SoftWare Architecture – Software Architecture depicts the organization of software components and how the software works.It shows – the arrangement of software components, Connection and interaction between software components and Distribution of software components across different computing systems. It is a blue print of a software system for the stakeholders to understand how the system would be once it is implemented. A blueprint should have higher level of abstractions (system functions) – it is not the detailing of how code should be written – at the same time should now have sea of information. Different stake holders expect certain information as a justification for building a software and all their expectations must be addressed in Software Architecture.

Business Req -> SA -> Software Design

Who are all the stake holders ?

Managing Director – What business purpose does it solve ? How unique is it from the competition?

Chief Technology Officer – Does it adhere to organization standard ? eg – A cloud based mail service provider would focus only on messaging technology.

Database Engineer – What information should be stored ? Where and how ? Access mechanisms and longevity ?

Application Development Team – How do I implement a complex scenario in code ? How should I organize my code ?

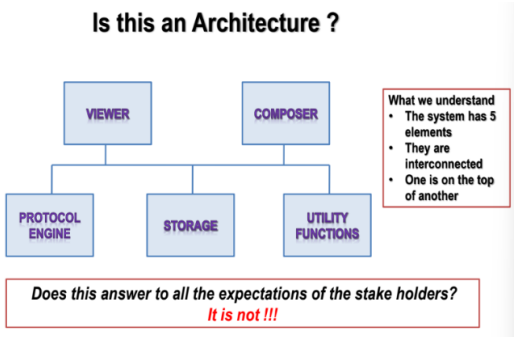
Users / Customers – Scalability , Performance and Reliability: Eas of use

Infrastructure Manager – Hardware system required .

Release and configuration Manager – Build Strategy Code Management at the highest level.

System Manitenance Team – How fast can I diagonise a fault and solve ?

The software architecture of a system is the set of structured needed to reason about the system which comprise software elements, relationships among them and properties of the both. The reason is to justify why is this required from the perspective of a stake holder who have vested interest in it.



What it does not address ?

Visible responsibilities :-

What do they do ?

How does their function relate to the system?

How have these elements been derived , is there any overlap?

Are these processes, or programs ? – How do they interact when the software executes ? Are they distributed ?

How are they deployed on a hardware ? What information does the system process ?

Significance of Connections – Sigfnify control or data, invoke each other synchronization and mechanism of communications.

Significance of Layout – Does level shown signify anything ? Was the type of drawing due to space constraints?

What an architecture should address ?-

A structure describing –

Modules – Services offered by each module, Their interactions to achieve the functionality.

Informing/Data Modeling

Achieving quality attributes

Processes and tasks that execute the software

Deployment onto hardware

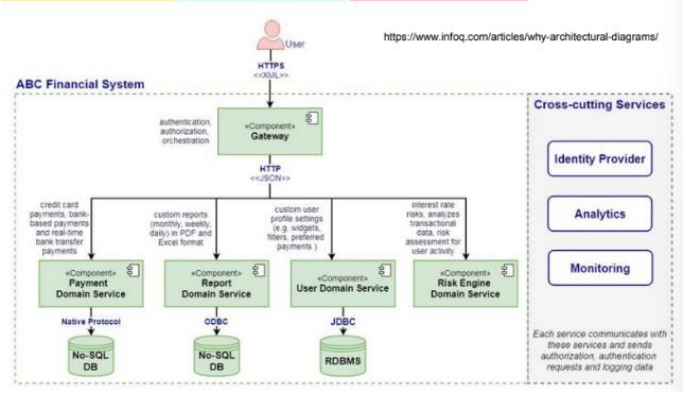
Development Plan

A behavioural Description –

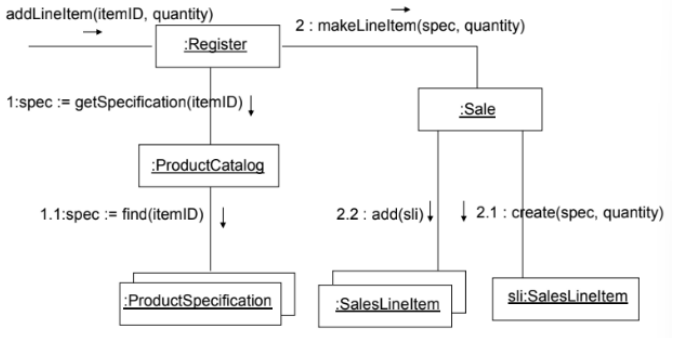
Describing how the structural elements execute “important” and “critical” scenarios

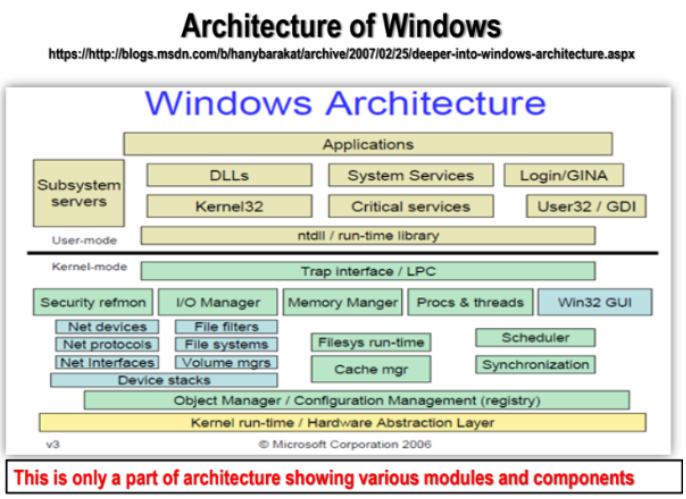
Eg - how does the system authenticates a mobile user, how does the system processes 1TB of data in a day ? How does it stream video uninterruptedly during peak load ?

These scenarios are mainly to implement various quality attributes

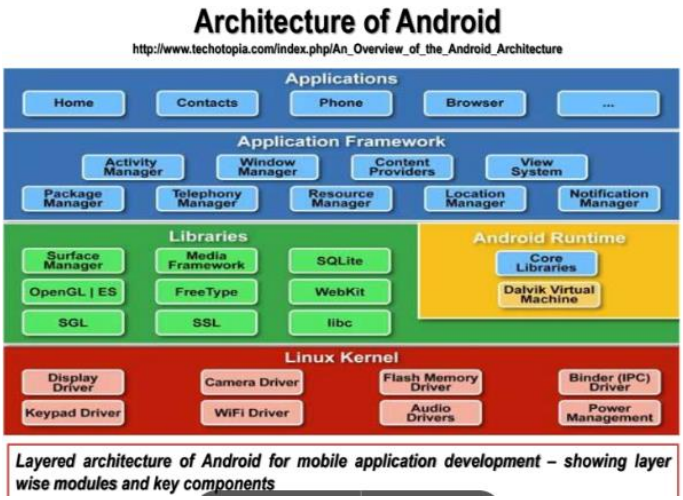


Example of during of Point-of-Sale system (partial design)





This is only a part of architecture showing various modules and components



Layered architecture of Android for mobile application development – showing layer wise modules and key components

Architecture Styles –

Architectural Styles is also termed as Architectural Pattern.

Style is a set of element types, a module or unit of computation(ex – an element to store data)

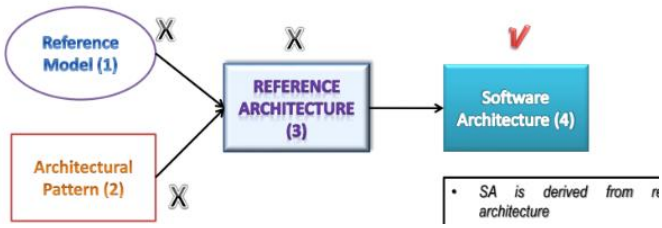
A set of interaction types – how do they interact with each other ?

Eg – Publisher / Subsriber is a modus operandii for exchange of information : A function call – Module “A” calling some components of “Module B” through a call manager.

Topolgy indicating interaction and interaction types.

Defining constraints- what are all allowed and what are all not allowed ?

Relationship between Reference Model and Architectural Pattern



Reference Model is a set of elements and the way they interact.

Architectural Pattern deal with non-functional requirements.

1,2 and 3 as a stand alone does not form software architecture.

Reference architecture is a best practice : J2EE /Java

Software Architecture is a derived from ref architecture. Actual Software System blueprint derived from requirement. Contains design decisions, describes how it is deployed and addresses quality of service concerns.

Structure Vs Views

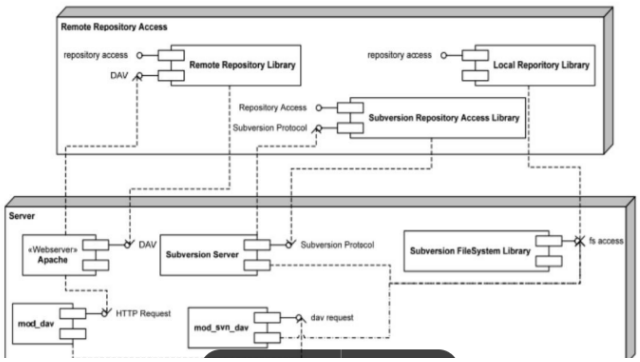
Structure – A set of elements itself as they exists in a software or hardware. Example – A module Structure is the set of the systems module and their organization.

Views:-

A view is a representation of architectural elements as written by and read by system stakeholders.

Example – A module view is the representation of a structure documented according to a template in a chosen notation.

Component - Connector View Example (Client-Server)



3 types of structure :-

Module Structure – Static in nature , modules (A set of code units) are assigned specific computational responsibilities. Modules are the basis of work assignments for programming Teams. Ex – Team “A” work on D/B , Team “B” on Business Rues and Team “C” on user Interface.

Component and Connector (C-C) Strcuture – Dynamic in nature. Focus on the way elements interact with each other at run-time to carryout the system functions. Elements are run-time components (responsible for computation) and Connectors (vehicle for communication)

Allocation Structure – Dynamic in nature, Show relationship between software elements and one or more external environments. Ex- What processor does each software element execute on ?

How do these structures help ?

Module Structure will bring answers to:-

What is the primary functional responsibility assigned to each module ?

What other software elements is a module allowed to use ?

What other software does it actually use and depends on ?

What modules are related to other modules by organization or specialization (inheritance)

Component and Connector (C-C) Structure will bring answer to :-

What are the major executing components and how do they interact at runtime ?

What are the major shared data stores ?

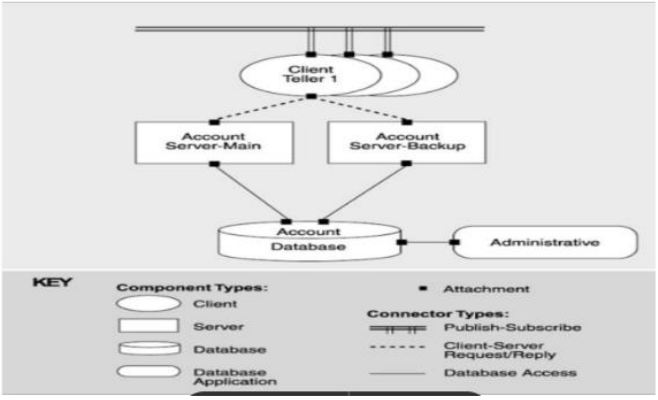
Which parts of the system are replicated ?

How does data progress through the system ?

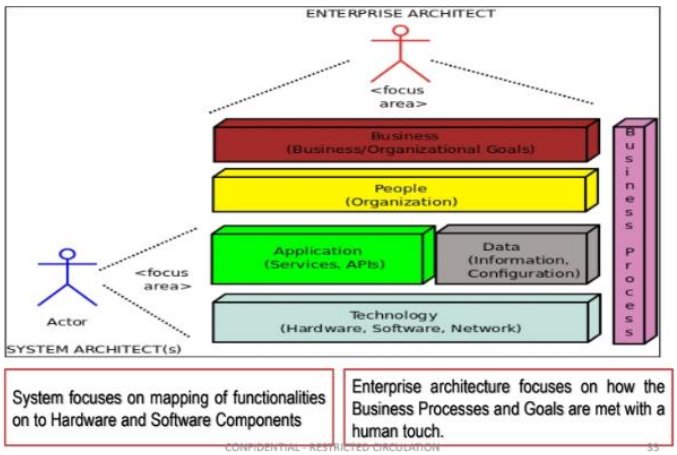
What parts of the system can run in parallel ?
Can the system's structure change as it executes and if so, how ?

Allocation Structure :-
What processor does each software element execute on ?
In what directories or files is each element stored during development , testing and system building ?
What is the assignment of each software element to development teams ?

A Typical C-C Structure Over View



System Vs Enterprise Architecture:-



Module Structure:-
Decomposition Structures – Modules are decomposed into smaller modules recursively until the modules are small enough to be easily understood. The decomposition structure determines, to a large degree, the system's modifiability, by assuring that likely changes are localized. Used as a basis for the development project organization.
Uses Structure – The units here are also modules, perhaps classes. A unit of software uses another in this model. The uses structure is used to engineer systems that can be extended to add functionality. Useful for incremental development.
Layer Structure – Components are hierarchical organized as a layer. This structure is used in system requiring portability, the ability to change the underlying computing platform (mobile devices)
Class Structure – The class structure allows one to reason about to reuse and the incremental addition of functionality. If any documentation exists for a project that has followed an object – oriented analysis and design process, it is typically this structure.
Data Model – The data model describes the static information structure in terms of data entities and their relationships. For example- in a banking system, entities will typically include account, customer and loan account has several attributes , such as account number, type (savings or checking) status and current balance.

Component and Connector Structure :-
Service Structure – The units here are services that interoperate with each other by service coordination mechanisms such as SOAP (Simple Object Access Protocol). This will help engineer a system composed of components developed anonymously and independently of each other.
Concurrency Structure – This component – and -connector structure allows the architect to determine opportunities for parallism and the locations where resource contention may occur. The concurrency structure is used early in the design process to identify the requirements to maange the issues associated with concurrent execution.

Allocation Structure:-
Deployment Structure – The deployment structure shows how software is assigned to hardware processing and communication elements. This structure can be used to reason about performance, data integrity, security and availability.
Implementation Structure – This structure shows how software elements (usually modules) are mapped to the file structure in the system's development, integration or configuration control environment.
Work Assignment Structure – This structure assigns responsibility for implementing and integrating the modules to the teams who will carry it out. This structure will also determine the major communication pathways among the teams :- regular teleconference, wikis, email lists and so forth.

Relationship Among the Structures

	Software Structure	Element Type	Relations	Useful for	Quality Attribute Affected
Module Structure	Decomposition	Module	"is' a sub module of	Resource Information hiding, encapsulation allocation,	Modifiability
	Uses	Module	Uses	Engineering subsets, Engineering extensions	Subsetability Extensibility
	Layers	Layer	Uses the service of	Incremental Development, Virtual Machines	Portability
	Class	Class, Object	Is an instance of	Object-Oriented Design Systems	Modifiability Extensibility
	Data Models	Data Entity	(One, many) to (one, many)	Consistency and performance	Modifiability Performance

Relationship Among the Structures

	Software Structure	Element Type	Relations	Useful for	Quality Attribute Affected
C & C Structure	Service	Service, ***Enterprise Software Bus*** (helps in distributing with the connected components), Registry	Runs concurrently with	Scheduling Analysis, Performance Analysis	Interoperability Modifiability
	Concurrency	Processes, Threads	Can run parallel	Identifying locations where resource contention exist	Performance, Availability

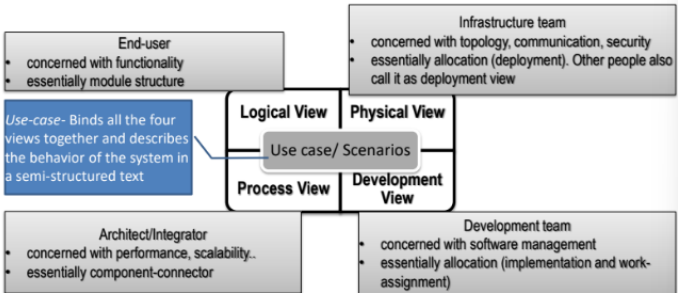
Relationship Among the Structures

	Software Structure	Element Type	Relations	Useful for	Quality Attribute Affected
Allocation Structure	Deployment	Components, Hardware elements	Allocated to Migrated to	Performance, Availability, Security Analysis	Performance, Security, Availability
	Implementation	Modules File structures	Stored in	Configuration control, Integration, test activities	Development efficiency
	Work Assignment	Modules, Organizational units	Assigned to	Project Management, Management of commonality	Development efficiency

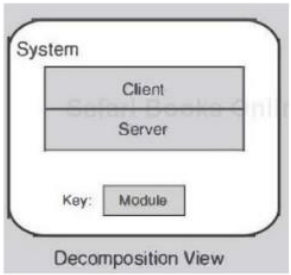
Which Structure to Choose?

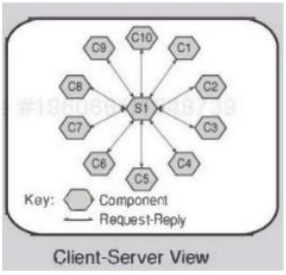
Philippe Kruchten's 4+1 View Model

- **Many opinions exist**
- **We will consider 4+1 view. This has been institutionalized as Rational Unified Process of Architecture description**



Two views of a Client-Server System





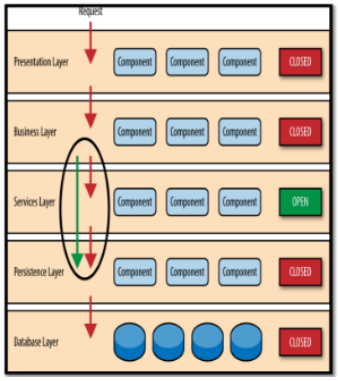
Client Software as one module & Server Software as other module
Module Decomposition View of Client-Server Architecture

11 Components and 10 Connections
CC Model View of Client-Server Architecture

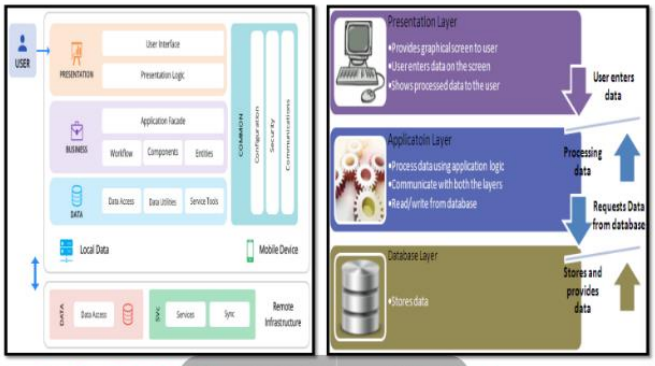
Architectural Patterns – Module Type

Layered pattern :

- When the uses *relation among software elements is strictly unidirectional*, a system of layers emerges
- A layer is a *coherent set of related functionality*. In a strictly layered structure, a layer can only use the services of the layer immediately below it.
- Layers are often designed as abstractions (virtual machines) that hide implementation specifics below from the layers above, engendering portability



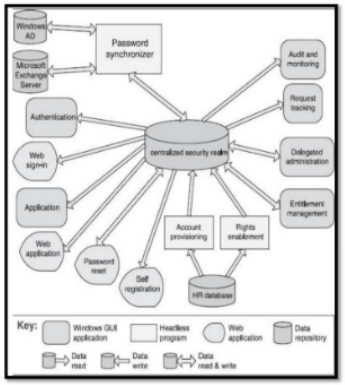
Layered Architecture - Example



Architecture Patterns : C-C- Type:-

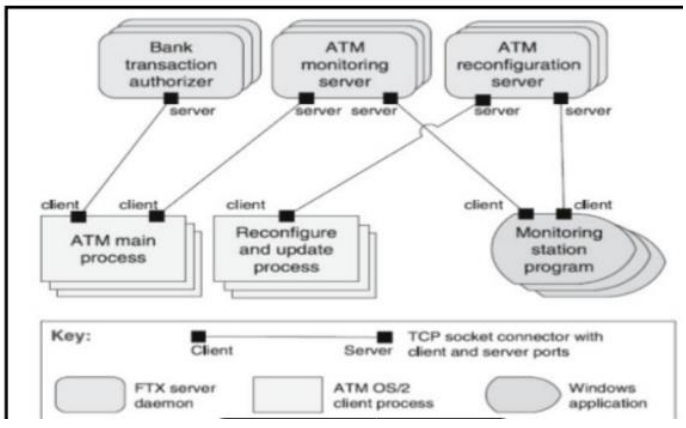
Shared-data pattern. This pattern comprises components and connectors that create, store, and access persistent data. The repository usually takes the form of a (commercial) database. The connectors are protocols for managing the data, such as SQL.

