BITS Pilani

Software Architecture

Software Quality Attributes

Viswanathan Hariharan
Edited by Vijayarajan

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

Contents

- Recap
- Introduction to quality attributes
- Tactics to achieve quality attributes

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>

Why do we Need Structure

Functionalities are articulated by Functional requirements

There are other requirements generally not articulated

They are implicit or not very explicit

They are domain specific

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

How to achieve quality attributes?

- Partly through coding
- Partly through architecture

How to achieve quality attributes?

Example

If we want a high performance software with fast response time, then

- We have to use the best algorithm which takes minimal time (coding)
- We have to distribute copies of the software on multiple servers to handle the load (architecture)

How to achieve quality attributes?

Example

If we want a software that is easy to modify, then

- We have to divide it into functional modules (architecture)
- We also have to code it using good naming conventions for variables and functions (coding)

Quality attributes Deep Dive

Quality Attribute Requirements

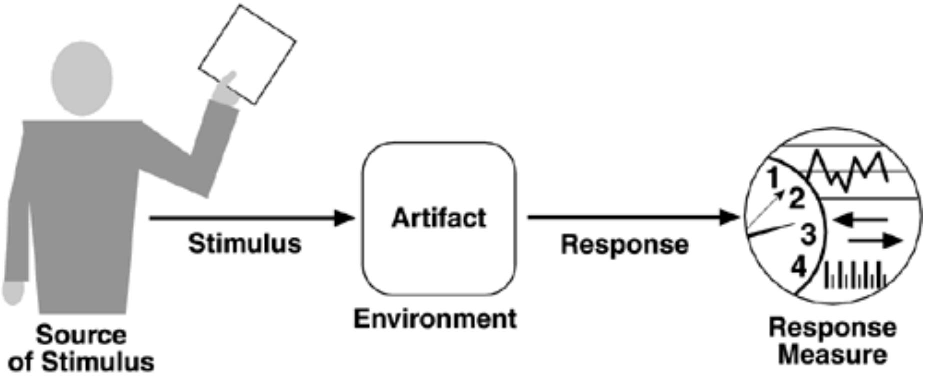
How do we articulate the Quality attribute requirements

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

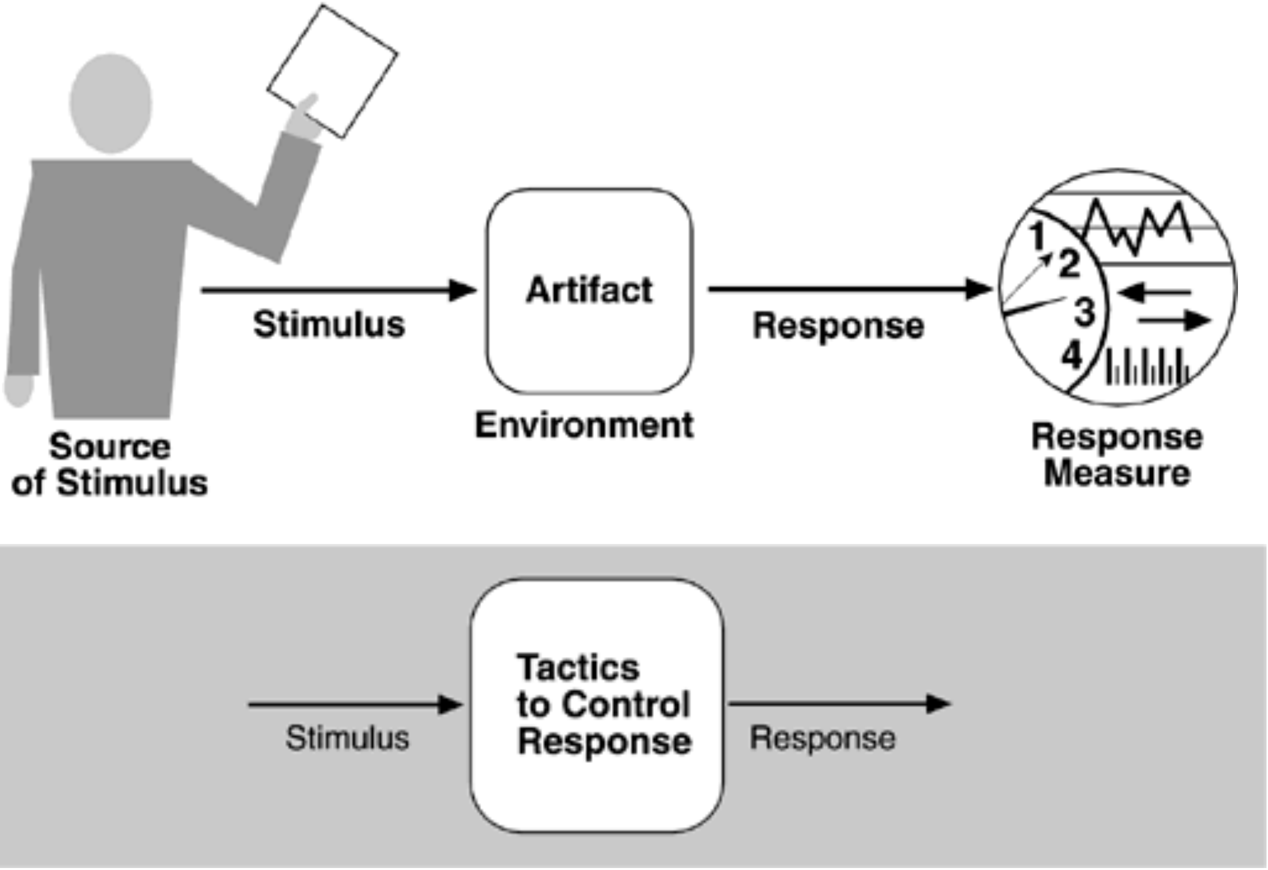
Response: activity undertaken after the arrival of the stimulus.

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

Quality attribute Scenario Framework - example

Refer to the PDF

Design Tactics



Tactic is a design option for the architect

Availability

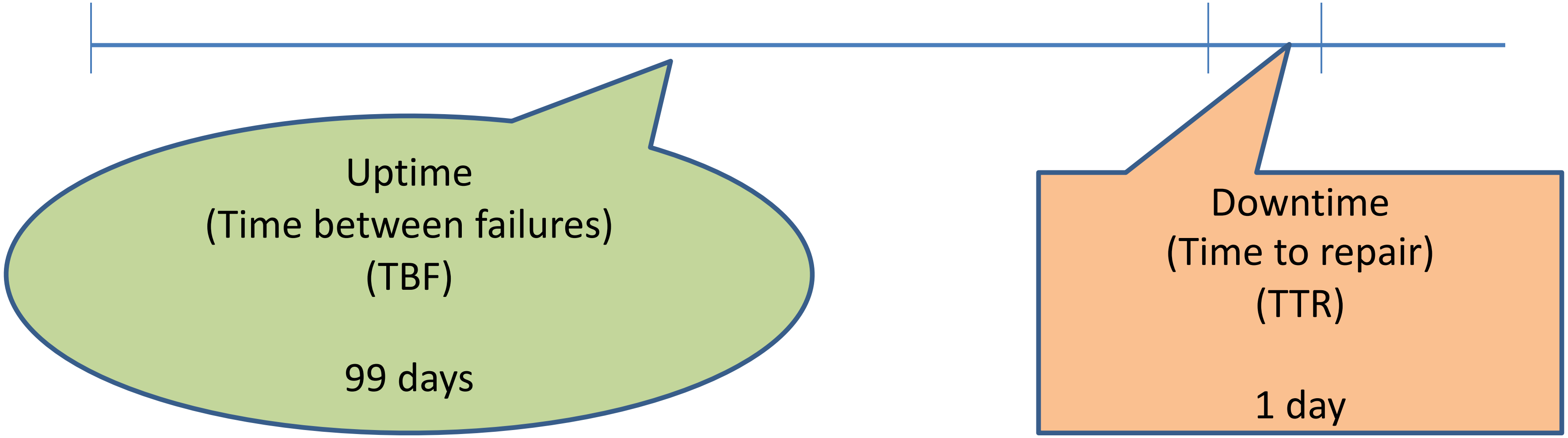
- Availability is the ability of a system to minimize system outages

- Availability is generally measured as % of up-time

- Ex. If a system is down for an average of 1 day in 100 days, its availability is 99%
- If a system is down for an average of 2 day in 100 days, its availability is 98%
- If a system is down for an average of 0.5 day in 100 days, its availability is 99.5%

Availability

Calculating availability



Availability = Mean TBF / (Mean TBF + Mean TTR)
= MTBF / (MTBF + MTTR)

Availability

What are the issues that affect availability?

