SS ZG653 (RL 6.1): Software Architecture

Security and Its Tactics

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

What is Security

A measure of the system's ability to resist unauthorized usage while still providing its services to legitimate users

- Ability to protect data and information from unauthorized access

An attempt to breach this is an "Attack"

- Unauthorized attempt to access, modify, delete data
  - Theft of money by e-transfer, modification records and files, reading and copying sensitive data like credit card number
- Deny service to legitimate users

Important aspects of Security

Security comprises of

Confidentiality

- prevention of the unauthorized disclosure of information. E.g. Nobody except you should be able to access your income tax returns on an online tax-filing site

Integrity

- prevention of the unauthorized modification or deletion of information. E.g. your grade has not been changed since your instructor assigned it

Availability

- prevention of the unauthorized withholding of information – e.g. DoS attack should not prevent you from booking railway ticket

Important aspects of Security

Non repudiation:: An activity (say a transaction) can't be denied by any of the parties involved. E.g. you cannot deny ordering something from the Internet, or the merchant cannot disclaim getting your order.

Assurance:: Parties in an activity are assured to be who they purport to be. Typically done through authentication. E.g. if you get an email purporting to come from a bank, it is indeed from a bank.

Auditing:: System tracks activities so that it can be reconstructed later

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

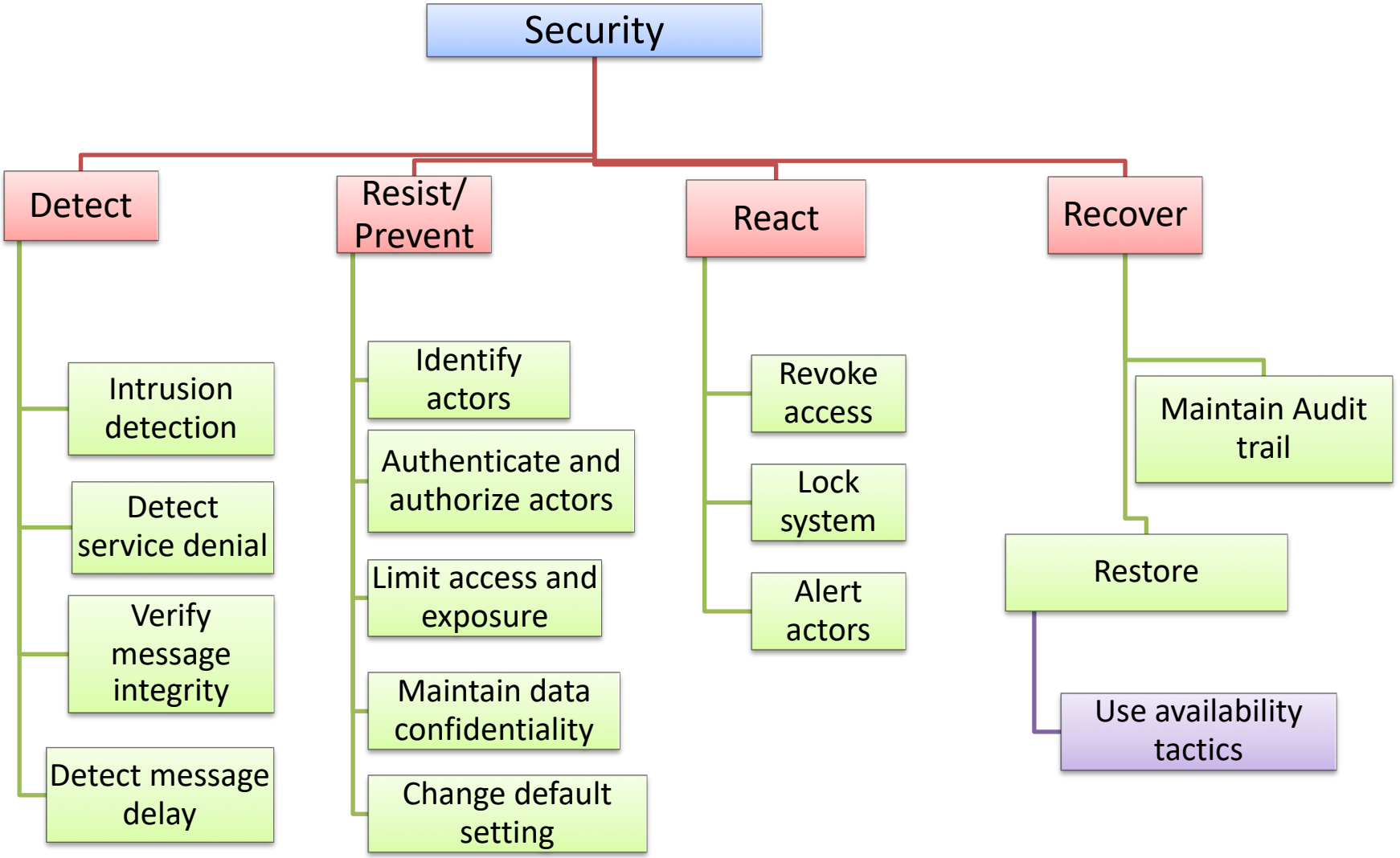
Security Scenario

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
• **Individual or a system** ✓Correctly identified OR ✓Incorrectly identified ✓OR unknown • **Who is** ✓Internal or external ✓Authorized or unauthrized • **Has access to** ✓Limited resource ✓Vast resource	•**Tries to** ✓Read data ✓Change/delete data ✓Access system services ✓Reduce availability of services	System services, data •Online/offline •Within firewall or Open	• **Authentication** ✓Authenticates ✓Hides identity • **Access control to data or services** ✓Blocks access ✓Grants/withdraws permission to access ✓Audits access/ modification attempts • **Corrective Actions** ✓Encrypts data ✓Detect anomalous situation (high access req.) and informs people/another system ✓Restricts availability	• Time required to circumvent security measure with Probability of success • Probability of detecting attack • Probability of detecting the individual responsible • %of services available during attack • Time to restore data/services • Extent of damage- how much data is vulnerable • No of access denials