Client-server pattern. The components are the clients and the servers, and the connectors are protocols and messages they share among each other to carry out the system's work.



Core Banking System is a classical Example:-

Client – Server Architecture



Architectural Pattern – Allocation Types:

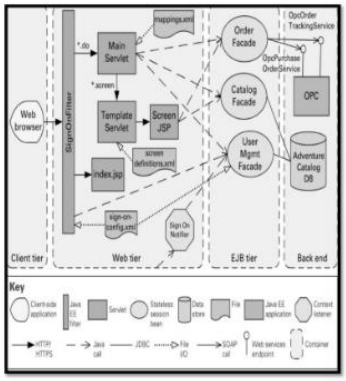
Multi-Tier pattern. Describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium.

Competence Center & Platform

In competence center, work is allocated to sites depending on the technical or domain expertise located at a site.

For example, *user-interface design is done at a site where usability engineering experts are located.*

In *platform, one site is tasked with developing reusable core assets* of a software product line, and other sites develop applications that use the core assets



What makes a good architecture ?

Process Perceptiveness

1. The architecture *should be the product of a single architect or a small group* of architects with an identified technical leader
2. The architect (or architecture team) should, on an on-going basis, *base the architecture on a prioritized list of well-specified quality attribute* requirements
3. The architecture *should be documented using views*. The views should address the concerns of the most important stakeholders in support of the project timeline
4. The architecture should be *evaluated for its ability to deliver* the system's important quality attributes.
5. The architecture should lend itself to *incremental implementation*, to avoid having to integrate everything at once as well as to discover problems early

Structural Rules of Thumb

1. The architecture should feature well-defined modules whose functional responsibilities are assigned on the *principles of information hiding and separation of concerns*
2. Your *quality attributes* should be achieved using *well-known architectural patterns and tactics*
3. The architecture should *never depend on a particular version* of a commercial product or tool.
4. Modules that *produce data should be separate* from modules that consume data
5. Don't expect a one-to-one correspondence between modules and components (*Place for parallelism*)
6. Every *process should be written* so that its assignment to a specific processor can be easily changed, perhaps even at runtime
7. The architecture should feature a small number of ways for *components to interact*
8. The architecture should contain a specific (and small) *set of resource contention areas*, the resolution of which is clearly specified and maintained.

Whys is Software Architecture so Important ?

An architecture will inhibit or enable a system's driving quality attributes (performance, modifiability, security, scalability, interoperability etc.)

The decisions made in architecture allows you to reason about and manage change as the system evolves (local, non-local and architecture)

The analysis of architecture enables early predictions of a system's qualities without waiting for the complete system developed.

A documented architecture enhances communication among stakeholderns (customer, user , Project manager, coder, tester)

The architecture is a carrier of the earliest and hence most fundamental, hardest – to – change design decisions (Single processor or distributed systems, layered or not, encryption required or not , OS dependency , etc).

An architecture defines a set of constraints on subsequent implementations (Trade off on quality attributes).

The architecture dictates the structure of an organization or vice-versa (Work Breakdown structure).

An architecture can provide the basis for evolutionary prototyping.

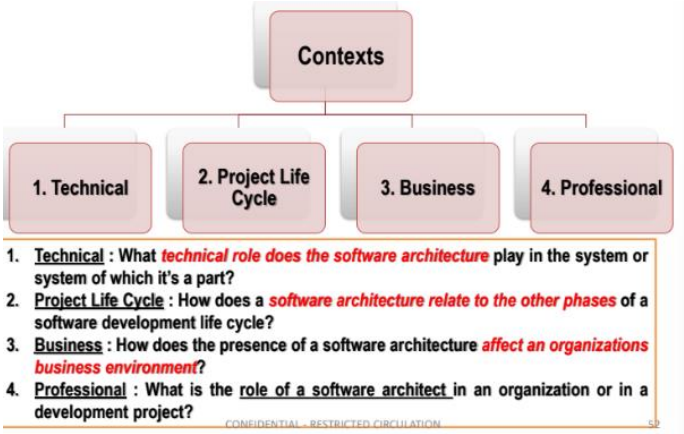
An architecture is the key artifact that allows the architect and project manager to reason about cost and schedule.

An architecture based development focuses attention on the assembly of components, rather than simply on their creation. (Transformation from KLOC to composing or assembling elements).

By restricting design alternatives, architecture channels the creativity of developers, reducing design and system complexity.

An architecture can be the foundation for training a new team member.

Many Contexts of Software Architecture:-



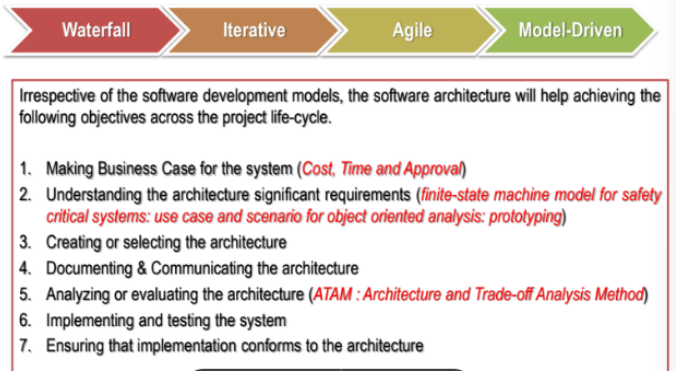
1. **Technical** : What *technical role does the software architecture* play in the system or system of which it's a part?
2. **Project Life Cycle** : How does a *software architecture relate to the other phases* of a software development life cycle?
3. **Business** : How does the presence of a software architecture *affect an organizations business environment*?
4. **Professional** : What is the role of a software architect in an organization or in a development project?

Technical Context of Software Architecture:-

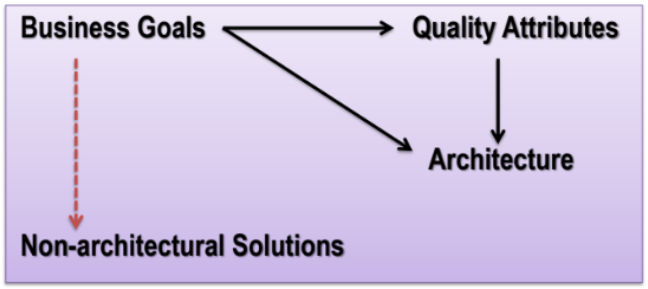
An Architecture will inhibit or enable a system's driving quality attributes (performance, modifiability, security, scalability, interoperability etc.,)

Sl. #	Software Attribute	Focus Elements of the Architecture
1	Performance	Time based behavior of elements, use of shared resources & frequency and volume of inter element communication
2	Availability	How components take over for each other in the event of a failure
3	Usability	Isolating the details of user interface & elements responsible for user experience
4	Testability	Testability of individual elements
5	Interoperability	Elements responsible for external interactions

Software Architecture in a Project Life Cycle Context:-

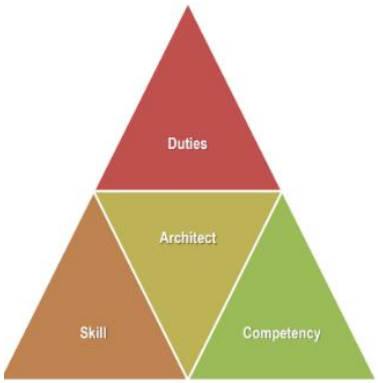


Software Architecture in Business Context:-



The Business Goals of a Development organization for acquiring a software system is different from a customer organization.

Software Architecture in a Professional Context:-



Must be able to meet requirements of different stakeholders:

Developing Organization : Low cost, keeping people employed

Marketing : Unique features, Low cost, parity with competition, short time to market

End User: UI, performance, reliability , security

Maintenance : Modifiability

Customer : Low cost, timely delivery

In Summary :-

- ✓ Software Architecture (SA) is a **BLUE PRINT** of the software (Visualization of the Final Outcome), will enable design and Coding.
- ✓ SA **BINDS ELEMENTS**, establishes relationship and reflects properties of them.
- ✓ SA deals primarily with **Non-Functional Requirements**
- ✓ SA is derived only from **Reference Architecture**
- ✓ **STRUCTURES & VIEWS** are the basic Software Architectural Engineering

- ✓ **Structure** refers to ***ELEMENTS*** in the software: **View** is a ***TEMPLATE*** to depict them.
- ✓ **Structure**: ***Module, Component & Connector and Allocation***
- ✓ **View** ***: Logical, Physical, Process and Deployment***
- ✓ Use **Philippe Kruchten's (4+1) Model to decide on the relevant structure**
- ✓ Understand the ***properties of a Good Architecture***
- ✓ The ***context of a Software Architecture*** varies from the person who views it (***Technical, Project Life Cycle, Business & Professional***)

Quality Attributes:-
Functionality – Ability of the system to fulfils its responsibility. However, in software quality attributes, only non-functional requirements are considered.
Constraints – is the condition that the system must satisfy while delivering its functionality (What is allowed and what is not allowed)
Design Decision – A constraint driven by external factors (ex- use of programming language : Service Oriented Architecture decision by top management)

Architectural Decision Vs Non-Architectural Decisions
User interface must be easy to use (this is vague) :- Radio button or check box (Non-architectural Decision) , Clear Text or screen layout (Non-architectural decions)
User interface should allow Redo/Undo at any level of depth (Architectural Decision).
Modifiability of the System (Vague) : Modular function (Architectural Deecison), Coding Techniques (Non-Architectural Decision).
Need to process 300 requests per seconds – Intercation among components, data sharing issues (Architectural Decision) , Choice of Algorithm (Non-Architectural Decision)

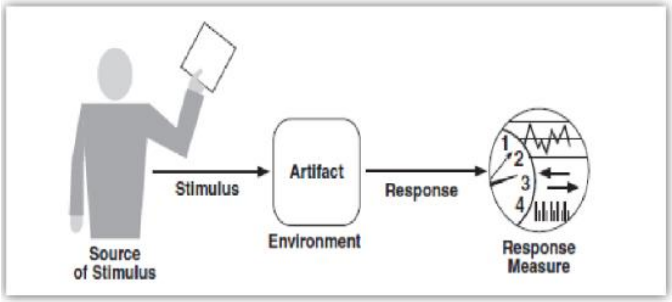
Quality Attribute – is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders. You can think of quality attribute as measuring the "goodness" of a product along some dimaension of interest to a stakeholder.
Availability, Interoperability, Modifiability, Perfomance, Security, Testability, Usbaility, Scalability, portability, mobility and reliability. '

Architecture Vs. Requirements

Requirements Type	Response of the Architecture
1. Functional Requirements : The very essence of the intended purpose of the system	Assigning sequence of responsibilities
2. Quality Attribute Requirements : These shall "Qualify" functional requirements	Various structures designed into the architecture
3. Constraints : Design decision with "Zero" degrees of freedom (static).	Accept the design decision and reconcile with other decisions.

Functionality does not decide a software architecture: It only helps to assign architectural responsibilities.

A typical template for defining a "Quality Attribute"
Source of Stimulus – This is sme entity (A human, computer system, or any other actuator) that generated the stimulus.
Stimulus – The stimulus is a condition that rquires a response when it arrives at a system.
Environment – The stimulus occurs under certain conditions. The system may be in an overhead condition or normal operation, or some other relevant state.
Artifact – Product produced during development. Some artifact is stimulated. This may be a collection of systems , the whole system or some piece or pieces of it.
Response – The response is the activity undertaken as the result of the arrival of the stimulus.
Response measure – When the response occurs, it should be measurable in some fashion so that the requirement can be tested.
Components of a Quality Attribute:-



Achieving Quality attributes thru' Architectural Tactics:- A tactic is a design decision that influences the achievement of a quality attribute response. There are 7 categories of Design Decision. These 7 categories are – Allocation of responsibility, Coordination model, data model, Management of resources, Mapping among architectural elements, Binding time decisions and Choice of Technology.

Design Decisions:-

1. **Allocation of responsibilities**
   - Identifying the important responsibilities, including basic system functions, architectural infrastructure, and satisfaction of quality attributes.
   - Determining how ***these responsibilities are allocated to*** non-runtime and runtime elements (namely, ***modules, components, and connectors***).
2. **Coordination model**
   - Identifying the elements of the system that must coordinate, or are prohibited from coordinating.
   - Determining the ***properties of the coordination, such as timeliness, currency, completeness, correctness, and consistency***.
   - Choosing the ***communication mechanisms*** (*stateful versus stateless, synchronous versus asynchronous, guaranteed versus nonguaranteed delivery, and performance-related properties such as throughput and latency*).
3. **Data model**
   - Choosing the major data abstractions, their operations, and their properties. (Determining ***how the data items are created, initialized, accessed, persisted, manipulated, translated, and destroyed***).
   - Compiling metadata needed for consistent interpretation of the data.
   - Organizing the data (data is going to be kept in a relational database, a collection of objects, or both)
4. **Management of resources**
   - These include hard resources (e.g., ***CPU, memory, battery, hardware buffers, system clock, I/O ports***) and soft resources (e.g., *system locks, software buffers, thread pools, and non-thread-safe code*).
5. **Mapping among architectural elements**
   - Two types of mappings.
   - Mapping between elements in different types of architecture structures (for example, ***mapping from units of development (modules) to units of execution*** (threads or processes).
   - Mapping between software elements and environment elements (for example, ***mapping from processes to the specific CPUs*** where these processes will execute).
6. **Binding time decisions**
   - Binding time decisions introduce *allowable ranges of variation* (For resource management, you can design a system to accept new peripheral devices plugged in at runtime)
7. **Choice of technology**
   - Example : ***Determining whether the available tools to support this technology choice*** (IDEs, simulators, testing tools, etc.) are adequate for development to proceed

Availability :-

Availability is Readiness of the software to carryout its task

Fault -> Error -> failure

Fault – A condition that causes a failure. Eg – a wrong program code.

Error – The difference between Actual Output vs expected output. Eg – Wrong code causing 4+3 as 6.

Failure – Inability of the system to perform a reqyired function: eg – wrong financial report.

The primary objective of availability attribute is to remove faults with serious consequences.

The secondary objective is to remove faults encountered most often by the users.

Availability refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval.

In Hardware Systems, Availability is measured as :-

MTBF /MTBF+MTTR

In software world, it is about what will make your system fail, how likely that is to occur (the probability), and that there will be some time required to repair it.

Failure Classification:-

Transient Failure – Only occurs with certain inputs and cannot be stimulated.

Permanent Failure – Occurs on all input.

Recoverable Failure – System can recovers without the help of operator.

Unrecoverable Failure – Operator has to intervene to recover the system.

Non-corrupting – Failure does not corrupt system state or data.

Corrupting Failure – System data gets corrupted.

Availability Vs Relaibility :-

Availability is seen as Percentage Uptime

Relaibility is the ability to continuously provide service without failure measured in terms of MTBF .. Over a Period of Time. So, the first thing as an architect , one must identify all SPOF (Single point of failures), introduce. Redundancy both in Hardware and Software System to introduce availability and Relaibilty.

General Scenario for Availability :-

Source of Stimulus – Internal/External , people, hardware, software, physical infrastructure, physical environment.

Stimulus – A fault of any of the following class can occur:-

Omission – A component fails to respond to an input.

Crash – A component repeatedly suffers omission faults.

Incorrect timing – A component responds but response is too early or late.

Incorrect response – A component responds with an incorrect value.

Environment – The stimulus occurs under certain conditions. The system may be in an overload condition or in normal operation or some other relevant state.

Artifact – Which resource is required to be nighly available ? Processors, Communication, Channels , Persistent Storage (devices which retains data even after the power is shut-off), Processes.

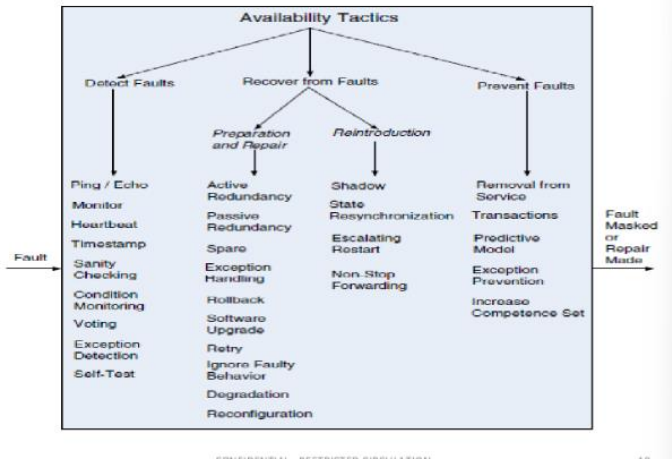
Response – Prevent the fault from becoming a failure

Detect a fault – Log the fault, Notify appropriate entitie4s.

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 damafe it causes, Operate in a degraded mode while repair is ebing effected.

Response measure – Availability percentage (eg – 99.999%) . Time to detect the fault time to repair the fault. Time or time interval in which system can be degraded mode

Proportion (eg 99%)



Availaibilty Tactics:-

Fault Detection Tactics:-

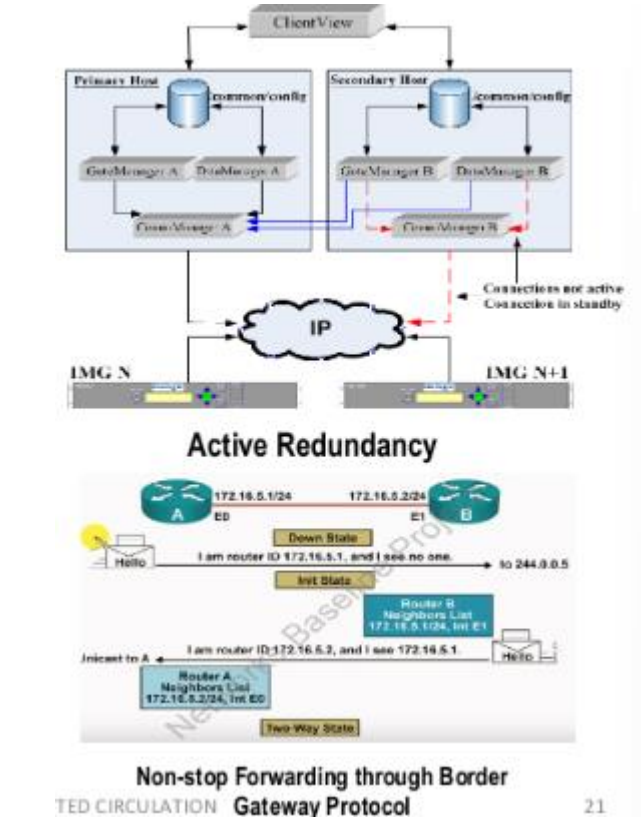
Ping/Echo – Used to determine reachability and the round-trip delay through the associated network path. Ping/echo requires a time threshold to be set.

Heartbeat – A server sends a periodical signal called heart beat. The signal might carry data as well. The listeners (software components, network components) listen for the signal. If no signal then it means the Server is DEAD. Eg – ATM (is a node) connected to a branch office can send the last transaction to branch office server.
Voting – TMR (Triple Modular Redundancy) which employs three components that do the same thing, each of which receives identical inputs, and forwards their outputs to voting logic. Majority voting wins!

Fault recovery Tactics :-
Active Redundancy – This refers to a configuration where all of the nodes (active or redundant. Spare) in a protection group receive and process identical inputs in parallel, allowing the redundant spares to maintain synchronous state with the active nodes.

Non-stop Forwarding -This predominantly done in network systems. The Data packets would be transferred through alternate path when one path fails (At the router level)

Roll-back :- This is a strategy through which a transaction is tracked for its success and in case of unexpected termination, the system is rolled-back to original state. ATM-pin-Fed amount – Power failure – Debit advise is a classical roll back.