Availability

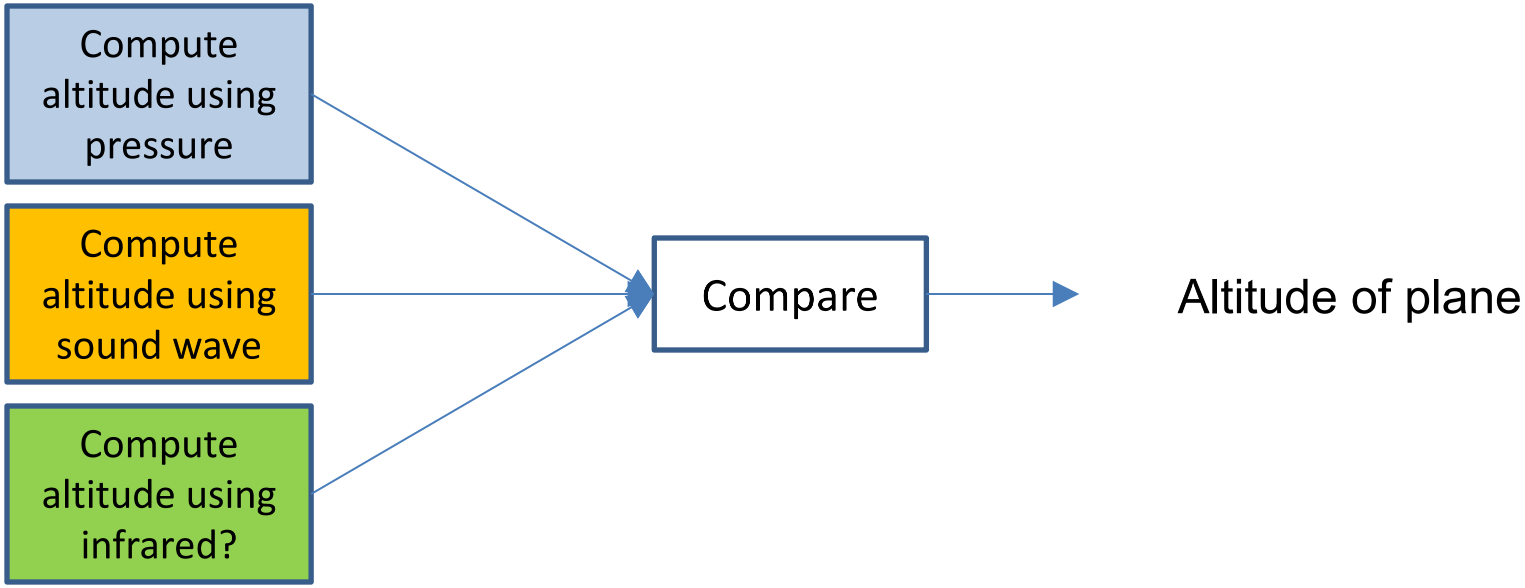
Issue	Tactic
Disk failure	
Server failure	
Link failure	
Malicious user floods the system with fictitious login	
Temporary loss of connection	
Unreliable code / algorithm	

Availability

Issue	Tactic
Disk failure	Real time replication (Hot standby)
Server failure	Backup server running the same application Master regularly updates the Backup about the progress. Upon failure, backup becomes master and continues from the last known state (Warm standby)
Link failure	Redundant communication paths
Malicious user floods the system with fictitious login	Block IP
Temporary loss of connection	Retry after some time. Ex. Mobile app unable to reach application in central server
Unreliable code / algorithm	Have multiple algorithms compute the result. Compare their results. Take the result given my most algorithms

Example: Boeing 777 Triple Modular Redundancy (TMR)

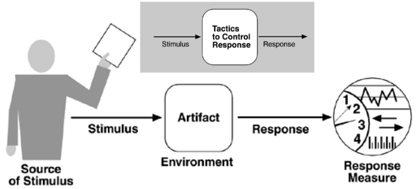
- The flight computer uses TMR with three dissimilar computation lanes
- Results of at least 2 computational modules need to match to go ahead


Compute altitude using pressure
Compute altitude using sound wave
Compute altitude using infrared?
Compare
Altitude of plane

Availability - Details

① **Source of stimulus.** entity (human, computer system, or any other actuator) that generated the stimulus
② **Stimulus -** condition that needs to be considered when it arrives at a system.
③ **Artifact**: Some artifact is stimulated – whole system or parts;
④ **Environment:** The stimulus occurs within certain conditions - Normal, safe mode, overload condition etc
⑤ **Response**: activity undertaken after the arrival of the stimulus.
⑥ **Response measure**: Measurable outcome, it should be measurable in some fashion so that the requirement can be tested.

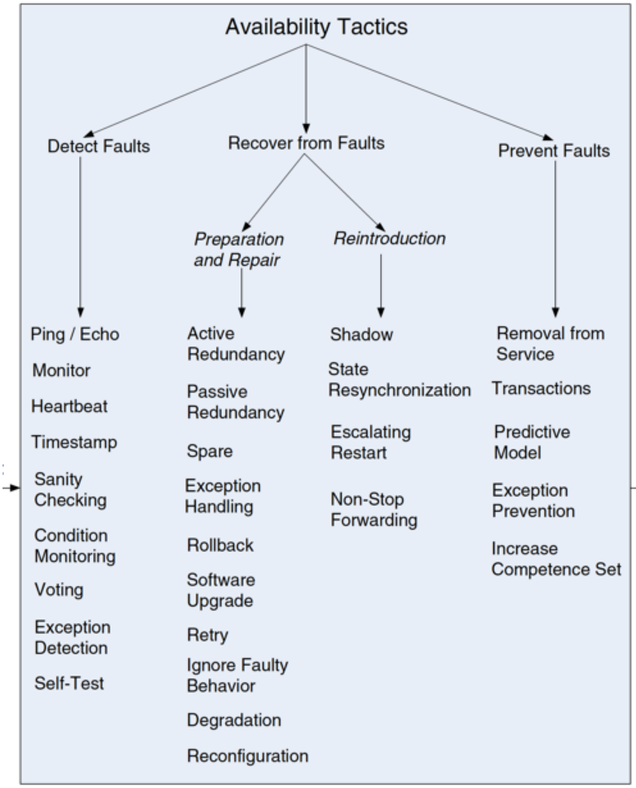
Tactic is a design option for the architect



Scenarios should cover a range of
- Anticipated uses of (use case scenarios),
- Anticipated changes to (growth scenarios), or
- Unanticipated stresses (exploratory scenarios) to the system.

Quality Attributes &Tactics

Identify the Quality attributes and Tactics require to achieve them



Availability Tactics

Detect Faults: Ping / Echo, Monitor, Heartbeat, Timestamp, Sanity Checking, Condition Monitoring, Voting, Exception Detection, Self-Test

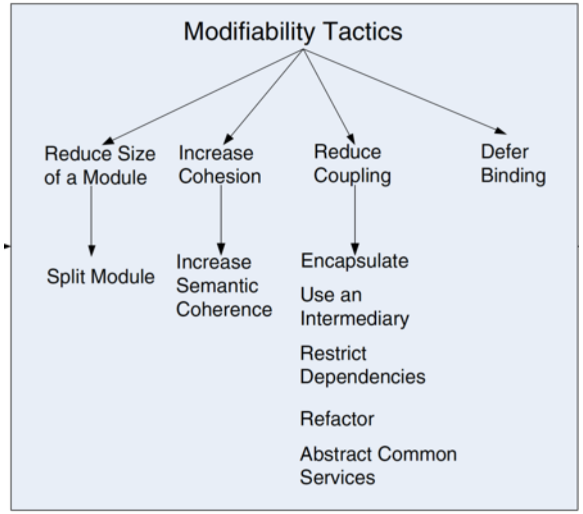
Recover from Faults
- *Preparation and Repair*: Active Redundancy, Passive Redundancy, Spare, Exception Handling, Rollback, Software Upgrade, Retry, Ignore Faulty Behavior, Degradation, Reconfiguration
- *Reintroduction*: Shadow, State Resynchronization, Escalating Restart, Non-Stop Forwarding

Prevent Faults: Removal from Service, Transactions, Predictive Model, Exception Prevention, Increase Competence Set

① Internal/external: people, hardware, software, physical infrastructure, physical environment
② Fault: omission, crash, incorrect timing, incorrect response
③ Processors, communication channels, persistent storage, processes
④ Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
⑤ Prevent the fault from becoming a failure
Detect the fault:
- Log the fault
- Notify appropriate entities (people or systems)

Recover from the fault:
- Disable source of events causing the fault
- Be temporarily unavailable while repair is being effected
- Fix or mask the fault/failure or contain the damage it causes
- Operate in a degraded mode while repair is being effected

⑥ Time or time interval when the system must be available
Availability percentage (e.g., 99.999%)
Time to detect the fault
Time to repair the fault
Time or time interval in which system can be in degraded mode
Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing



Modifiability Tactics

Reduce Size of a Module: Split Module

Increase Cohesion: Increase Semantic Coherence

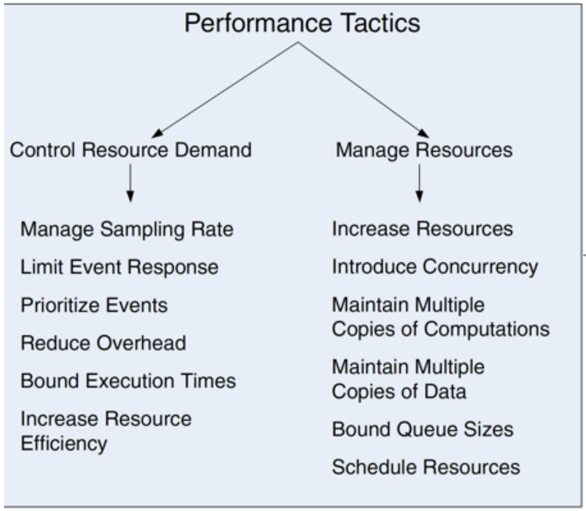
Reduce Coupling: Encapsulate, Use an Intermediary, Restrict Dependencies, Refactor, Abstract Common Services

Defer Binding

① End user, developer, system administrator
② A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
③ Code, data, interfaces, components, resources, configurations, . . .
④ Runtime, compile time, build time, initiation time, design time
⑤ One or more of the following:
- Make modification
- Test modification
- Deploy modification

⑥ Cost in terms of the following:
- Number, size, complexity of affected artifacts
- Effort
- Calendar time
- Money (direct outlay or opportunity cost)
- Extent to which this modification affects other functions or quality attributes
- New defects introduced

① Internal or external to the system
② Arrival of a periodic, sporadic, or stochastic event
③ System or one or more components in the system
④ Operational mode: normal, emergency, peak load, overload
⑤ Process events, change level of service
⑥ Latency, deadline, throughput, jitter, miss rate

