BITS Pilani

Software Architecture

Software Quality Attributes

Viswanathan Hariharan
Edited by Vijayarajan

Contents

- Recap
- Quality attributes and Tactics Continued
- Architecturally significant Requirements

avijayarajan@wilp.bits-pilani.ac.in

Assignment

A1 - Build an Architecture for an App

- The App should at minimum include the technologies Web, Mobile, IOT, cloud and analytics.
- Each team will select an application and get the approval of the TA.
- Duplication May not be allowed …
- The submitted architecture will be evaluated by the TA
- The team will be evaluated for their developed architecture

A2 – Research paper on real life architecture / latest trends etc

- Each team has to select a topic and get the approval of the faculty.
- Duplication may not be allowed.
- Final Paper should be submitted as per the agreed upon template and schedule

Schdule

Evolution of SW Architecture

We design and implement information systems to solve problems and process data.

As problems become larger and more complex and data becomes more voluminous, so do the associated information systems

- Structured programming, Data Structure, Higher Level languages, software engineering, Object Oriented etc

Computing become Distributed, on the cloud, Mobile as a front end

As the problem size and complexity increase, algorithms and data structures become less important than getting the right structure for the information system.

Specifying the right structure of the information system becomes a critical design problem itself

< Example from Construction Industry>

Importance of Quality attributes

- Functional requirements help us to define the modules
- Quality attributes help us to structure the system

Importance of Quality attributes

- If we have to design an ERP system, we can design the different modules based on functional requirements
- But if the goal is to have a portable software that can be easily ported to other operating systems, then we need to have a layer that shields us from variations in OS

Quality attributes & Architecture

- Input for Architecture are:
  - Business goals
  - Functional requirements
  - Non-functional requirements or Quality attributes such as response time needs, system uptime needs, security needs, etc.
  - Constraints such as choice of technology (client server, Web, Cloud), language, availability of staff, external systems (Mainframe) with which it needs to interact, etc.

Examples of quality attributes

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

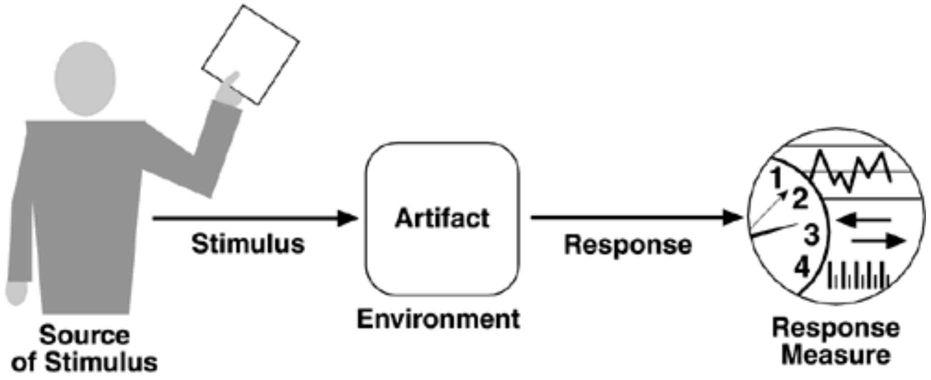
Scalability

Quality Attribute Scenario

Quality attribute requirements are expressed with Scenarios

Quality attribute Scenario Framework

Source of stimulus. entity (human, computer system, or any other actuator) that generated the stimulus
Stimulus – is a condition that requires a response when it arrives at a system