Availability Tactics – Preventing from fault
Removal from service - This tactics refers to temporarily placing a system component in an out-of-service state for the purpose of mitigating potential system failures (Memory Leak).

Software Upgrade – periodic upgrade of software from patching prevents known vulnerability.

Predictive Model – A predictive model, when combined with a monitor, is employed to monitor the state of health of a system process to ensure that the system is operating within its nominal operating parameters (eg – monitoring software can predict the processor capacity of 80% threshold will in advance for upgradation).

Design Check List for Availability :-

Category - Allocation of Responsibility
Checklist – Ensure that there are responsibilities to do the following :- Log the fault, Notify appropriate entities (people or system), Disable the source of events causing the fault, Be temporarily unavailable,

Category – Coordination Model
Checklist – Ensure that coordination mechanism can detect an omission, crash, incorrect timing or incorrect response, Ensure that coordination mechanisms enable the logging of the fault, notification of appropriate entities, disabling of the source of the events causing the fault, Ensure that the coordination model supports the replacement of the artifacts used.

Category – Data Model
Checklist – Determine which portions of the system need to be highly available. Within those portions of the system need to be highly available . Within those portions, determine which data abstractions, along with their operations or their properties, could cause a fault of omission , a crash , incorrect timing behavior , or an incorrect response. For those data abstractions, operations and properties ensures that they can be disabled , be temporarily unavailable, or be fixed or masked in the event of a fault.

Category – Mapping Among Architectural
Checklist – Which processes on failed processors need to be reassigned at runtime. Which processors, data stores or communication channels can be activated or reassigned at runtime, How data on failed processors or storage can be served by replacement units,how quickly the system can be reinstalled based on the units of delivery provided. How to reassign runtime elements to processors, communication channels, and data stores.

Category – Resource Management
Checklist – Determine what critical resources are necessary to continue operating in the presence of a fault. Determine the availability time for critical resources.

Category – Binding
Checklist – Determine how and when architectural elements are bound. If late binding is used to alternate between components that can themselves be sources of faults, ensure the chosen availability strategy is sufficient to cover faults introduced by all sources.

Categiry - Choice of terminology
Checklist – Determine the available technologies that can (help) detect fault, recover from faults, or reintroduce failed components. Determine what technologies are available that help the response to a fault (eg – event logggers)

Explicit Software Availability Measures:-
1.POFOD: (Probability of On Demand): Likelihood when a system will fail, when a request is made. 0.001 POFOD means, 1 in every 1000 request will fail. Used in safety related system where every failure is critical (aircraft system).

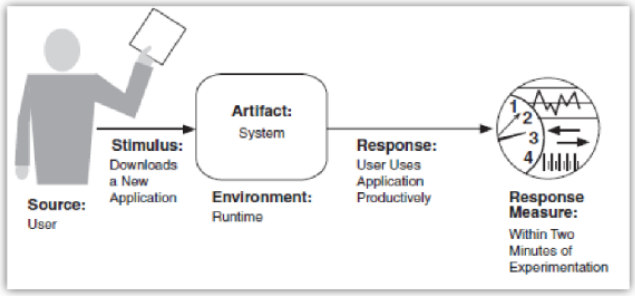
2.ROCOF: (Rate of Occurrence Of Failure). Its frequency of occurrence of failures. ROCOF 0.02 means 2 failures in each 100 time units

3. MTTF: (Mean Time To Failure). It's a measure of time between failures. A 500 MTTF means an average of 500 time units passes between failures. Used in transactions taking a lot of processing time., CAD Systems.

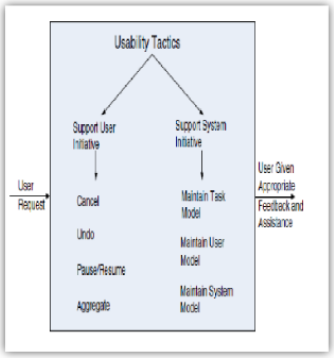
Software Usability – Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides. Usability comprises of 2 important domains namely "Learnability and Operability"
Learnability – Learning system features (how does the system help user learn quickly)
Operability – Using a system efficiently (suspending a task – its impact)
Operability – Minimizing the impact of error (cancelling a command issues incorrectly)
Operability – Adapting the system to user needs (automatically file in URLs based on a user's past entities)
Operability- Increasing confidence and satisfaction

General Scenario for Usability
Source of Stimulus – End user, possibly in a specialized role (Admin)
Stimulus – End user tries to use a system efficiently, learn to use the system, minimize the impact of errors, adapt the system, or configure the system.
Environment – Runtime or configuration time.
Artifact – Systems or the specific portion of the system with which the user is interacting.
Response – the system should provide the user with the features needed. (Learn -> Efficient -> Error Impact)
Response Measure – One or more of the following: task time, number of errors, number of tasks accomplished, user satisfaction, gain of user knowledge, ration of successful operations to total operations or amount of time or data lost when an error occurs.

Usability Scenario



Usability Tactics



Maintain task model. Example, knowing that *sentences start with capital letters* would allow an application to correct a lowercase letter in that position.

Maintain user model. Example, maintaining a user model allows the system to *pace mouse selection* so that not all of the document is selected when scrolling is required.

Maintain system model. A common manifestation of a system model is a *progress bar that predicts the time needed to complete* the current activity

Design Check list for Usability

SL #	Category	Checklist
1	Allocation of Responsibilities	Identify Modules / Components - Ensure that there are responsibilities to do the following: ❑ Learning how to use the system ❑ Efficiently achieving the task at hand ❑ Adapting and configuring the system ❑ Recovering from user and system errors
2	Coordination Model	❑ Check if the system needs to respond to : User action Ex., *can the system respond to mouse events and give semantic feedback in real time?* ❑ *Can long running events be cancelled?*
3	Data Model	❑ Determine the major data abstractions that are involved with user-perceivable behaviour. For example, the data abstractions should be designed to support *undo and cancel operations*: the transaction granularity should not be so great that cancelling or undoing an operation takes an excessively long time.

F

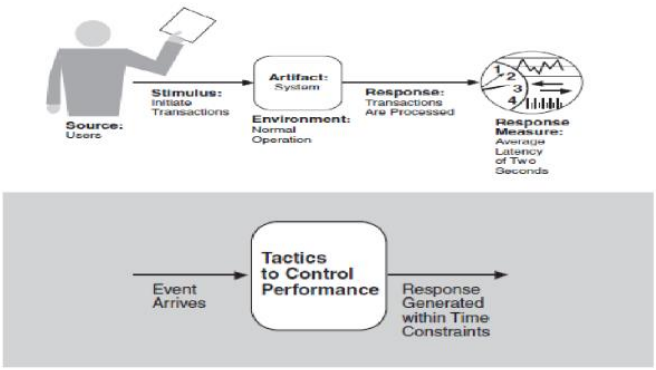
SL #	Category	Checklist
4	Mapping Among Architectural Elements	❑ Determine what mapping among architectural elements is visible to the end user (*for example, the extent to which the end user is aware of which services are local and which are remote*)
5	Resource Management	❑ Determine how the user can adapt and configure the system's use of resources. Ensure that resource limitations under all user-controlled configurations will not make users less likely to achieve their tasks. For example, *attempt to avoid configurations that would result in excessively long response times.*
6	Binding Time	❑ Determine which binding time decisions should be under user control and ensure that users can make decisions that aid in usability. For example, if *the user can choose, at runtime, the system's configuration, or its communication protocols*, or its functionality via plug-ins, you need to ensure that *such choices do not adversely affect the user's ability to learn system features*

SL #	Category	Checklist
7	Choice of Technology	❑ Ensure the chosen technologies help to achieve the usability scenarios that apply to your system. For example, *do these technologies aid in the creation of online help, the production of training materials*, and the collection of user feedback?

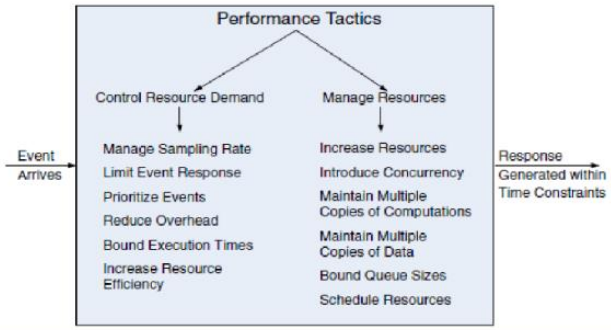
Performance:- It's about time and the software system's ability to meet timing requirements.
Throughput and Response time are the two terms associated with Performance.
Throughput refers to the number of transactions per second.
Response time refers to the time taken for completing a task.
An event can be Periodic (Regular interval), Stochastic (some probabilistic) and
Sporadic (Random). The response time can be measured as:

"General Scenario" for Performance
1. Source of stimulus. Internal or external to the system environment.
2. Stimulus. Arrival of a periodic, sporadic, or stochastic event
3. Environment. Operational, normal, emergency or peak load
4. Artifact. System or one or more components in the system.
5. Response . Process events, change level of service
6. Response measure. Latency, deadline, throughput, jitter, miss rate

Tactics Vs. Performance Attributes



Performance Tactics



One way to increase the performance of the system is by reducing the number of events processed by enforcing a sampling rate, or by limiting the rate at which the system responds to events.

Understanding Performance related Tactics

Control Resource Demand

1	Managing Sampling Rate	In Signal Processing Systems
2	Limiting Event Response	Discrete events can't be sampled- Achieved through queue size
3	Prioritizing Events	High, Medium & Low
4	Reducing Overload	Collocating coordinating elements in the same processor to avoid latency-eliminate all intermediaries - Reinitialize hash tables & virtual memory maps
5	Bound Execution Times	Limit execution time used to respond to events
6	Increase resource efficiency	Improving algorithms used in critical areas will decrease latency

Manage Resources

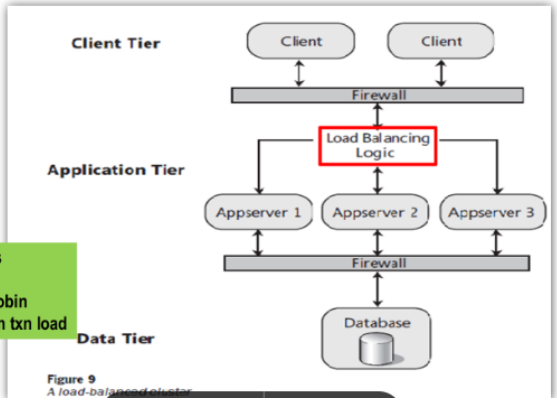
1	Increase Resource	Faster Processor; Additional processor & memories: SAN, Fiber Optic networks
2	Introduce concurrency	Parallel processing of events in different threads-schedule policies
3	Multiple copies of computation	To reduce Contention (conflict) – multiple servers in client Server pattern – load balancers
4	Maintain Multiple copies of data	Caching – local disk- SAN-DR Center
5	Bound Queue size	Controlling the maximum # of queue arrivals
6	Schedule Resources	Schedule based on FIFO the resources Processor, RAM , memories in case of conflict

P

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

P

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

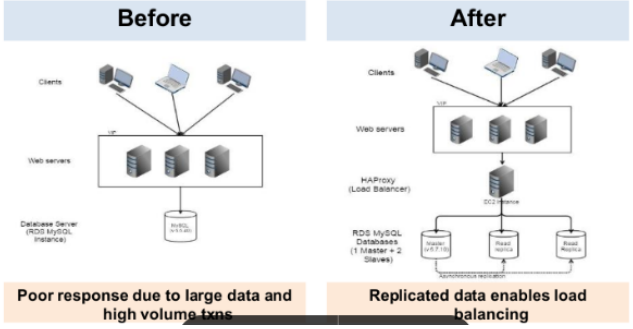


Approaches
- Round robin
- Based on txn load

Figure 9
A load-balanced cluster

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



Poor response due to large data and high volume txns

Replicated data enables load balancing

Design Check list for Performance

SL #	Category	Checklist
1	Allocation of Responsibilities	❑ Determine *the system's responsibilities that will involve heavy loading, have time-critical response requirements*, are heavily used, or impact portions of the system where heavy loads or time-critical events occur ❑ Responsibilities that result from a thread of control crossing process or processor boundaries ❑ Responsibilities to manage the threads of control ❑ Responsibilities for scheduling shared resources or managing performance
2	Coordination Model	❑ Determine the elements of the system that must coordinate with each other -directly or indirectly ❑ Support any introduced *concurrency , event prioritization*, or *scheduling strategy* ❑ Ensure that the required *performance response* can be delivered ❑ Can capture *periodic, stochastic, or sporadic event* arrivals, as needed

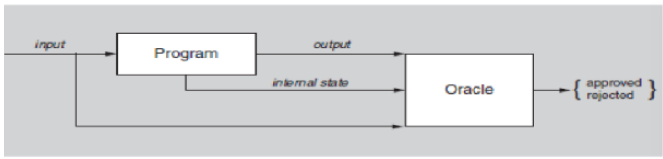
SL #	Category	Checklist
3	Data Model	❑ Determine those portions of the data model that will be *heavily loaded, have time-critical response requirements,* are heavily used ❑ Whether maintaining *multiple copies of key data* would benefit performance ❑ Whether *partitioning data would* benefit performance ❑ Whether *reducing the processing requirements* , ❑ Whether *adding resources to reduce bottlenecks*
4	Mapping Among Architectural Elements	❑ 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*. ❑ Determine where *introducing concurrency* (that is, allocating a piece of functionality to two or more copies of a component running simultaneously) *is feasible* and has a significant positive effect on performance.

SL #	Category	Checklist
5	Resource Management	❑ Determine which resources in your system are critical for performance. For example ❑ System elements that need to be aware of, and manage, time and other performance-critical resources ❑ Process/thread models ❑ *Prioritization of resources* and access to resources ❑ *Scheduling and locking strategies* ❑ *Deploying additional resources on demand to meet increased loads*
6	Binding Time	❑ Time necessary to complete the binding ❑ Additional overhead introduced by using the late binding mechanism

SL #	Category	Checklist
7	Choice of Technology	❑ Does your choice of technology give you the ability to set the following: ❑ *Scheduling policy* ❑ *Priorities* ❑ *Policies for reducing demand* ❑ Allocation of portions of the technology to processors ❑ Does your choice of technology introduce excessive overhead for heavily used operations?

Testability - Software testability refers to the ease with which software can be made to demonstrate
its faults through testing. Specifically, testability refers to the probability, assuming that
the software has at least one fault, that it will fail on its next test execution.

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)



G General Scenario for Testability:-
1.Source of stimulus. Unit testers, integration testers, system testers, acceptance testers, end users,
either running tests manually or using automated testing tools

2. Stimulus. A set of tests is executed due to the completion of a coding increment such as a class layer
or service, the completed integration of a subsystem, the complete implementation of the whole system, or
the delivery of the system to the customer.

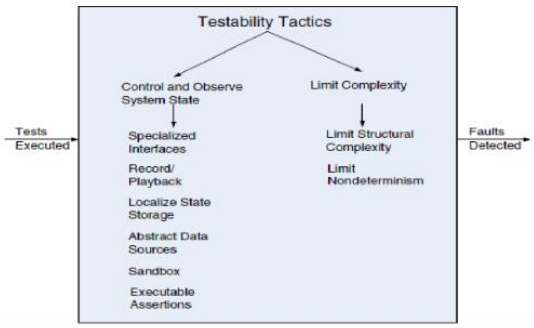
3. Environment. Design time, development time, compile time, integration time, deployment time,
run time

4. Artifact. The portion of the system being tested

5. Response. One or more of the following: execute test suite and capture results, capture activity that
resulted in the fault, control and monitor the state of the system

6. Response measure. Effort to find a fault, effort to achieve a given percentage of coverage,
probability of fault being revealed by the next test, time to perform tests, effort to detect faults, length of
time to prepare test environment, reduction in risk exposure (size(loss) × prob (loss))

Testability Tactics



common form of sandboxing is to virtualize resources. Using a sandbox, you can build a ersion of the resource whose behaviour is under your control. Non-deterministic systems are ard to test as they correspond to unpredictable events.

T

Testability Issues and Tactics:-

Issue	Tactic
We need to test during every release	Automated testing
Sometimes external systems are not available	Simulators
Sometimes lot of test data needs to be prepared	Copy production data into Test DB but remove / change sensitive data
When an error is found in a Complex system, we are not sure which module is the cause	Incremental integration of smaller modules Log the steps completed and intermediate results after each step

Design Checklist for Testability:-

SL #	Category	Checklist
1	Allocation of Responsibilities	❑ Determine which system responsibilities are most critical and hence need to be most thoroughly tested ❑ Execute test suite and capture results (external test or self-test) ❑ Capture (log) the activity that resulted in a fault *or* that resulted in unexpected (perhaps emergent) behaviour that was not necessarily a fault ❑ Control and observe relevant system state for testing Make sure the allocation of functionality provides high cohesion, low coupling, strong separation of concerns, and low structural complexity
2	Coordination Model	❑ Ensure the system's coordination and communication mechanisms: ❑ Support the execution of a test suite and capture the results within a system or between systems ❑ Support capturing activity that resulted in a fault within a system or between systems ❑ Support injection and monitoring of state into the communication channels for use in testing, within a ❑ system or between systems

SL #	Category	Checklist
3	Data Model	❑ Ensure that it is possible to capture the values of instances of these data abstractions ❑ Ensure that the values of instances of these data abstractions can be set when state is injected into the system, so that system state leading to a fault may be re-created ❑ Ensure that the creation, initialization, persistence, manipulation, translation, and destruction of instances of these data abstractions can be exercised and captured
4	Mapping Among Architectural Elements	❑ Determine how to test the possible mappings of architectural elements (especially mappings of processes to processors, threads to processes, and modules to components) so that the desired test response is achieved and potential race conditions identified. ❑ In addition, determine whether it is possible to test for illegal mappings of architectural elements.

SL #	Category	Checklist
5	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 do the following: ❑ Test resource limits ❑ Capture detailed resource usage for analysis in the event of a failure ❑ Inject new resource limits into the system for the purposes of testing ❑ Provide virtualized resources for testing
6	Binding Time	❑ Ensure that components that are bound later than compile time can be tested in the late-bound context ❑ Ensure that the full range of binding possibilities can be tested

SL #	Category	Checklist
7	Choice of Technology	❑ Determine what technologies are available to help achieve the testability scenarios that apply to your architecture. ❑ Are Technologies available to help with regression testing, fault injection, recording and playback, and so on? ❑ Determine how testable the technologies are that you have chosen (or are considering choosing in the future) and ensure that your chosen technologies support the level of testing appropriate for your system

I Interoperability – is about the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context. Not only having the ability to eexchange data (syntactic interoperability) but also having the ability to correctly interpret the data being exchanged (semantic interoperability).There are 5 levels of interoperability : Lowest being systems that do not share data at all and highest level indicates systems that work together seamlessly.

InInteroperability is the ease with which systems residing on didifferent servers and developed in different technologies can interact with each other. Service oriented architecture (SoA) isis uuseful in such situations, airlines – banks and hotels

G

G General Scenario for Interoperability:-

'. ***Source of stimulus***. A system initiates a request to interoperate with another system.

. ***Stimulus***. A request to exchange information among system

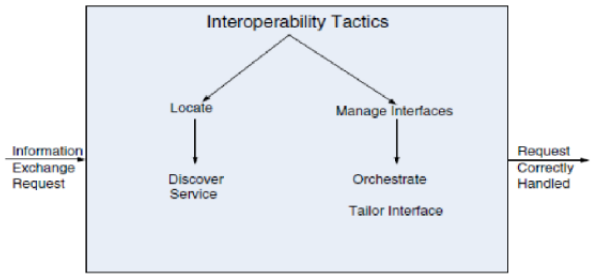
. ***Environment***. System(s) wishing to interoperate are discovered at runtime

. ***Artifact***. The systems that wish to interoperate

. ***Response*** . One or more of the following:
- ❑ The request is (appropriately) rejected and appropriate entities (people or systems) are notified.
- ❑ The request is (appropriately) accepted and information is exchanged successfully.
- ❑ The request is logged by one or more of the involved systems.

