BITS Pilani

Pilani|Dubai|Goa|Hyderabad

SS ZG653: Software Architecture

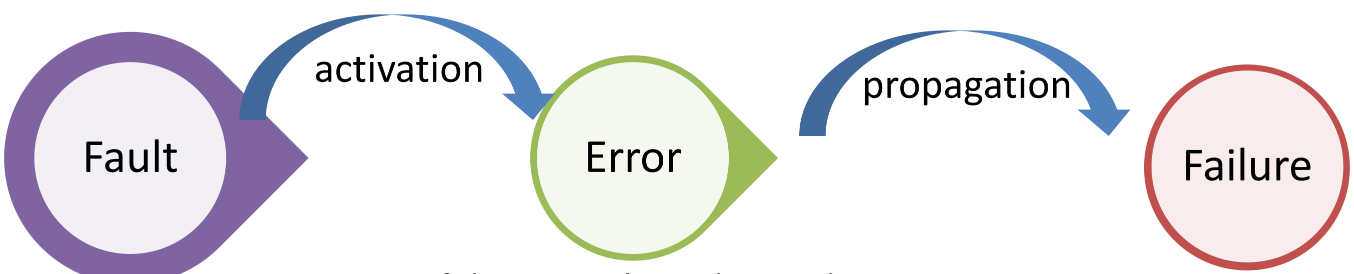
Lecture 3: System Quality- Availability, Modifiability, Performance

Instructor: Prof. Santonu Sarkar

Jan 20, 2015

Availability

- Readiness of the software to carry out its task
  - 100% available (which is actually impossible) means it is always ready to perform the intended task
- A related concept is Reliability
  - Ability to "continuously provide" correct service without failure
- Availability vs Reliability
  - A software is said to be available even when it fails but recovers immediately
  - Such a software will NOT be called Reliable



- Hypothesized cause of error
- Part of the system's total state that can leads to failure
- event that occurs when the delivered service deviates from correct service

Availability

- Faults and failures
  - Faults can be a failure if it is not corrected or masked
  - Once the system fails
    - It is NOT available
    - It must recover within certain time

Availability A= $\dfrac{Mean\ time\ to\ failure\ (MTTF)}{MTTF + Mean\ time\ to\ recover\ (MTTR)}$

  - Scheduled downtime is not considered
  - Availability 100% means it recovers instantenously
  - Availability 99.9% ➔ there is 0.01% probability that it will not be operational when needed

Availability Scenarios

WHO

Internal or External to System

STIMULUS

Fault causing
- System does not respond
- Crash
- Delay in response
- Erroneous Response