An insider	Updates payment details	Pay database table	Anomaly detected Audit trail maintained	Corrective action taken in 1 day

Security Tactics- Close to Physical Security

- Detection:
  - Limit the access through security checkpoints
  - Enforces everyone to wear badges or checks legitimate visitors
- Resist
  - Armed guards
- React
  - Lock the door automatically
- Recover
  - Keep backup of the data in a different place

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
  - Compare the pattern or signature of network traffic *cominginto* a system to historic profiles of known Denial of Service (DoS) attacks.
- Verify Message Integrity
  - Use 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 & Authorize Actors:
  - Use strong passwords, OTP, digital certificates, biometric identity
  - Use access control pattern, define proper user class, user group, role based access
- Limit Access
  - Restrict access based on message source or destination ports
  - Use of DMZ

Resist Attacks

- Limit Exposure: minimize the attack surface of a system by allocating limited number of services to each hosts
- Data confidentiality:
  - Use encryption to encrypt data in database
  - User encryption based communication such as SSL for web based transaction
  - Use Virtual private network to communicate between two trusted machines
- 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.

Design Checklist- Allocation of Responsibilities

- Identify the services that needs to be secured
  - Identify the modules, subsystems offering these services
- For each such service
  - Identify actors which can access this service, and implement authentication and level of authorization for those
  - verify checksums and hash values
  - Allow/deny data associated with this service for these actors
  - record attempts to access or modify data or services
  - Encrypt data that are sensitive
  - Implement a mechanism to recognize reduced availability for this services
  - Implement notification and alert mechanism
  - Implement recover from an attack mechanism

Design Checklist- Manage Data

- Determine the sensitivity of different data fields
- 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.

Design Checklist- Manage Coordination

- For inter-system communication (applied for people also)
  - Ensure that mechanisms for authenticating and authorizing the actor or system, and encrypting data for transmission across the connection are in place.
- Monitor communication
  - Monitor anomalous communication such as
    - unexpectedly high demands for resources or services
    - Unusual access pattern
  - Mechanisms for restricting or terminating the connection.