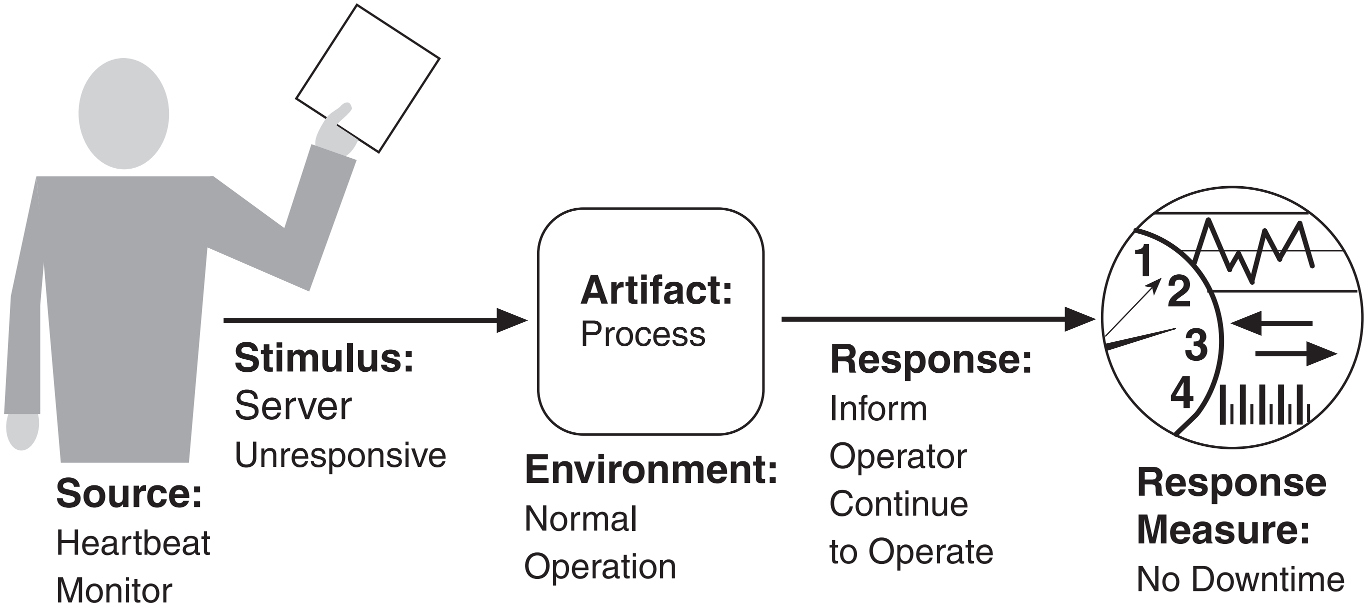


Performance Tactics

Control Resource Demand: Manage Sampling Rate, Limit Event Response, Prioritize Events, Reduce Overhead, Bound Execution Times, Increase Resource Efficiency

Manage Resources: Increase Resources, Introduce Concurrency, Maintain Multiple Copies of Computations, Maintain Multiple Copies of Data, Bound Queue Sizes, Schedule Resources

Availability Scenario



Source: Heartbeat Monitor

Stimulus: Server Unresponsive

Artifact: Process

Environment: Normal Operation

Response: Inform Operator Continue to Operate

Response Measure: No Downtime

Experience sharing

Have you come across situations where you faced Availability challenges?

How did you address them?

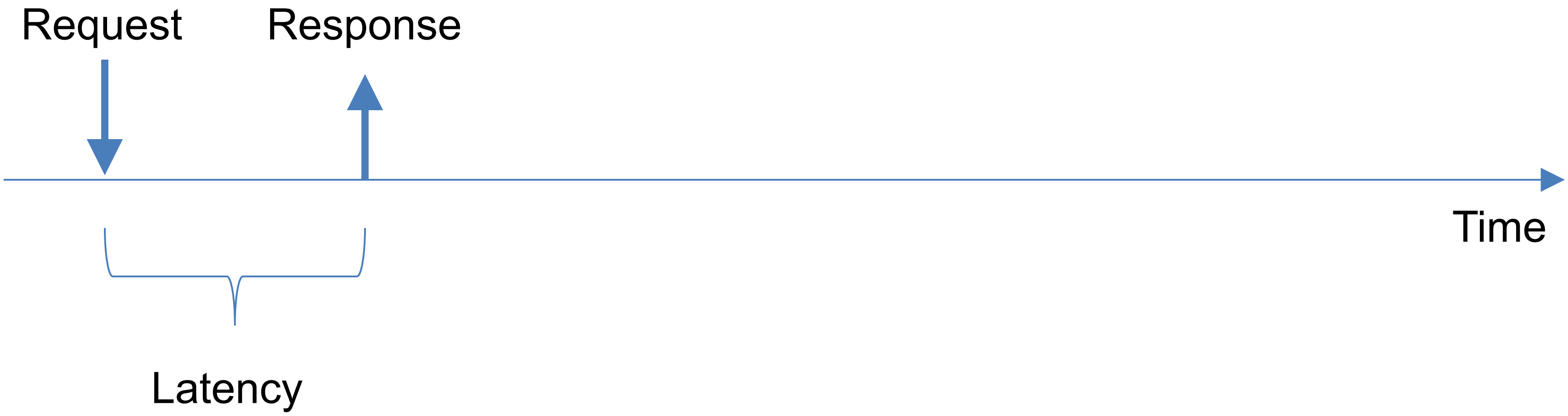
Performance

It is the system’s ability to meet timing requirements.

Measurements

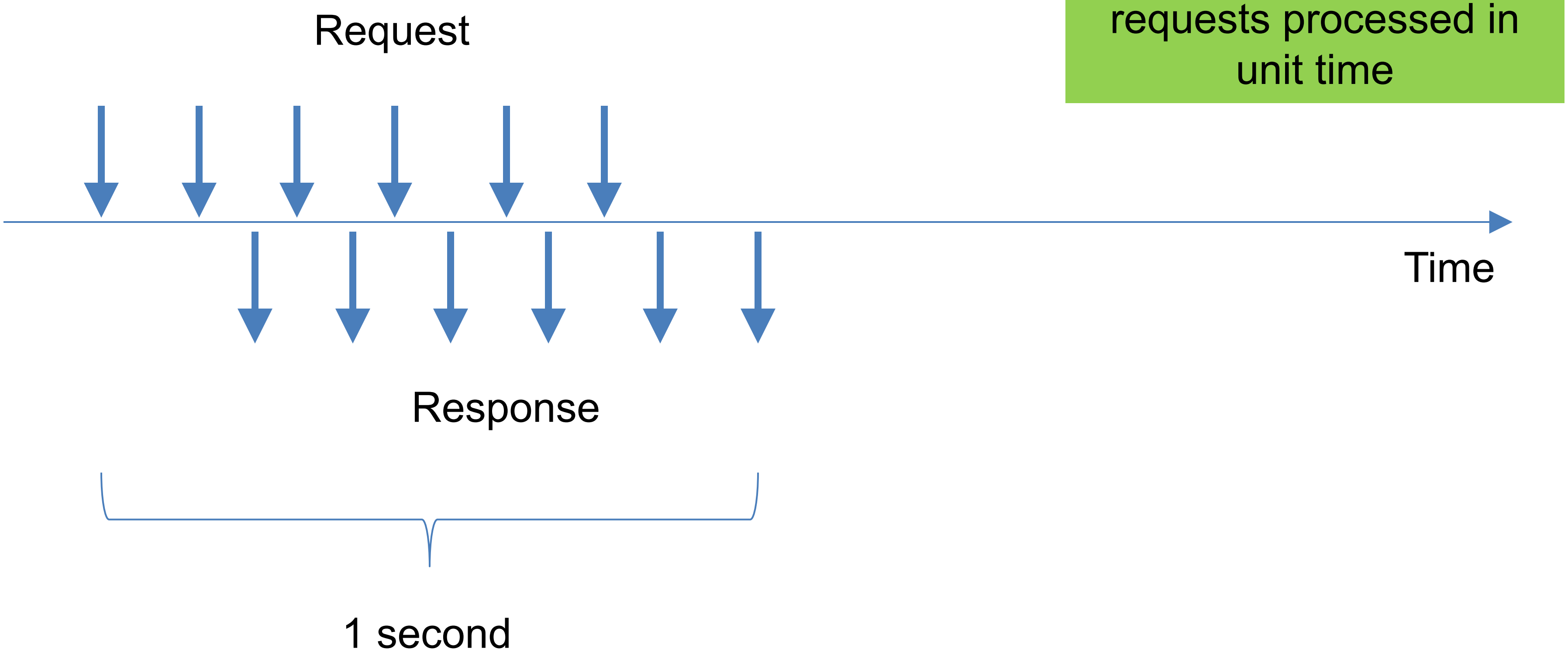
- Latency: Time to respond: Ex. 3 seconds
- Throughput: The number of requests handled per second: Ex. 3000 txns per second
- Jitter of response: Variation in response time: Response time is 3 seconds when # of users is 50, 5 seconds when # of users is 100
- Deadline: The max time within which it should respond (Requirement)