. ***Response measure***. ***Percentage of information exchanges correctly processed*** and ***ercentage of information exchanges correctly rejected***

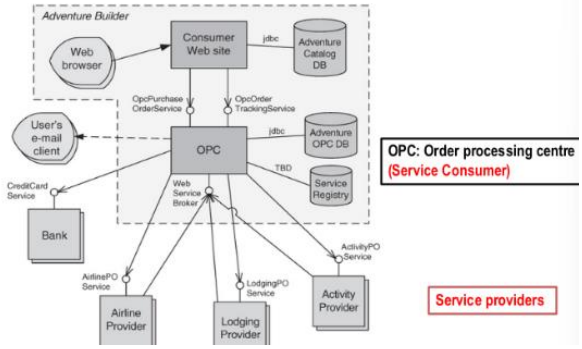
Interoperability Tactics



rchestrate : Managing sequence of invocations . Ex: Work flow engine in ERP : **Tailor Interface** fers to add / remove capabilities to interface Ex: Hiding particular function from untrusted user: **ocate and Discover services** : Google Map integration on the go.

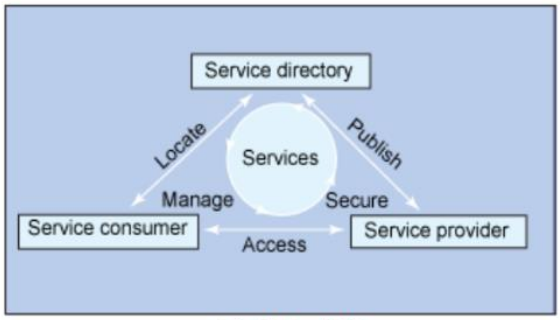
Example of Service Oriented Architecture in a Travel website

pplication is composed of different business services offered by internal systems and external systems



SOA - How does it work?

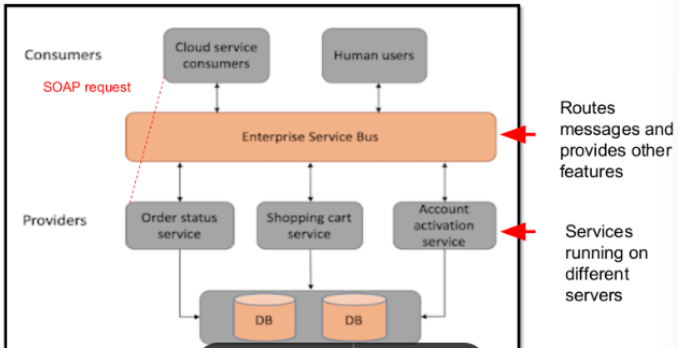
Service registry stores the physical location of services (service end-points)



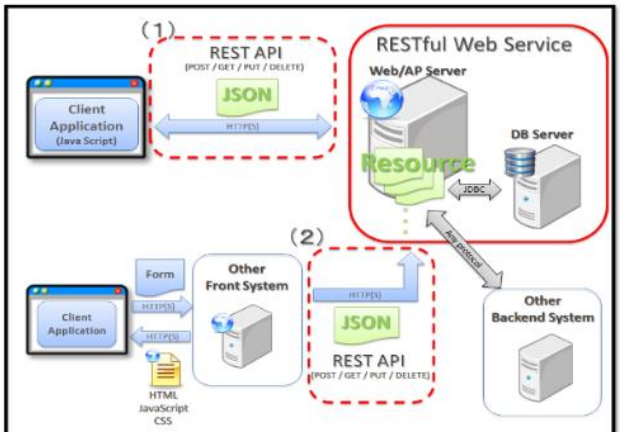
SOAP Vs. ReST

SOAP	ReST
• Simple Object Access Protocol • Technology to allow web-based applications to interoperate • Protocol specification for XML-based information that distributed applications can use • SOAP relies on HTTP and RPC (Remote Procedure Call) for message transmission • SOAP offers completeness • Implementation has greater support for security, availability	• Representation State Transfer • Technology to allow web-based applications to interoperate • Is a client-server-based architectural style that is structured around a small set of create, read, update, delete (CRUD) operations (called POST, GET, PUT, DELETE respectively in the REST world) • REST offers simplicity: *any HTTP client can talk to any HTTP server.* • Is more appropriate for read-only functionality, typical of mashups, where there are minimal **QoS**

Use of Enterprise Service Bus in SOA



Rest



Design Check list for Interoperability

SL #	Category	Checklist
1	Allocation of Responsibilities	❑ Ensure that responsibilities have been allocated to carry out the following tasks: ❑ Accept the request ❑ Exchange information ❑ Reject the request ❑ Notify appropriate entities (people or systems) ❑ Log the request (for interoperability in an untrusted environment, logging for nonrepudiation is essential)
2	Coordination Model	❑ Considerations for performance include the following ❑ Volume of traffic on the network ❑ Timeliness of the messages being sent by your systems ❑ Currency of the messages being sent by your systems ❑ Jitter of the messages' arrival times

SL #	Category	Checklist
3	Data Model	❑ Determine the syntax and semantics of the major data abstractions that may be exchanged among interoperating systems. ❑ Ensure that these major data abstractions are consistent with data from the interoperating systems
4	Mapping Among Architectural Elements	❑ For interoperability, the critical mapping is that of components to processors. The components that communicate externally are hosted on processors that can reach the network. ❑ The primary considerations deal with meeting the security, availability, and performance requirements for the communication.

SL #	Category	Checklist
5	Resource Management	❑ Ensure that interoperation with another system (accepting a request and/or rejecting a request) can never exhaust critical system resources ❑ Ensure that the resource load imposed by the communication requirements of interoperation is acceptable. ❑ Ensure that if interoperation requires that resources be shared among the participating systems, an adequate arbitration policy is in place
6	Binding Time	❑ Ensure that it has a policy for dealing with binding to both known and unknown external systems. ❑ Ensure that it has mechanisms in place to reject unacceptable bindings and to log such requests

SL #	Category	Checklist
7	Choice of Technology	❑ For any of your chosen technologies, are they "visible" at the interface boundary of a system? If so, what interoperability effects do they have? ❑ Do they support, undercut, or have no effect on the interoperability scenarios that apply to your system? ❑ Ensure the effects they have are acceptable. ❑ Consider technologies that are designed to support interoperability, such as web services. ❑ Can they be used to satisfy the interoperability requirements for the systems under your control?

M Modifiability:-

)eals with the ease with which we can make changes to a system

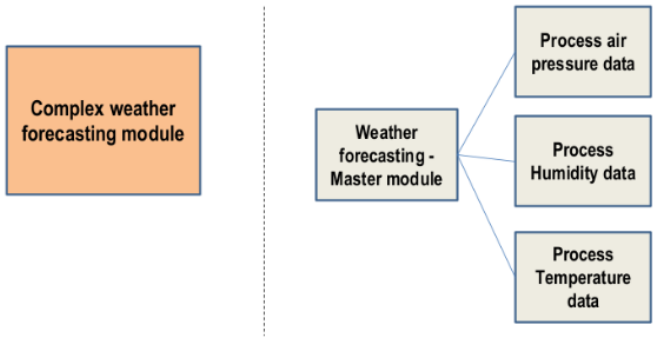
Tactics for Modifiability

- **Reduce complexity – ex. smaller modules**
- **Encapsulate aspects that are likely to change – ex. interface to external systems**
  (Information hiding principle was introduced by David Parnas)
- **Increase cohesion & Reduce coupling**

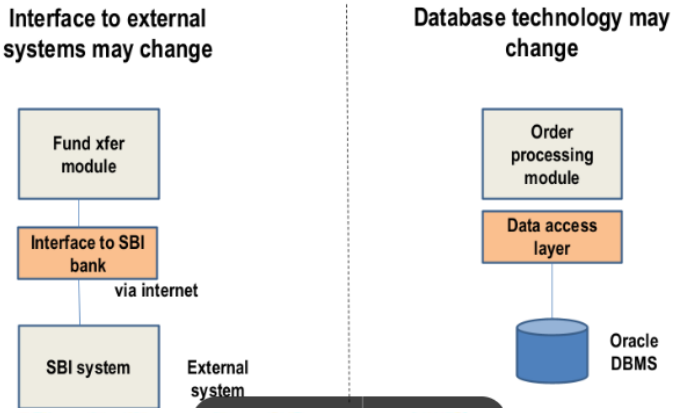
S

Reduce Complexity

Weather forecasting



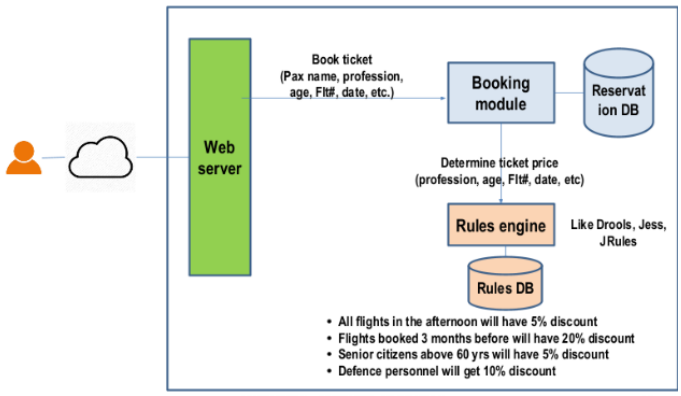
Encapsulate things that are likely to change



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



➔ Security

Primary Goals of Security : The CIA Triad

The First Principle

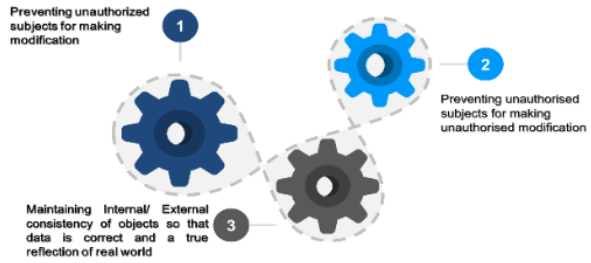
- Security Controls are evaluated on how well they address these 3 tenets (belief)
- Vulnerabilities & Risks are assessed Based on the threats they pose against CIA

Confidentiality: The measure used to ensure the *protection of Secrecy of data,* objects or resources

Goal	To prevent or minimize unauthorized access to data
Security Measures	❑ Access Control ❑ Data Classification ❑ Encryption ❑ Steganography ❑ Network Traffic Padding ❑ Training
Attacks focused on Confidentiality	❖ Capturing Network Traffic ❖ Stealing Password Files ❖ Social Engineering ❖ Port Scanning ❖ Shoulder Surfing ❖ Sniffing ❖ Escalation of privileges
Object is a passive element such as file, computer and a network connection **Subject** is an active element such as user, programs and computers	

Integrity - Integrity: The Act of Protecting the correctness & Reliability of
data. This ensures that data remains correct, unaltered and preserved. Integrity must be ensured when the data is at rest, in transit or in process.

3 Different Dimensions of Integrity

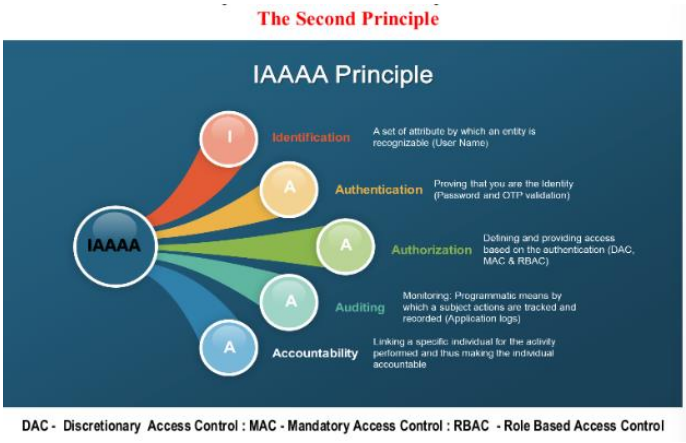


Technology Controls for Maintaining Integrity	
✔ Restrict Access to data & Objects	✔ Rigorous authentication process
✔ Activity Logging	✔ Intrusion Detection System
✔ Object/ Data Encryption	✔ Verification of object integrity across the storage
✔ Hash total verification	
Attacks on Integrity	
✔ Virus attacks	✔ Logic Bombs
✔ Unauthorized access	✔ Errors in coding & application
✔ Malicious modification	✔ Intentional Replacement
✔ System backdoors	✔ Trojan Horse

Availability - It refers to the ability of the system to grant data access to authorized objects timely and uninterrupted. It mean that efficient, uninterrupted access to objects & prevention of Denial of Service (DoS).

Technology Controls for improving availability	
Elimination of Single Point of Failure (SPOF)	Acceptable level of performance
SLA's for correction of interruption	Build redundancy
Reliable back-up & restore	Prevent data loss/ destruction
Design effective intermediary delivery system	Effectively using access control
Monitoring performance & Network Traffic	Defense-in depth mechanism
Threats to Availability	
Device Failure	Software Errors
Environmental Issues (heat, flooding, power loss)	DoS attacks
Object Destruction	Communication interruption

Primary Goals of IAAAA



Security Tactics:-

A hacker may gain access to confidential data such as credit card number , password stored in DB – Encrypt data in DB , Firewall

A hacker may snoop on messages (intercept messages) flowing in the internet – Encrypt messages

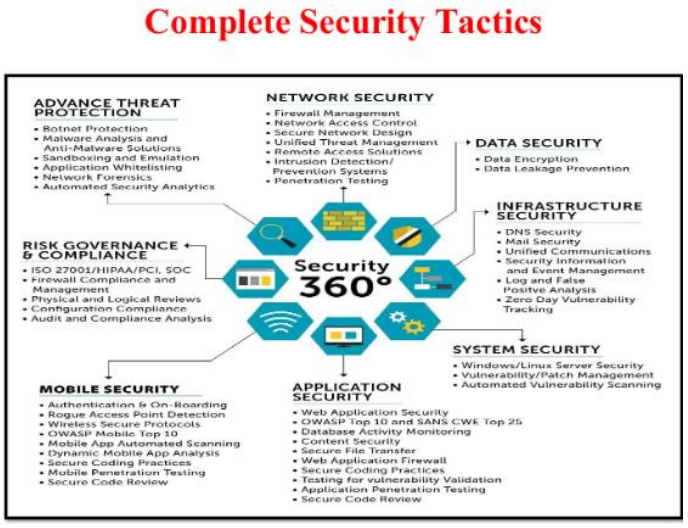
Malicious people may Forge electronic documents – Digital Signature

Malicious people may create websites that look like original websites to gather your user id and password – Digital Certificate

Difference between Digital Signature and Digital Certificate

A digital signature is a mechanism that is used to verify the authenticity of a document or a message.

A digital certificate is a certificate issues by a trusted third party called a Certificate Authority (CA) to verify the identity of the certificate holder.



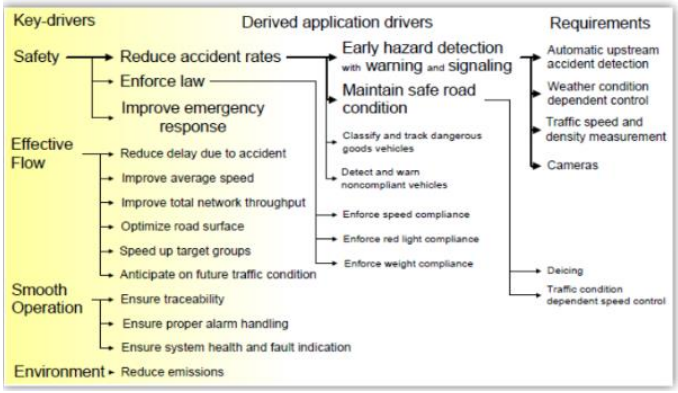
Architecturally Significant Requirements (ASR) – is a requirement that will have a profound effect on the architecture - that is, the architecture might well be dramatically different in the absence of such a requirement. The more difficult and important the QA requirement, the more likely it is to significantly affect the architecture, and hence to be an ASR.

Drivers: - Need for Categorizing requirements into Critical, Important and Useful. Researchers indicate that only 40 % of software features are used even in highly critical applications.

Some requirement are technically more challenging than others - Simple functional requirements like 'System should be able to create an order', 'System should keep track of inventory', etc. are straight forward to implement. Some functional requirements such as 'We should be able to detect fraud in real time and block the user' may not be that straight forward. Some non-functional requirements like the 'System should be easy to customize for a different organization" may not be straight forward to implement. Such challenging requirements are called 'Architecturally Significant Requirements' (ASR)

Examples of ASR - Audit trail, Alert & Notifications, Variety of data input mechanisms, Third party interaction, Workflow, User behaviour analysis (ex. in e-commerce), All such requirements need to be probed in detail and we need to understand the specific scenarios in order to decide the design approach.

Example of probing the requirements – Traffic Management System



ASR from Requirement Documents:- Do requirements specification follow the principle of "MoSCoW" style (must, should, could, won't), or as a collection of "user stories "? Do the requirement capturing exercise include Quality Attributes as well in equal proportions as per Software Engineering Body of Knowledge (SWEBOK) recommendations?Have you come across BSR (Basic Skill Requirements) with explicit requirements on " high usability", "System Modularity", "performance through Resilience", "Fault detection" etc.,? So, in the absence of BSR not reflecting the QA clearly, it is a challenge for an ARCHITECT to extract ASR . However, that makes the Role of an Architect requisite.

Look for leads from Design Decision Documents ..,

Design Decision Category	Look for Requirements addressing ...,
Allocation of Responsibilities	Planned evolution of responsibilities, user roles, system modes, major processing steps, commercial packages
Coordination Model	Properties of the coordination (timeliness, currency, completeness, correctness, and consistency) Names of external elements, protocols, sensors or actuators (devices), middleware, network configurations (including their security properties) Evolution requirements on the list above
Data Model	Processing steps, information flows, major domain entities, access rights, persistence, evolution requirements
Management of Resources	Time, concurrency, memory footprint, scheduling, multiple users, multiple activities, devices, energy usage, soft resources (buffers, queues, etc.) Scalability requirements on the list above
Mapping among Architectural Elements	Plans for teaming, processors, families of processors, evolution of processors, network configurations.
Binding Time Decisions	Extension of or flexibility of functionality, regional distinctions, language distinctions, portability, calibrations, configurations
Choice of Technology	Named technologies, changes to technologies (planned and unplanned)

ASR from interviewing stakeholders (QAW) – Quality Attributes Workshop – Expectations : List of architectural Drivers.

Presentation & Introduction : Everyone introduces themselves, briefly stating their background, their role in the organization, and their relationship to the system being built.

Business Mission Presentation : The stakeholder representing the business concerns behind the system (typically a manager or management representative) spends about one hour presenting the system's business context, broad functional requirements, constraints, and known quality attribute requirements. The quality attributes that will be refined in later steps.

Architectural Plan Presentation : At this point in the workshop, the architect will present the system architectural plans as they stand. This lets stakeholders know the current architectural thinking, to the extent that it exists.

Identification of Architectural Drivers : The facilitators will share their list of key architectural drivers that they assembled during steps 2 and 3, and ask the stakeholders for clarifications, additions, deletions, and corrections. The idea is to reach a consensus on a distilled list of architectural drivers that includes overall requirements, business
drivers, constraints, and quality attributes.

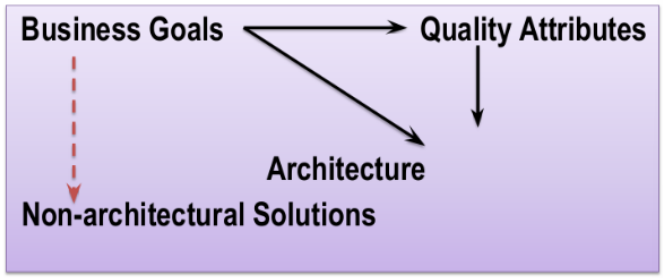
Scenario Brainstorming : Each stakeholder expresses a scenario representing his or her concerns with respect to the system. Facilitators ensure that each scenario has an explicit stimulus and response. The facilitators ensure that at least one representative scenario exists for each architectural driver listed in step 4.