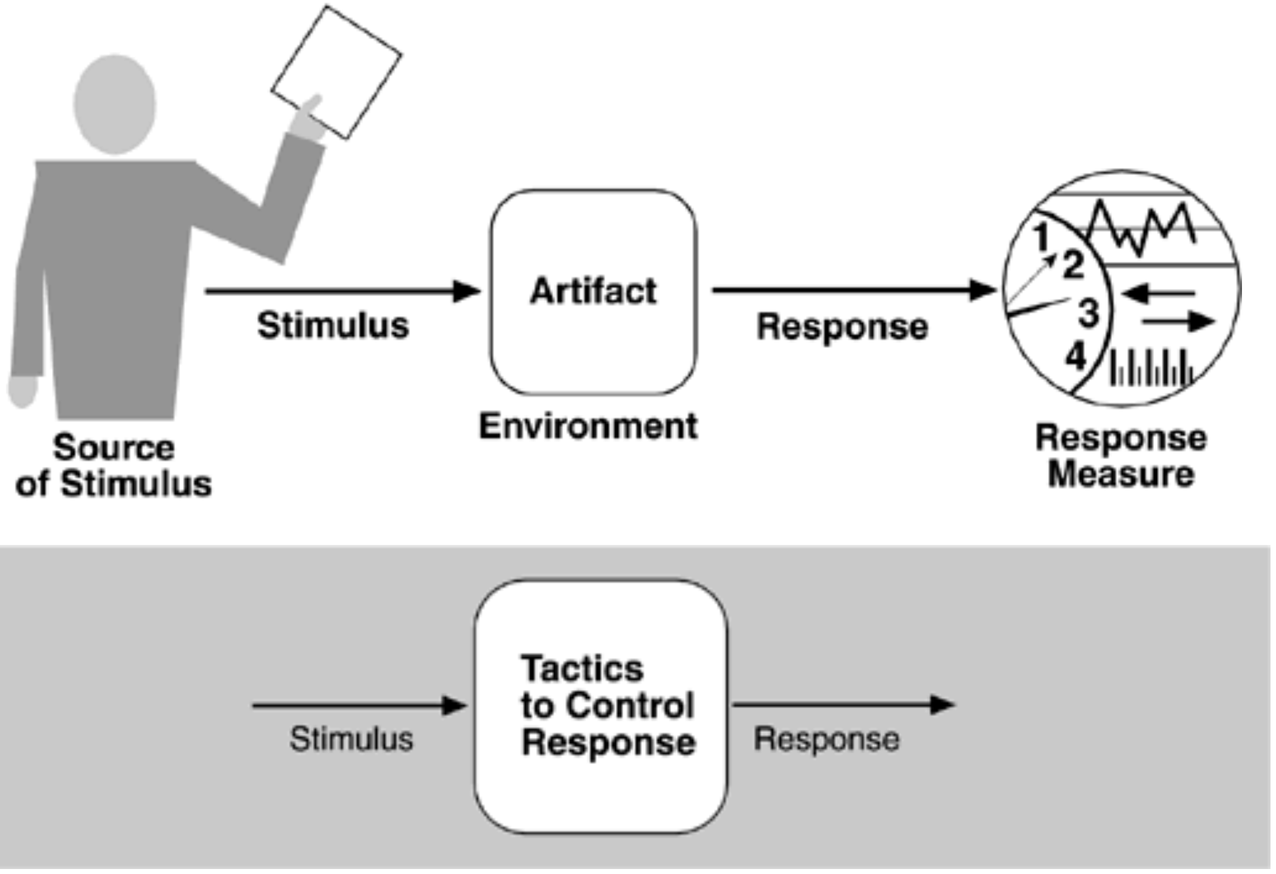
Environment: The stimulus occurs within certain conditions - Normal, safe mode, overload condition etc

Artifact: Some artifact is stimulated – whole system or parts

Response: activity undertaken after the arrival of the stimulus.

Response measure: Measurable outcome, it should be measurable in some fashion so that the requirement can be tested.

Design Tactics



Tactic is a design option for the architect

Testability

Testability deals with reducing the testing effort

Industry estimates indicate that between 30 and 50 percent (or in some cases, even more) of the cost of developing well-engineered systems is taken up by test- ing. If the software architect can reduce this cost, the payoff is large.

Testability

What are some of the issues that lead to increased testing effort or lead to delay in testing ?