IMPACTED PART

Infrastructure and/or application

- During normal operation
- During degraded mode of operation

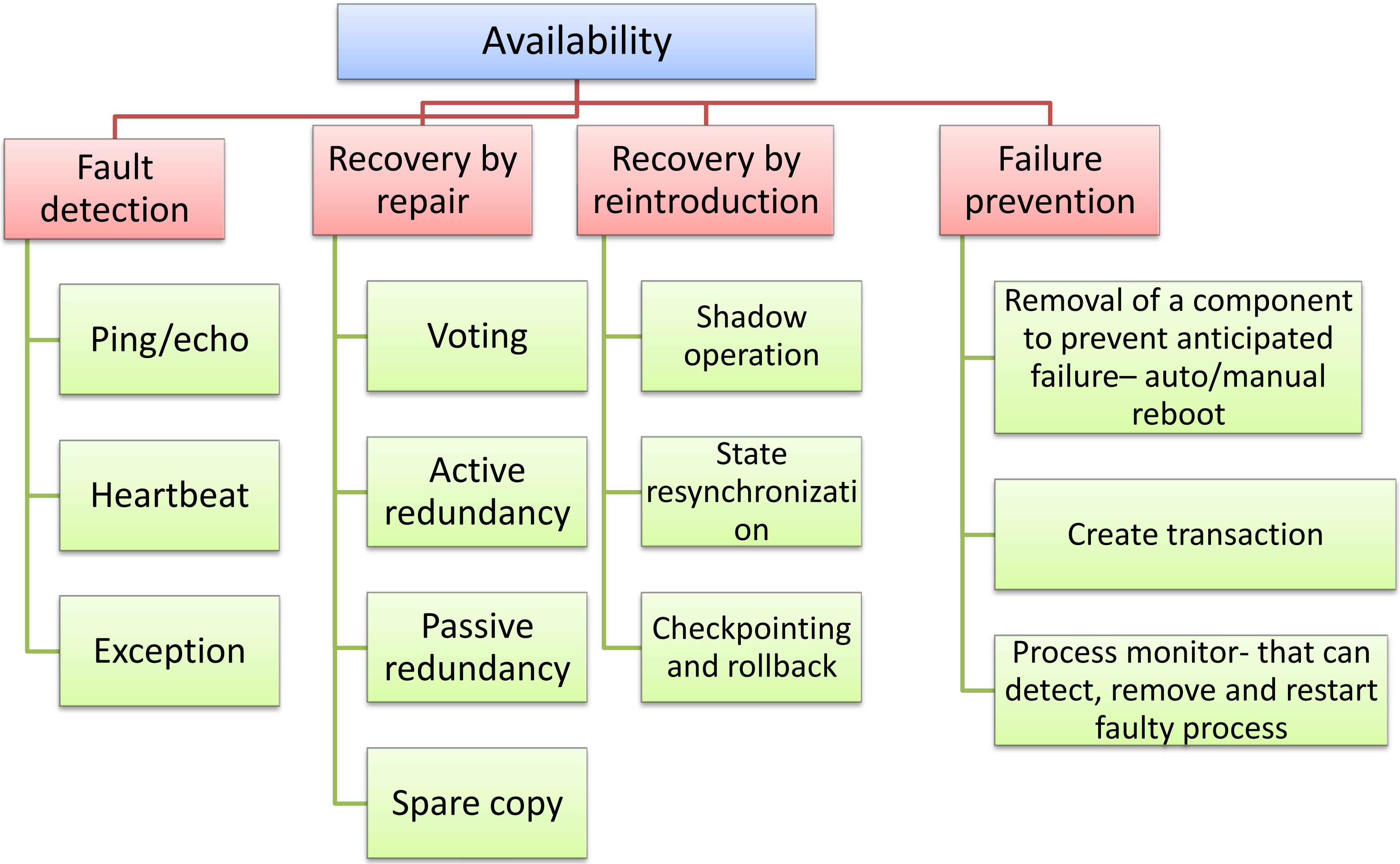
MITIGATING ACTION

When fault occurs it should do one or more of
- ✓ detect and log
- ✓ Notify the relevant stakeholders
- ✓ Disable the source of failure
- ✓ Be unavailable for a predefined time interval
- ✓ Continue to operate in a degraded mode

MEASURABLE RESPONSE

- Specific time interval for availability
- Availability number
- Time interval when it runs in degraded mode
- Time to repair

Availability Tactics



Availability Tactics- Fault Detection

- Ping
  - Client (or fault-detector) pings the server and gets response back
  - To avoid less communication bandwidth- use hierarchy of fault-detectors, the lowest one shares the same h/w as the server
- Heartbeat
  - Server periodically sends a signal
  - Listeners listen for such heartbeat. Failure of heartbeat means that the server is dead
  - Signal can have data (ATM sending the last txn)
- Exception
  - And Exception handler

Availability Tactics- Fault Recovery through quick replacement

- Voting
  - Redundant processes- all of them perform the same task and their outputs are compared
  - Any mismatch is considered failure.
    - Majority voting, or preferred component wins
- Hot restart (Active redundancy)
  - Every redundant process is active
  - When one fails, another one is taken up
  - Downtime is millisec
- Warm restart (Passive redundancy)
  - Standbys keep syncing their states with the primary one
  - When primary fails, backup starts
- Spare copy
  - Restarted when primary fails
  - Restarts to the checkpointed position
  - Downtime in min