Design Checklist- Manage Resource

- Define appropriate grant or denial of resources
- Record access attempts to resources
- Encrypt data
- Monitor resource utilization
  - Log
  - Identify suddenly high demand to a particular resource- for instance high CPU utilization at an unusual time
- 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.
  - .

Design checklist- Binding

- Runtime binding of components can be untrusted. Determine the following
  - Based on situation implement certificate based authentication for a component
    - Implement certification management, validation
  - Define access rules for components that are dynamically bound
  - Implement audit trail for whenever a late bound component tries to access records
  - System data should be encrypted where the keys are intentionally withheld for late bound components

Design Checklist- Technology choice

Choice of technology is often governed by the organization mandate (enterprise architecture)

- Decide tactics first. Based on the tactics, ensure that your chosen technologies support the tactics

- Determine what technology are available to help user authentication, data access rights, resource protection, data encryption
- Identify technology and tools for monitoring and alert

SS ZG653 (RL 6.2): Software Architecture

Testability and Its Tactics

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

What is Testability

The ease with which software can be made to demonstrate its faults through testing

If a fault is present in a system, then we want it to fail during testing as quickly as possible.

At least 40% effort goes for testing

- Done by developers, testers, and verifiers (tools)

Specialized software for testing

- Test harness
- Simple playback capability
- Specialized testing chamber

Testable Software

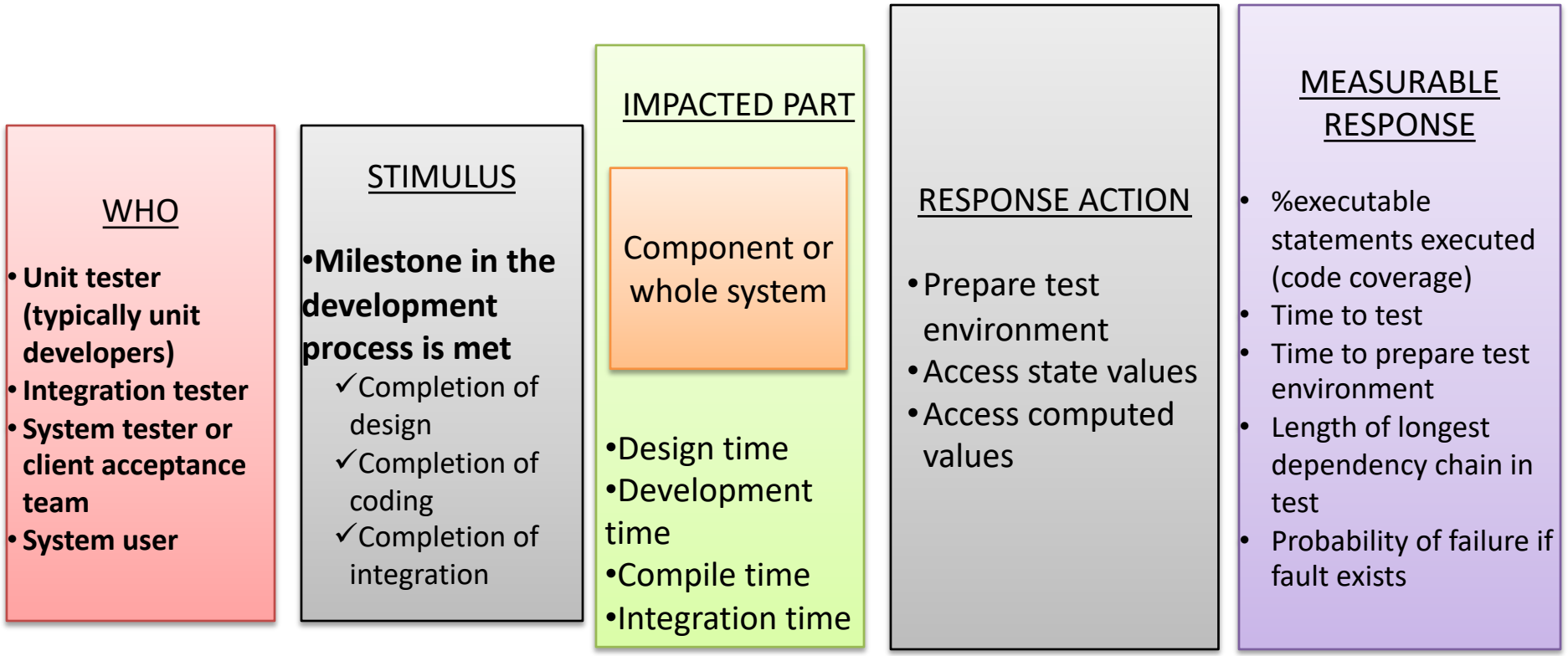
Dijkstra's Thesis

- Test can't guarantee the absence of errors, but it can only show their presence.
- Fault discovery is a probability
  - That the next test execution will fail and exhibit the fault

- A perfectly testable code – each component's internal state must be controllable through inputs and output must be observable

- Error-free software does not exist.