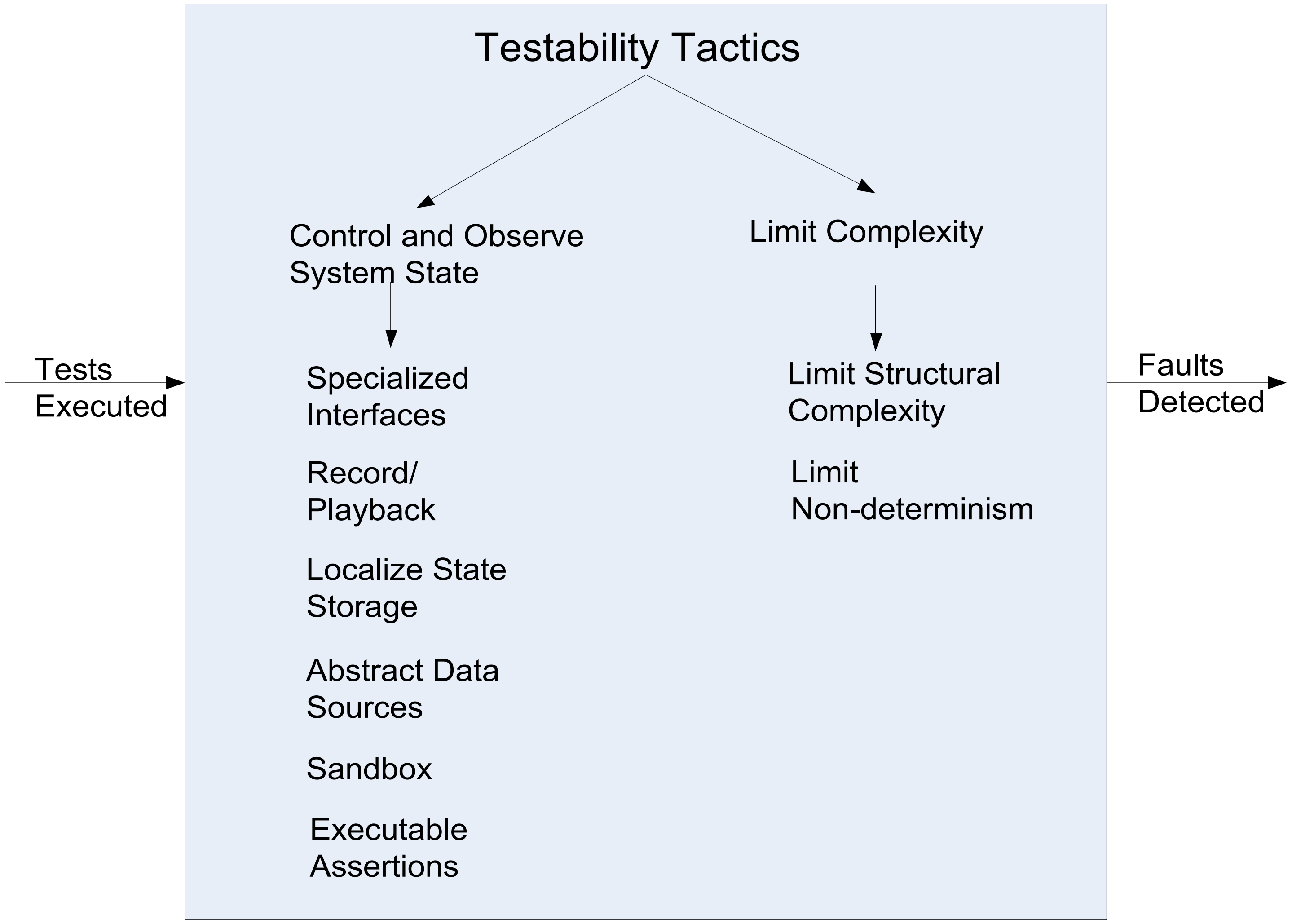
innovate achieve lead

Example

Fetal Lite Video

For a system to be properly testable, it must be possible to control each component’s inputs (and possibly manipulate its internal state) and then to observe its outputs (and possibly its internal state, either after or on the way to computing the outputs).

Testability Tactics



Control and Observe System State

Specialized Interfaces: to control or capture variable values for a component either through a test harness or through normal execution.

Record/Playback: capturing information crossing an interface and using it as input for further testing.

Localize State Storage: To start a system, subsystem, or module in an arbitrary state for a test, it is most convenient if that state is stored in a single place.

Control and Observe System State

Abstract Data Sources: Abstracting the interfaces lets you substitute test data more easily.

Sandbox: isolate the system from the real world to enable experimentation that is unconstrained by the **worry** about having to undo the consequences of the experiment.

Executable Assertions: assertions are (usually) hand coded and placed at desired locations to indicate when and where a program is in a faulty state.

Limit Complexity

Limit Structural Complexity:

- avoiding or resolving cyclic dependencies between components,
- isolating and encapsulating dependencies on the external environment,

Limit Non-determinism: finding all the sources of non-determinism, such as unconstrained parallelism, and weeding them out as far as possible. (parallel threads!)

Security

Security is a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized

Security

What are the common security issues in information systems?

Security

Security has three main characteristics, called CIA:

Confidentiality is the property that data or services are protected from unauthorized access. For example, a hacker cannot access your income tax returns on a government computer.

Integrity is the property that data or services are not subject to unauthorized manipulation. For example, your grade has not been changed since your instructor assigned it.

Availability is the property that the system will be available for legitimate use. For example, a denial-of-service attack won't prevent you from ordering a book from an online bookstore.