Latency

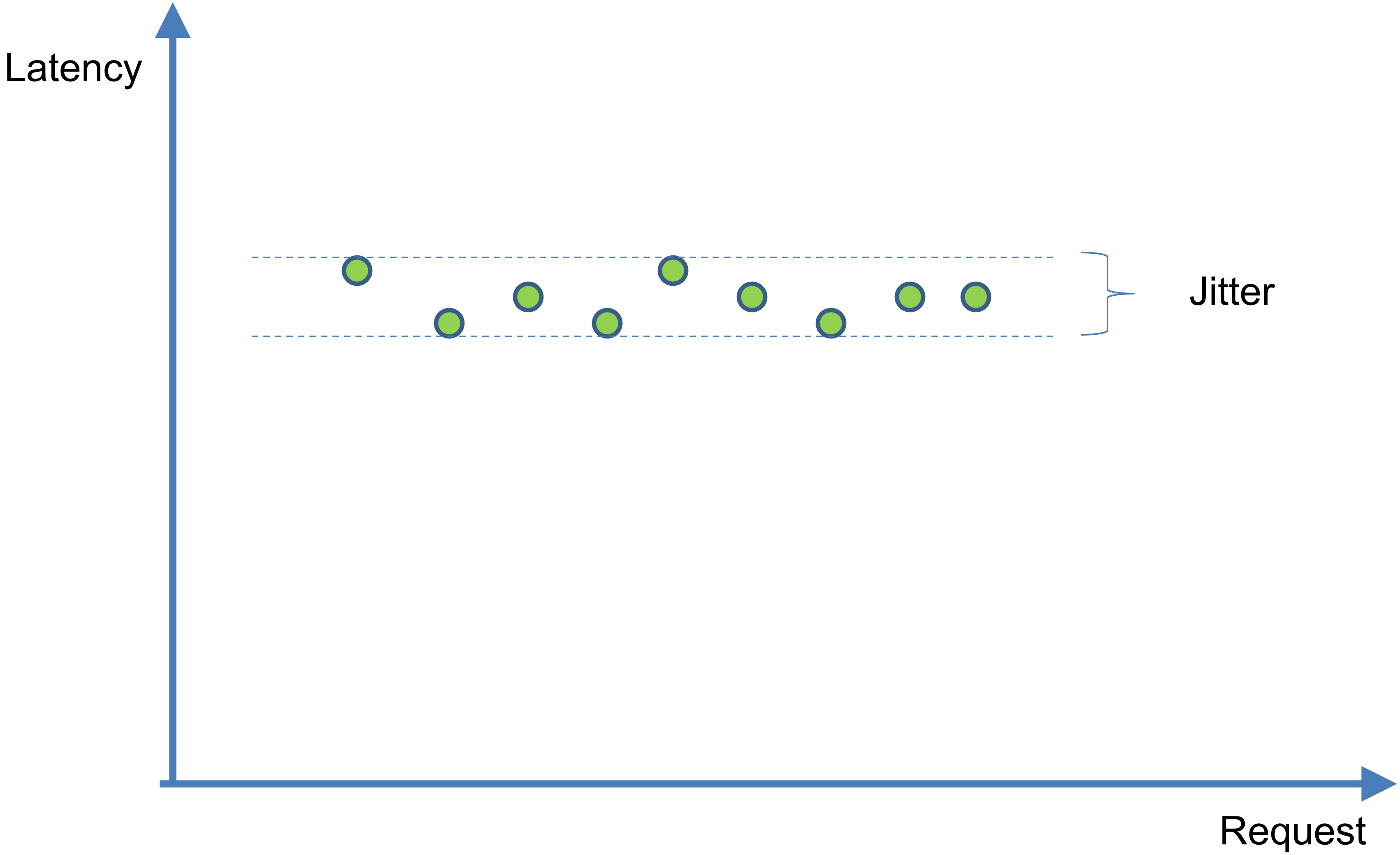


Throughput

Throughput = # of requests processed in unit time



Jitter



Performance tactics

What are the causes that impact performance?

Performance tactics

Cause	Tactic
Poor algorithm	
Database retrieval is slow	
Volume of data to be processed is high	
Number of concurrent users is high	

Performance tactics

Cause	Tactic
Poor algorithm	Improve search algorithm
Database retrieval is slow	• Index data – Hash index, B+ tree index • Partition data, De-normalize, etc. • Run reports in night • Maintain multiple copies of data
Volume of data to be processed is high	Distribute data on multiple servers and parallelize data processing (ex Map-Reduce)
Number of concurrent users is high	Distribute requests to different servers and do load balancing

How to improve performance when # of users is very large?

- **Load balancing**: Distribute requests to different servers in a server farm

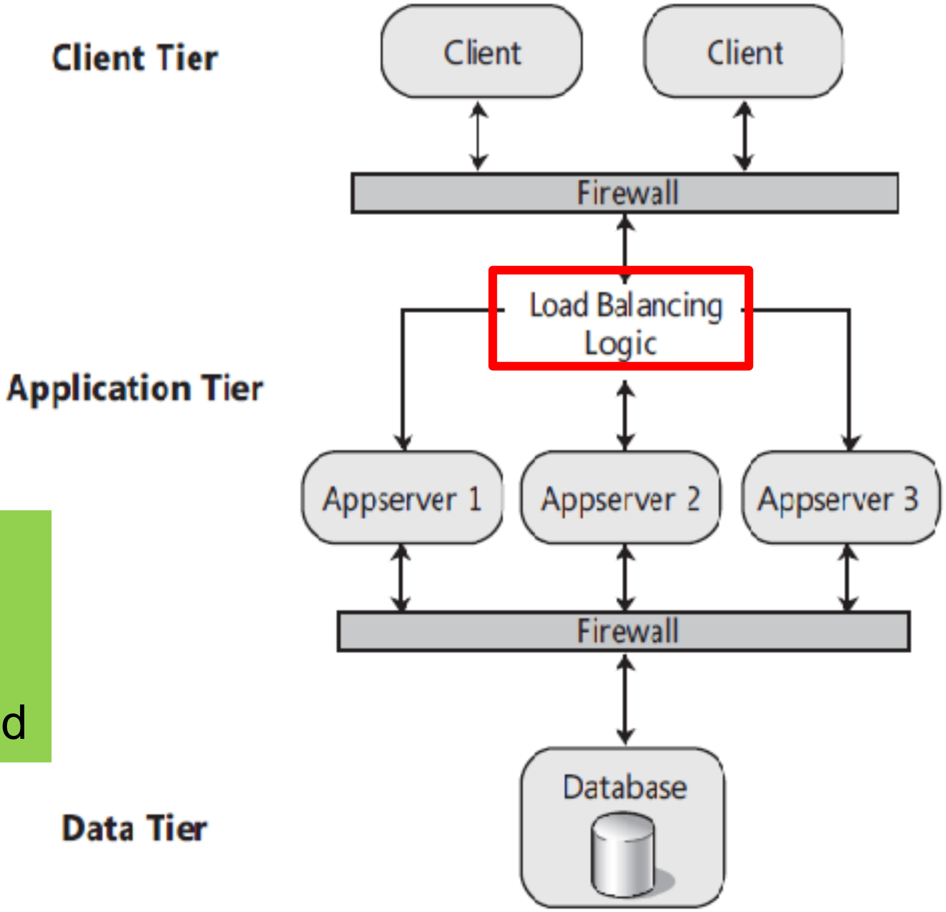


Figure 9
A load-balanced cluster

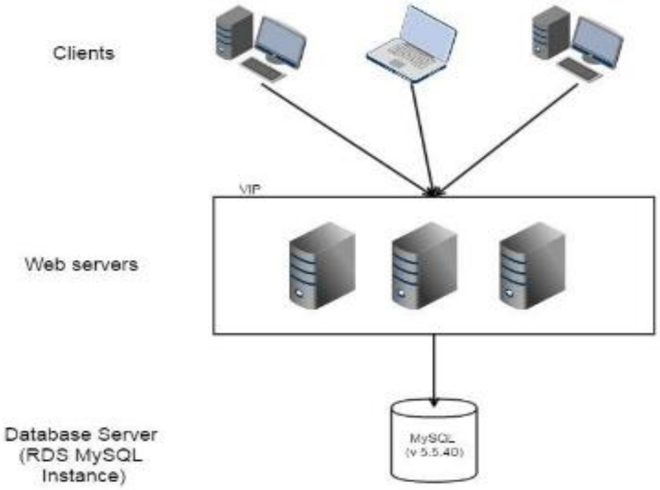
Approaches

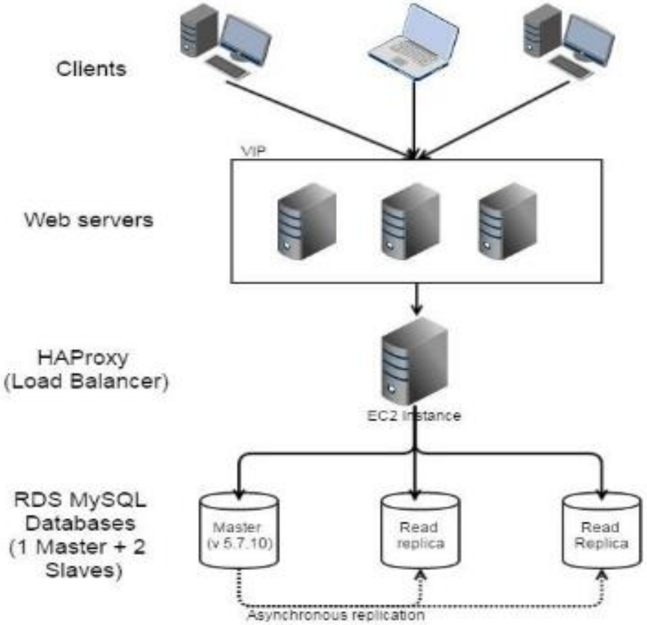
- Round robin
- Based on txn load

How to improve performance when multiple users want to READ the same data?

