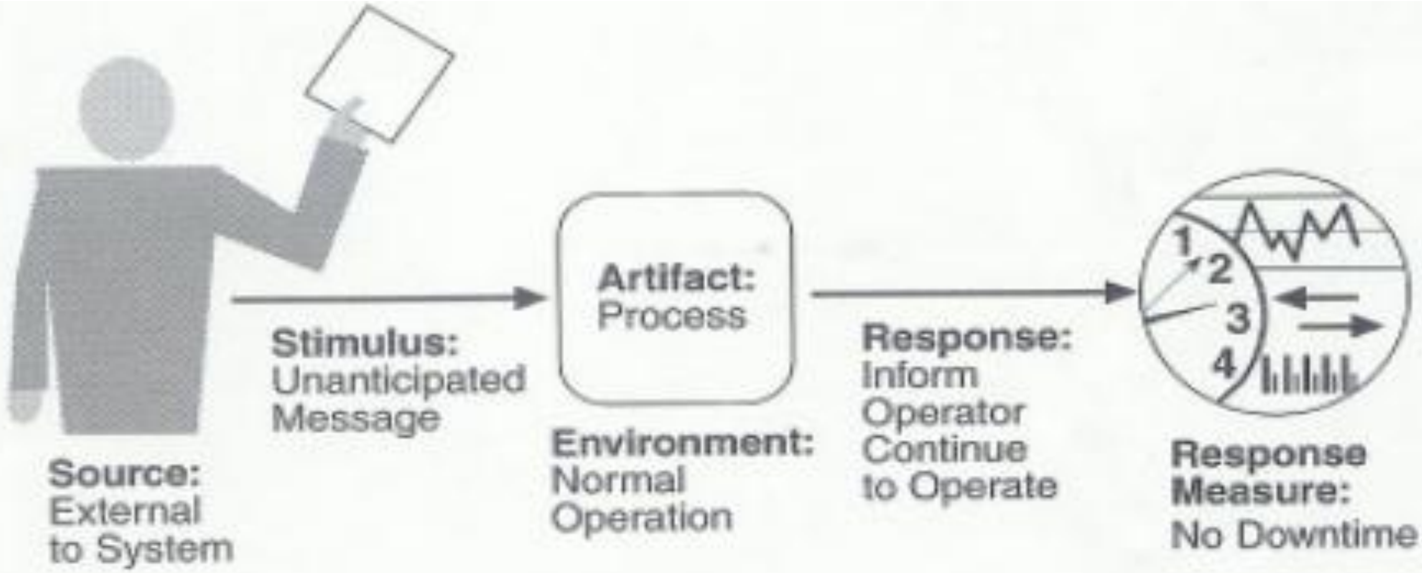
General vs. Concrete Quality Attribute Scenarios

- A general scenario is system independent and can, potentially, pertain to any system.
- A concrete scenario is specific to the particular system under consideration.
- Concrete scenarios are needed to make the quality requirements operational.

