# SOFTWARE DESIGN

Quality attributes

## CONTENT

Requirements Engineering

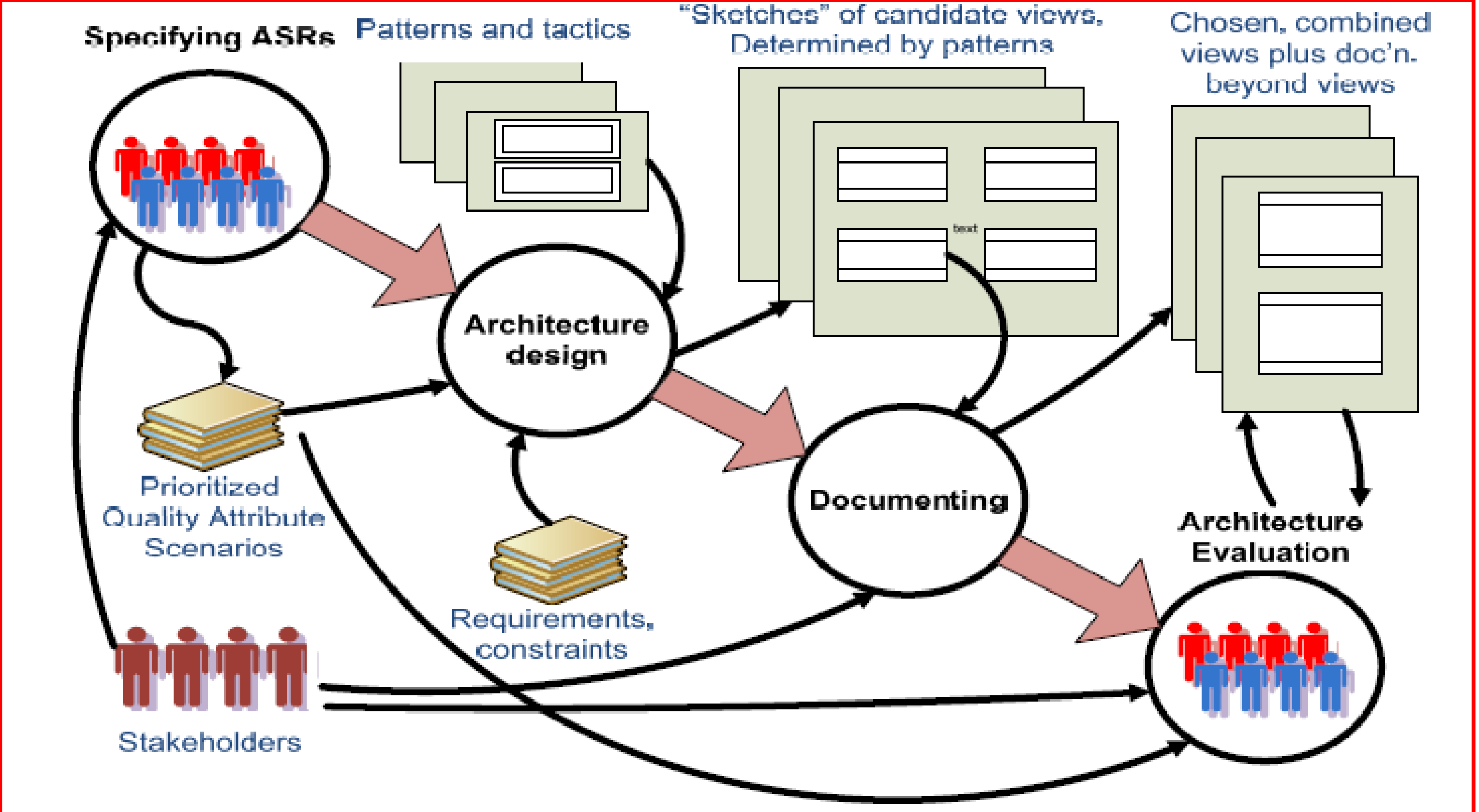
Achieving Quality Attributes

Attribute-Driven Design (ADD)

## REFERENCES

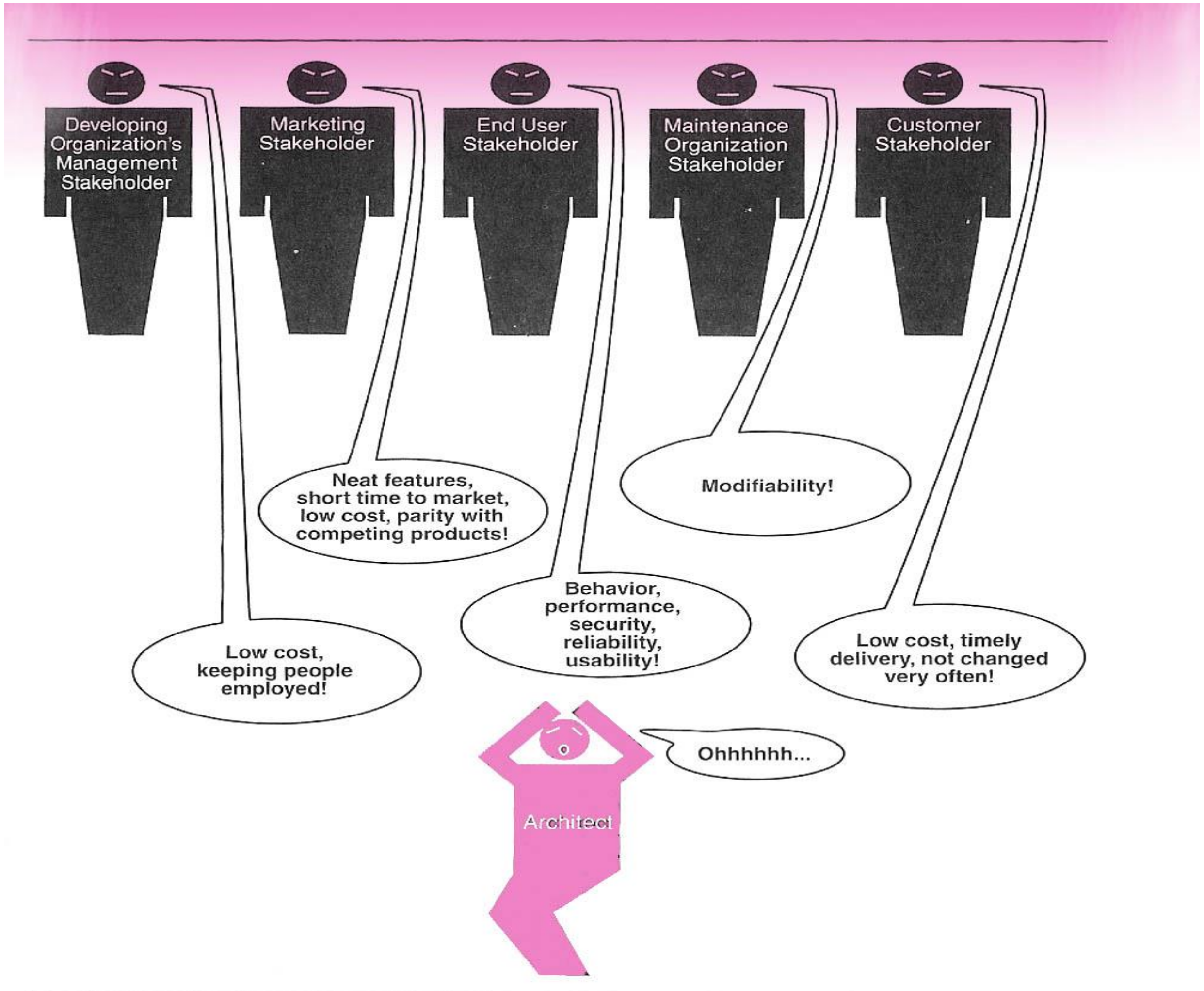
- Len Bass, Paul Clements, Rick Kazman, **Software Architecture in Practice, Second Edition**, Addison Wesley, 2003, ISBN: 0-321-15495-9
- Felix Bachmann, Len Bass, Mark Klein, **Deriving Architectural Tactics: A Step Toward Methodical Architectural Design,**
- TECHNICAL REPORT CMU/SEI-2003-TR-004
- IBM Rational
- Microsoft MSF
- http://www.cs.uu.nl/wiki/bin/view/Swa/CourseLiterature