Testability Scenario



A unit tester

Performs unit test

Component that has controllable interface
After component is complete

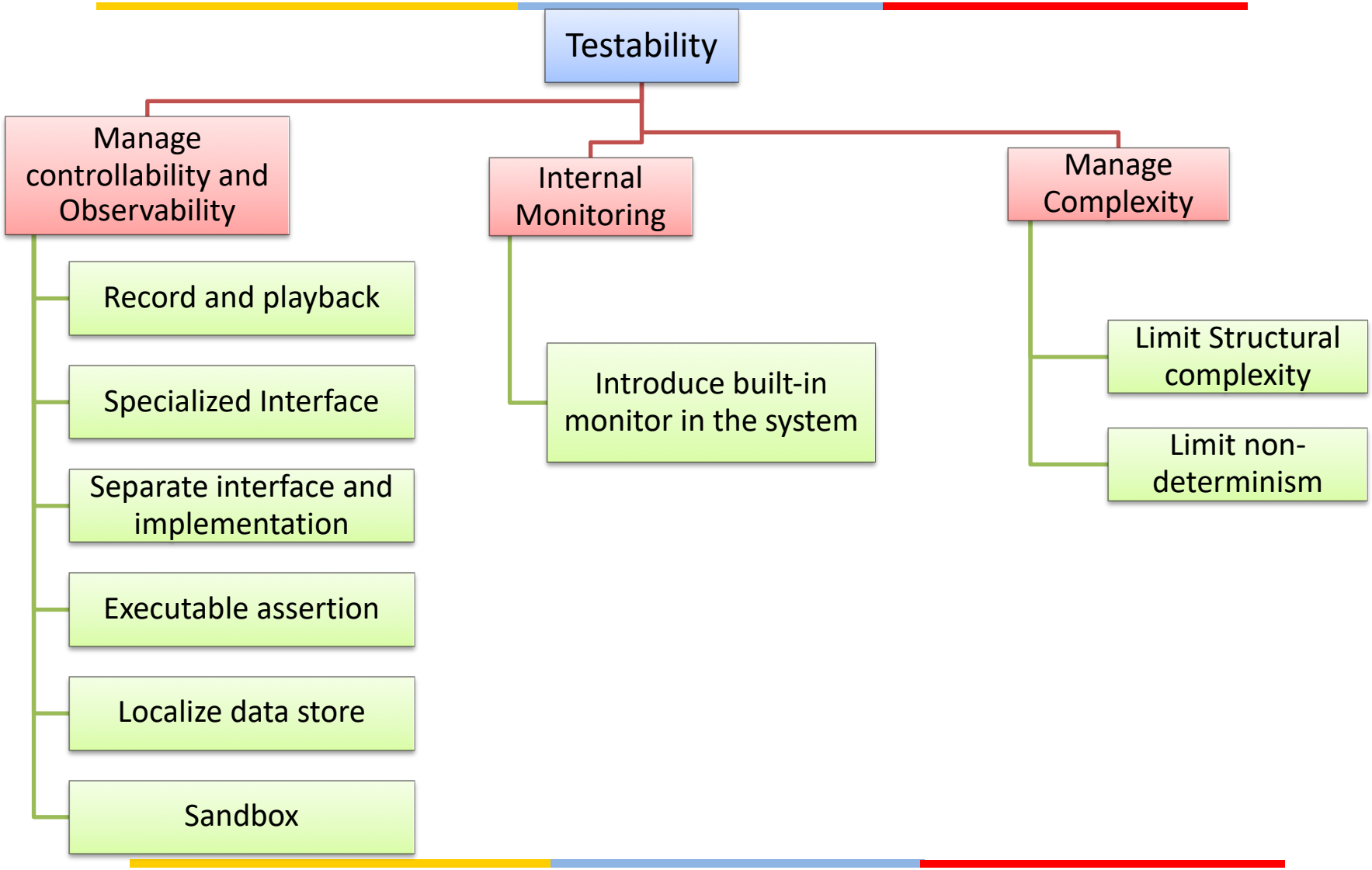
Observe the output for inputs provided

Coverage of 85% is achieved in 2 hours

Goal of Testability Tactics

- Using testability tactics the architect should aim to reduce the high cost of testing when the software is modified
- Two categories of tactics
  - Introducing controllability and observability to the system during design
  - The second deals with limiting complexity in the system's design

Testability Tactics



Control and Observe System State

- Specialized Interfaces for testing:
  - to control or capture variable values for a component either through a test harness or through normal execution.
  - Use a special interface that a test harness can use
  - Make use of some metadata through this special interface

- 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

- Interface and implementation
  - If they are separated, implementation can be replaced by a stub for testing rest of the system
- 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.

Manage Complexity

- Limit Structural Complexity:
  - avoiding or resolving cyclic dependencies between components,
  - isolating and encapsulating dependencies on the external environment
  - reducing dependencies between components in general.
- Limit Non-determinism: finding all the sources of non-determinism, such as unconstrained parallelism, and remove them out as far as possible.

Internal Monitoring

- Implement a built-in monitoring mechanism
  - One should be able to turn on or off
    - one example is logging
  - Performed typically by instrumentation- AOP, Preprocessor macro. Instrument the code to introduce recorder at some point

Design Checklist- Allocation of Responsibility

Identify the services are most critical and hence need to be most thoroughly tested.

- Identify the modules, subsystems offering these services

- For each such service
  - Ensure that internal monitoring mechanism like logging is well designed
  - Make sure that the allocation of functionality provides
    - low coupling,
    - strong separation of concerns, and
    - low structural complexity.

Design Checklist- Testing Data

- Identify the data entities that are related to the critical services need to be most thoroughly tested.

- Ensure that creation, initialization, persistence, manipulation, translation, and destruction of these data entities are possible--
  - State Snapshot: Ensure that the values of these data entities can be captured if required, while the system is in execution or at fault
  - Replay: Ensure that the desired values of these data entities can be set (state injection) during testing so that it is possible to recreate the faulty behavior

Design Checklist- Testing Infrastructure

- Is it possible to inject faults into the communication channel and monitoring the state of the communication
- Is it possible to execute test suites and capture results for a distributed set of systems?
- Testing for potential race condition- check if it is possible toexplicitly map
  - processes to processors
  - threads to processes

So that the desired test response is achieved and potential race conditions identified

Design Checklist- Testing resource binding

- Ensure that components that are bound later than compile time can be tested in the late bound context
  - E.g. loading a driver on-demand
- 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.