Scenario Consolidation : Similar scenarios are consolidated where reasonable. Facilitators ask stakeholders to identify those scenarios that are very similar in content. Scenarios that are similar are merged, as long as the people who proposed them agree and feel that their scenarios will not be diluted in the process

Scenario Prioritization : Prioritization of the scenarios is accomplished by allocating each stakeholder a number of votes equal to 30 percent of the total number of scenarios generated after consolidation.

3. ASR from Business GOALS

Format : "X" to "Y" by when? : "X " - Current State : "Y" - Future state



1. Business goals often lead to quality attribute requirements
2. Business goals may directly affect the architecture without precipitating a quality attribute requirement at all
3. No influence at all

Business GOAL Categories

Format : "X" to "Y" by when? : "X " - Current State : "Y" - Future state

#	Category	Example
1	Contributing to the **growth and continuity** of the organization	Market share by 15%
2	Meeting **financial** objectives	30 % Profit after tax
3	Meeting **personal** objectives	Learn New Technologies
4	Meeting **responsibility to employees**	Reduce work load by 8 %
5	Meeting responsibility to **society**	Green Computing – Less CO2
6	Meeting responsibility to **state**	Regulatory Compliance
7	Meeting responsibility to **shareholders**	1:3 Share Dividend
8	Managing **market position**	IPR - 50 Design Registrations
9	Improving **business processes**	Productivity Improvement
10	Managing the **quality and reputation** of products	Automated Testing Strategy
11	Managing change in **environmental factors**	US Dollar Volatility

Example of Business Goal – Govt. wanted to speed-up cashless transactions. So, government introduced BHIM app . This translates into QAs:- Usability (easy steps) and Security (3-factors)

Format for Presenting Final Business Goals

2. Goal-subject	3. Goal-object	5. Goal	4. Environment	6. Goal-measure (examples, based on goal categories)	7. Value
Any stakeholder or stakeholder group identified as having a *legitimate* interest in the system	Individual System Portfolio Organization's employees Organization's shareholders Organization Nation Society	Contributing to the growth and continuity of the organization Meeting financial objectives Meeting personal objectives Meeting responsibility to employees Meeting responsibility to society Meeting responsibility to state Meeting responsibility to shareholders Managing market position Improving business processes Managing quality and reputation of products Managing change in environmental factors	Social (includes political) Legal Competitive Customer Technological	Time that business remains viable Financial performance vs. objectives Promotion or raise achieved in period Employee satisfaction; turnover rate Contribution to trade deficit/ surplus Stock price, dividends Market share Time to carry out a business process Quality measures of products Technology-related problems Time window for achievement	1–n 1–10 H-M-L Resources willing to expend

PALM – Pedigreed Attribute eLicitation Methods Capturing Business Goal

"Pedigree" means that the business Goal has a strong background:

PALM overview presentation. Overview of PALM, the problem it solves, its steps, and its expected outcomes.

Business driver's presentation. Briefing of business drivers by project management. What are the goals of the customer organization for this system? What are the goals of the development organization?

Architecture driver's presentation. Briefing by the architect on the driving business and quality attribute requirements: the ASRs.

Business goals elicitation. Using the standard business goal categories to guide discussion, capture the set of important business goals as scenarios, consolidate and prioritize them to identify important Goals.

Identification of potential quality attributes from business goals. For each important business goal scenario, participants describe a quality attribute that would help achieve it

Assignment of pedigree to existing quality attributes drivers. For each architectural driver named in step 3, we identify which business goals it is there to support. For example: Why is there a 40-millisecond performance requirement? Why not 60 milliseconds? Or 80 milliseconds?

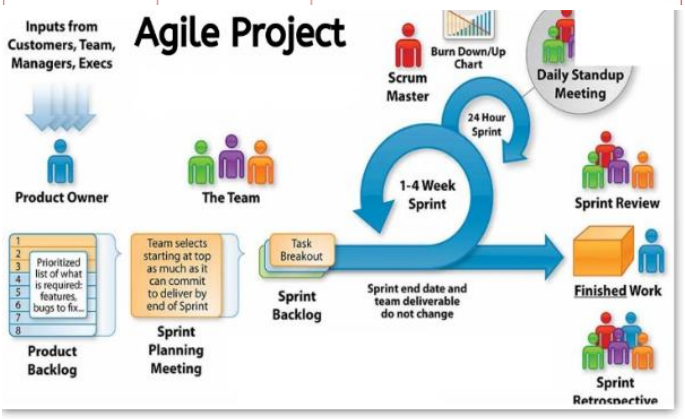
Exercise conclusion. Review of results, next steps, and participant feedback.

ASR – Utility Tree

Quality Attribute	Attribute Refinement	ASR
Performance	Transaction response time	A user updates a patient's account in response to a change-of-address notification while the system is under peak load, and the transaction completes in less than 0.75 second. (H,M)
		A user updates a patient's account in response to a change-of-address notification while the system is under double the peak load, and the transaction completes in less than 4 seconds. (L,M)
	Throughput	At peak load, the system is able to complete 150 normalized transactions per second. (M,M)
Usability	Proficiency training	A new hire with two or more years' experience in the business becomes proficient in Nightingale's core functions in less than 1 week. (M,L)
		A user in a particular context asks for help, and the system provides help for that context, within 3 seconds. (H,M)
	Normal operations	A hospital payment officer initiates a payment plan for a patient while interacting with that patient and completes the process without the system introducing delays. (M,M)

Quality Attribute	Attribute Refinement	ASR
Configurability	User-defined changes	A hospital increases the fee for a particular service. The configuration team makes the change in 1 working day; no source code needs to change. (H,L)
Maintainability	Routine changes	A maintainer encounters search- and response-time deficiencies, fixes the bug, and distributes the bug fix with no more than 3 person-days of effort. (H,M)
		A reporting requirement requires a change to the report generating metadata. Change is made in 4 person-hours of effort. (M,L)
	Upgrades to commercial components	The database vendor releases a new version that must be installed in less than 3 person-weeks. (H,M)
Extensibility	Adding new product	A product that tracks blood bank donors is created within 2 person-months. (M,M)

Quality Attribute	Attribute Refinement	ASR
Security	Confidentiality	A physical therapist is allowed to see that part of a patient's record dealing with orthopaedic treatment but not other parts nor any financial information. (H,M)
	Integrity	The system resists unauthorized intrusion and reports the intrusion attempt to authorities within 90 seconds. (H,M)
Availability	No Downtime	The database vendor releases new software, which is hot-swapped into place, with no downtime. (H,L)
		The system supports 24/7 web-based account access by patients. (L,L)



Architecture in Agile Projects

Agile Development	Traditional Development
Individuals and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Agile Manifesto : 12 Principles

#	Principles	#	Principles
1	Early and continuous delivery of valuable software	7	Working software is the primary measure of progress
2	Welcome changing requirements, even late in development	8	Agile processes promote sustainable development
3	Deliver working software frequently, from a couple of weeks to a couple of months	9	Continuous attention to technical excellence and good design enhances agility.
4	Business people and developers must work together	10	Simplicity is essential
5	Build projects around motivated individuals	11	The best architectures, requirements, and designs emerge from self-organizing teams
6	Face-to-Face conversation	12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly

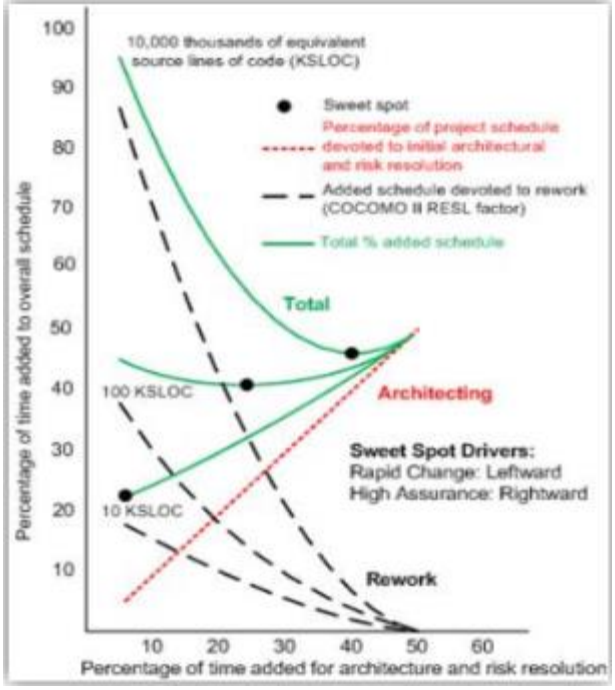
Analytics – Up-front Work Vs Agility

There is one line representing each of these three projects, starting near the Y axis and descending, at different rates, to the X axis at the 50 mark. This shows that adding time for up-front work reduces later rework.

when you sum each of those downward-trending lines (for the 10, 100, and 1,000 KSLOC projects) with the upward sloping line for the up-front (initial architecture and risk resolution) work, you get the second set of three lines, which start at the Y axis and meet the upward sloping line at the 50 mark on the X axis.

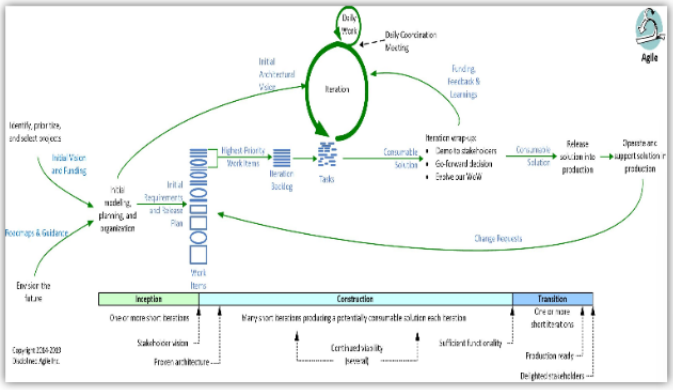
These lines show that there is a sweet spot for each project. For the 10 KSLOC project, the sweet spot is at the far left. This says that devoting much, if any, time to up-front work is a waste for a small project.

For the 100 KSLOC project, the sweet spot is at around 20 percent of the project schedule. And for the 1,000 KSLOC project, the sweet spot is at around 40 percent of the project schedule.



A project with a million lines of code is enormously complex, and it is difficult to imagine how Agile principles alone can cope with this complexity if there is no architecture to guide and organize the effort.

BDUF (Big Design Up-front Model) for Agile



Attribute Driven Design :- ADD is a method to architect the system based on the quality attributes

The method is as follows:

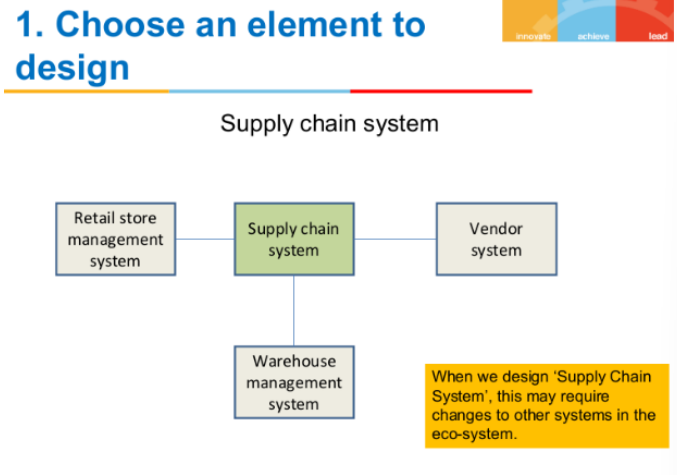
Choose one element to design (in case of a new system, the element is the whole system)

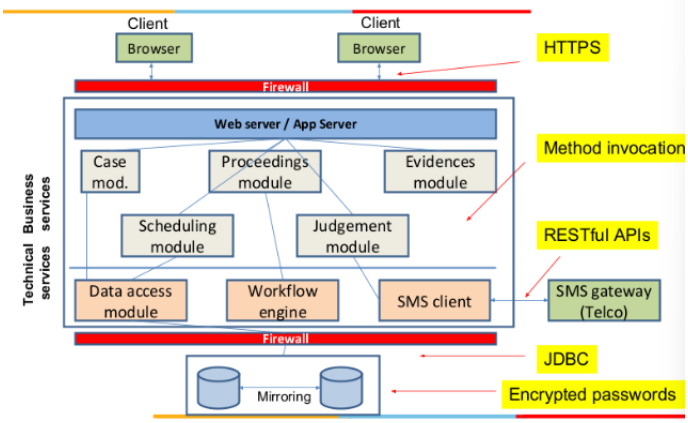
Identify the ASRs pertaining to this element

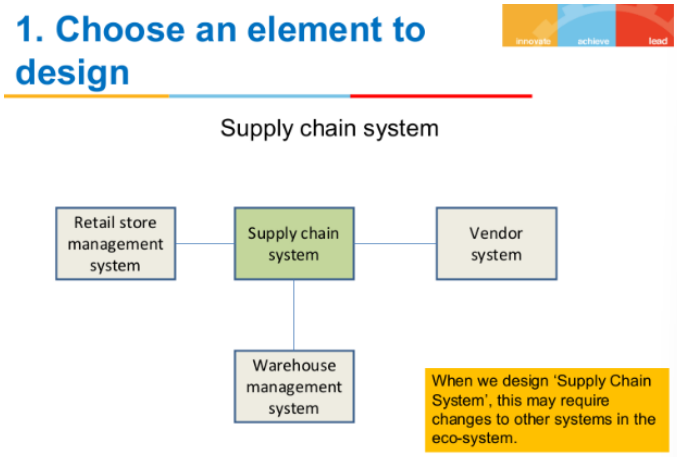
Generate a design solution for the chosen element

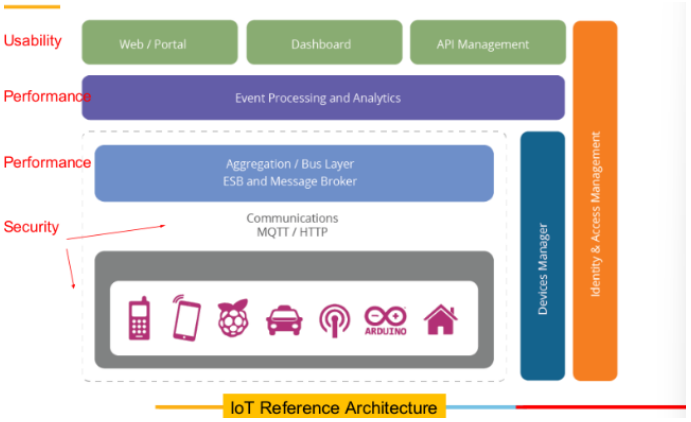
Verify & refine requirements and generate input for next iteration

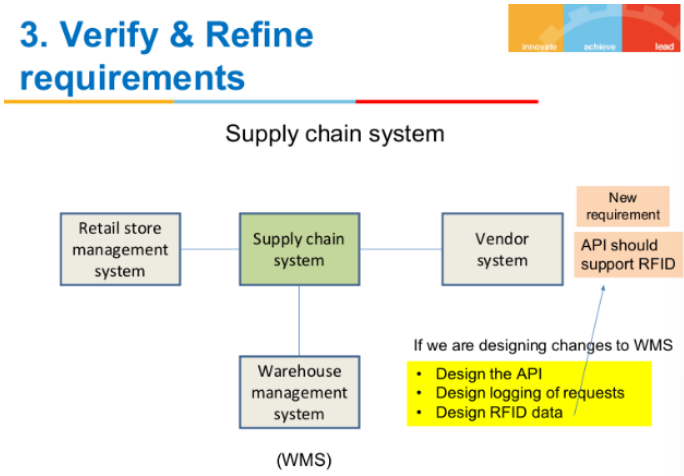
Repeat the above steps until done







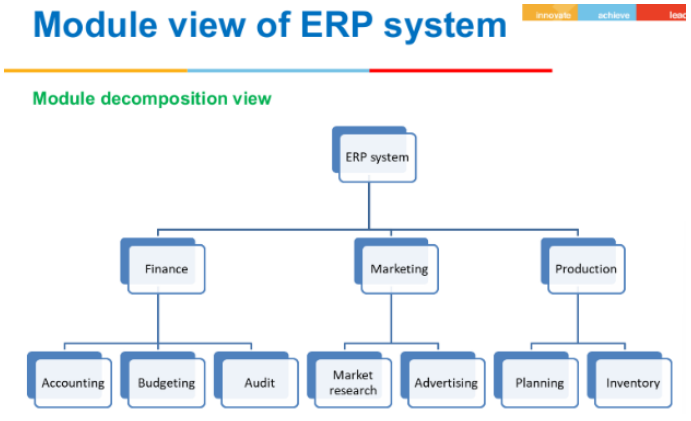


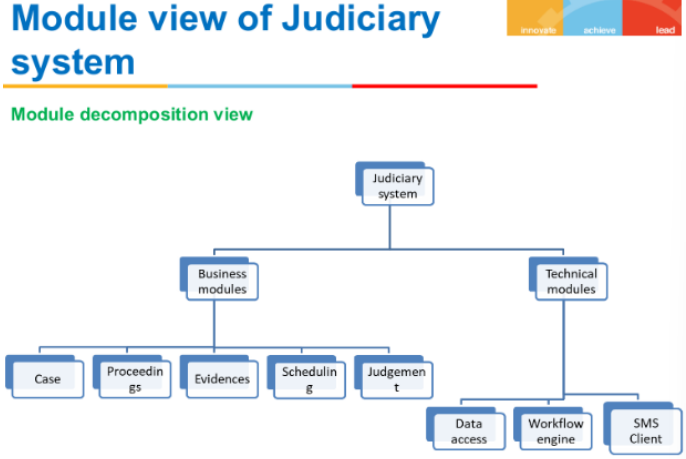


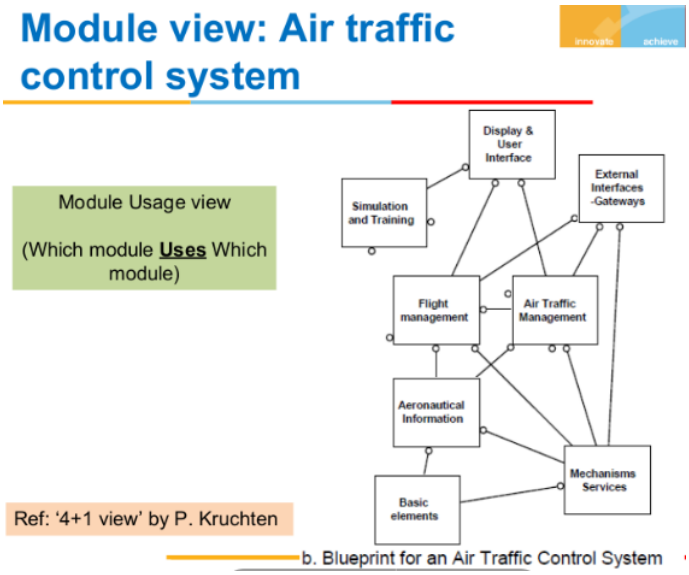
Architecture in Agile Projects:-
Agile method is used when requirements are not very clear and / or technology is new. Therefore detailed architecture & its verification may not be possible. One approach to take is to - Try an approach / technology (experiment), If it does not work, try another approach. When new requirements come in, we may have to refactor the architecture

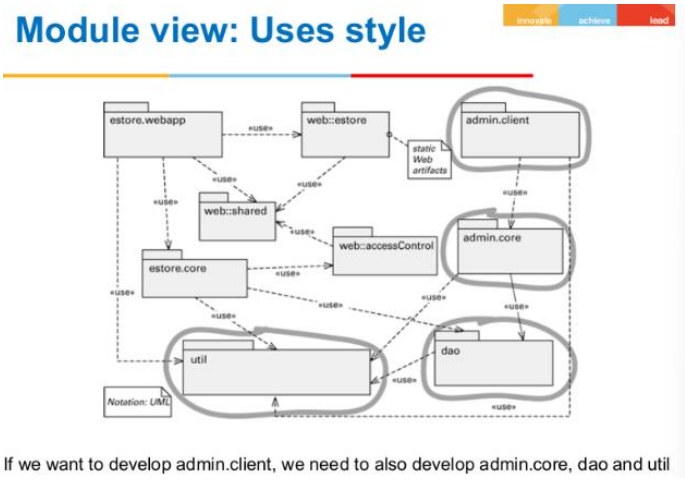
Importance of documenting architecture
Communication Vehicle – Architects, sponsor, business managers, end users, developers, testers
Basis for evaluation- Analyse how well the architecture meets the quality attributes
Different Architectural Views - Architecture of a large system can be complex. We may not be able to understand the system with just one diagram. We may need different types of diagrams to understand different aspects such as structure and behaviour.