Maintain multiple copies of data. This reduces contention. However we need to keep them synchronized

Before	After



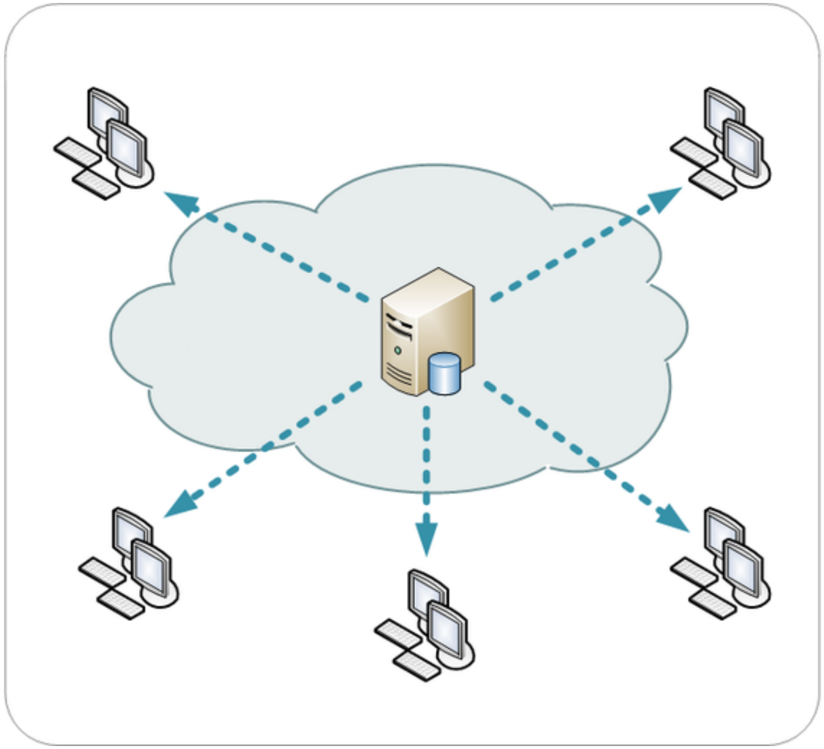


Poor response due to large data and high volume txns	Replicated data enables load balancing

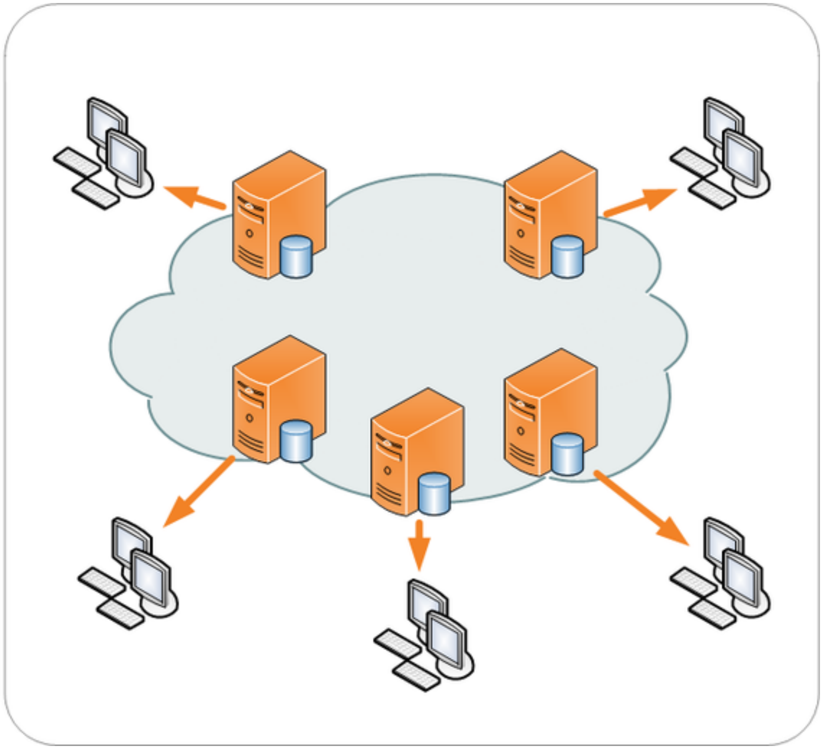
YouTube Example

- Supports the delivery of over 100 million videos per day.
- Most popular content is moved to a CDN (**content delivery network**):
- CDNs replicate content in multiple places. There's a better chance of content being closer to the user, with fewer hops, and content will run over a more friendly network.

CDN – Content Delivery Network



No CDN



CDN with multiple servers serving neighbouring clients

How does Google Return Results So Damn Fast?!?

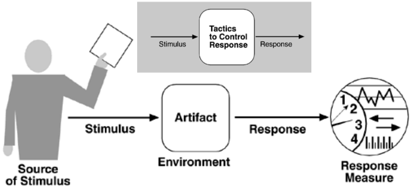
- Search the index not the internet
- Direct the search to nearest datacentre
- Hundreds of computers in each data center perform distributed look up
- Store index in RAM instead of disk

shutterstock.com • 400503175

● Google Data centers

① **Source of stimulus.** entity (human, computer system, or any other actuator) that generated the stimulus
② **Stimulus -** condition that needs to be considered when it arrives at a system.
③ **Artifact**: Some artifact is stimulated – whole system or parts;
④ **Environment:** The stimulus occurs within certain conditions - Normal, safe mode, overload condition etc
⑤ **Response**: activity undertaken after the arrival of the stimulus.
⑥ **Response measure**: Measurable outcome, it should be measurable in some fashion so that the requirement can be tested.

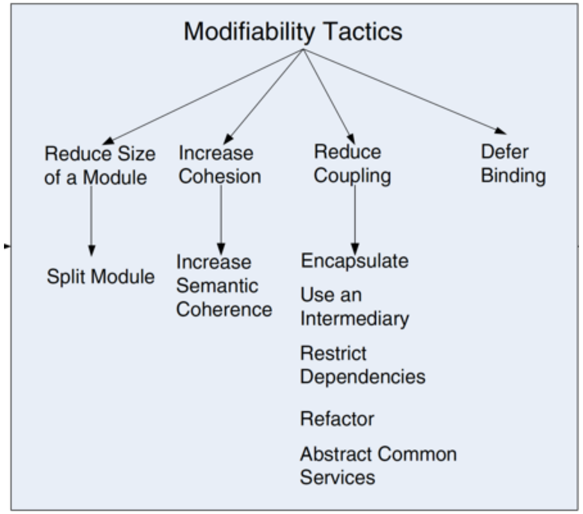
Tactic is a design option for the architect



Scenarios should cover a range of
- Anticipated uses of (use case scenarios),
- Anticipated changes to (growth scenarios), or
- Unanticipated stresses (exploratory scenarios) to the system.

Quality Attributes &Tactics

Identify the Quality attributes and Tactics require to achieve them



Modifiability Tactics

Reduce Size of a Module
- Split Module

Increase Cohesion
- Increase Semantic Coherence

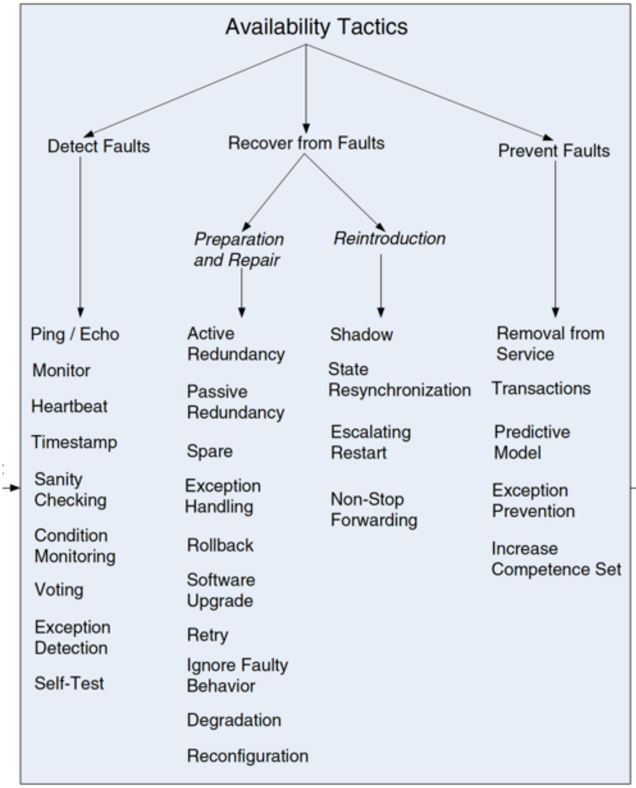
Reduce Coupling
- Encapsulate
- Use an Intermediary
- Restrict Dependencies
- Refactor
- Abstract Common Services

Defer Binding

① End user, developer, system administrator
② A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
③ Code, data, interfaces, components, resources, configurations, . . .
④ Runtime, compile time, build time, initiation time, design time
⑤ One or more of the following:
- Make modification
- Test modification
- Deploy modification

⑥ Cost in terms of the following:
- Number, size, complexity of affected artifacts
- Effort
- Calendar time
- Money (direct outlay or opportunity cost)
- Extent to which this modification affects other functions or quality attributes
- New defects introduced

① Internal or external to the system
② Arrival of a periodic, sporadic, or stochastic event
③ System or one or more components in the system
④ Operational mode: normal, emergency, peak load, overload
⑤ Process events, change level of service
⑥ Latency, deadline, throughput, jitter, miss rate



Availability Tactics

Detect Faults
- Ping / Echo
- Monitor
- Heartbeat
- Timestamp
- Sanity Checking
- Condition Monitoring
- Voting
- Exception Detection
- Self-Test

Recover from Faults

Preparation and Repair
- Active Redundancy
- Passive Redundancy
- Spare
- Exception Handling
- Rollback
- Software Upgrade
- Retry
- Ignore Faulty Behavior
- Degradation
- Reconfiguration