# SOFTWARE ARCHITECTURE PROCESS


Specifying ASRs
Patterns and tactics
"Sketches" of candidate views, Determined by patterns
Chosen, combined views plus doc'n. beyond views
text
Architecture design
Prioritized Quality Attribute Scenarios
Documenting
Architecture Evaluation
Requirements, constraints
Stakeholders

Adapted from Hofmeister et al., 2006.

## STAKEHOLDERS



Developing Organization's Management Stakeholder
Marketing Stakeholder
End User Stakeholder
Maintenance Organization Stakeholder
Customer Stakeholder
Neat features, short time to market, low cost, parity with competing products!
Modifiability!
Behavior, performance, security, reliability, usability!
Low cost, keeping people employed!
Low cost, timely delivery, not changed very often!
Ohhhhhh...
Architect

**FIGURE 1.2** Influence of stakeholders on the architect

- Developing Organization Management
- Marketing
- End User
- Maintenance Organization
- Customer

## REQUIREMENTS ELICITATION

Requirements elicitation = the activities involved in discovering the requirements of the system

Techniques:

- **Asking**: interview, (structured) brainstorm, questionnaire
- **Task analysis**
- **Scenario-based analysis**: 'think aloud', use case analysis
- **Ethnography**: active observation
- **Form and document analysis**
- Start from **existing system**
- **Prototyping**
- **Own insight**

General advice: use more than one technique!

## REQUIREMENTS SPECIFICATIONS

Requirements specification = rigorous modeling of requirements, to provide formal definitions for various aspects of the system

Requirements specification document should be

- as **precise** as possible: starting point for architecture/design
- as **readable** as possible: understandable for the user

Other preferred properties:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance
- Verifiable
- Modifiable
- Traceable

## SPECIFICATION TECHNIQUES

Specification techniques:

- Entity-Relationship (E-R) modeling
- Use cases (UML)
- Epics and User stories

## REQUIREMENTS VALIDATION

- Requirements validation = checking the requirements document for consistency, completeness and accuracy
- Validation of requirements needs user interaction!
- Techniques:
  - Reviews (reading, checklists, discussion)
  - Prototyping
  - Animation

## USE-CASES

- Technique for specifying **functional** requirements (What should the system do?)
- A **use case** captures a **contract** between the stakeholders of a system about its behavior
- The use case describes the system's behavior **under various conditions** as the system responds to a request from one of the stakeholders, called the primary actor
- Use cases are fundamentally a text form describing use scenarios

## BASIC USE-CASE FORMAT

- Use case: <use case goal>
- Level: <one of: summary level, user-goal level, subfunction>
- Primary actor: <a role name for the actor who initiates the use case>
- Description: <the steps of the **main success scenario** from trigger to goal delivery and any cleanup after>
- Extension: <alternate scenarios of success or failure>

## USE-CASES: BEST PRACTICES

- A use case is a prose essay
- Should be easy to read
- Sentence form for the scenario: active voice, present tense, describing an actor successfully achieving a goal
- Include sub-use cases where appropriate
- Keep the GUI out
- Use UML use case diagrams to visualize relations between actors and use cases or among use cases. Use text to specify use cases themselves!
- It is hard, and important, to keep track of the various use cases

## FUNCTIONAL REQUIREMENTS

Pitfalls:

- Undefined or inconsistent system boundary (scope!)
- System point of view instead of actor centered
- Spiderweb actor-to-use case relations
- Long, too many, confusing use case specifications, incomprehensible to customer

Beware of:

- ‘Shopping cart’ mentality
- The ‘all requirements are equal’ fallacy
- Stakeholders who won’t read use case descriptions because they find them too technical or too complicated

## ARCHITECTURAL DRIVERS

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

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

=> functionality is largely independent of structure

- Architecture constrains the mapping of functionality on various structures if **quality attributes** are important

## ARCHITECTURAL DRIVERS [2]

- Get the functionality right and then accommodate nonfunctional requirements – NOT POSSIBLE!!!

- Non-functional requirements must be taken into account EARLY ON!!!

- There are two broad categories of non-functional requirements
  - Design-time
  - Run-time

## SOFTWARE ARCHITECTURE AND 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 the qualities attributes, which are over and above of system's functionality!!!

## QUALITY REQUIREMENTS OBJECTIVES

- Input for architecture definition
- Driving architecture evaluation
- Communicating architectural parts and requirements
- Finding missing requirements
- Guiding the testing process

## QUALITY REQUIREMENTS: BEST PRACTICES

Not all quality attributes are equally important: prioritize!

Make the quality requirements measurable

- Not: ‘The system should perform well’ but ‘The response time in interactive use is less than 200 ms’

Link the quality requirements to use cases

- Example: ‘The ATM validates the PIN within 1 second’

## CHANGE SCENARIOS

Some quality requirements do not concern functionality but other aspects of the system. These cannot be linked to any use case

- Maintainability and Portability, e.g. Changeability and Adaptability

Link these quality requirements to specific change scenarios

- **Not** ‘The system should be very adaptable’ **but** ‘The software can be installed on all Windows and Unix platforms without change to the source code
- **Not** ‘The system should be changeable’ **but** ‘Functionality can be added to the ATM within one month that makes is possible for users to transfer money from savings to checking account

## CONSTRAINTS

- Functional and quality requirements specify the goal, constraints limit the (architecture) solution space

- Stakeholders should therefore not only specify requirements, but also constraints
- Possible constraint categories:
  - Technical, e.g. platform, reuse of existing systems and components, use of standards
  - Financial, e.g. budget
  - Organizational, e.g. processes, availability of customer
  - Time, e.g. deadline

## ACHIEVING QUALITY

Once determined, the quality requirements provide guidance for **architectural decisions**

An architectural decision that influences the qualities of the product is sometimes called a **tactic**

Mutually connected tactics are bundled together into **architectural patterns**: schemas for the structural organization of entire systems

# TYPES OF REQUIREMENTS (FURPS MODEL)

**F**unctionality

- feature sets
- capabilities
- security

**U**sability

- human factors
- aesthetics
- consistency in the user interface
- online and context-sensitive help
- wizards and agents
- user documentation
- training materials

**R**eliability

- frequency and severity of failure
- recoverability
- predictability
- accuracy
- mean time between failure

## TYPES OF REQUIREMENTS [2]

Performance

- speed
- efficiency
- availability
- accuracy
- response time
- recovery time
- resource usage