3 Types of Architecture Views:-
Module View – Shows structure consisting of different modules and sub-modules.
Component & Connector view – Shows how the different components interact.
Allocation view – Shows where the components reside.





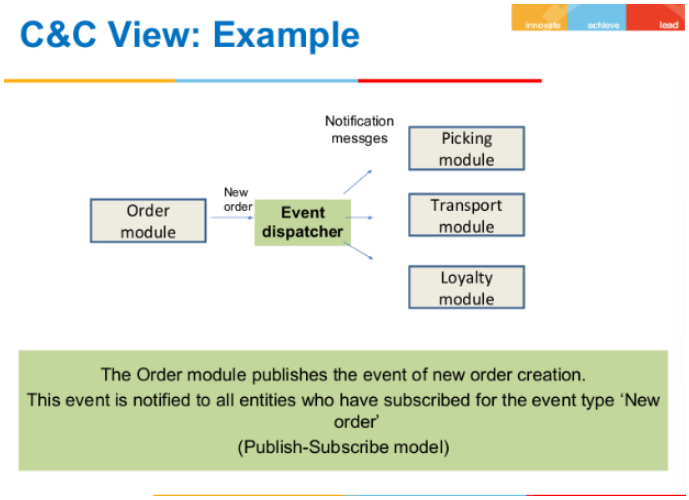




Benefits of Module view ? – It gives an idea about the scope of construction, Plan incremental development and Requirement traceability

Module Documentation Example –
Name :- Inventory Module
Responsibilities – Maintain quantity in stock of different items, Handles inventory tracnsactions and Automatically generate request when qualities goes below reorder level.
Interface provided – Issue inventory (Item code, quantity) Returns Success or Error Code, Receive Inventory (Item Code, quantity) Returns Success or Error Code, Inquire quantity in stock (Item code) Returns quantity.

Module Documentation Example
Name - Proceedings
Responsibilities – Maintain information about proceedings of different cases, Logs change
Interface – ADD PROCEEDINGS (Case code, date , test)
Retrieve Proceedings (case code, date) returns text of proceedings.
C & C Model



Component and Connector View :-
Component examples – Packages, objects, processes, databases, webservers, load balancer, firewall, off-the-shelf packages.
Connector Example – Method invocation, message , REST , HTTPS, FTPs, sFTP, RMI

Types of Connections :-
Data access module – database – JDBC
SMS Client – SMS Gateway – REST
Client – Webserver – HTTP
Order Module – Other module – Message queue
Communication over network – Socket
Secure Communication – Secure Sockets
Process Synchronization – Semaphore

Inter-process communication mechanisms :-
Message queue – Multiple processes can write to and read from message queues (eg – Simple message queue from Amazon, MQ queries from IBM)
Socket Programming – A data stream sent over a netwok (TCP/IP).
Semaphore – Used for process synchronization. Example when we have limited DB connection to be shared among several processes.

C& C view helps us to understand how systems works, analyse reliability , performance, security etc.

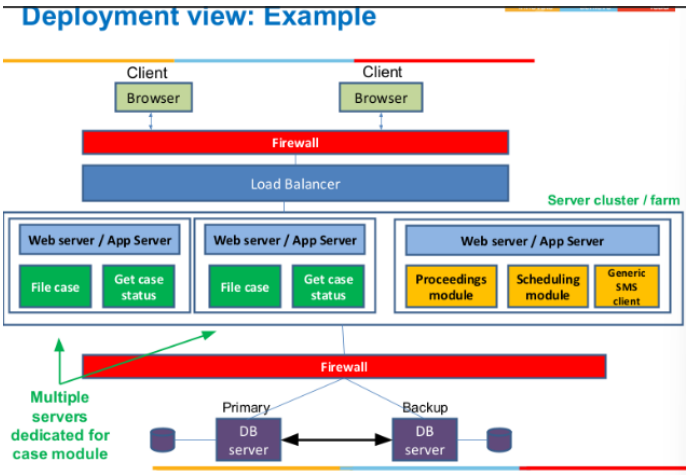
Example of analysis of architecture:-
A system uses SMS gateway. Upon analysis we discover that the SMS gateway
is unreliable and goes down a few times in a day. Then we may want to revise

the architecture to store unsent messages in a queue and then retry after some
time

A system imports critical data from a service provider such as Dow Jones using
FTP. Upon analysis if realize that data is not encrypted, we may consider S-FTP
(Secure FTP protocol)

A real-time system has too many layers. If performance is likely to be impacted,
we may think of combining some layers into one layer

Allocation view shows where the module/components reside.



Allocation view is useful to analyse performance, security, availability etc.
Different views – Module view, C& C view and Deployment view.
Quality Attribute views :-
Apart from structural views, we can have other views to explain the architecture
to achieve specific quality attributes

Security view: Shows different types of security measures such as encryption,
protocols, certificate servers, VPN, firewalls

Communication view: Shows different communication aspects

Type of network such as private and public networks,

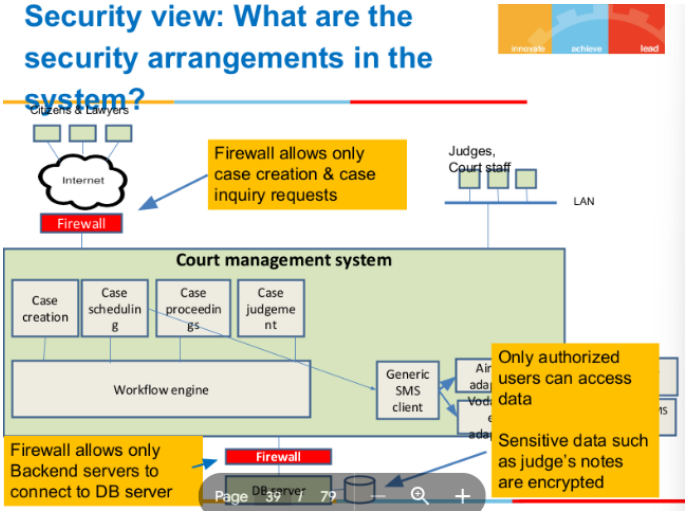
Medium of communication such as LAN, broadband and satellite communication,

Communication protocols such as TCP/IP sockets, HTTP,

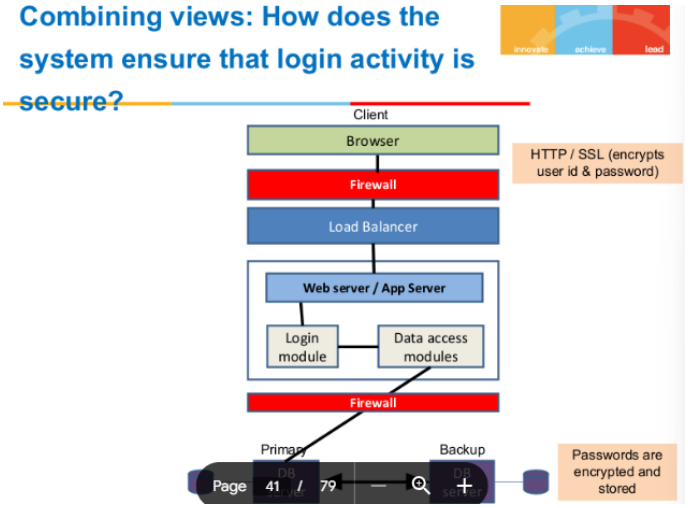
Message formats such as XML, JSON,

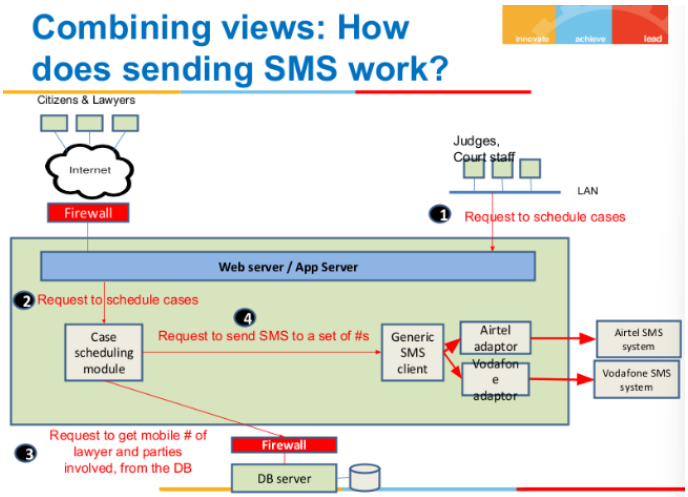
Communication mechanisms such as message queues, file transfers, publish-subscribe

Reliability view: Shows mechanisms used to implement reliability such as replication, switch over

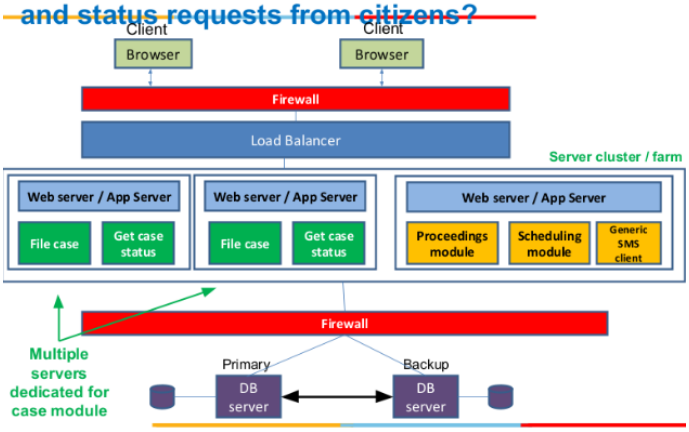


Combining View – Many a times we need to combine views – Module view, C&C view and deployment view.
This is useful to answer stakeholder concerns or queries and explain how the system works to address their needs.





Combining views: How does the system handle the large volume of case filing and status requests from citizens?



Philippe Kruchten's 4+1 view

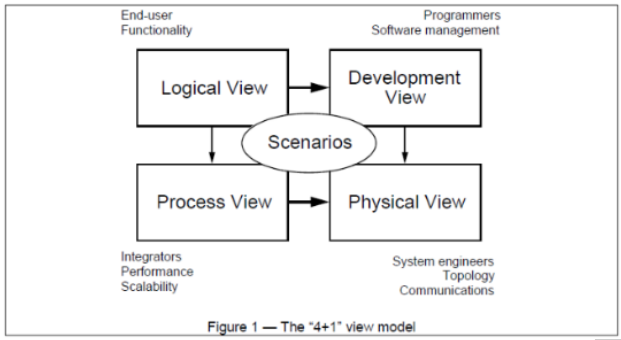
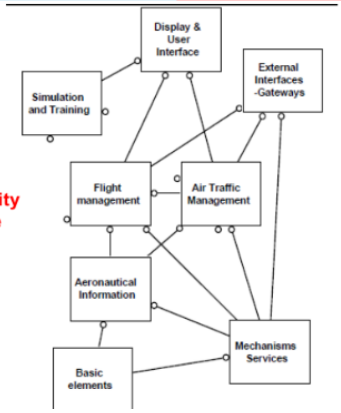


Figure 1 — The "4+1" view model

Logical view

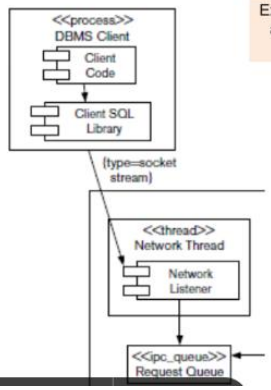
Air Traffic Control system



Shows functionality provided by the system

Object diagrams such as Class diagram, ER diagram, are used as appropriate

Process View



Example of a DB Client accessing Database server

Shows processes, concurrency & synchronization of processes

Deployment View:-

Describes the static organization of the software including Modules, libraries, databases, layers, etc.

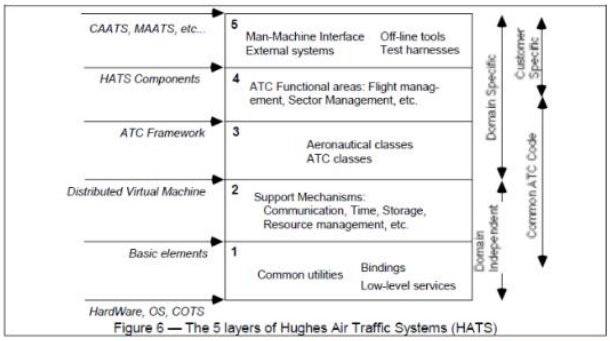
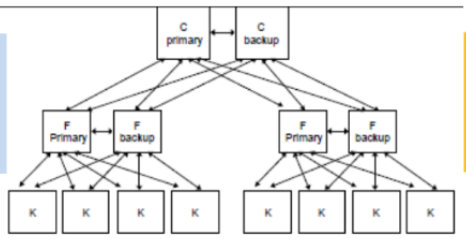


Figure 6 — The 5 layers of Hughes Air Traffic Systems (HATS)

Physical View :-

Describes the mapping(s) of the software onto the hardware and reflects its distributed aspect.

Shows infrastructural elements such as servers, networking, etc.



Example

Redundancy in telephone exchange to ensure reliability

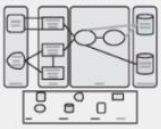
Figure 8 — Physical blueprint for the PABX

C, F and K are three types of computers of different capacity, supporting three different executables.

Ref: Architectural Blueprints—The "4+1" View Model of Software Architecture by *Philippe Kruchten, IEEE Software Nov 1995*

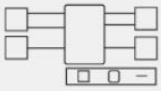
Documentation package

Section 1. Primary Presentation



Section 2. Element Catalog
- Section 2.A. Elements and Their Properties
- Section 2.B. Relations and Their Properties
- Section 2.C. Element Interfaces
- Section 2.D. Element Behavior

Section 3. Context Diagram



Section 4. Variability Guide

Section 5. Rationale

Documentation package: Another example

Architecture Documentation Template
Project Name: XXX

1 Project Context
2 Architecture Requirements
- 2.1 Overview of Key Objectives
- 2.2 Architecture Use Cases
- 2.3 Stakeholder Architectural Requirements
- 2.4 Constraints
- 2.5 Non-functional Requirements
- 2.6 Risks

3 Solution
- 3.1 Relevant Architectural Patterns
- 3.2 Architecture Overview
- 3.3 Structural Views
- 3.4 Behavioral Views
- 3.5 Implementation Issues

4 Architecture Analysis
- 4.1 Scenario analysis
- 4.2 Risks

by Ian Gortan
In the book
"Essential Software Architecture"

Fig. 52. Architecture documentation outline

Typical Layers in Layered Architecture :-

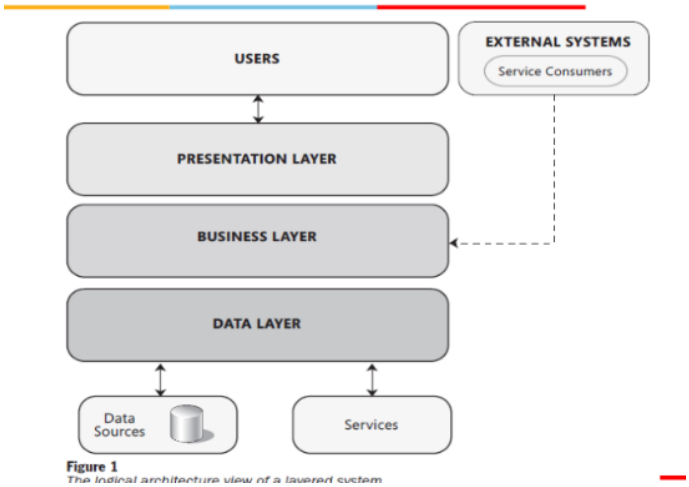


Figure 1
The logical architecture view of a layered system

What are the benefits of layered structure ?
Clearly defined functional layers (Separation of concerns)
Loose Coupling
Reusable lower layer components
Exchangeable parts

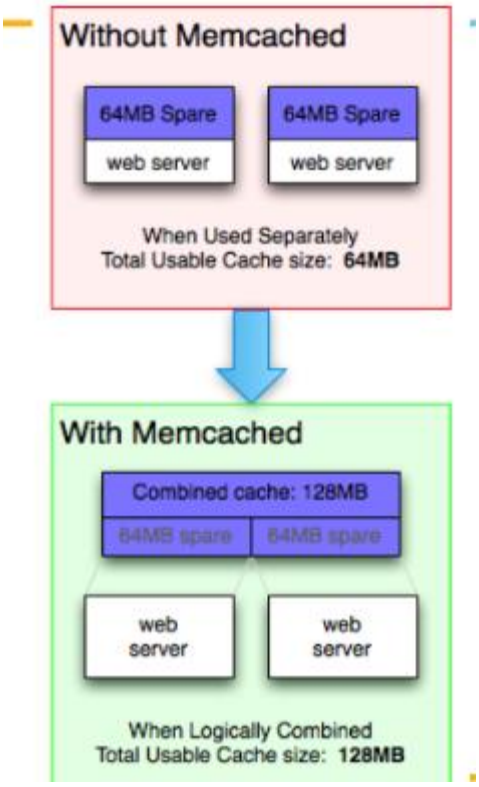
Presentation Layer Techniques:-
Cache frequently used data such as Product catalog (Client side caching vs Server side caching)
Use asynchronous communication between UI & Webserver to
update parts of the web page without loading the whole web page (ex. AJAX)
Different rendering for different form-factors: Responsive design.

Client Side Caching Vs Server Side Caching:-
Client-side caching: Here we store frequently used data in the browser of the client machine – example user preference such as payment method, delivery address, etc.

Server-side caching: Here we store frequently used data needed by
different back-end modules on the server-side – example: Product
codes and description, promotions, etc.

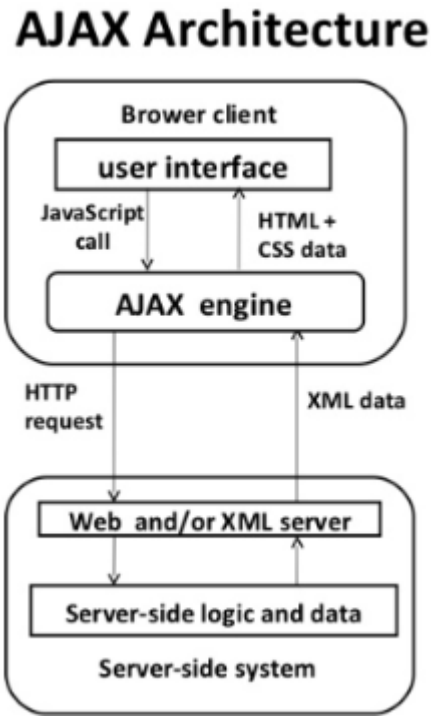
Combination: In practice, a combination of the above 2 techniques is used.

Memcached :-
memcached is a high-performance,
distributed memory object caching system,
generic in nature, but originally intended for
use in speeding up dynamic web
applications by alleviating database load.
You can think of it as a short-term memory
for your applications.memcached allows you to take
memory from parts of your system where you have
more than you need and make it accessible to areas where
you have less than you need.



Asynchronous Communication with Web Server:-
Example use cases: Show product list depending on the product category (eg. Mobile phone or
laptop) selected by user , Validating a user input such as loan amount - whether it is within permissible limits, depending on user profile, Displaying a chat panel
and Reloading Captcha.

AJAX Architecture:- AJAX stands for Asynchronous JavaScript and XML. AJAX is a technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.Some famous web applications that use AJAX: Google Maps (Drag entire map) Google Suggest (Google suggests as you type).



Responsive web design is an approach that renders web pages well on a variety of devices or screen sizes.