Other characteristics that support CIA are

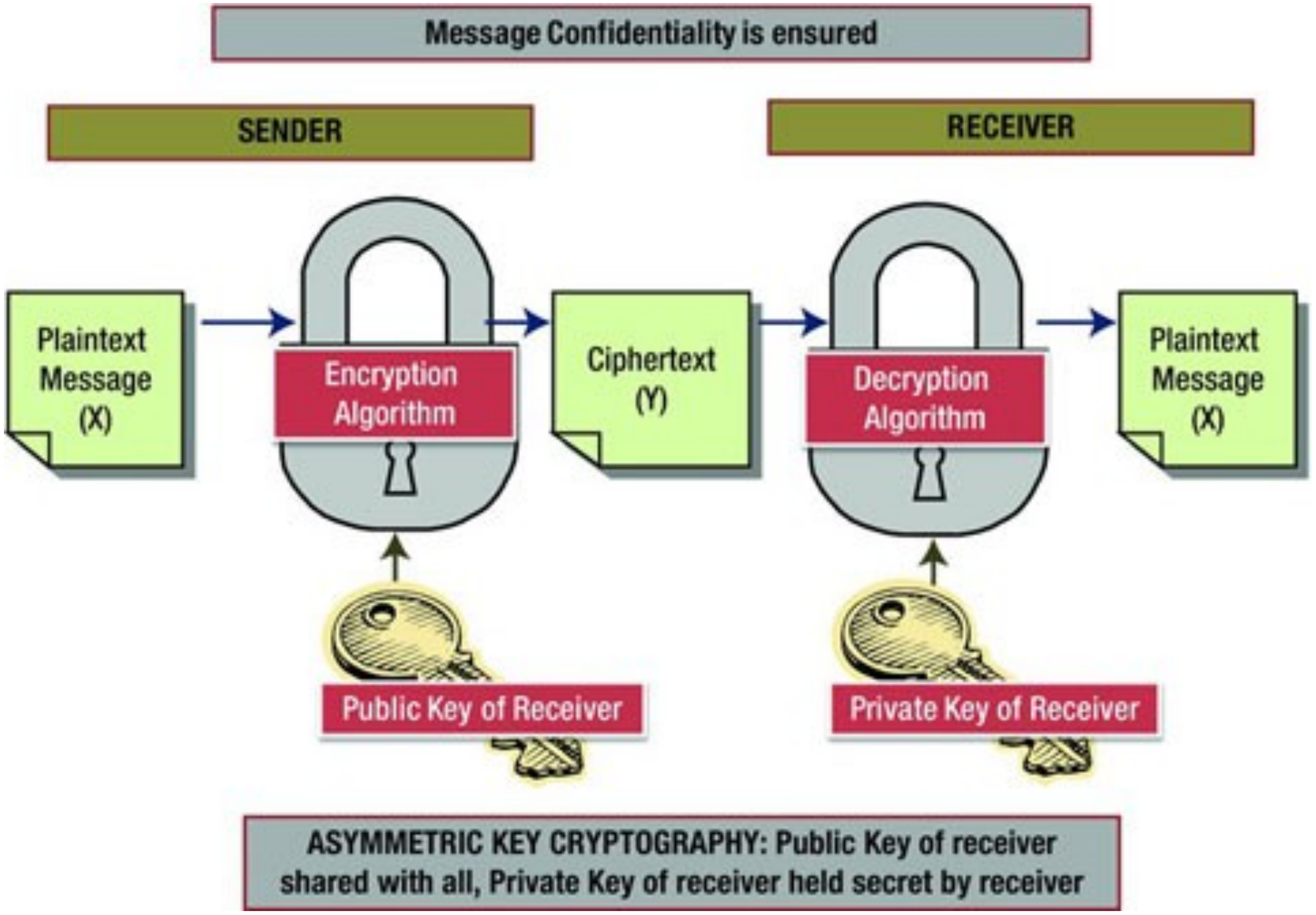
Authentication verifies the identities of the parties to a transaction and checks if they are truly who they claim to be. For example, when you get an e-mail purporting to come from a bank, authentication guarantees that it actually comes from the bank.

Nonrepudiation guarantees that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message. For example, you cannot deny ordering something from the Internet, or the merchant cannot disclaim getting your order.