General Scenario for Availability

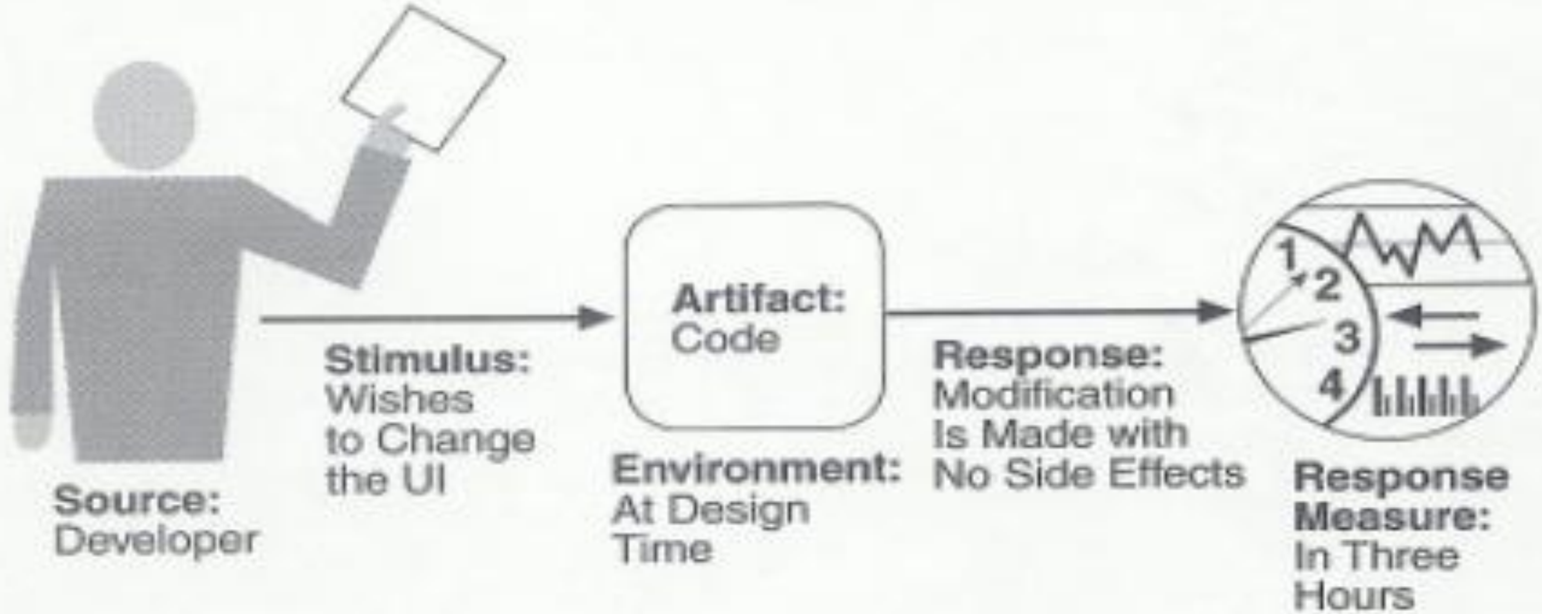


Sample Concrete Availability Scenario



Sample availability scenario. An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and the system continues to operate with no downtime.

Sample Modifiability Scenario



Sample modifiability scenario. A developer wishes to change the UI code at design time; modification is made with no side effects in three hours.

Features of Concrete Scenarios

- A collection of concrete scenarios can be used as the quality attribute requirements of a system.
- Each scenario is concrete enough to be meaningful to the architect.
- The details of the responses are meaningful enough so that it is possible to test whether the system has achieved the response.

Features of Concrete Scenarios

- Concrete scenarios play the same role in the specification of quality attributes that use cases play in the specification of functional requirements.

Quality Attribute Scenario Generation

- Theoretically quality attribute requirements should be obtained during requirements analysis, but in practice is seldom done.
- It is the architect's task to ensure that this is accomplished by generating concrete quality attribute scenarios.
- Quality-attribute-specific tables are used to create general scenarios and from them concrete scenarios are specified.

Quality Attribute Scenario Generation

- The tables serve as checklist to ensure that all possibilities have been considered.
- It doesn’t matter to generate the same or similar scenarios from different quality attributes as the redundancies can easily be removed.
- The important is that no important requirements are omitted.

Quality Attribute Scenarios in Practice

- The general scenario approach provides a framework for generating a large number of system-independent, quality-attribute-specific scenarios.
- To make the general scenarios useful, you must make them system specific.