Business Layer:-

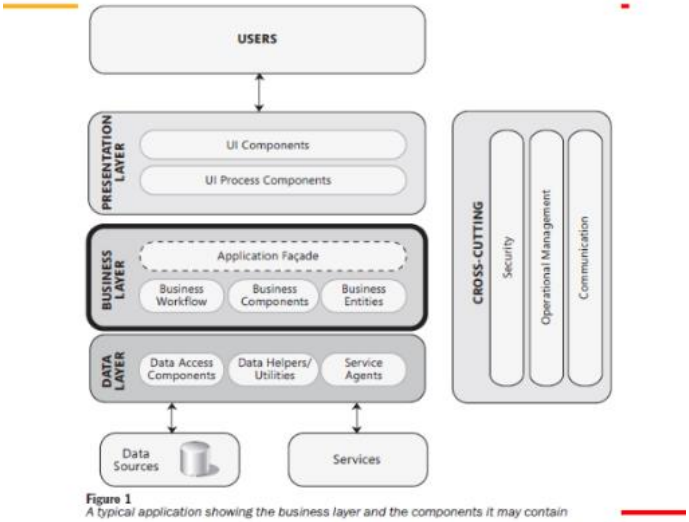
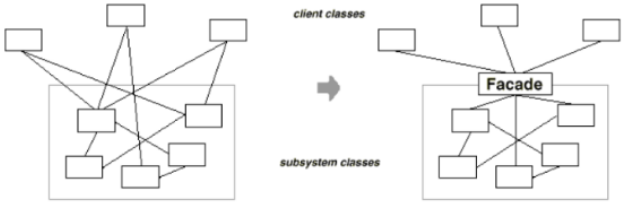


Figure 1
A typical application showing the business layer and the components it may contain

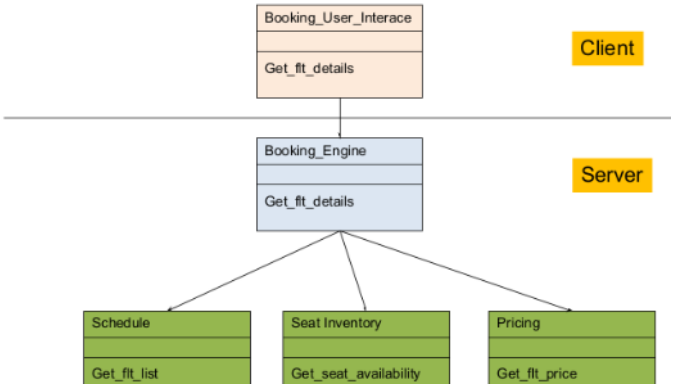
Business Layer Techniques:- Use ap]plication façade to hide internal modules, implement session management and use work flow engine, rules engine etc for modifiability.

Facade: Making sub-system easier to use

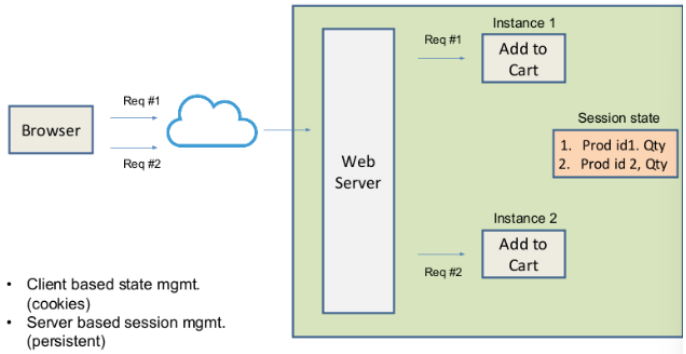


Intent:
•Provide a unified interface to a set of interfaces in a subsystem.
•Facade defines a higher-level interface that makes the subsystem easier to use.
•It typically involves a single wrapper class which contains a set of members required by client.

Example of Application façade: Flight booking



Session management

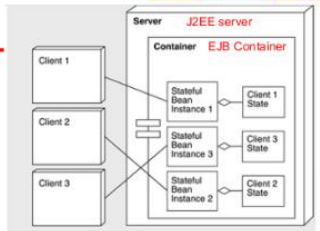


- Client based state mgmt. (cookies)
- Server based session mgmt. (persistent)

EJB: Stateless & Stateful session beans

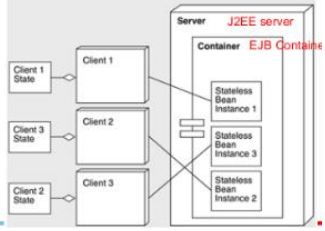
Stateful session Bean: All requests from a Client goes to the same instance of the Bean.

Bean maintains the state of the session



Stateless session Bean: Subsequent request from a client to a Bean may go to another instance of the Bean

Hence Client needs to maintain the state of the session



Work flow engine

Insurance policy processing

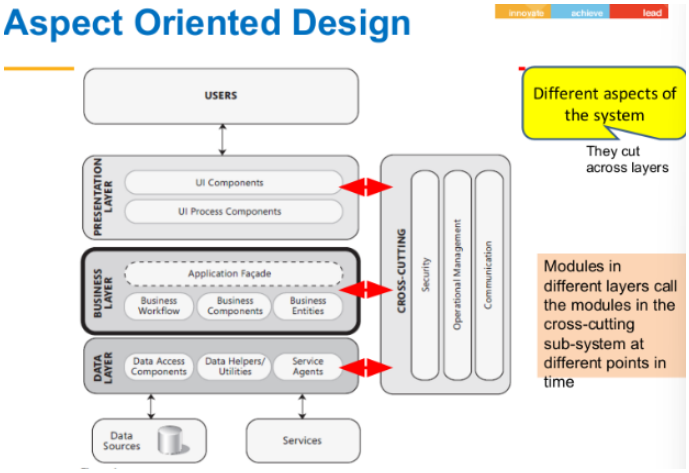
Example Activities

Once an activity is completed, the engine invokes the next step in the process
The next step / activity could be a manual one or an automated one

Popular engines: BizTalk Server, Oracle BPEL processor, IBM WebSphere Process Server

Aspect oriented design for cross cutting concerns – aims to encapsulate cross-cutting concerns into aspects (enhancing modularity). Example of aspects – Logging and instrumentation, auditing – who is doing what, caching and security.

Aspect Oriented Design –



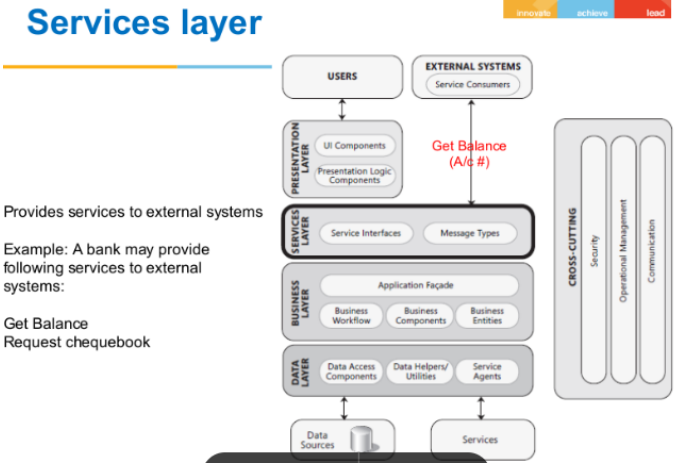
Logging and instrumentation
Log actionable information such as error – database is unavailable, information useful to diagnose errors.
Logging is expensive. Log only the most essential data.
Instrumentation – Helps in understanding the usage and behavior of the system. Request rate, error rate, duration of requests and Queue length.

Data Layer – Prominent technique used
Use DB connection pool, if there are too many users
Use multiple copies of data for faster access
Use transactions to achieve atomicity
Use Object - Relational mapping
Use stored procedures to improve performance
Use parameterized SQL queries to reduce SQL injection attacks

Object-Relational Mapping – Helps in mapping objects in our program to database table. Hibernate is a framework that provides an Object /Relational Mapping (O/RM)

SQL Injection attacks :- Do not concantenate user entered string to SQL string. – Malicious input can result in the following:-
SELECT product-data FROM table WHERE product-name = “Toothpaste” OR “x” = “x”
This will return all product data.

Use parametrized SQL queries:-
SELECT product-data FROM table WHERE product-name=?
? is a parameter which contains the value entered user.



Service Layer – Situations that need to be handled by service layer. Handle same request coming twice, Handle message coming out of sequence, Handle communication failure (using retry mechanism or by queuing work and sending once communication is restored)

What are some of the components of Departure Control System ?- Presentation layer, Business layer, data layer and Service Layer.

Exercsie :-\ What software mechanism will you use to improve the performance in the following situation: One of the frequently used screens in Aadhaar system is citizen registration.This screen has a fields “State” and “District” and “Town”, which are Drop-down fields. It takes time to load this screen due to the large number of states, districts and towns in India.

Solution – Cache information about states, districts and towns in the backend. Use AJAX for dynamic loading of districts and towns.

Exercise - Suppose you are developing a hotel reservation system. You want to provide an interface to external applications such as Makemytrip.com, to inquire about availability of rooms and book rooms. What layer would you build and what will be the component (s) in that layer?

Answer - Services layer is needed to provide interface to external systems. This layer will have components such as Get room availability (from date, to date) return (# rooms available by type of room). Reserve room (# of rooms, type of room, from date, to date) Return (Success /Failure).

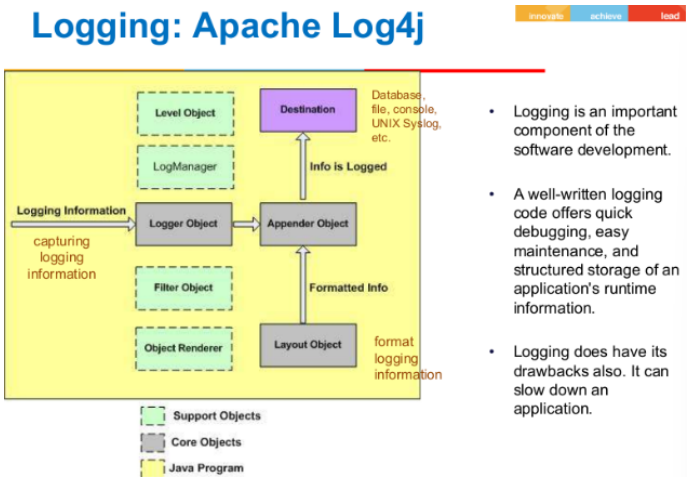
Exercise - Suppose you are building a complex logistics system for a shipping & container Company Users wanting to send goods via containers will login to the system and place their requests. A number of modules need to be called for this: Find out the nearest location of an empty container. Find a the best transporter to pick up the goods Find a ship that is scheduled leave to the desired destination and has spare capacity to load the container

As an architect you want to hide these modules from the client layer. What technique will you use and in which layer?

Answer - In the Business layer, provide a unified API – façade – to place a shipping request.The façade will in turn make use of modules to search for appropriate container, appropriate transporter, appropriate ship, etc.

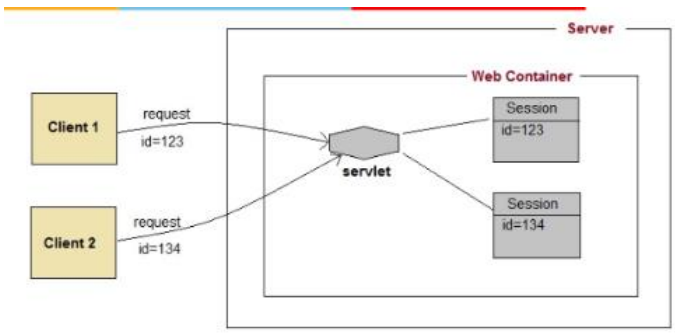
SSL - Browser connects to a web server (website) secured with SSL (https). Browser requests that the server identify itself. Server sends a copy of its SSL Certificate, including the server's public key. Browser checks the certificate root against a list of trusted CAs and that the certificate is unexpired, unrevoked, and that its common name is valid for the website that it is connecting to. If the browser trusts the certificate, it creates, encrypts, and sends back a symmetric session key using the server's public key. Server decrypts the symmetric session key using its private key and sends back an acknowledgement encrypted with the session key to start the encrypted session. Server and Browser now encrypt all transmitted data with the session key.

Logging: Apache Log4j

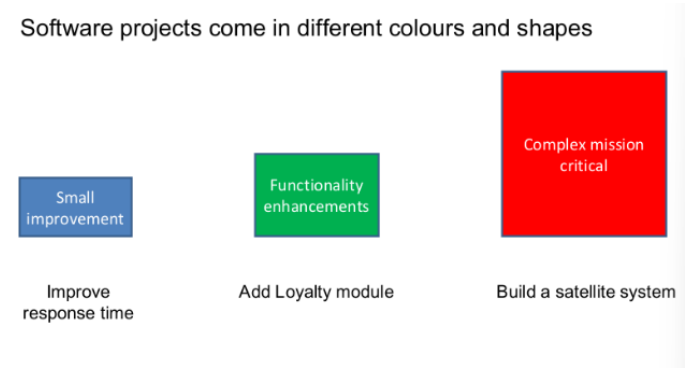


IPSec – Internet Protocol Security – IPSec is an Internet Engineering Task Force (IETF) standards suite of protocols that provides data authentication, integrity and confidentiality as data is transferred between communication points across IP networks.
IPSec provides data security at the IP packet level.

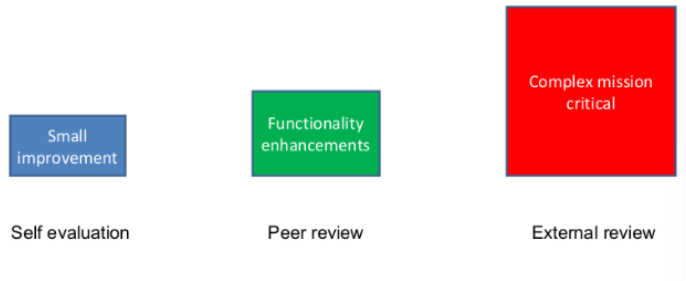
Session Management:-



A web session is a sequence of network HTTP request and response transactions associated to the same user. Modern and complex web applications require the retaining of information or status about each user for the duration of multiple requests.



Review techniques differ



Example of self evaluations checklists –
Availability – Do we have a mechanism to detect failure?, Do we have a mechanism to switch to a backup component?
Do we have a mechanism to save state periodically?

Performance – What is the approach to determine the number of servers needed ? ,What is the mechanism to add & remove resources dynamically? If there is a common resource that is needed by multiple clients, what is the mechanism to reduce the bottleneck ?

How to evaluate the availability of a system that has 2 servers – one primary and one hot standsby ?
In case of single server system – Availability = 99% = Probability of failure = 1% = 0.01

In case of a 2 server system –
Probability that both servers fail at the same time = 0.01 * 0.01 = 0.0001
Availability of 2-server system = 1 -0.0001 = 0.9999 = 99.99%

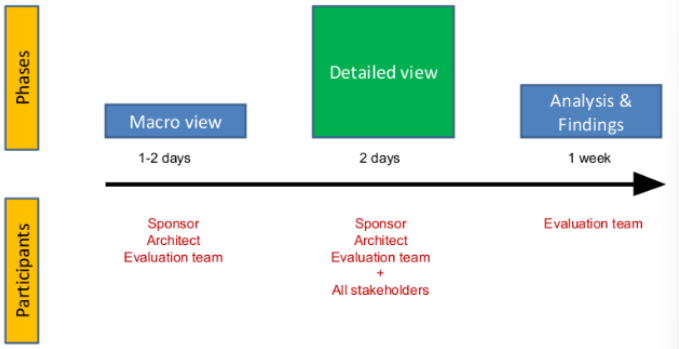
Examples of Self – evaluation Models :- Let us request passes through 3 modules in order to get a response. What is the latency experienced by the client?
Latency experienced by the client = 1+2+3 = 6 sec

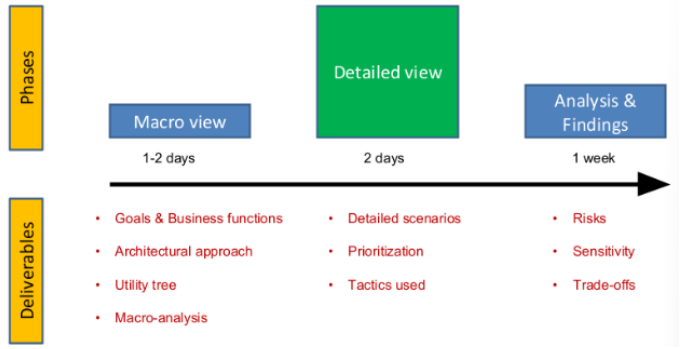
Architecture Trade – off Analysis Method (ATAM) :- ATAM is a method used to evaluate architecture of large systems. It assumes that reviewers are not familiar with the business goals and the architecture of the systesm.

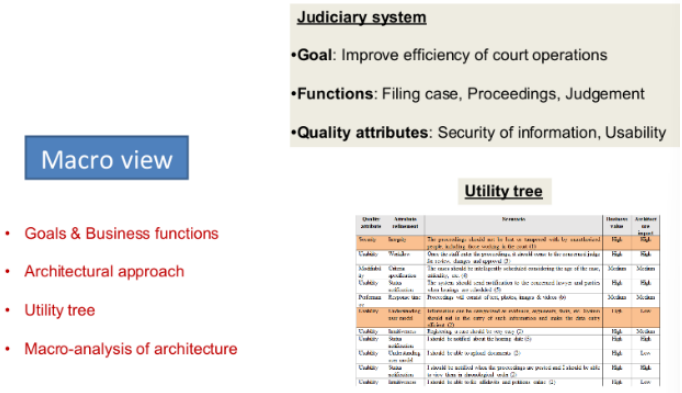
It is suitable for many domains such as :- Finance, Defence, Automotive.

Reviewers – External, from different organization . they will give unbiased opinion and independent perspective.

ATAM Phases –

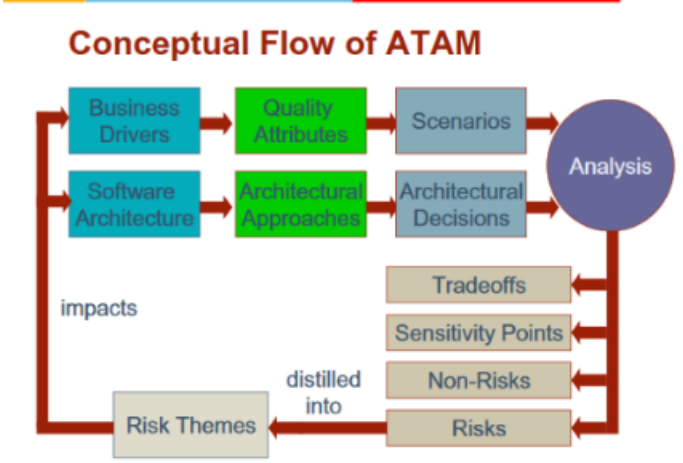






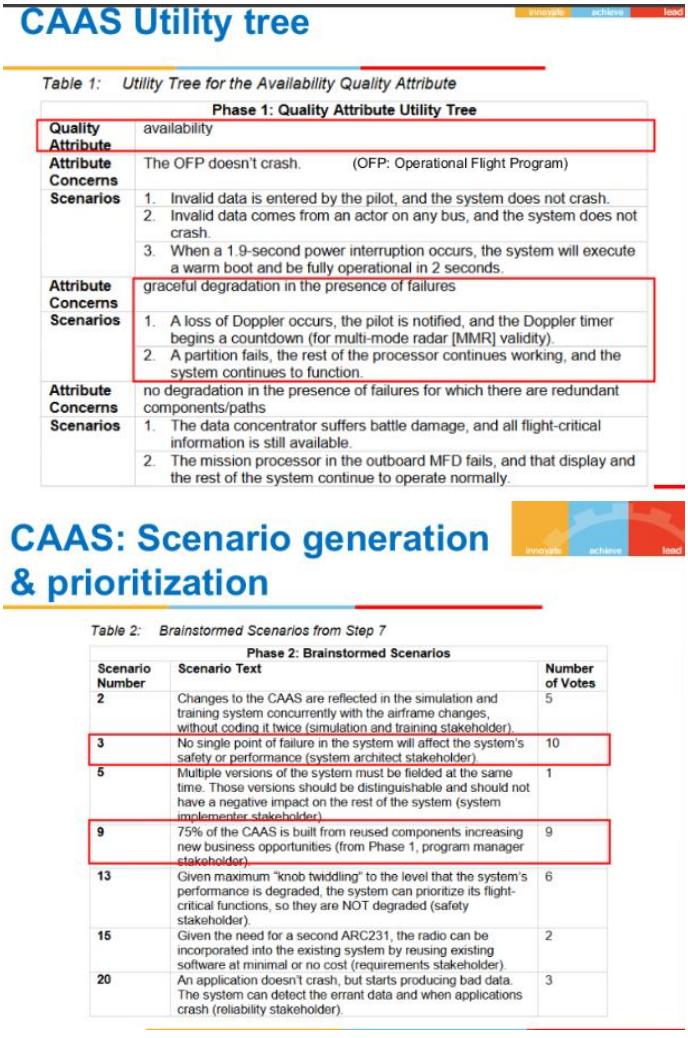
ATAM activities :-Voting
Detailed Views – detailed scenarios from all stakeholders, Priortization of secnarios and Tactics used to address key scenarios

Analysis and Findings – Risks, Sensitivity and trade-offs – Analyse the architecture to determine the sufficiency of design to meet the requirements
Risks: Ex. Design is not secure enough. Hackers can tamper with evidence data.
Sensitivity: Ex. The workflow tool uses proprietary language. Migrating to a new tool may be expensive
Trade-offs: Ex. Multiple levels of security (user pwd, txn pwd, OTP) may impact usability.