Design Checklist- 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 the environment in which the system will run
- Ensure that the system provides the means to:
  - test resource limits
  - capture detailed resource usage for analysis in the event of a failure
  - inject new resources limits into the system for the purposes of testing
  - provide virtualized resources for testing

Choice of Tools

- Determine what tools are available to help achieve the testability scenarios
  - Do you have regression testing, fault injection, recording and playback supports from the testing tools?
- Does your choice of tools support the type of testing you intend to carry on?
  - You may want a fault-injection but you need to have a tool that can support the level of fault-injection you want
  - Does it support capturing and injecting the data-state

SS ZG653 (RL 6.3): Software Architecture

Interoperability and Its Tactics

Instructor: Prof. Santonu Sarkar

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

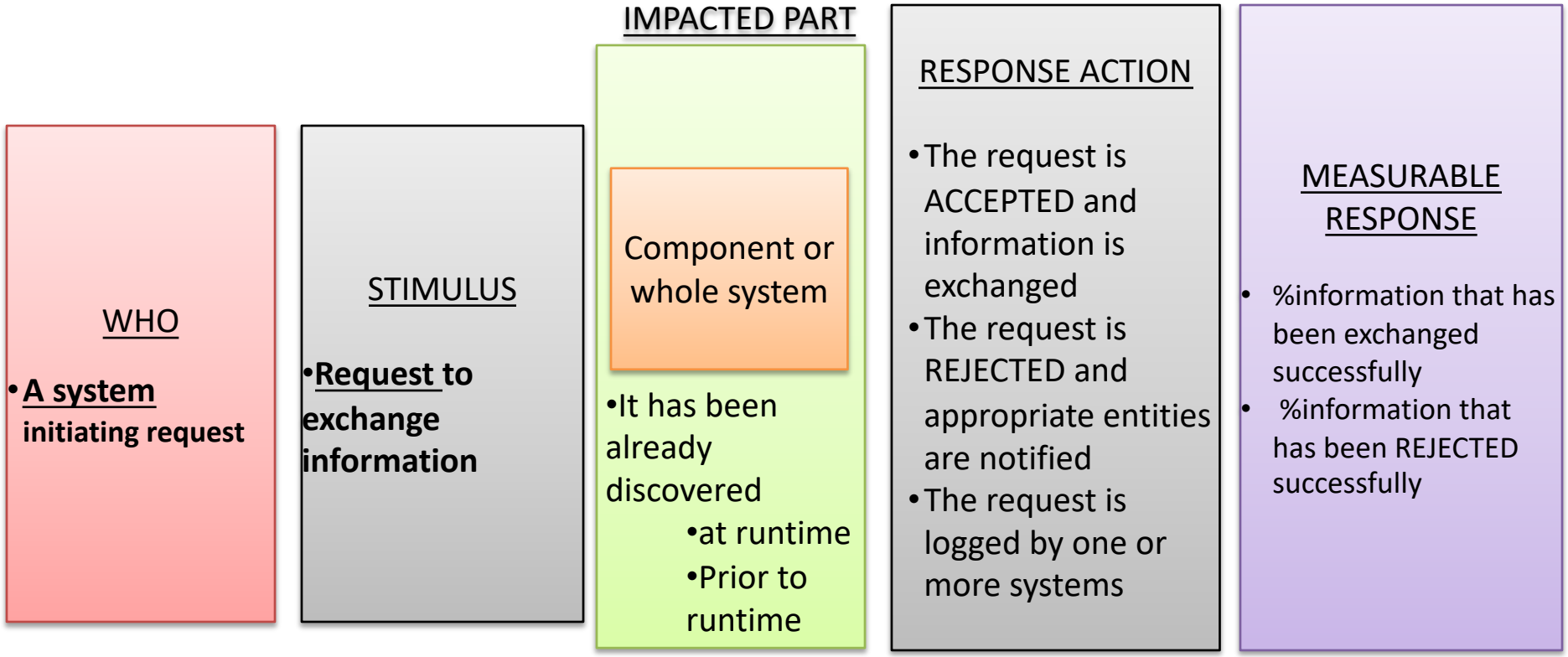
Interoperability

- Ability that two systems can usefully exchange information through an interface
  - Ability to transfer data (syntactic) and interpret data (semantic)
- Information exchange can be direct or indirect
- Interface
  - Beyond API
  - Need to have a set of assumptions you can safely make about the entity exposing the API
- Example- you want to integrate with Google Maps

Why Interoperate?