Availability Tactics- Fault Recovery through reintroduction after repair

- Shadow
  - Repair the component
  - Run in shadow mode to observe the behavior
  - Once it performs correctly, reintroduce it
- State resynch
  - Related to the hot and warm restart
  - When the faulty component is started, its state must be upgraded to the latest state.
    - Update depends on downtime allowed, size of the state, number of messages required for the update..
- Checkpointing and recovery
  - Application periodically "commits" its state and puts a checkpoint
  - Recovery routines can either roll-forward or roll-back the failed component to a checkpoint when it recovers

Availability Tactics- Fault Prevention

- Faulty component removal
  - Fault detector predicts the imminent failure based on process's observable parameters (memory leak)
  - The process can be removed (rebooted) and can be auto-restart
- Transaction
  - Group relevant set of instructions to a transaction
  - Execute a transaction so that either everyone passes or all fails
- Process Monitor
  - Which can detect a faulty process and restart

Modifiability

- **Ability to Modify the system based on the change in requirement so that**
  - **the time and cost to implement is optimal**
  - **Impact of modification such as testing, deployment, and change management is minimal**

Modifiability Scenarios

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE

• Enduser
• Developer
• SysAdm
	They want to modify
	➢ Functionality
	    ➢ Add, modify, delete
	➢ Quality
	    ➢ Capacity
		UI, platform or System
		
		• Runtime
		• Compile time
		• Design time
		• Build time
			When fault occurs it should do one or more of
			✓ Locate (Impact analysis)
			✓ Modify
			✓ Test
			✓ Deploy again
				• Volume of the impact of the primary system
				• Cost of modification
				• Time and effort
				• Extent of impact to other systems

Developer	Tries to change UI	Artifact – Code Environment: Design time	Changes made and unit test done	Completed in 4 hours

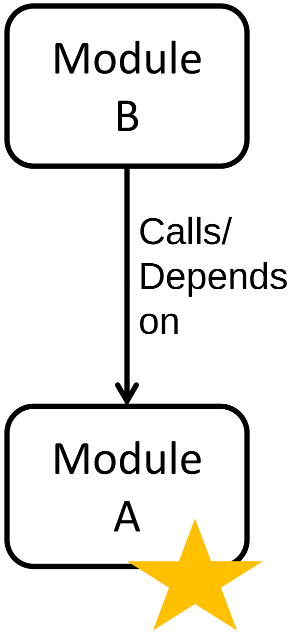
SS ZG653 Second Semester 2014-15

Modifiability Tactics

- Modifiabilty
  - Localize change
    - Factoring common service
    - Anticipate change
    - Generaliz e module
    - Limit possible options
  - Prevent Ripple effect
    - Hide Information
    - Maintain Existing Interface
    - Restrict Communication Paths
    - Use intermediary between modules
  - Defer binding
    - Runtime registration
    - Configuration files
    - Polymorphism
    - Component replacement
    - Adherence to defined protocols

Localize Modifications

1. Factoring common service
   - Common services through a specialized module (only implementing module should be impacted)
     - Heavily used in application framework and middleware
   - Reduce Coupling and increase cohesion
2. Anticipate Expected Changes
   - Quite difficult to anticipate, hence should be coupled with previous one
   - Allow extension points to accommodate changes
3. Generalize the Module
   - Allowing it to perform broader range of functions
   - Externalize configuration parameters (could be described in a language like XML)
     - The module reconfigure itself based on the configurable parameters
   - Externalize business rules
4. Limit Possible options
   - Do not keep too many options for modules that are part of the framework



Dependency between two modules (B→ A)

Syntax (compile+runtime)

- Data : B uses the type/format of the data created by A
- Service B uses the API signature provided by A

Semantics of A

- Data: Semantics of data created by A should be consistent with the assumption made by B
- Service: Same …..