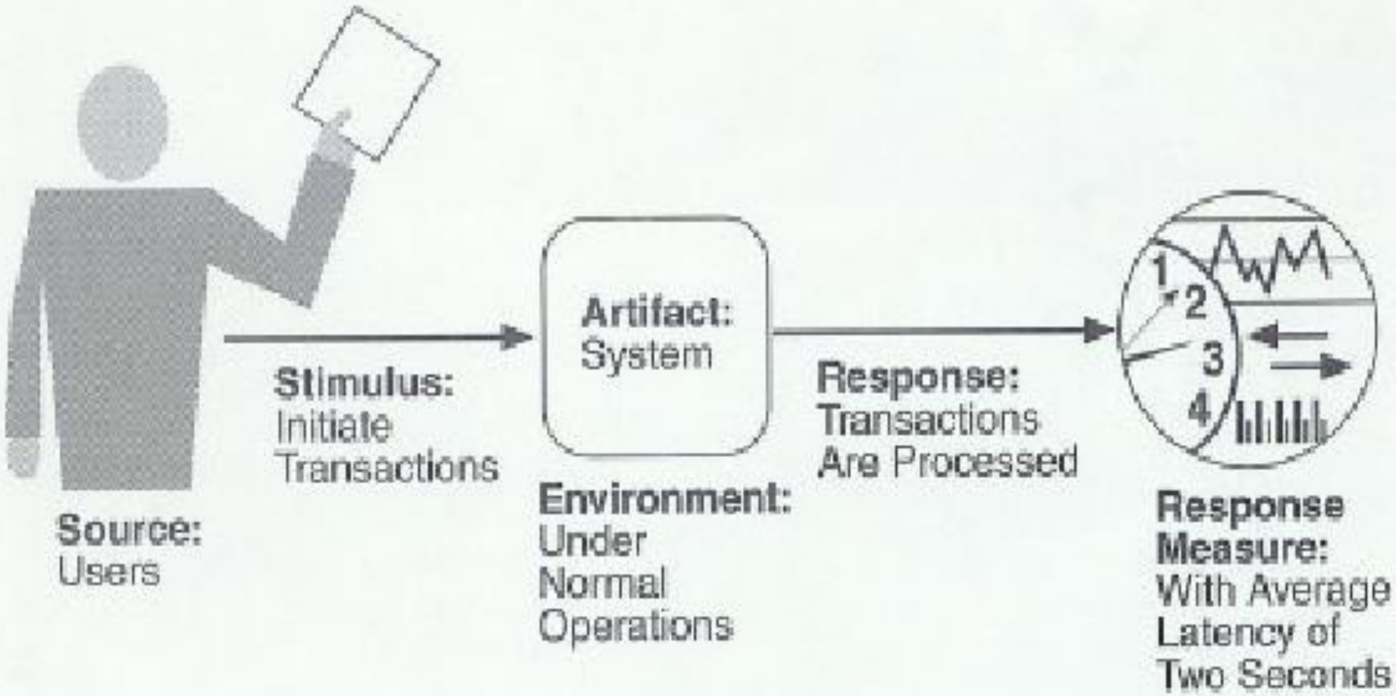
Performance

- Many examples of poor performance in enterprise applications
- Performance requires a:
  - Metric of amount of work performed in unit time
  - Deadline that must be met
- Enterprise applications often have strict performance requirements, e.g.
  - 1000 transactions per second
  - 3 second average latency for a request
- Localize critical operations and minimize communications. Use large rather than fine-grain components.

Table for Generation of General Performance Scenario

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

Sample Performance Scenario



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

Performance - Throughput

- Measure of the amount of work an application must perform in unit time
  - Transactions per second
  - Messages per minute
- Is required throughput:
  - Average?
  - Peak?

Performance - Response Time

- measure of the latency an application exhibits in processing a request
- Usually measured in (milli)seconds
- Is required response time:
  - Guaranteed?
  - Average?

Performance - Deadlines

- something must be completed before some specified time
  - Payroll system must complete by 2am so that electronic transfers can be sent to bank
  - Weekly accounting run must complete by 6am Monday so that figures are available to management
- Deadlines are often associated with batch jobs in IT systems.

Scalability

- Scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged.
- 4 common scalability issues in IT systems:
  - Request load
  - Connections
  - Data size
  - Deployments