- The service provided by Google Maps are used by unknown systems
  - They must be able to use Google Maps w/o Google knowing who they can be
- You may want to construct capability from variety of systems
  - A traffic sensing system can receive stream of data from individual vehicles
  - Raw data needs to be processed
  - Need to be fused with other data from different sources
  - Need to decide the traffic congestion
  - Overlay with Google Maps

Interoperability Scenario



Avehicle information system

Sends current location

Traffic monitoring system
-- discovered prior to runtime

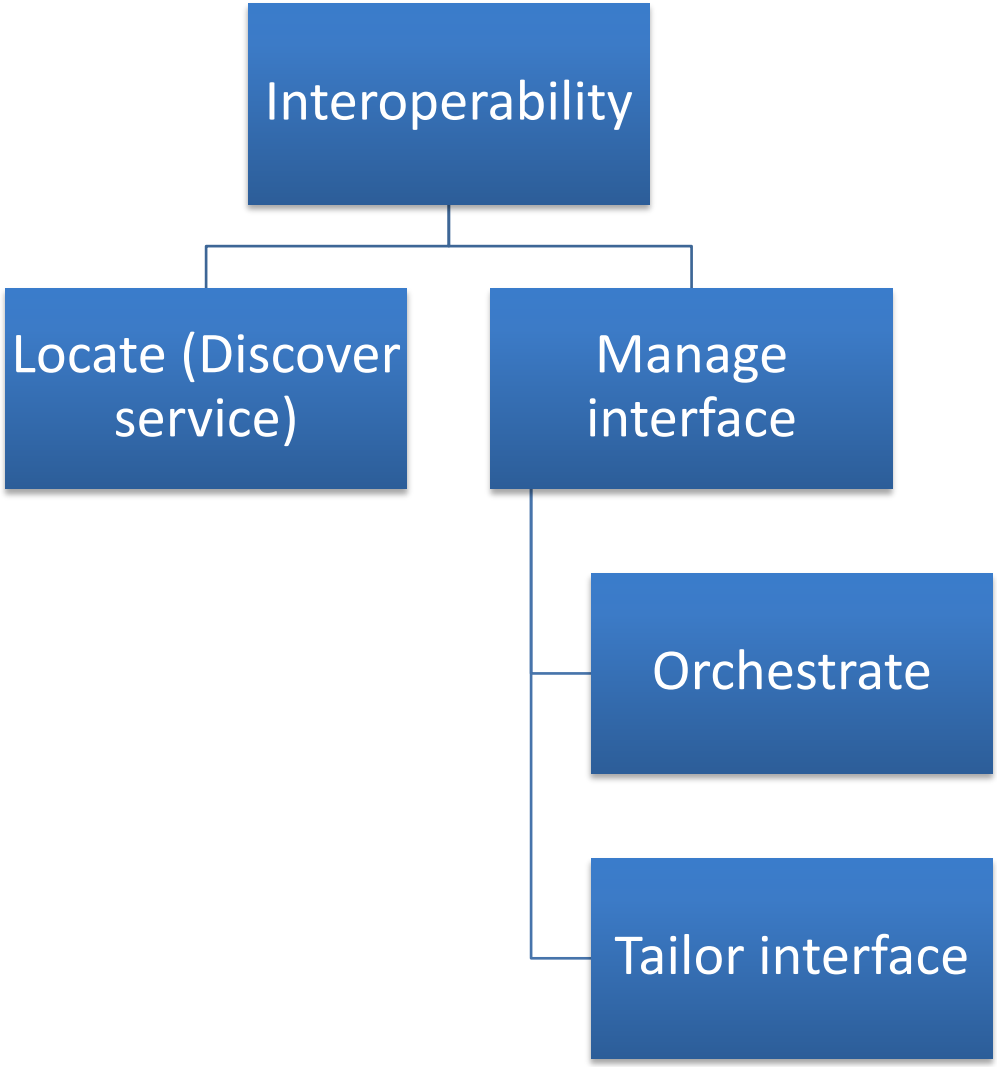
Combines the location information with other details, overlays with Google Maps, and broadcasts