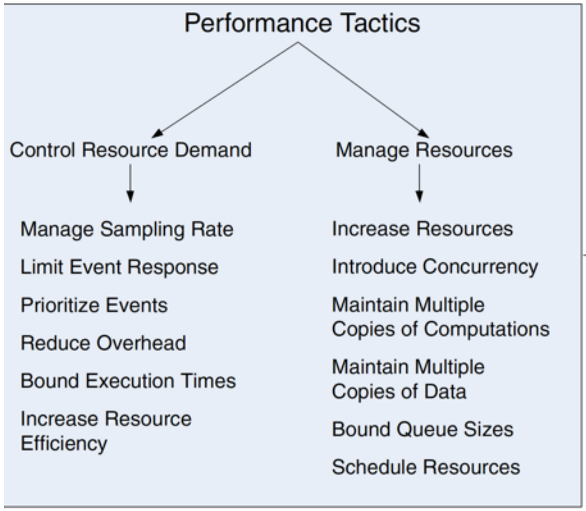
Reintroduction
- Shadow
- State Resynchronization
- Escalating Restart
- Non-Stop Forwarding

Prevent Faults
- Removal from Service
- Transactions
- Predictive Model
- Exception Prevention
- Increase Competence Set

① Internal/external: people, hardware, software, physical infrastructure, physical environment
② Fault: omission, crash, incorrect timing, incorrect response
③ Processors, communication channels, persistent storage, processes
④ Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation
⑤ Prevent the fault from becoming a failure
Detect the fault:
- Log the fault
- Notify appropriate entities (people or systems)

Recover from the fault:
- Disable source of events causing the fault
- Be temporarily unavailable while repair is being effected
- Fix or mask the fault/failure or contain the damage it causes
- Operate in a degraded mode while repair is being effected

⑥ Time or time interval when the system must be available
Availability percentage (e.g., 99.999%)
Time to detect the fault
Time to repair the fault
Time or time interval in which system can be in degraded mode
Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

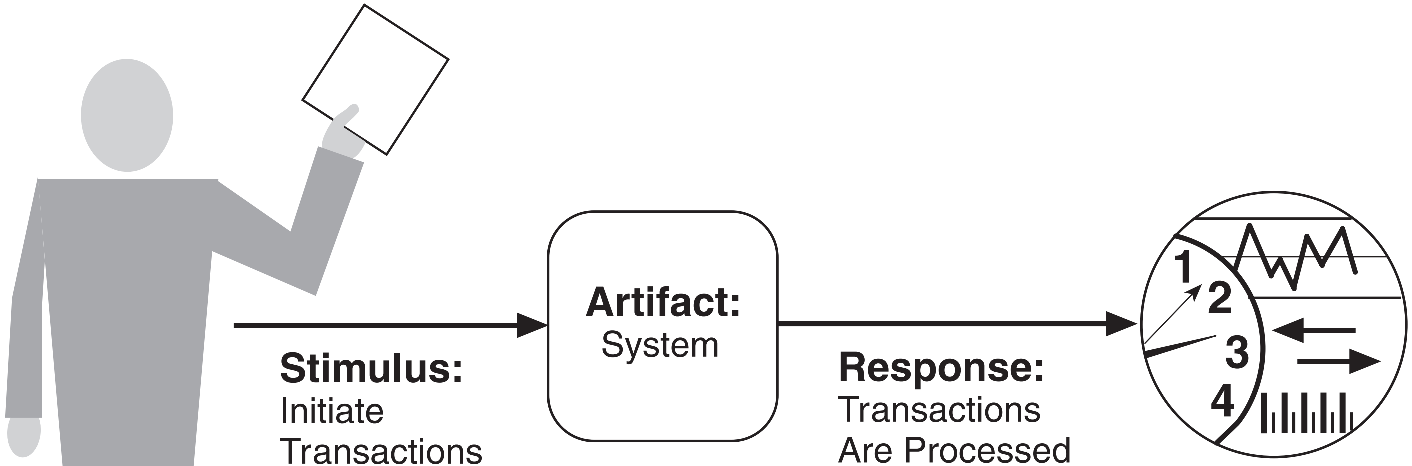


Performance Tactics

Control Resource Demand
- Manage Sampling Rate
- Limit Event Response
- Prioritize Events
- Reduce Overhead
- Bound Execution Times
- Increase Resource Efficiency

Manage Resources
- Increase Resources
- Introduce Concurrency
- Maintain Multiple Copies of Computations
- Maintain Multiple Copies of Data
- Bound Queue Sizes
- Schedule Resources

Performance Scenario



Source:
Users

Stimulus:
Initiate
Transactions

Artifact:
System

Environment:
Normal
Operation

Response:
Transactions
Are Processed

Response Measure:
Average
Latency
of Two
Seconds

Experience sharing

Have you come across any performance issues?

How did you address them?

Modifiability

- Deals with the ease with which we can make changes to a system

Modifiability

- What tactics can we use to increase modifiability?

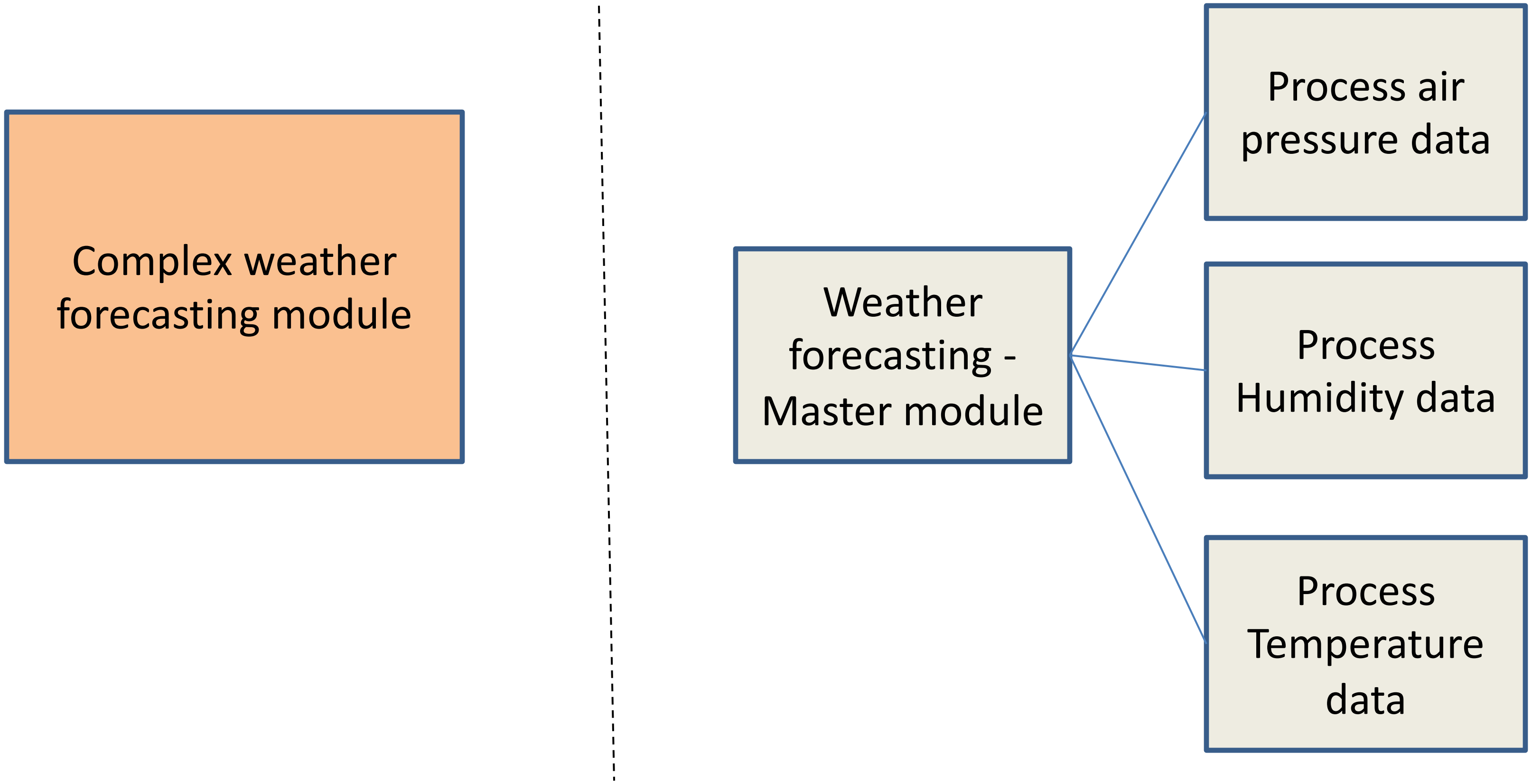
Modifiability

Tactics for Modifiability

- Reduce complexity – ex. smaller modules
- Encapsulate aspects that are likely to change – ex. interface to external systems
  (Information hiding)
- Increase cohesion & Reduce coupling

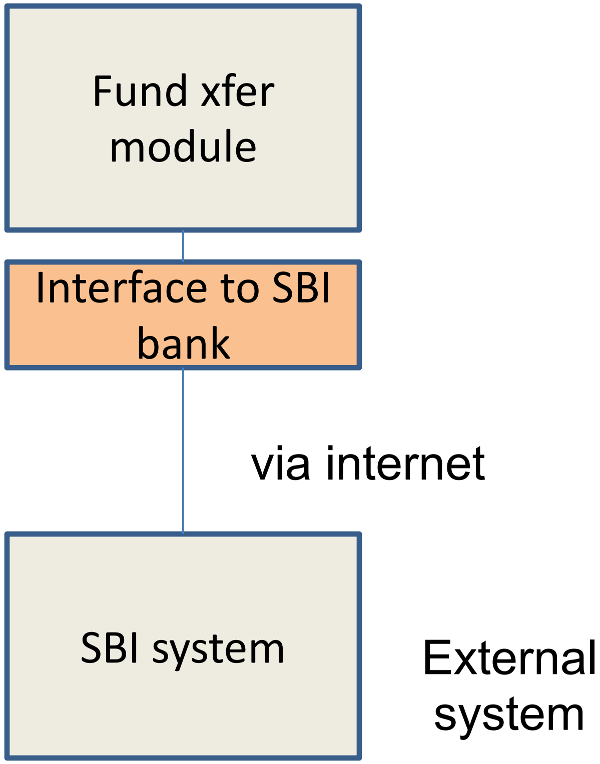
Reduce complexity

Weather forecasting

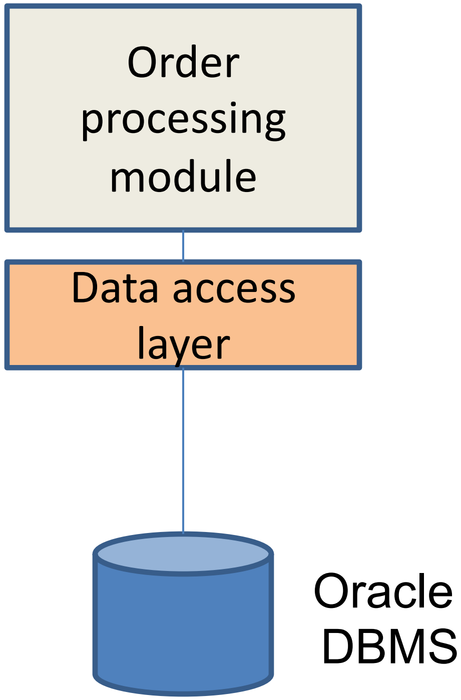

Complex weather forecasting module
Weather forecasting - Master module
Process air pressure data
Process Humidity data
Process Temperature data