Supportability

- testability
- extensibility
- adaptability
- maintainability
- compatibility
- configurability
- installability
- localizability (internationalization)

## QUALITY ATTRIBUTES

Business quality:

- Time to market – “Time”
- Cost and benefit – “Economy”
- Projected lifetime – “Form”
- Target market – “Function”
- Rollout schedule – “Time”
- Integration with legacy – “Time”

## QUALITY ATTRIBUTES

Architectural quality

- Conceptual integrity
- Correctness and completeness
- Buildability

## QUALITY ATTRIBUTES

System quality

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

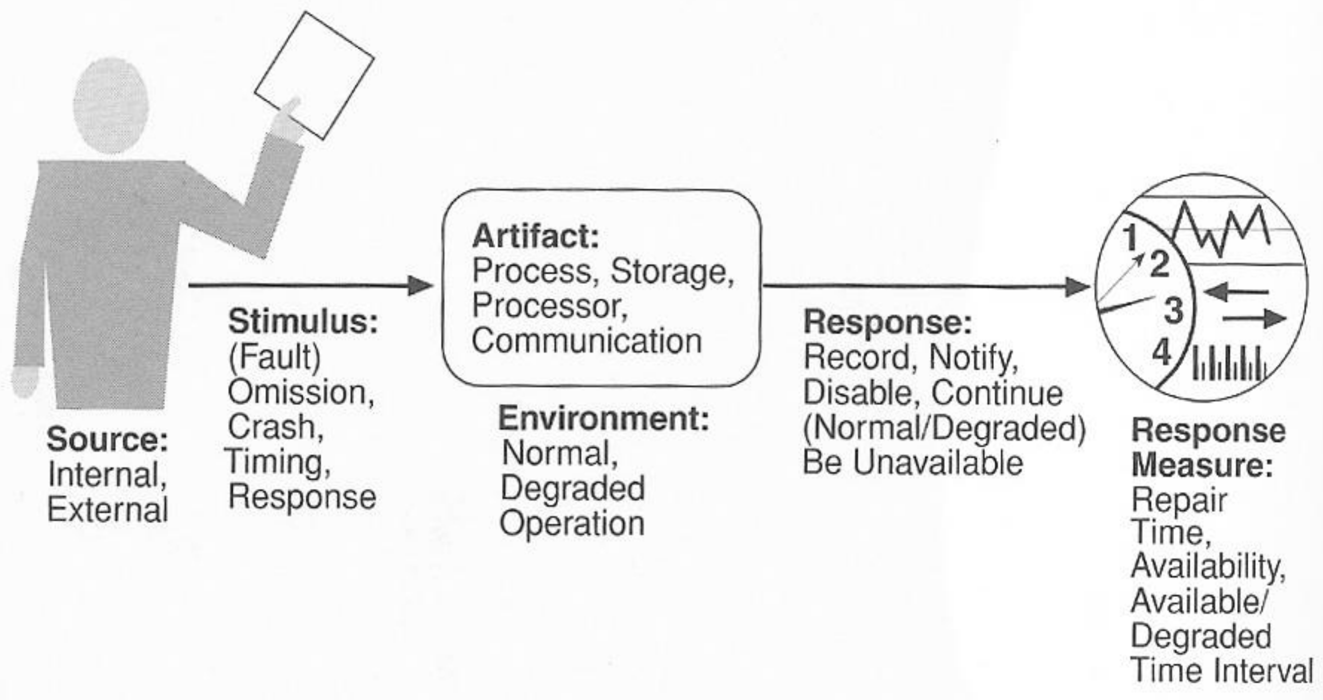
## SYSTEM QUALITY ATTRIBUTES

Are defined in terms of scenarios

- **source of stimulus** [the entity (human or another system) that generated the stimulus or event.] **who?**
- **stimulus** [a condition that determines a reaction of the system.] **what?**
- **environment** [the current condition of the system when the stimulus arrives.] **when?**
- **artifact** [is a component that reacts to the stimulus. It may be the whole system or some pieces of it.] **where?**
- **response** [the activity determined by the arrival of the stimulus.] **which?**
- **response measure** [the quantifiable indication of the response.] **how?**

# GENERAL SCENARIO

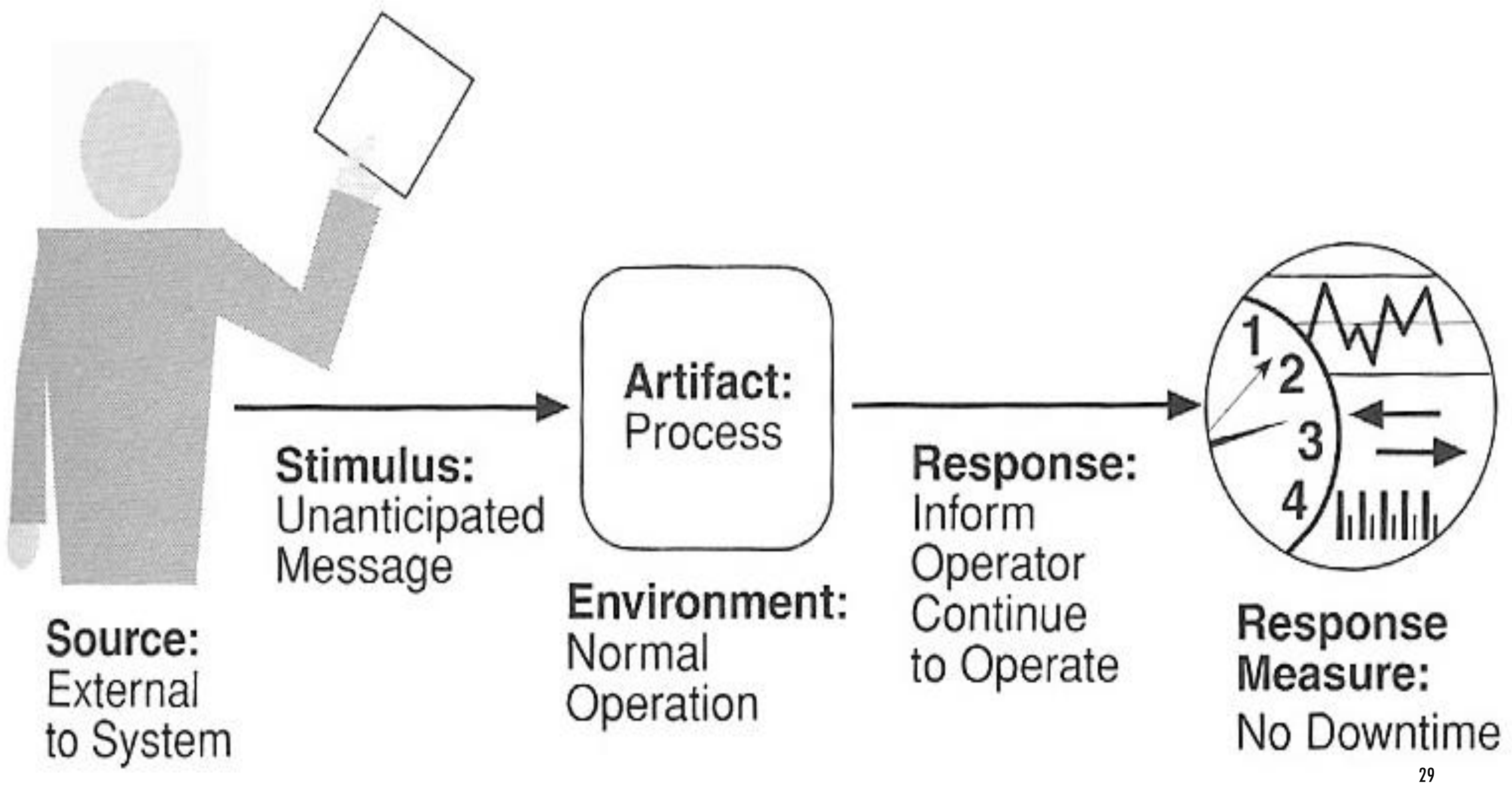
76 Part Two Creating an Architecture 4—Understanding Quality Attributes