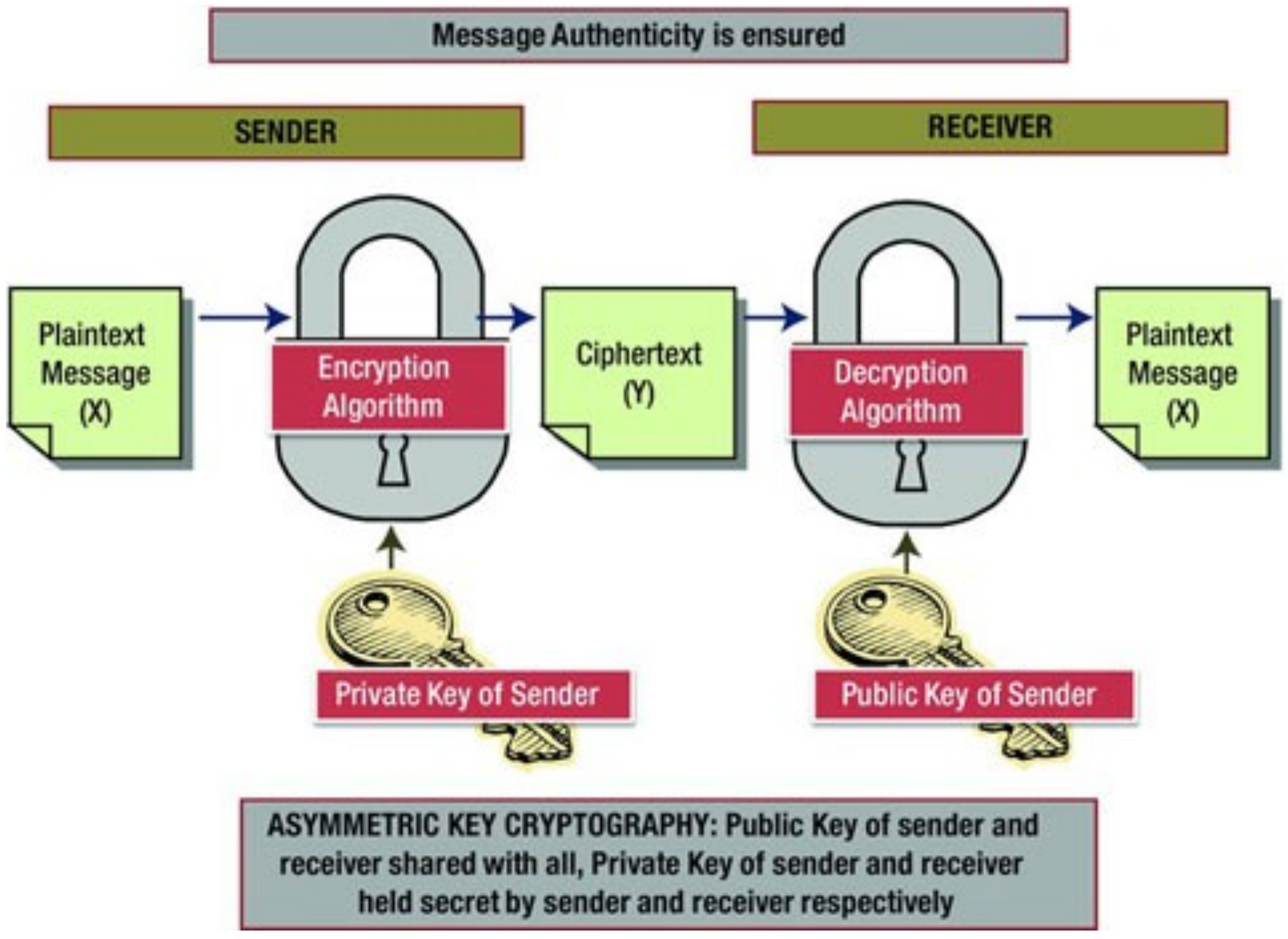
Authorization grants a user the privileges to perform a task. For example, an online banking system authorizes a legitimate user to access his account.