Sequence

- Data -- data packets created by A should maintain the order as understood by B
- Control– A must execute 5ms before B. Or an API of A can be called only after calling another API

Interface identity

- Handle of A must be consistent with B, if A maintains multiple interfaces

Location of A

- B may assume that A is in-process or in a different process, hardware..

Quality of service/data provided by A

- Data quality produced by A must be > some accuracy for B to work

Existence of A

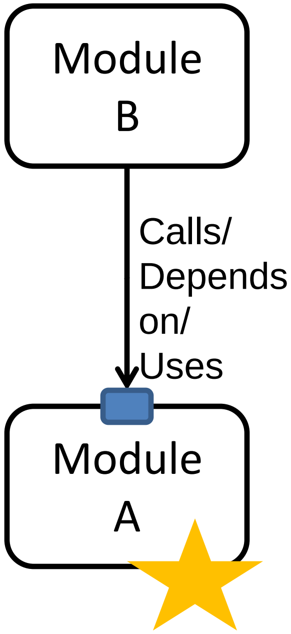
- B may assume that A must exist when B is calling A

Resource behavior of A

- B may assume that both use same memory
- B needs to reserve a resource owned by A

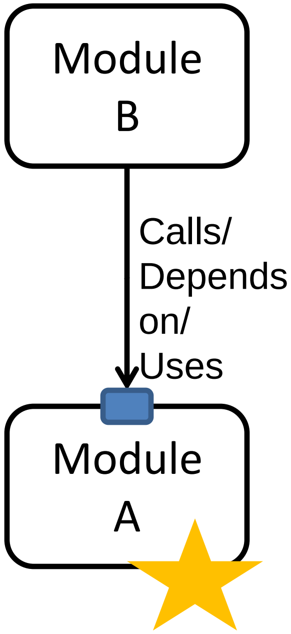
Prevent Ripple Effect Tactics

1. Hide Information (of A)
   – Use interfaces, allow published API based calls only
2. Maintain existing Interface (of A)
   – Add new interfaces if needed
   – Use Wrapper, adapter to maintain same interface
   – Use stub
3. Restrict Communication Paths
   – Too many modules should not depend on A
4. Use an intermediary between B and A
   – Data
     • Repository from which B can read data created by A (blackboard pattern)
     • Publish-subscribe (data flowing through a central hub)
     • MVC pattern
   – Service: Use of design patterns like bridge, mediator, strategy, proxy
   – Identity of A – Use broker pattern which deals with A's identity
   – Location of A – Use naming service to discover A
   – Existence of A- Use factory pattern



Defer Binding Time

1. Runtime registration of A (plug n play)
   – Use of pub-sub
2. Configuration Files
   – To take decisions during startup
3. Polymorphism
   – Late binding of method call
4. Component Replacement (for A)
   – during load time such as classloader
5. Adherence to a defined protocol- runtime binding of independent processes