Source: Internal, External
Stimulus: (Fault) Omission, Crash, Timing, Response
Artifact: Process, Storage, Processor, Communication
Environment: Normal, Degraded Operation
Response: Record, Notify, Disable, Continue (Normal/Degraded) Be Unavailable
Response Measure: Repair Time, Availability, Available/ Degraded Time Interval
1
2
3
4

**FIGURE 4.2** Availability general scenarios

# CONCRETE SCENARIO



Source: External to System
Stimulus: Unanticipated Message
Artifact: Process
Environment: Normal Operation
Response: Inform Operator Continue to Operate
1
2
3
4
Response Measure: No Downtime

## AVAILABILITY

Typically defined as the probability of a system to be operational when needed in terms of

```
mean time to failure / (mean time to
 failure + mean time to repair)
```

## AVAILABILITY SCENARIO

**Source of stimulus:** internal or external

**Stimulus:**

- omission: a component fails to respond to an input.
- crash: a component repeatedly suffers omission faults.
- timing: a component responds, but the response is early or late.
- response: a component responds with an incorrect value.

**Artifact:** the resource that is required to be available (i.e. processor, communication channel, process, or storage device).

## AVAILABILITY SCENARIO

**Environment:** defines the state of the system when the fault or failure occurred: normal, degraded

**Response:** logging, notification, switching to backup, restart, shutdown

**Response measure:**

- the availability percentage,
- a time for repair,
- certain times during which the system must be available,
- the duration for which the system must be available.

## POS – QUALITY ATTRIBUTE SCENARIO 1

**Scenario**(s): The barcode scanner fails; failure is detected, signalled to user at terminal; continue in degraded mode

**Stimulus Source** : Internal to system

**Stimulus**: Fails

**Environment**: Normal operation

**Artefact** (If Known): Barcode scanner

**Response**: Failure detected, shown to user, continue to operate

**Response Measure**: No downtime, React in 2 seconds

## POS – QUALITY ATTRIBUTE SCENARIO 2

**Scenario(s):** The inventory system fails and the failure is detected. The system continues to operate and queue inventory requests internally; issue requests when inventory system is running again

**Stimulus Source** : Internal to system

**Stimulus:** Fails

**Environment:** Normal operation

**Artefact** (If Known): Inventory system

**Response:** Failure detected, operates in degraded mode, queues requests, detects when inventory system is up again

**Response Measure:** Degraded mode is entered for maximum one hour

## TACTICS TO ACHIEVE AVAILABILITY

- for fault detection
- for fault recovery
- for fault prevention

## TACTICS FOR FAULT DETECTION

- Ping/echo
  - Signal is issued, response is waited for
  - Estimates round-trip time and rate of package loss
- Heartbeat
  - Periodic signal is broadcasted
- Exception handling

## TACTICS FOR FAULT RECOVERY

- Voting
  - run the same algorithm on different processors.
  - "majority rules" approach uncovers any deviant behavior in the processors.
- Active redundancy
  - set up redundant components and keep them synchronized
- Passive redundancy
  - have only one component respond to events, but have that component inform redundant components about the changes

## TACTICS FOR FAULT RECOVERY [2]

- Spare
  - a standby spare computing platform is prepared to replace many different failed components. Backup + logs needed.
- 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 component before it is restored to service.

## TACTICS FOR FAULT PREVENTION

- Removal from service
  - removal of a component of the system from operation in order to update it and avoid potential failures.
    - Automatic
    - Manual
- Transactions
  - set of operations where either all or none are executed successfully.
- Process monitor
  - if a fault is detected in a process, an automated monitoring process can delete the failed process and create a new instance of it, initializing it to some appropriate state as in the spare tactic

## PERFORMANCE

**Performance** refers to the time it takes the system to respond to an event. The event can be fired by:

- a user,
- another system,
- the system itself.

## PERFORMANCE SCENARIO

**Source of stimulus:** The stimuli arrive either from external (possibly multiple) or internal sources.

**Stimulus:** The stimuli are the event arrivals. The arrival pattern can be characterized as periodic, stochastic, or sporadic.

- **Periodic** means that the events arrive in regular intervals of time
- **Stochastic** means that the arrival of events is based on some probabilistic distribution
- **Sporadic** means that the events arrive rather randomly.

**Artifact.** The artifact is always the system's service, which has to respond to the event.

**Environment.** The system can be in various operational modes, such as normal, emergency, or overload. The response varies depending on the current state of the system.