innovate achieve lead

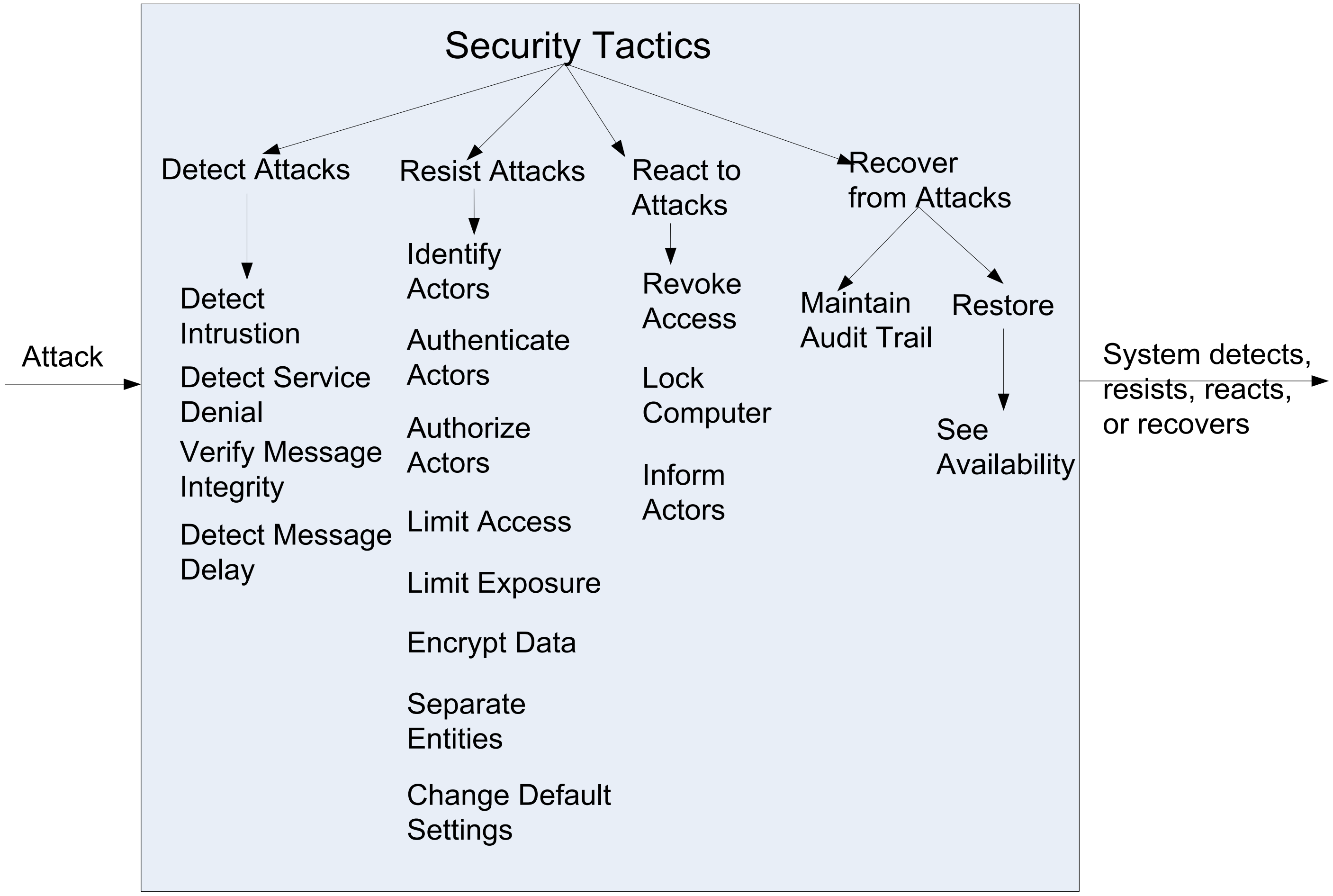
Confidentiality



Authenticity


Message Authenticity is ensured
SENDER
RECEIVER
Plaintext Message (X)
Encryption Algorithm
Ciphertext (Y)
Decryption Algorithm
Plaintext Message (X)
Private Key of Sender
Public Key of Sender
ASYMMETRIC KEY CRYPTOGRAPHY: Public Key of sender and receiver shared with all, Private Key of sender and receiver held secret by sender and receiver respectively

Security Tactics



Detect Attacks

Detect Intrusion: compare network traffic or service request patterns *within* a system to a set of signatures or known patterns of malicious behavior stored in a database.

Detect Service Denial: comparison of the pattern or signature of network traffic *coming into* a system to historic profiles of known Denial of Service (DoS) attacks.

Verify Message Integrity: use techniques such as checksums or hash values to verify the integrity of messages, resource files, deployment files, and configuration files.

Detect Message Delay: checking the time that it takes to deliver a message, it is possible to detect suspicious timing behavior.

Resist Attacks

Identify Actors: identify the source of any external input to the system.

Authenticate Actors: ensure that an actor (user or a remote computer) is actually who or what it purports to be.