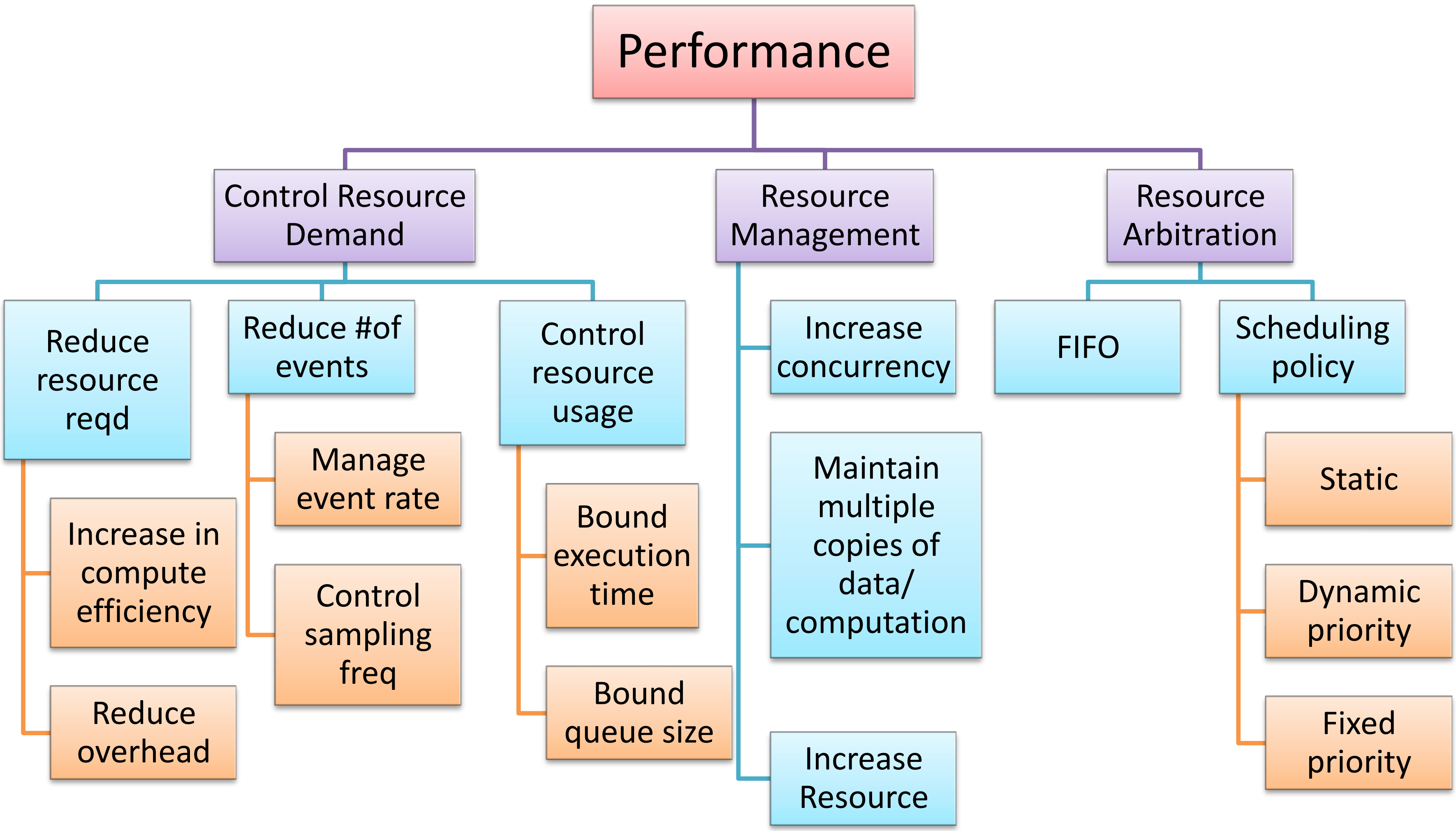
What is Performance?

- Software system's ability to meet timing requirements when it responds to an event
- Events are
  - interrupts, messages, requests from users or other systems
  - clock events marking the passage of time
- The system, or some element of the system, must **respond to them** in time

Performance Scenarios

WHO	STIMULUS	IMPACTED PART	MITIGATING ACTION	MEASURABLE RESPONSE
• External • Internal	• **Events that are** • Periodic • Sporadic • Stochastic (with avg 1000 txn/min)	System • Running in normal mode • Overloaded	✓ Processes event in time ✓ Changes the service-level	• Latency or deadline by which it must be processed • Throughput- #of events processed/time • Jitter- variation • Miss rate • Data loss
Users	Submits transactions and expect response in 3s		Transactions processed	Average latency of 2s

Performance Tactics



Why System fails to Respond?