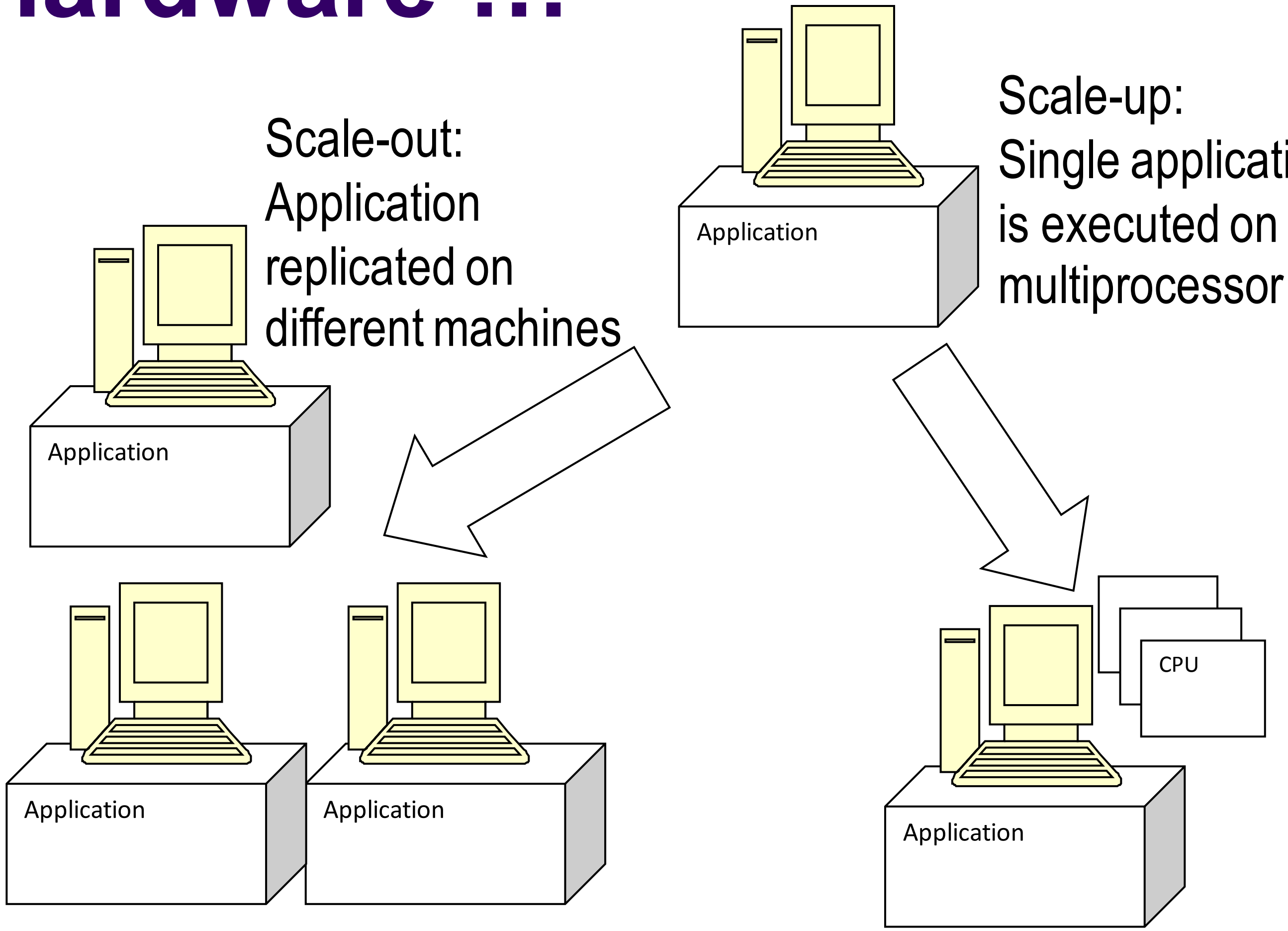
Scalability – Request Load

- How does an 100 tps application behave when simultaneous request load grows?
  - From 100 to 1000 requests per second?
- Ideal solution, without additional hardware capacity:
  - as the load increases, throughput remains constant (i.e. 100 tps), and response time per request increases only linearly (i.e. 10 seconds).

Scalability – Add more hardware …

Scale-out:
Application replicated on different machines

Scale-up:
Single application instance is executed on a multiprocessor machine



Scalability - connections

- What happens if number of simultaneous connections to an application increases
  - If each connection consumes a resource?
  - Exceed maximum number of connections?

Scalability – Data Size

- How does an application behave as the data it processes increases in size?
  - Chat application sees average message size double?
  - Database table size grows from 1 million to 20 million rows?
  - Image analysis algorithm processes images of 100MB instead of 1MB?
- Can application/algorithms scale to handle increased data requirements?