Authorize Actors: ensuring that an authenticated actor has the rights to access and modify either data or services.

Limit Access: limiting access to resources such as memory, network connections, or access points.

Resist Attacks

Limit Exposure: minimize the attack surface of a system by having the fewest possible number of access points.

Encrypt Data: apply some form of encryption to data and to communication.

Separate Entities: can be done through physical separation on different servers attached to different networks, the use of virtual machines, or an “air gap”.

Change Default Settings: Force the user to change settings assigned by default.

React to Attacks

Revoke Access: limit access to sensitive resources, even for normally legitimate users and uses, if an attack is suspected.

Lock Computer: limit access to a resource if there are repeated failed attempts to access it.

Inform Actors: notify operators, other personnel, or cooperating systems when an attack is suspected or detected.

Recover From Attacks

In addition to the Availability tactics for recovery of failed resources there is Audit.

Audit: keep a record of user and system actions and their effects, to help trace the actions of, and to identify, an attacker.

Usability

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of support it provides to users

Dimensions of Usability

- **Learnability**: How easy is it for users to accomplish basic tasks the first time they encounter the SW?
- **Efficiency**: Once users have learned the SW, how quickly can they perform tasks?
- **Memorability**: When users return to the SW after a period of not using it, how easily can they re-establish proficiency?
- **Errors**: how easily can users recover from the errors?
- **Satisfaction**: How pleasant is it to use the SW?