- Resource Consumption
  - CPU, memory, data store, network communication
  - A buffer may be sequentially accessed in a critical section
  - There may be a workflow of tasks one of which may be choked with request
- Blocking of computation time
  - Resource contention
  - Availability of a resource
  - Deadlock due to dependency of resource

Control Resource Demand

- Increase Computation Efficiency: Improving the algorithms used in performance critical areas
- Reduce Overhead
  - Reduce resource consumption when not needed
    - Use of local objects instead of RMI calls
    - Local interface in EJB 3.0
  - Remove intermediaries (conflicts with modifiability)
- Manage
  - event rate: If you have control, don't sample too many events (e.g. sampling environmental data)
  - sampling time: If you don't have control, sample them at a lower speed, leading to loss of request
- Bound
  - Execution: Decide how much time should be given on an event. E.g. iteration bound on a data-dependent algorithm
  - Queue size: Controls maximum number of queued arrivals

Manage Resources

- Increase Resources (infrastructure)
  - Faster processors, additional processors, additional memory, and faster networks
- Increase Concurrency
  - If possible, process requests in parallel
  - Process different streams of events on different threads
  - Create additional threads to process different sets of activities
- Multiple copies
  - Computations : so that it can be performed faster (client-server), MapReduce computation
  - Data:
    - use of cache for faster access and reduce contention
    - Hadoop maintains data copies to avoid data-transfer and improve data locality

Resource Arbitration

- Resources are scheduled to reduce contention
  - Processors, buffer, network
  - Architect needs to choose the right scheduling strategy
- FIFO
- Fixed Priority
  - Semantic importance
    - Domain specific logic such as request from a privileged class gets higher priority
  - Deadline monotonic (shortest job first)
- Dynamic priority
  - Round robin
  - Earliest deadline first- the job which has earliest deadline to complete
- Static scheduling
  - Also pre-emptive scheduling policy

Design Checklist for a Quality Attribute

- Allocate responsibility
  - Modules can take care of the required quality requirement
- Manage Data
  - Identify the portion of the data that needs to be managed for this quality attribute
  - Plan for various data design w.r.t. the quality attribute
- Resource Management Planning
  - How infrastructure should be monitored, tuned, deployed to address the quality concern
- Manage Coordination
  - Plan how system elements communicate and coordinate
- Binding

Performance- Design Checklist- Allocate responsibilities

- Identify which requirements may cause performance bottlenecks
- Analyze the scenarios that can cross process or processor boundaries and check for performance bottleneck
- Managing the threads of control —allocation and de-allocation of threads, maintaining thread pools, and so forth
- Assign responsibilities that will schedule shared resources or appropriately select, manage performance-related artifacts such as queues, buffers, and caches

Performance- Design Checklist- Manage Data

- Identify the data that's involved in time critical response requirements, heavily used, massive size that needs to be loaded etc. For those data determine
  - whether maintaining multiple copies of key data would benefit performance
  - 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.

Performance- Design Checklist- Manage Coordination

- Look for the possibility of introducing concurrency (and obviously pay attention to thread-safety), event priorization, or scheduling strategy
  - Will this strategy have a significant positive effect on performance? Check
  - Determine whether the choice of threads of control and their associated responsibilities introduces bottlenecks

- Consider appropriate mechanisms for example
  - stateful, stateless, synchronous, asynchronous, guaranteed delivery

Performance Design Checklist- Resource Management

- Determine which resources (CPU, memory) in your system are critical for performance.
  - Ensure they will be monitored and managed under normal and overloaded system operation.
- Plan for mitigating actions early, for instance
  - 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.
- Prioritization of resources and access to resources
  - scheduling and locking strategies
- Deploying additional resources on demand to meet increased loads
  - Typically possible in a Cloud and virtualized scenario

Performance Design checklist- Binding

- For each element that will be bound after compile time, determine the
  - 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.

Performance Design Checklist- Technology choice

- Choice of technology is often governed by the organization mandate (enterprise architecture)
- Find out if the chosen technology will let you set and meet real time deadlines?
  - Do you know its characteristics under load and its limits?
- Does your choice of technology give you the ability to set
  - scheduling policy
  - Priorities
  - policies for reducing demand
  - allocation of portions of the technology to processors
- Does your choice of technology introduce excessive overhead?

Thank You