Encapsulate things that are likely to change

Interface to external systems may change



Database technology may change



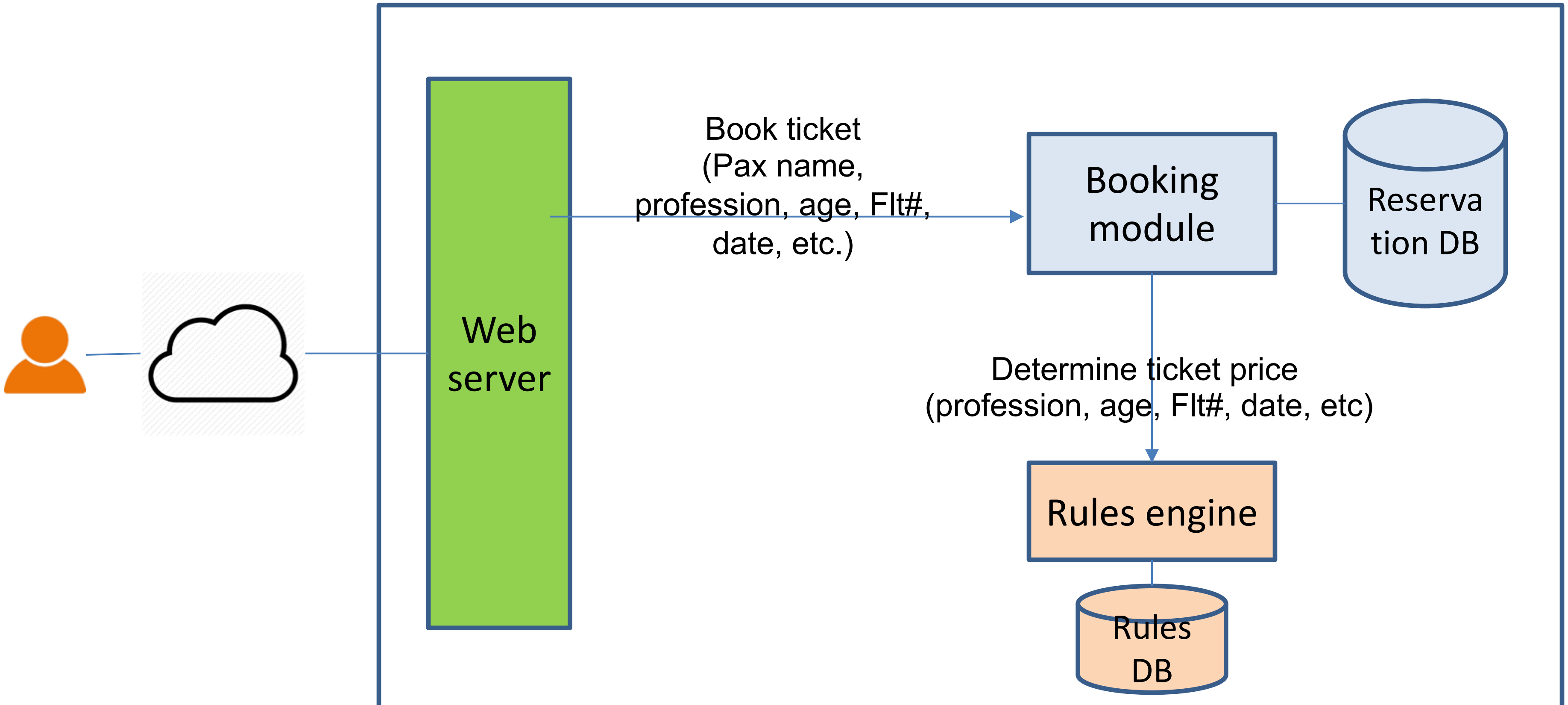
Protect system from variations

Variation	Example	
Changes in business rules	Income Tax rules, Pricing rules in airline change frequently	
Change in business process	Need one more approval	
Changes in module interface	Need to interface with a different SMS service provider	
Need to notify additional recipients about an event	‘New order’ event needs to be notified to Order Fulfilment module & Transport module	
Enhancing browser to handle new audio file format	mp3, wav, …	

Protect system from variations

Variation	Example	Tactic (Binding)
Changes in business rules	Income Tax rules, Pricing rules in airline change frequently	Rules engine (Deployment time)
Change in business process	Need one more approval	Work flow engine, Configuration files (Deployment time)
Changes in module interface	Need to interface with a different SMS service provider	Adaptor pattern (Build time)
Adding new recipients of notification of event	'New order' event needs to be notified to Order Fulfilment module & Transport module	Publish – Subscribe (Deployment time)
Enhancing browser to handle new audio file format	mp3, wav, …	Plugins (Deployment time)

Example of Rules engine


Web server
Book ticket (Pax name, profession, age, Flt#, date, etc.)
Booking module
Reservation DB
Determine ticket price (profession, age, Flt#, date, etc)
Rules engine
Rules DB

- All flights in the afternoon will have 5% discount
- Flights booked 3 months before will have 20% discount
- Senior citizens above 60 yrs will have 5% discount
- Defence personnel will get 10% discount

Exercise - RTO

Regional Transport Office (RTO) gave a project to Tata Infotech to develop a software to manage the service they provide to citizens, namely issue of driving license and registration of vehicles.

The requirement consisted of functional requirements such as application for license, recording results of test, etc. and a few reports.

As the project entered the design stage, new requirements started coming in. These consisted of many more reports. They also said that they may require more reports in the future.

What architecture should be adopted in this scenario?

Exercise - RTO

Regional Transport Office (RTO) gave a project to Tata Infotech to develop a software to manage the service they provide to citizens, namely issue of driving license and registration of vehicles.

The requirement consisted of functional requirements such as application for license, recording results of test, etc. and a few reports.

As the project entered the design stage, new requirements started coming in. These consisted of many more reports. They also said that they may require more reports in the future.

What tactic should be adopted in this scenario?

A module for generating ad-hoc reports which can generate a report based on specification provided by the user, can address most of the reports. Some examples of report generation tools are Ubiq, Zoho Analytics, BIRT, etc.

Exercise – Airline Loyalty module

An airline is building a comprehensive flight information & reservation system. It has several modules such as Flight planning, Flight schedules, Pricing, Reservations, Departure control, Analytics, etc.

As the airline industry is very competitive, the airline foresees many new requirements such as Loyalty card, etc. to come up in the future. The new modules to be built in the future would most probably be dependent on existing modules. For example, the Loyalty module will have to calculate Loyalty points based on flights undertaken by the customer, which is an output of Departure control module.

What architecture approach should be take to address this?

Exercise

An airline is building a comprehensive flight information & reservation system. It has several modules such as Flight planning, Flight schedules, Pricing, Reservations, Departure control, Analytics, etc.

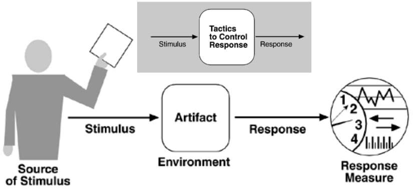
As the airline industry is very competitive, the airline foresees many new requirements such as Loyalty card, etc. to come up in the future. The new modules to be built in the future would most probably be dependent on existing modules. For example, the Loyalty module will have to calculate Loyalty points based on flights undertaken by the customer, which is an output of Departure control module.

What architecture approach should be take to address this?

Since new modules would require information from other modules, it is good idea to have a common database which can be accessed by all modules. If information is needed in real-time, one can adopt a Publish-Subscribe approach where all important events taking place in different modules are identified and published to interested subscribing modules

① **Source of stimulus.** entity (human, computer system, or any other actuator) that generated the stimulus
② **Stimulus -** condition that needs to be considered when it arrives at a system.
③ **Artifact**: Some artifact is stimulated – whole system or parts;
④ **Environment:** The stimulus occurs within certain conditions - Normal, safe mode, overload condition etc
⑤ **Response**: activity undertaken after the arrival of the stimulus.
⑥ **Response measure**: Measurable outcome, it should be measurable in some fashion so that the requirement can be tested.