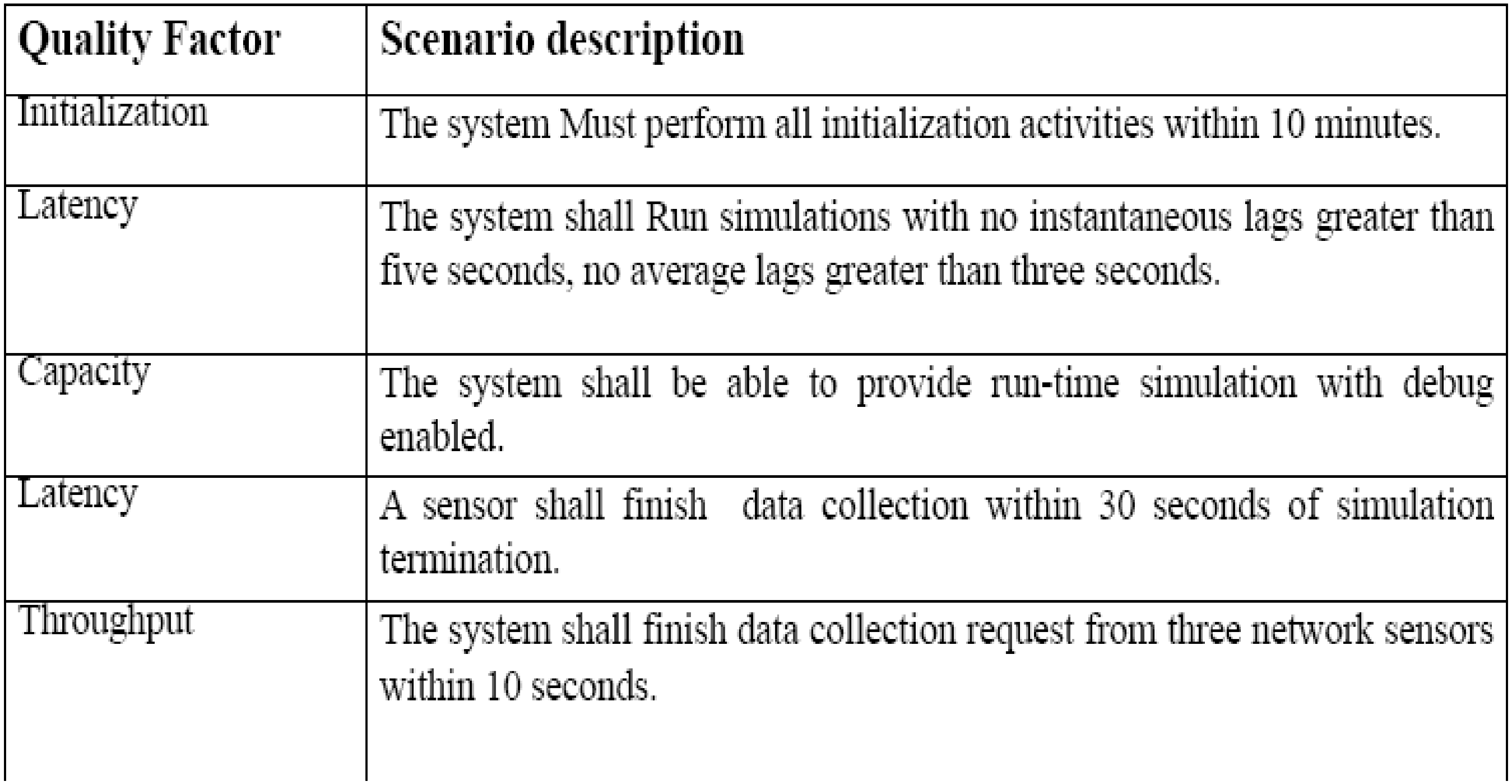
## PERFORMANCE SCENARIO

**Response.** The system must process the arriving events. This may cause a change in the system environment (e.g., from normal to overload mode). The response of the system can be characterized by:

- **latency** (the time between the arrival of the stimulus and the system's response to it),
- **deadlines** in processing (a specific action should take place before another),
- **throughput** of the system (e.g., the number of transactions the system can process in a second),
- **jitter** of the response (the variation in latency),
- **number of events not processed** because the system was too busy to respond,
- **lost data** because the system was too busy.

**Response measure.** Response measures include the time it takes to process the arriving events (latency, or deadlines by which the events must be processed), variations in this time (jitter), the number of events that can be processed within a particular time interval (throughput), and the characterization of the events that cannot be processed (miss rate, data loss).

## SCENARIO PROFILE FOR PERFORMANCE



| Quality Factor | Scenario description |
|---|---|
| Initialization | The system Must perform all initialization activities within 10 minutes. |
| Latency | The system shall Run simulations with no instantaneous lags greater than five seconds, no average lags greater than three seconds. |
| Capacity | The system shall be able to provide run-time simulation with debug enabled. |
| Latency | A sensor shall finish data collection within 30 seconds of simulation termination. |
| Throughput | The system shall finish data collection request from three network sensors within 10 seconds. |

## POS CASE STUDY

**Scenario**(s): The POS system scans a new item, item is looked up, total price updated within two seconds

**Stimulus Source** : End user

**Stimulus**: Scan item, fixed time between events for limited time period

**Environment**: Development time

**Artefact** (If Known): POS system

**Response**: Item is looked up, total price updated

**Response Measure**: Within two seconds

## THROUGHPUT

Measure of the amount of work an application must perform in unit time

- Transactions per second (TPS)
- Messages per second (MPS)

Is required throughput:

- Average?
- Peak?

Many system have low average but high peak throughput requirements

## RESPONSE TIME

- Measure of the latency an application exhibits in processing a request
- Usually measured in (milli)seconds
- Often an important metric for users
- Is required response time:
  - Guaranteed?
  - Average?

E.g. 95% of responses in sub-4 seconds, and all within 10 seconds

## 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 often associated with batch jobs in IT systems.

## ATTENTION!

What is a

- Transaction?
- Message?
- Request?

All are application specific measures.

Ex. System must achieve 100 mps throughput

- BAD!!

System must achieve 100 mps peak throughput for *PaymentReceived* messages

- GOOD!!!

## FACTORS AFFECTING PERFORMANCE

- Resource Consumption
- Blocked time
  - Contention for resources
  - Availability of resources
  - Dependency on other computation

## TACTICS TO ACHIEVE PERFORMANCE

- resource demand,
- resource management
- resource arbitration.

## RESOURCE DEMAND

**Increase computational efficiency.**

- Improving the algorithms,
- Trading one resource for another.

**Reduce computational overhead.**

- Use local class vs. RMI
- Use of intermediaries => the classic flexbility/performance tradeoff.

## RESOURCE DEMAND - REDUCE THE NUMBER OF PROCESSED EVENTS

**Manage event rate.**

- Reduce the sampling frequency at which environmental variables are monitored

**Control frequency of sampling.**

- If there is no control over the arrival of externally generated events, queued requests can be sampled at a lower frequency, possibly resulting in the loss of requests.

## RESOURCE DEMAND – CONTROL THE USE OF RESOURCES

**Bound execution times.**

- Place a limit on how much execution time is used to respond to an event.

**Bound queue sizes.**

- This controls the maximum number of queued arrivals and consequently the resources used to process the arrivals.

## RESOURCE MANAGEMENT

**Introduce concurrency.**

- process different streams of events on different threads
- create additional threads to process different sets of activities.
- appropriately allocate the threads to resources (load balancing)

**Maintain multiple copies of either data or computations.**

- clients in a client-server pattern are replicas of the computation.
- caching is a tactic in which data is replicated,
- keeping the copies consistent and synchronized becomes a responsibility that the system must assume.

**Increase available resources.**

- faster processors, additional processors, additional memory, faster networks.

## RESOURCE ARBITRATION - SCHEDULING

**First In, First Out (FIFO).**

- queues that treat all requests equally (all have the same priority). Requests are ordered by time of arrival.

**Fixed priority.**

- assign to each resource requester a priority, and treat the requests issued by high-priority requesters first.
- Strategies:
  - **semantic importance**. Priority is assigned based on some domain characteristic.
  - **deadline monotonic.** This is a static strategy that assigns higher priority to requesters with shorter deadlines.
  - **rate monotonic.** This is a static strategy for periodic requesters; higher priority is assigned to requesters with shorter periods.

## RESOURCE SCHEDULING

**Dynamic priority.**

- ordering according to a criterion and then, at every assignment possibility, assign the resource to the next request in that order.
- earliest deadline: the priority is assigned to the pending request with the earliest deadline.

**Static scheduling.**

- determine the sequence of assignment offline.

## SECURITY

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

An attempt to breach security is called an attack

## SECURITY FEATURES

**Nonrepudiation** - a transaction (access to or modification of data or services) cannot be denied by any of the parties to it.