The vehicle information is included correctly 99.9% of the time

Notion of Interface

- Information exchange
  - Can be as simple as A calling B
  - A and B can exchange implicitly w/o direct communication
  - Operation Dessert Storm 1991: Anti-missile system failed to exchange information (intercept) an incoming ballistic rocket
    - The system required periodic restart in order to recalibrate its position. Since it wasn't restarted, the information wasn't correctly captured due to error accumulation
- Interface
  - Here it also means that a set of assumptions that can be made safely about this entity
  - E.g. it is safe to assume that the API of anti-missile system DOES NOT give information about gradual degradation

Tactics


Interoperability
Locate (Discover service)
Manage interface
Orchestrate
Tailor interface

Interoperability Tactics

- Locate (Discover service)
  - Identify the service through a known directory service. Here service implies a set of capabilities available through an interface
  - By name, location or other attributes

Interoperability Tactics

Manage interface

- Orchestrate
  - Co-ordinate and manage a sequence of services. Example- workflow engines containing scripts of interaction
  - Mediator design pattern for simple orchestration. BPEL language for complex orchestration
- Tailor interface
  - Add or remove capability from an interface (hide a particular function from an untrusted user)
  - Use Decorator design pattern for this purpose

REpresentational State Transfer (REST)

REST is an architectural pattern where services are described using an uniform interface. REST*ful* services are viewed as a hypermedia resource. REST is stateless.

REST Verb	CRUD Operation	Description
POST	CREATE	Create a new resource.
GET	RETRIEVE	Retrieve a representation of the resource.
PUT	UPDATE	Update a resource.
DELETE	DELETE	Delete a resource.

Google Suggest : http://suggestqueries.google.com/complete/search?output=toolbar&hl=en&q=satyajit%20ray

Yahoo Search:
http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=YahooDemo&query=accenture

REST vs. SOAP/WSDL

- Simply put, the community has claimed that SOAP and WSDL have become too grandiose and comprehensive to achieve the "agility" touted by SOA (Seeley, R., "Burton sees the future of SOA and it is REST," SearchWebService.com, May 30, 2007)

	SOAP/WSDL	REST
Purpose	Message exchange between two applications/systems	Access and manipulating a hypermedia system
Origin	RPC	WWW
Functionality	Rich	Minimal
Interaction	Orchestrated event-based	Client/server (request/response)
Focus	Process-oriented	Data-oriented
Methods/ operations	Varies depending on the service	Fixed
Reuse	Centrally governed	Little/no governance (focus on ease of use instead)
Interaction context	Can be maintained in both client and server	Only on client
Format	SOAP in, SOAP out	URI (+POX) in, POX out
Transport	Transport independent	HTTP only
Security	WS-Security	HTTP authentication + SSL

Design Checklist- Interoperability

- Allocation of Responsibilities: Check which system features need to interoperate with others. For each of these features, ensure that the designers implement
  - Accepting and rejecting of requests
  - Logging of request
  - Notification mechanism
  - Exchange of information
- Coordination Model: Coordination should ensure performance SLAs to be met. Plan for
  - Handling the volume of requests
  - Timeliness to respond and send the message
  - Currency of the messages sent
  - Handle jitters in message arrival times

Design Checklist-Interoperability

Data Model

- Identify the data to be exchanged among interoperating systems
- If the data can't be exchanged due to confidentiality, plan for data transformation before exchange

Identification of Architectural Component

- The components that are going to interoperate should be available, secure, meet performance SLA (consider design-checklists for these quality attributes)

Design Checklist- Interoperability

Resource Management

- Ensure that system resources are not exhausted (flood of request shouldn't deny a legitimate user)
- Consider communication load
- When resources are to be shared, plan for an arbitration policy

Binding Time

- Ensure that it has the capability to bind unknown systems
- Ensure the proper acceptance and rejection of requests
- Ensure service discovery when you want to allow late binding

Technology Choice

- Consider technology that supports interoperability (e.g. web-services)

Thank You