Tactic is a design option for the architect

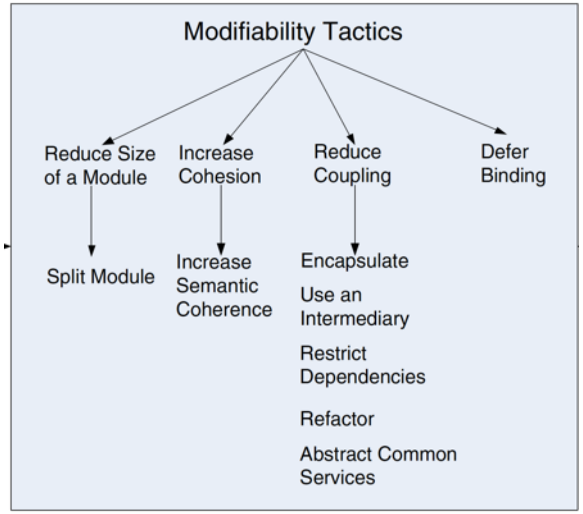


Scenarios should cover a range of

- Anticipated uses of (use case scenarios),
- Anticipated changes to (growth scenarios), or
- Unanticipated stresses (exploratory scenarios) to the system.

Quality Attributes &Tactics

Identify the Quality attributes and Tactics require to achieve them



Modifiability Tactics

Reduce Size of a Module
- Split Module

Increase Cohesion
- Increase Semantic Coherence

Reduce Coupling
- Encapsulate
- Use an Intermediary
- Restrict Dependencies
- Refactor
- Abstract Common Services

Defer Binding

① End user, developer, system administrator

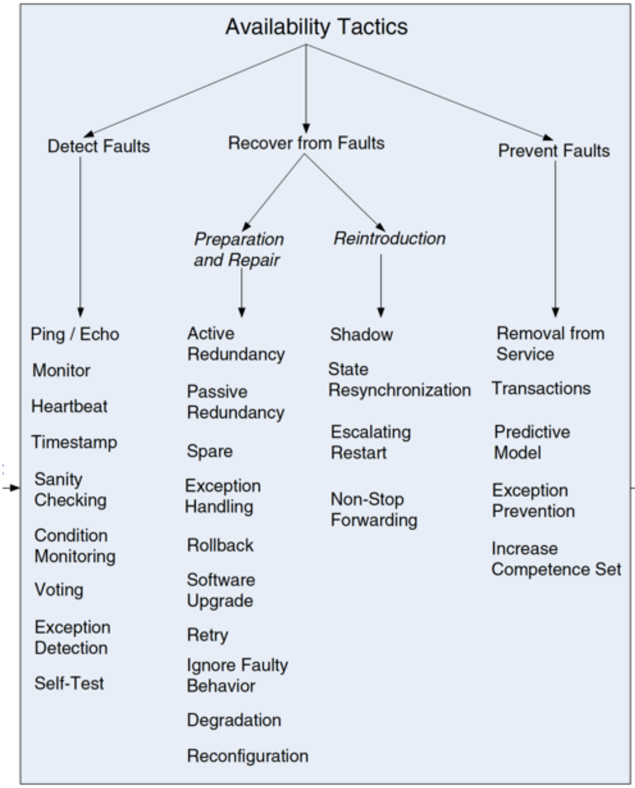
② A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology

③ Code, data, interfaces, components, resources, configurations, . . .

④ Runtime, compile time, build time, initiation time, design time

⑤ One or more of the following:
- Make modification
- Test modification
- Deploy modification

⑥ Cost in terms of the following:
- Number, size, complexity of affected artifacts
- Effort
- Calendar time
- Money (direct outlay or opportunity cost)
- Extent to which this modification affects other functions or quality attributes
- New defects introduced



Availability Tactics

Detect Faults
- Ping / Echo
- Monitor
- Heartbeat
- Timestamp
- Sanity Checking
- Condition Monitoring
- Voting
- Exception Detection
- Self-Test

Recover from Faults

Preparation and Repair
- Active Redundancy
- Passive Redundancy
- Spare
- Exception Handling
- Rollback
- Software Upgrade
- Retry
- Ignore Faulty Behavior
- Degradation
- Reconfiguration

Reintroduction
- Shadow
- State Resynchronization
- Escalating Restart
- Non-Stop Forwarding

Prevent Faults
- Removal from Service
- Transactions
- Predictive Model
- Exception Prevention
- Increase Competence Set

① Internal/external: people, hardware, software, physical infrastructure, physical environment

② Fault: omission, crash, incorrect timing, incorrect response

③ Processors, communication channels, persistent storage, processes

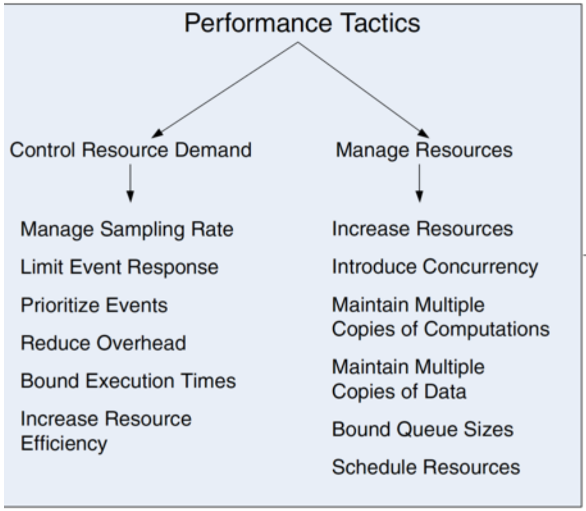
④ Normal operation, startup, shutdown, repair mode, degraded operation, overloaded operation

⑤ Prevent the fault from becoming a failure
Detect the fault:
- Log the fault
- Notify appropriate entities (people or systems)

Recover from the fault:
- Disable source of events causing the fault
- Be temporarily unavailable while repair is being effected
- Fix or mask the fault/failure or contain the damage it causes
- Operate in a degraded mode while repair is being effected

⑥ Time or time interval when the system must be available
Availability percentage (e.g., 99.999%)
Time to detect the fault
Time to repair the fault
Time or time interval in which system can be in degraded mode
Proportion (e.g., 99%) or rate (e.g., up to 100 per second) of a certain class of faults that the system prevents, or handles without failing

① Internal or external to the system
② Arrival of a periodic, sporadic, or stochastic event
③ System or one or more components in the system
④ Operational mode: normal, emergency, peak load, overload
⑤ Process events, change level of service
⑥ Latency, deadline, throughput, jitter, miss rate

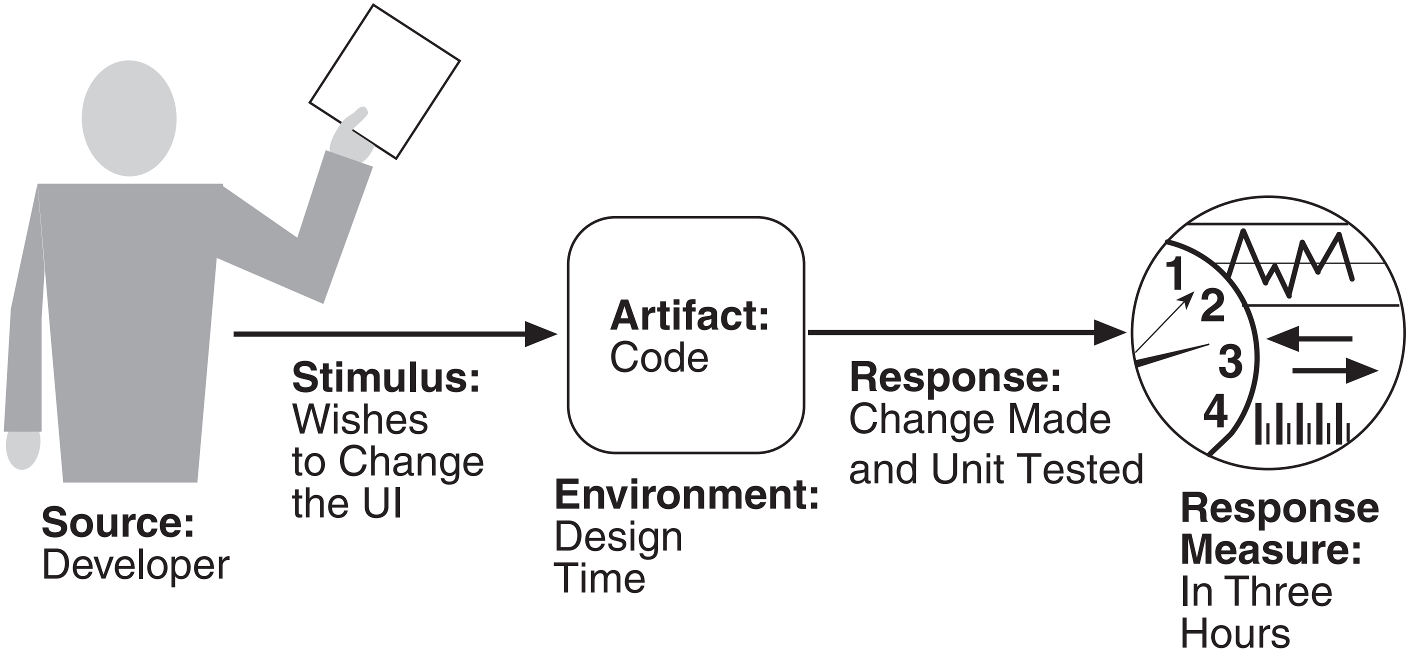


Performance Tactics

Control Resource Demand
- Manage Sampling Rate
- Limit Event Response
- Prioritize Events
- Reduce Overhead
- Bound Execution Times
- Increase Resource Efficiency

Manage Resources
- Increase Resources
- Introduce Concurrency
- Maintain Multiple Copies of Computations
- Maintain Multiple Copies of Data
- Bound Queue Sizes
- Schedule Resources

Modifiability Scenario



Source: Developer

Stimulus: Wishes to Change the UI

Artifact: Code

Environment: Design Time

Response: Change Made and Unit Tested

Response Measure: In Three Hours

Experience sharing

Have you come across any situation where you had tough time modifying the design to accommodate a requirement change?

Can you explain the situation? How did you address it?