**Confidentiality** - data or services are protected from unauthorized access.

**Integrity** - data or services are being delivered as intended.

**Assurance** - the parties to a transaction are who they purport to be.

**Availability** - the system will be available for legitimate use.

**Auditing** - the system tracks activities within it at levels sufficient to reconstruct them.

## SECURITY SCENARIO

**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** -
- online or offline,
- connected or disconnected,
- firewalled or open

## SECURITY SCENARIO CONTINUED

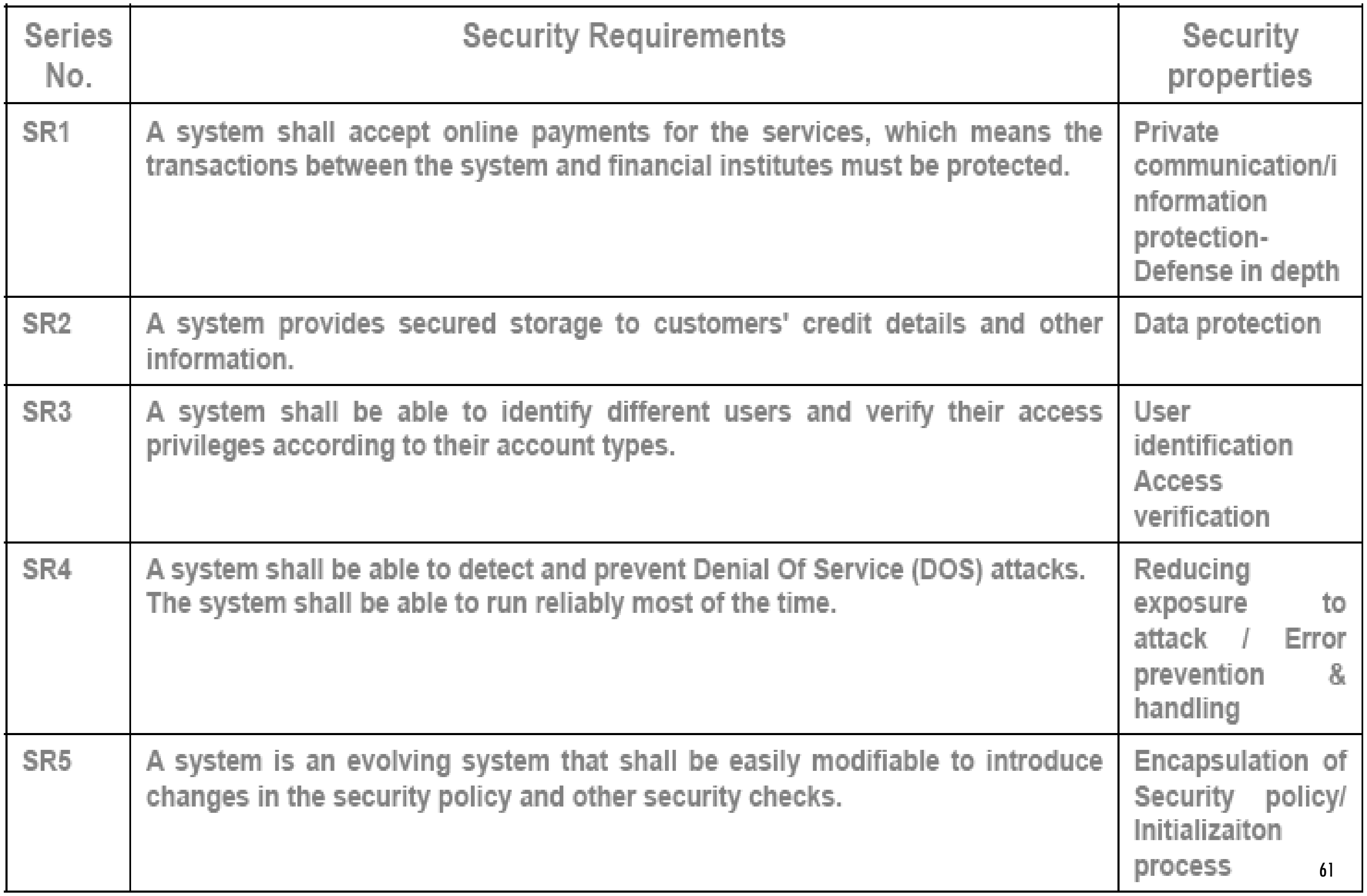
**Response**

- 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**

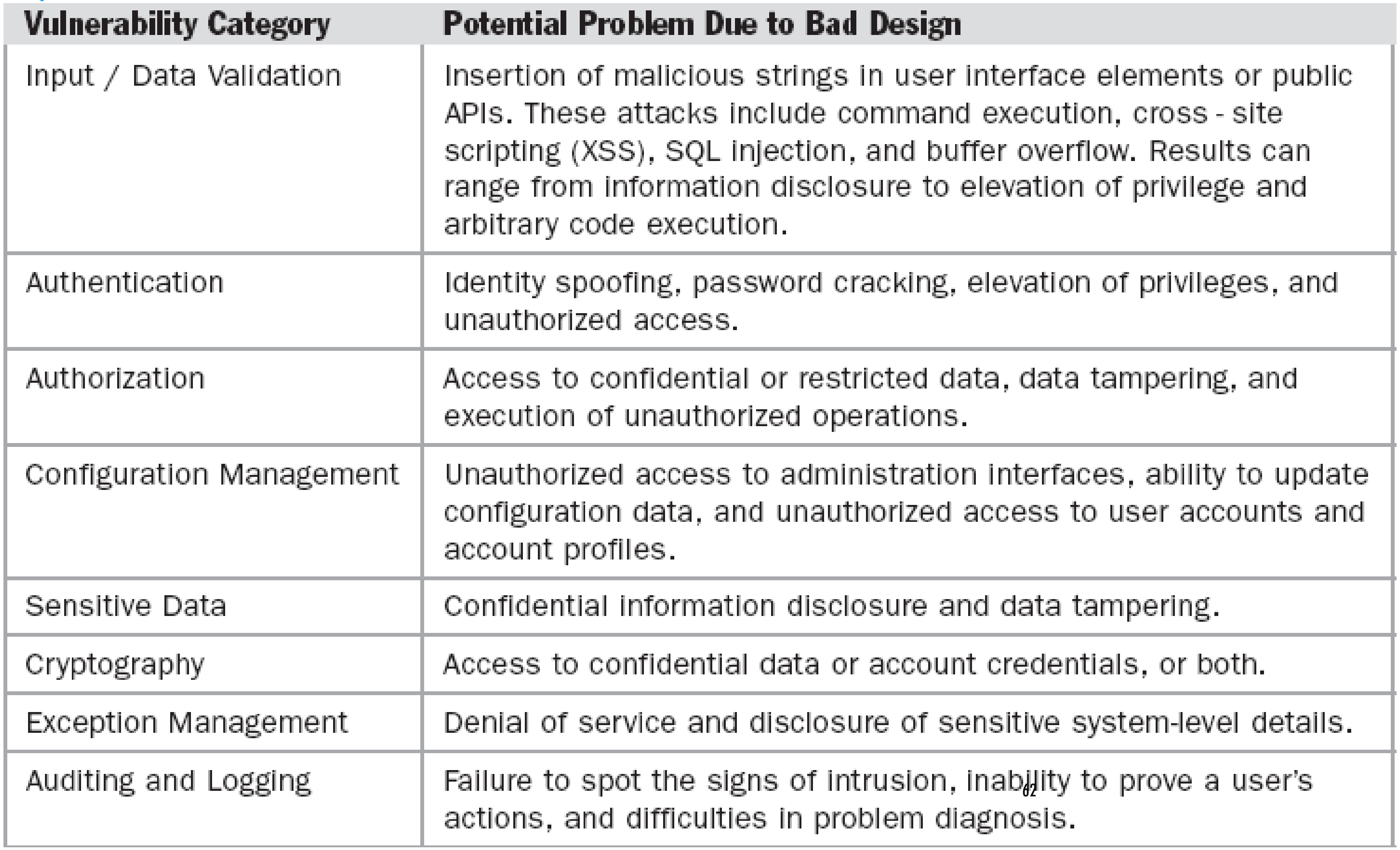
- Time/effort/resources required to circumvent security measures with probability of success; probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied

## SECURITY SCENARIOS



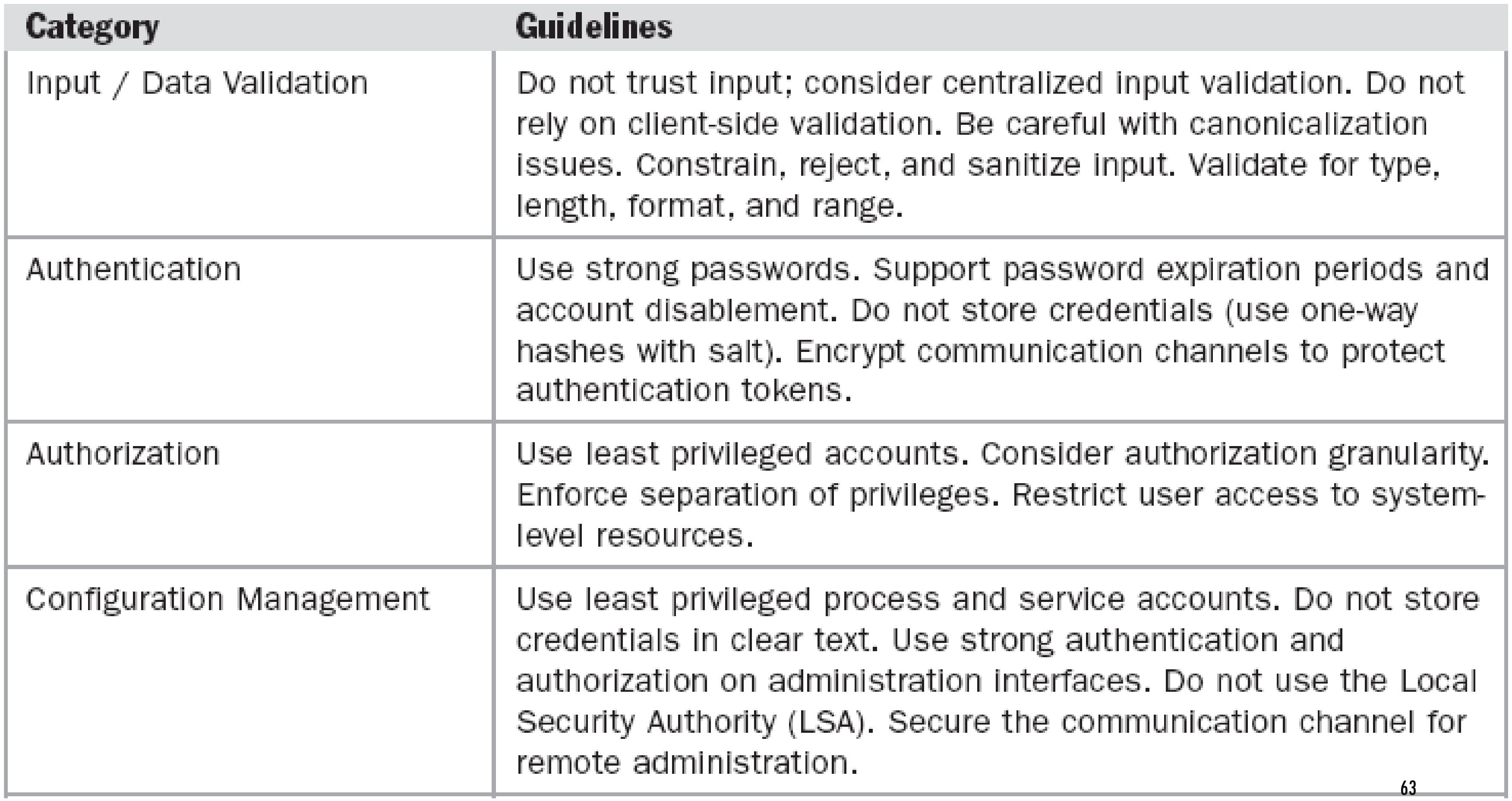
| Series No. | Security Requirements | Security properties |
|---|---|---|
| SR1 | A system shall accept online payments for the services, which means the transactions between the system and financial institutes must be protected. | Private communication/information protection-Defense in depth |
| SR2 | A system provides secured storage to customers' credit details and other information. | Data protection |
| SR3 | A system shall be able to identify different users and verify their access privileges according to their account types. | User identification Access verification |
| SR4 | A system shall be able to detect and prevent Denial Of Service (DOS) attacks. The system shall be able to run reliably most of the time. | Reducing exposure to attack / Error prevention & handling |
| SR5 | A system is an evolving system that shall be easily modifiable to introduce changes in the security policy and other security checks. | Encapsulation of Security policy/ Initializaiton process |

## DEALING WITH SECURITY RISKS



| Vulnerability Category | Potential Problem Due to Bad Design |
|---|---|
| Input / Data Validation | Insertion of malicious strings in user interface elements or public APIs. These attacks include command execution, cross - site scripting (XSS), SQL injection, and buffer overflow. Results can range from information disclosure to elevation of privilege and arbitrary code execution. |
| Authentication | Identity spoofing, password cracking, elevation of privileges, and unauthorized access. |
| Authorization | Access to confidential or restricted data, data tampering, and execution of unauthorized operations. |
| Configuration Management | Unauthorized access to administration interfaces, ability to update configuration data, and unauthorized access to user accounts and account profiles. |
| Sensitive Data | Confidential information disclosure and data tampering. |
| Cryptography | Access to confidential data or account credentials, or both. |
| Exception Management | Denial of service and disclosure of sensitive system-level details. |
| Auditing and Logging | Failure to spot the signs of intrusion, inability to prove a user's actions, and difficulties in problem diagnosis. |

# PRINCIPLES FOR SECURITY STRATEGIES



| Category | Guidelines |
|---|---|
| Input / Data Validation | Do not trust input; consider centralized input validation. Do not rely on client-side validation. Be careful with canonicalization issues. Constrain, reject, and sanitize input. Validate for type, length, format, and range. |
| Authentication | Use strong passwords. Support password expiration periods and account disablement. Do not store credentials (use one-way hashes with salt). Encrypt communication channels to protect authentication tokens. |
| Authorization | Use least privileged accounts. Consider authorization granularity. Enforce separation of privileges. Restrict user access to system-level resources. |
| Configuration Management | Use least privileged process and service accounts. Do not store credentials in clear text. Use strong authentication and authorization on administration interfaces. Do not use the Local Security Authority (LSA). Secure the communication channel for remote administration. |

## PRINCIPLES FOR SECURITY STRATEGIES

| | |
|---|---|
| Sensitive Data | Avoid storing secrets. Encrypt sensitive data over the wire. Secure the communication channel. Provide strong access controls for sensitive data stores. |
| Cryptography | Do not develop your own. Use proven and tested platform features. Keep unencrypted data close to the algorithm. Use the right algorithm and key size. Avoid key management (use DPAPI). Cycle your keys periodically. Store keys in a restricted location. |
| Exception Management | Use structured exception handling. Do not reveal sensitive application implementation details. Do not log private data such as passwords. Consider a centralized exception management framework. |
| Auditing and Logging | Identify malicious behavior. Know what good traffic looks like. Audit and log activity through all of the application tiers. Secure access to log files. Back up and regularly analyze log files. |

## SECURITY TACTICS

- Resisting attacks
- Detecting attacks
- Recovering from attacks

## RESISTING ATTACKS

**Authenticate users.**

- Ensure that a user or remote computer is actually who it purports to be. Passwords, one-time passwords, digital certificates, and biometric identifications provide authentication.