Scalability - Deployment

- How does effort to install/deploy an application increase as installation base grows?
  - Install new users?
  - Install new servers?
- Solutions typically revolve around automatic download/installation
  - E.g. downloading applications from the Internet

Scalability thoughts

- Scalability often overlooked.
  - Major cause of application failure
  - Hard to predict
  - Hard to test/validate
  - Reliance on proven designs and technologies is essential

Modifiability

- Modifications to a software system during its lifetime are a fact of life.
- Modifiable systems are easier to change/evolve
- Modifiability should be assessed in context of how a system is likely to change
  - No need to consider changes that are highly unlikely to occur
- Modifiability is about the cost of change.

Modifiability

- Modifiability measures how easy it **may** be to change an application to cater for new (non-) functional requirements.
  - **‘may’** – nearly always impossible to be certain
  - Must estimate cost/effort

Modifiability Analysis

- Impact is rarely easy to quantify
- The best possible is a:
  - Convincing impact analysis of changes needed
  - A demonstration of how the solution can accommodate the modification without change.

Modifiability Analysis

- Minimizing dependencies increases modifiability
  - Changes isolated to single components likely to be less expensive than those that cause ripple effects across the architecture.

Security

- Security Properties
  - **Authentication:** Applications can verify the identity of their users and other applications with which they communicate.
  - **Authorization:** Authenticated users and applications have defined access rights to the resources of the system.
  - **Encryption:** The messages sent to/from the application are encrypted.

Security

- **Integrity:** This ensures the contents of a message are not altered in transit.
- **Non-repudiation:** The sender of a message has proof of delivery and the receiver is assured of the sender's identity. This means neither can subsequently refute their participation in the message exchange, i.e. a transaction cannot be denied by any of the parties to it.
- **Auditing:** the property that the system tracks activities within it at levels sufficient to reconstruct them.