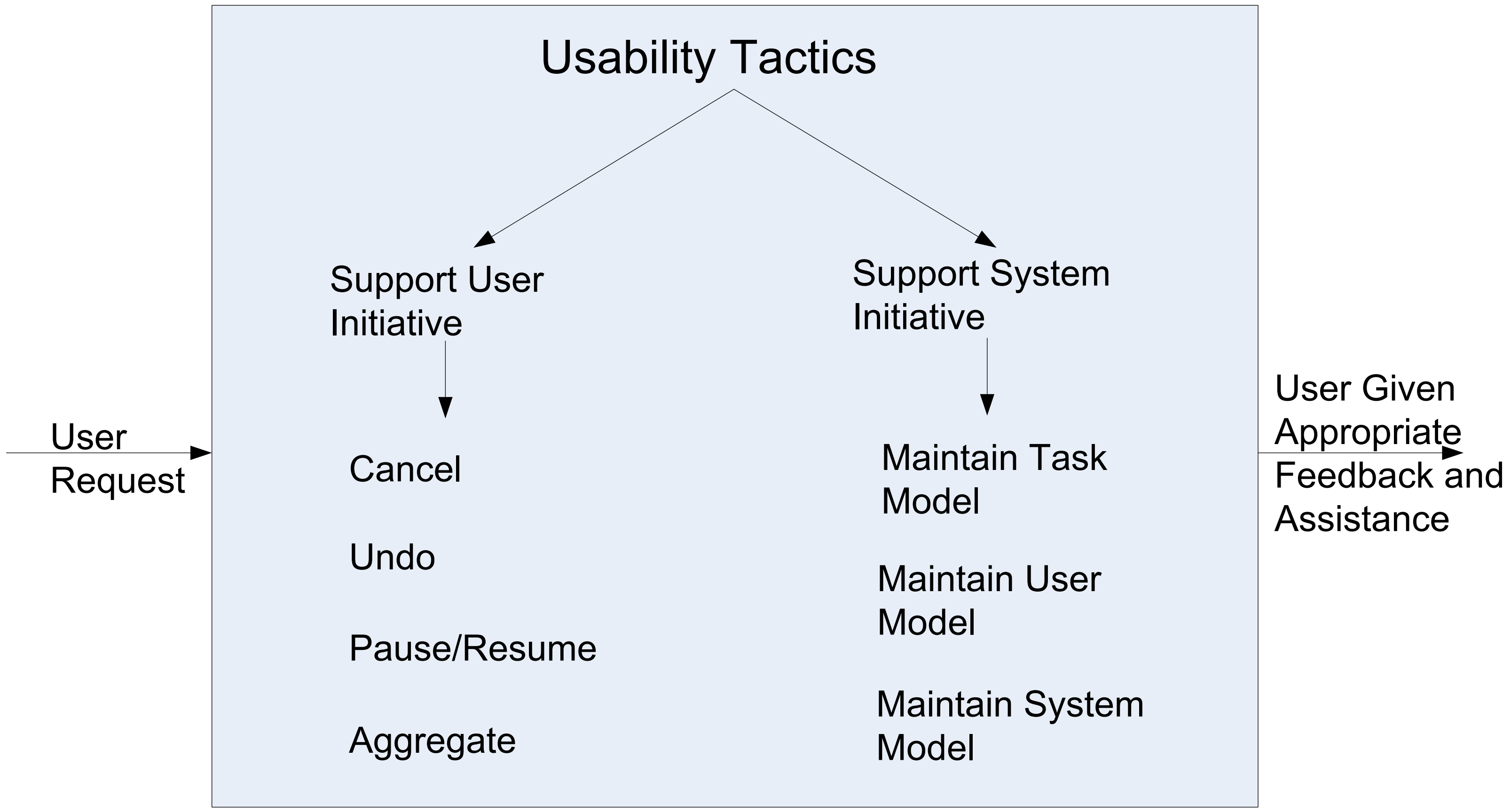
Usability

What are the ways in which we can make the User interface easy to use?

Usability Scenarios

- Aggregating Data
- Aggregating Commands
- Cancelling Commands
- Using Applications Concurrently
- Checking for Correctness
- Maintaining Device Independence
- Evaluating the System (tester)
- Recovering from Failure
- Retrieving Forgotten Passwords
- Providing Good Help
- Reusing Information
- Supporting International Use
- Leveraging Human Knowledge
- Modifying user Interfaces
- Supporting Multiple Activities
- Navigating Within a Single View
- Observing System State
- Working at the User’s Pace
- Predicting Task Duration
- Supporting Comprehensive Searching
- Supporting Undo
- Working in an Unfamiliar Context (novice)
- Verifying Resources (disk space? Battery?)
- Operating Consistently Across Views
- Making Views Accessible
- Supporting Visualization (print layout)

Usability Tactics



Support User Initiative

Cancel: the system must listen for the cancel request; the command being canceled must be terminated; resources used must be freed; and collaborating components must be informed.

Pause/Resume: temporarily free resources so that they may be re-allocated to other tasks.

Undo: maintain a sufficient amount of information about system state so that an earlier state may be restored, at the user's request.

Aggregate: ability to aggregate lower-level objects into a group, so that a user operation may be applied to the group, freeing the user from the drudgery.

Support System Initiative

Maintain Task Model: determines context so the system can have some idea of what the user is attempting and provide assistance. (capital letter)

Maintain User Model: explicitly represents the user's knowledge of the system, the user's behavior in terms of expected response time, etc. (configuration, running help)

Maintain System Model: system maintains an explicit model of itself. This is used to determine expected system behavior so that appropriate feedback can be given to the user. (progress bar)

Other quality attributes

- **Interoperability**
- **Scalability**
- Portability
- Deployability
- Monitorability

Design trade-offs

Examples

- Modifiability vs Performance: Too much modularity may impact performance as the request passes through multiple modules
- Security vs performance: High security in the form of encryption will delay response to end users
- Availability vs Testability: If we introduce redundancy to improve availability, we need to have mechanism to test if the system is recovering from faults

Summary

- Architecture is determined not only by functionality but also quality attributes and constraints

- We looked at some important quality attributes (QA) & corresponding tactics

- Trying to satisfy one QA may impact another QA. Hence we need to prioritize the QA

Exercise: Identify one dominant quality attribute (QA) of systems given below

System	Dominant Quality attribute to be addressed
IRCTC	
YouTube	
Flipkart.com	
OnlineSBI.com	
Uber	

Exercise: Identify one dominant quality attribute (QA) of systems given below

System	Dominant Quality attribute to be addressed
IRCTC	Availability, Performance
YouTube	Performance
FlipKart.com	Availability, Performance, Usability
OnlineSBI.com	Security
Uber	Usability

Exercise: Identify a tactic to address the Quality Attribute

System	Dominant Quality attribute to be addressed	Tactics
IRCTC	Availability, Performance	
YouTube	Performance	
Flipkart.com	Availability, Performance, Usability	
OnlineSBI.com	Security	
Uber	Usability	

Exercise: Identify a tactic to address the Quality Attribute

System	Dominant Quality attribute to be addressed	Tactics
IRCTC	Availability, Performance	Multiple servers
YouTube	Performance	Content Delivery Network (data replication)
Flipkart.com	Availability, Performance, Usability	Easy to search and buy (navigation)
OnlineSBI.com	Security	Encryption of sensitive data in DB, Digital certificate
Uber	Usability	Easy to book a cab and cancel