**Authorize users.**

- ensure that an authenticated user has the rights to access and modify either data or services. This is usually managed by providing some access control patterns within a system. Access control can be by user or by user class. Classes of users can be defined by user groups, by user roles, or by lists of individuals.

**Maintain data confidentiality.**

- encryption to data and to communication links. The link can be implemented by a virtual private network (VPN) or by a Secure Sockets Layer (SSL) for a Web-based link. Encryption can be symmetric (both parties use the same key) or asymmetric (public and private keys).

## RESISTING ATTACKS

**Maintain integrity.**

- Data should be delivered as intended. It can have redundant information encoded in it, such as checksums or hash results, which can be encrypted either along with or independently from the original data.

**Limit exposure.**

- The architect can design the allocation of services to hosts so that limited services are available on each host.

**Limit access.**

- Firewalls restrict access based on message source or destination port. It is not always possible to limit access to known sources. One configuration used in this case is the so-called demilitarized zone (DMZ).

## SECURITY TACTICS

**Detecting attacks**

- Intrusion detection system

**Recovering from attacks**

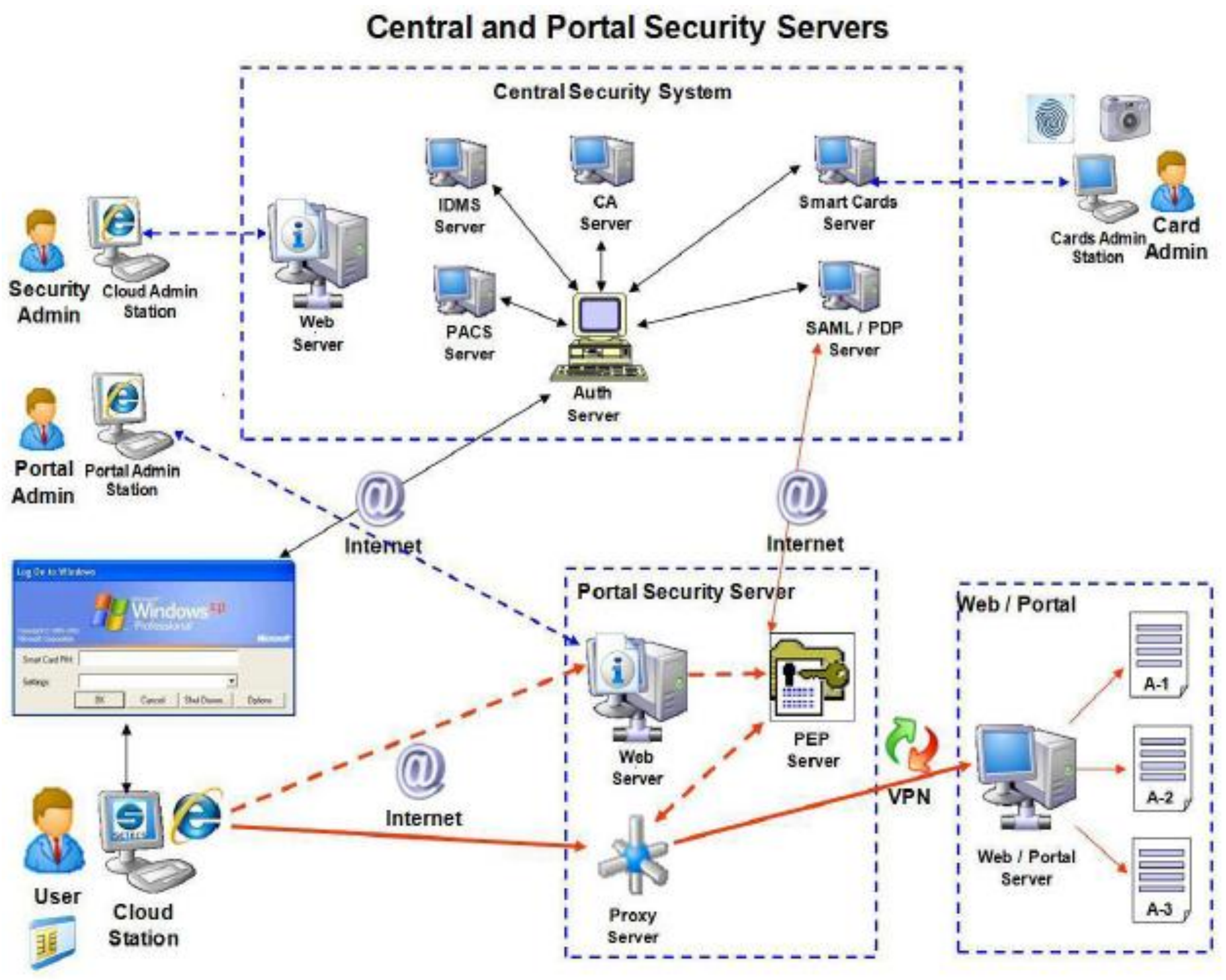
- restoring state (availability)
- attacker identification (nonrepudiation)

## CLOUD-BASED SECURITY VULNERABILITIES

Identified by CSA, NIST and ENISA

- Data Privacy and Reliability
- Data Integrity
- Authentication and Authorization
  - authentication frameworks like OpenID, SAML, Shibboleth

# CLOUD SECURITY INFRASTRUCTURE EXAMPLE



## OTHER QUALITY ATTRIBUTES

Modifiability

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

## TESTABILITY

**Source:** developer, tester, user

**Stimulus:** milestone completed

**Artifact:** design, code component, system

**Environment:** design, development, compile, deployment, run

**Response:** can be controlled and observed

**Measure:** coverage, probability, time

## EXAMPLE TESTABILITY SCENARIO

**Source:** Unit tester

**Stimulus:** Performs unit test

**Artifact:** Component of the system

**Environment:** At the completion of the component

**Response:** Component has interface for controlling behavior, and output of the component is observable

**Response Measure:** Path coverage of 85% is achieved within 3 hours

## USABILITY

**Source:** end user

**Stimulus:** wish to learn/use/minimize errors/adapt/feel comfortable

**Artifact:** system

**Environment:** configuration or runtime

**Response:** provide ability or anticipate

**Measure:** task time, number of errors, user satisfaction, efficiency

## LESSONS LEARNED

Requirements engineering is important and not trivial. Involves:

- Elicitation
- Specification
- Validation

Architecture is driven by requirements:

- Functional
- Non-functional (quality attributes)

Quality attributes

- Defined as scenarios
- Achieved using appropriate tactics

## DESIGN TRADE-OFFS

**QAs are rarely orthogonal**

- They interact, affect each other
- highly secure system may be difficult to integrate
- highly available application may trade-off lower performance for greater availability
- high performance application may be tied to a given platform, and hence not be easily portable

Architects must create solutions that makes **sensible design compromises**

- Not possible to fully satisfy all competing requirements
- Must satisfy all stakeholder needs
- This is the difficult bit!

## NEXT TIME

- Review
- Your questions