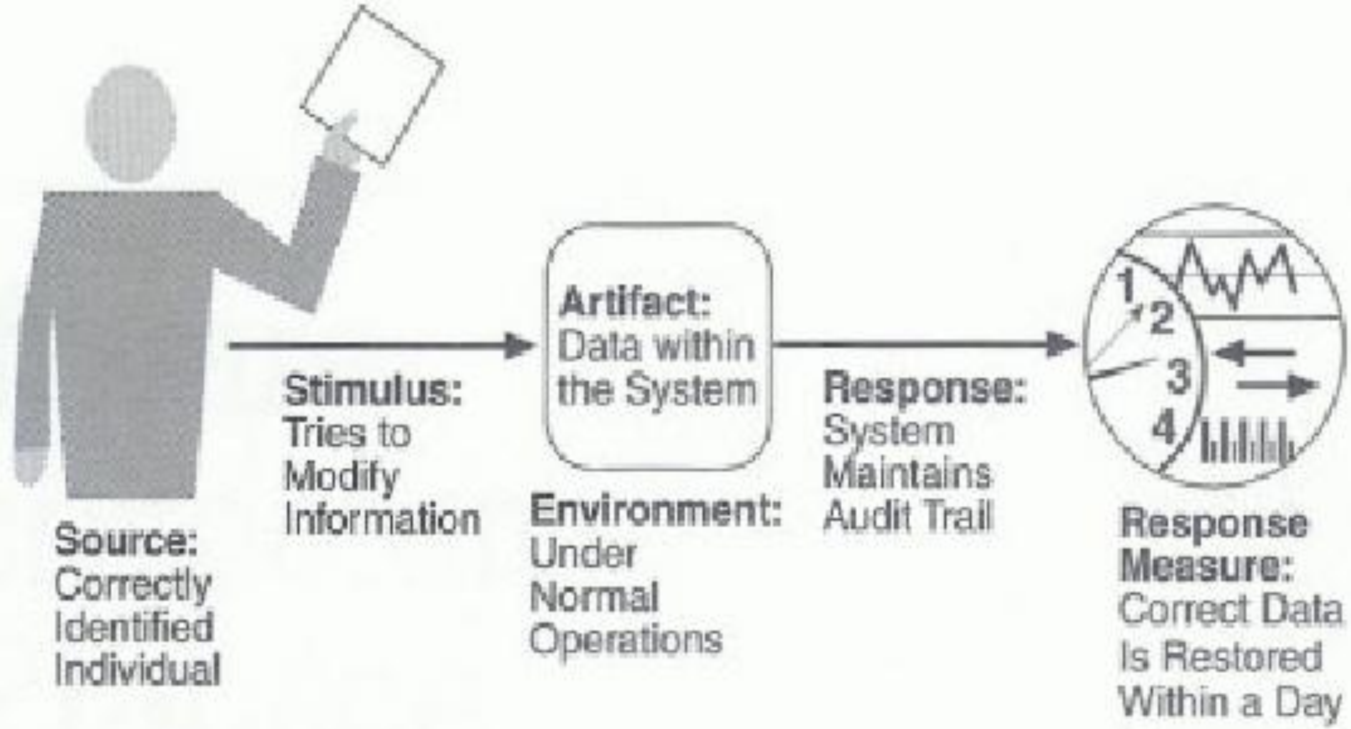
Security Specifics

- Security is a measure of the system’s ability to resist unauthorized usage while providing its services to legitimate users.
- An attempt to breach security is an attack – it could be to gain access to data or services or to deny service to others.

Table for Generation of General Security Scenario

Portion of Scenario	Possible Values
Source	Individual or system that is correctly identified, identified incorrectly, of unknown identity who is internal/external, authorized/not authorized with access to limited resources, vast resources
Stimulus	Tries to display data, change/delete data, access system services, reduce availability to system services
Artifact	System services; data within system
Environment	Either online or offline, connected or disconnected, firewalled or open
Response	Authenticates user; hides identity of the user; blocks access to data and/or services; allows access to data and/or services; grants or withdraws permission to access data and/or services; records access/modifications or attempts to access/modify data/services by identity; stores data in an unreadable format; recognizes an unexplainable high demand for services, and informs a user or another system, and restricts availability of services
Response Measure	Time/effort/resources required to circumvent security measures with probability of success; probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied

Sample Security Scenario



Sample security scenario. A correctly identified individual tries to modify system data from an external site; system maintains an audit trail and the correct data is restored within one day.

Availability

- Availability is concerned with system failure and its consequences.
- Key requirement for most IT applications
- Measured by the proportion of the required time it is useable.
  - 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
  - the capability to reestablish performance levels and recover affected data after an application or system failure

Availability Specifics

- Areas of concern:
  - How the system failure is detected
  - How frequently system failures occur
  - What happens when a failure occurs
  - How long a system is allowed to be out of operation
  - When failures may occur safely
  - How failures can be prevented
  - What kinds of notifications are required when a failure occurs

Generation of General Availability Scenario

Portion of Scenario	**Possible Values**
Source	Internal to the system; external to the system
Stimulus	Fault; omission, crash, timing, response
Artifact	System's processors, communication channels, persistent storage processes
Environment	Normal operation; degraded mode (i.e., fewer features, a fall-back solution.)
Response	System should detect event and do one or more of the following: record it notify appropriate parties, including the user and other systems disable sources of events that cause fault or failure be unavailable for a prespecified interval continue to operate in normal or degraded mode
Response Measure	Time interval when the system must be available; availability time; time interval in which system can be in degraded mode; repair time

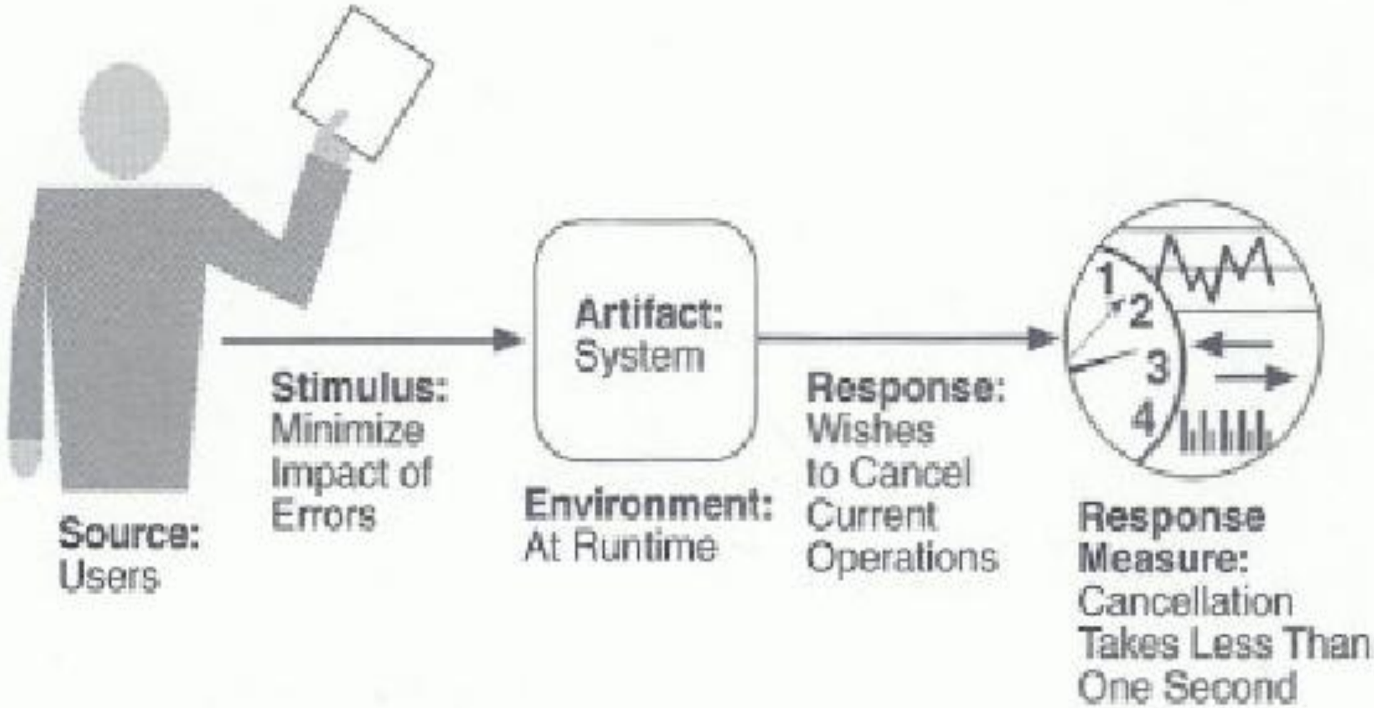
Usability

- Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support provided.
- Areas of usability:
  - Learning system features
  - Using a system efficiently
  - Minimizing the impact of errors
  - Adapting the system to user needs
  - Increasing confidence and satisfaction

Generation of General Usability Scenario

Portion of Scenario	Possible Values
Source	End user
Stimulus	Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable
Artifact	System
Environment	At runtime or configuration time
Response	System provides one or more of the following responses: to support "learn system features" – help system is sensitive to context; interface is familiar to user; interface is usable in an unfamiliar context to support "use system efficiently" – aggregation of data and/or commands; re-use of already entered data and/or commands; support for efficient navigation within a screen; distinct views with consistent operations; comprehensive searching; multiple simultaneous activities to minimize "impact of errors" – undo, cancel, recover from system failure, recognize and correct user error, retrieve forgotten password, verify system resources to "adapt system" – customizability; internationalization to "feel comfortable" – display system state; work at the user's pace
Response Measure	Task time, number of errors, number of problems solved, user satisfaction, gain of user knowledge, ratio of successful operations to total operations; amount of time/data lost

Sample Usability Scenario



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

Design Trade-offs

- QAs are rarely orthogonal, they interact and affect each other
- Architects must create solutions that makes sensible design compromises
  - not possible to fully satisfy all competing requirements
  - Must satisfy all stakeholder needs

Summary

- QAs are part of an application's non-functional requirements
- Many QAs
- Architect must decide which are important for a given application
  - Understand implications for application
  - Understand competing requirements and trade-offs.