ATAM –
Result consist of :-
Arch approaches , Arch decision, Risks, Non risks, Sensitivity points & Trade-offs, Risk Themes

Other Outputs of ATM – Business goals, Presentation of architecture, utility tree and mapping of architecture decisions to quality requirements (scenarios)



CAAS: Sample observations from analysis

Risk	There are no built-in hooks to connect to simulators. So the software can not drive both the simulators and the actual helicopters
Sensitivity	Isolating operating system dependencies will enhance portability.
Trade-off	Letting pilots set parameters such as turbine gas temperature limits, increases flexibility but decreases safety
Themes	Several scenarios dealt with performance. However performance requirements were not spelled out clearly

Additional Risks- External evaluation can also reveal additional risks not previously imagined.

Architecture patterns are frequently used architecture solutions for commonly occurring situations.
List of patterns:- Layered pattern, Service Oriented Architecture, Model-View- controller, Pipe & filter pattern, Publish-Subscribe, Shared Data, Client-Server, Multi-tier pattern, Map-reduce, Peer-to-Peer and broker

Layered Pattern:-

Layered pattern: Example: Architecture of an Information system

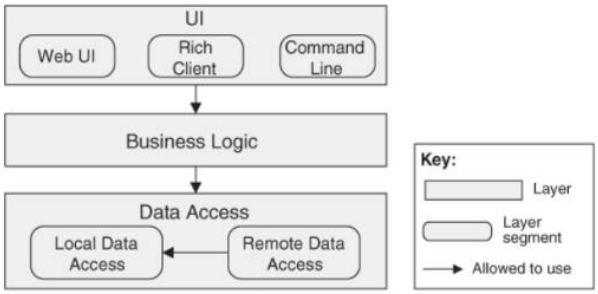
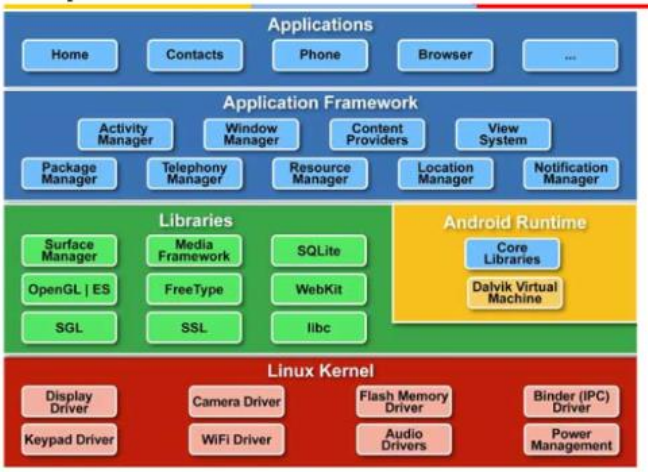


Figure 13.5. Layered design with segmented layers

Layered pattern Example: Architecture of Android



What are the benefits of layered pattern ?

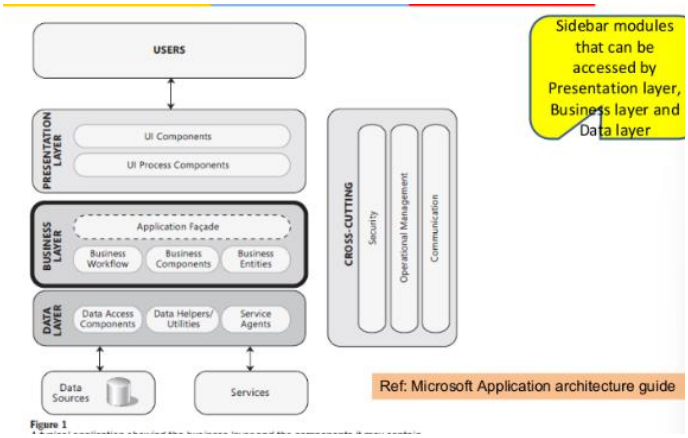
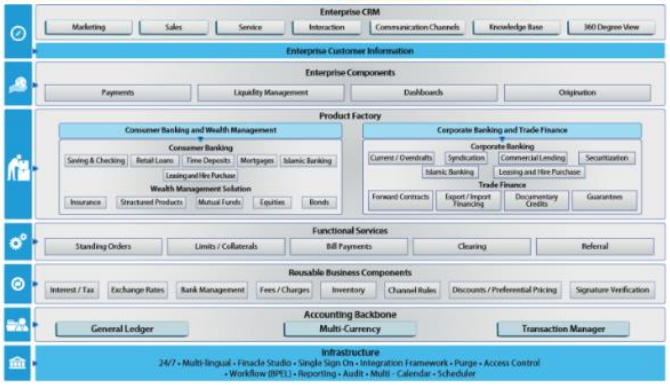


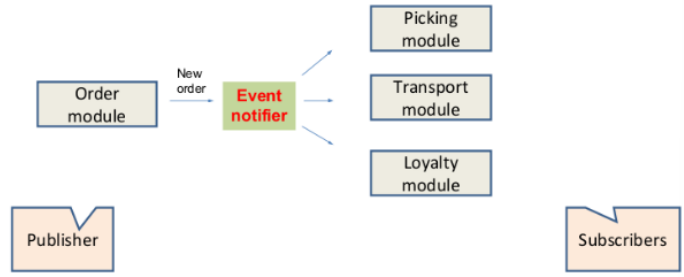
Figure 1

Layered architecture: Finacle Banking solution

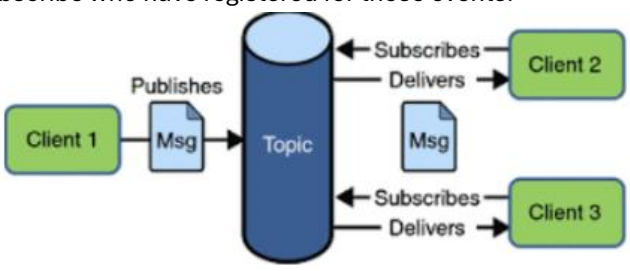


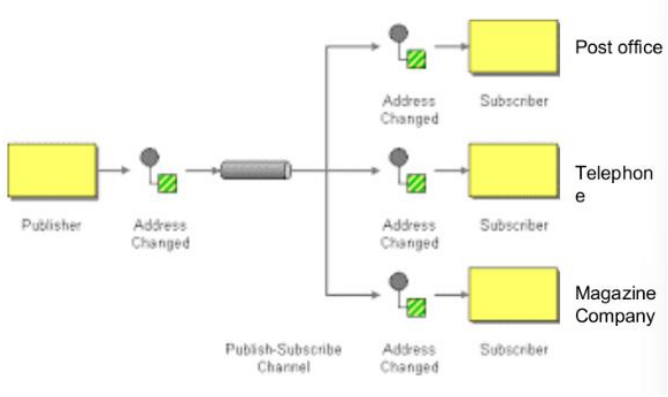
Layered Pattern – Each layer contains modules which together provides a cohesive set of services. Each layer provides an interface to layers above. This enables changes to implementation of a layer without affecting the upper layers. Have too many layers may lead to complecity and poorer performance.

Publish – Subscriber :- Example:- Order creation module event is subscribed to by picking module, transport module, loyalty module.

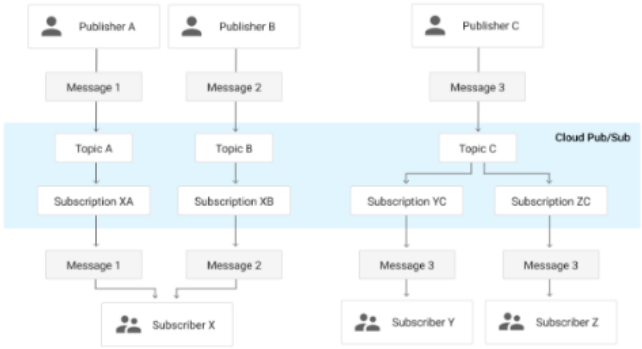


Publish – Subscribe :- Components interested is knowing about events , subscribe to relevant events. Publisher place events on an Event Bus. Event nbug delivers events to subscribe who have registered for those events.





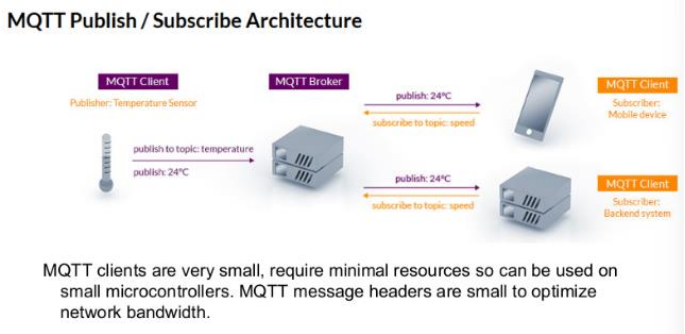
A process may subscribe to more than one topic/cahnnel.
A topic/Channel can be subscribed to by more than 1 process

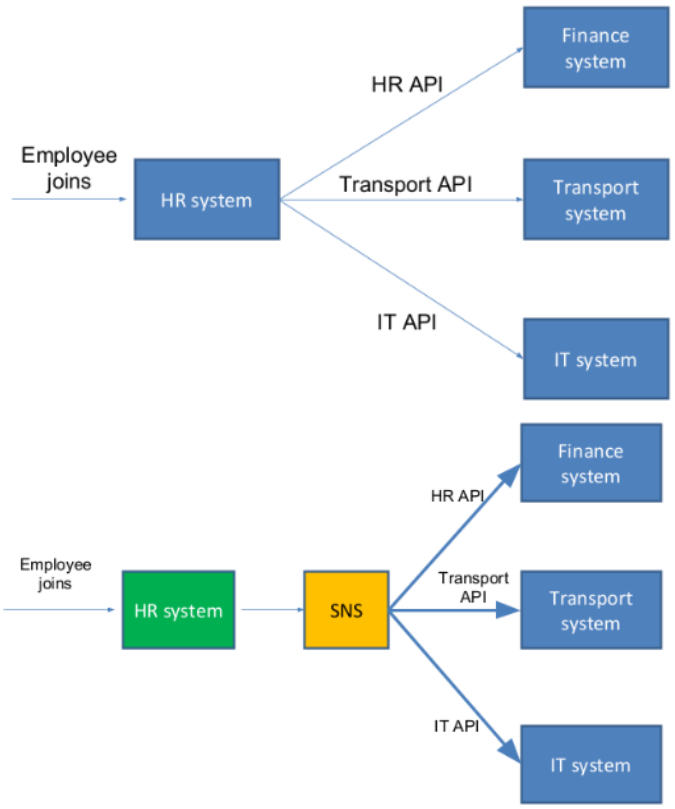


Publish – Subscribe Implementation:-
3 forms :-
List based – All subscribers in the list, will be notified.
Content based – Topic based – eg – sports, business, Politics, one can subscribe for a topic.

Publish-Subscribe Tools :- Amazon Simple Notifications Service (SNS)
Azure Service Bus
MQTT (MQ Telemetry transport) for IoT
Apache Kafka , Apache Pulsar

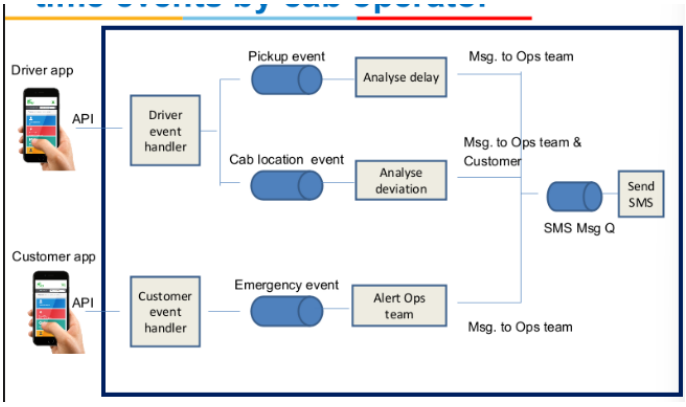
Amazon SNS : Notification end points :- SNS supports a variety of subscription types , allowing you to push messages directly to Amazon SQS queues, AWS Lambda functions, and HTTP endpoints.

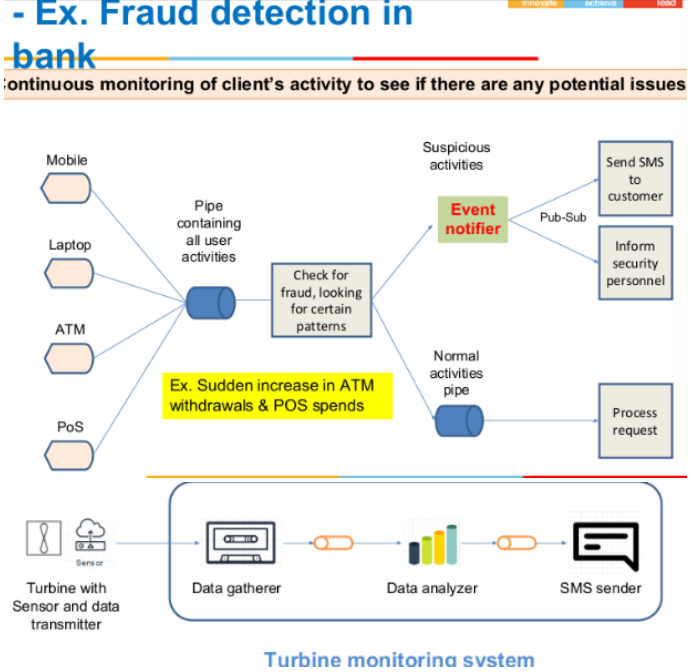




Benefits of Publish-Subscribe
Loose coupling between producer and consumer
New Listeners can be added without impacting producer.
Eliminating polling

Pipes and Filters:- This architecture is useful when we need to perform a number of transformations in a sequence.
An App based cab operator (like Ola / Uber) wants to monitor real time events and take appropriate action as follows:
If the cab deviates from the designated route and if the deviation is significant, then we want to alert the passenger as well as the ops team (inform via SMS). If the customer presses the emergency button, we want to inform the police and the ops team immediately (inform via SMS)
Using Pipe & filter pattern, draw an architecture diagram to implement the above requirements. Indicate how the pipes and filters are used.





Turbine monitoring system

When to use pipes ?-

When you pass on information to a module but do not want to wait for the module to complete its job. (Asynchronous communication). When the module being called is expected to take a long time, you can send a message and do other things in the meantime. When the module has completed the job, it can reply back via pipe (duplex communication). When there is an increase in transaction volume, we can queue up the transactions to be processed. This sometimes avoids crashing of system. It may be more efficient to insert 100 records, than one at a time. Queues can be used to create a batch of inserts.

Pipes and Filters Applications:-

Applications where Pipes & Filters are useful

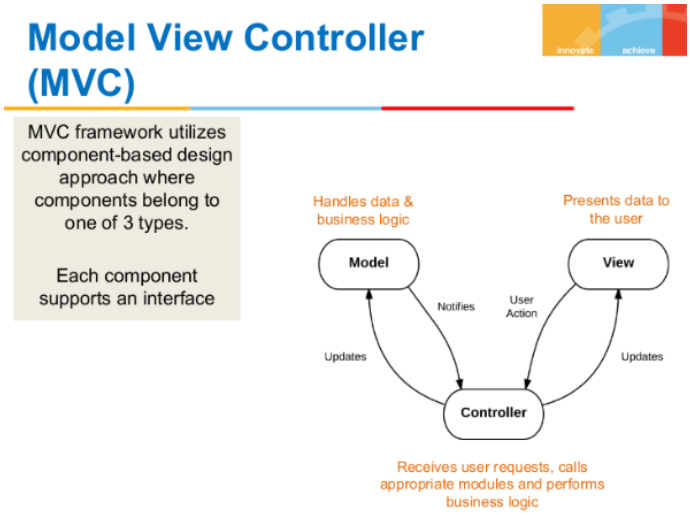
Satellite signal processing – noise reduction, transformations

Extract – Transform – Load (ETL) is a good fit for Pipes & filters architecture

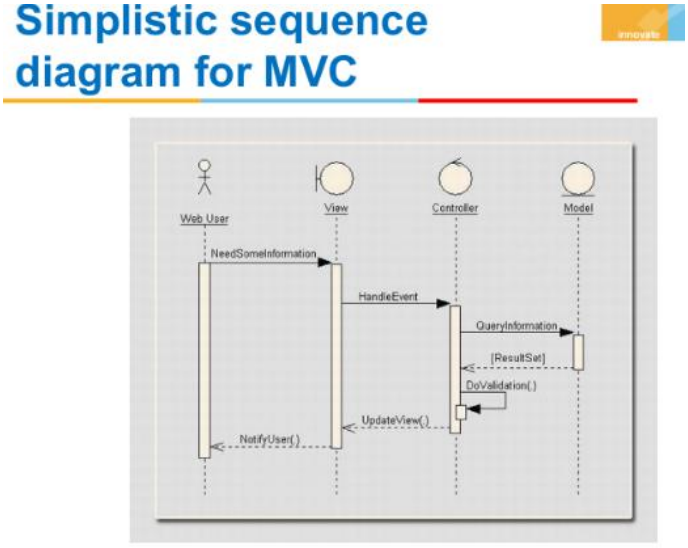
Real time recommendations to customer in ecommerce

Pipes and filters : Tools

Apache Kafka, IBM MQ Series , Amazon SQS

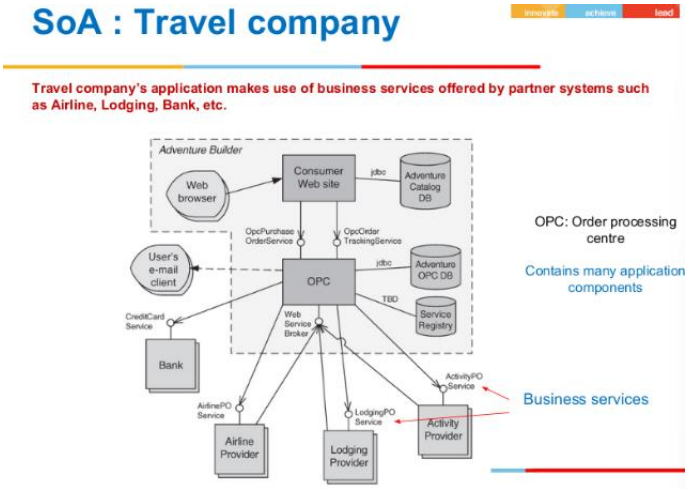


Model View Controller is useful when we need different views of data. Eg – Bar chart and Pie chart of data.

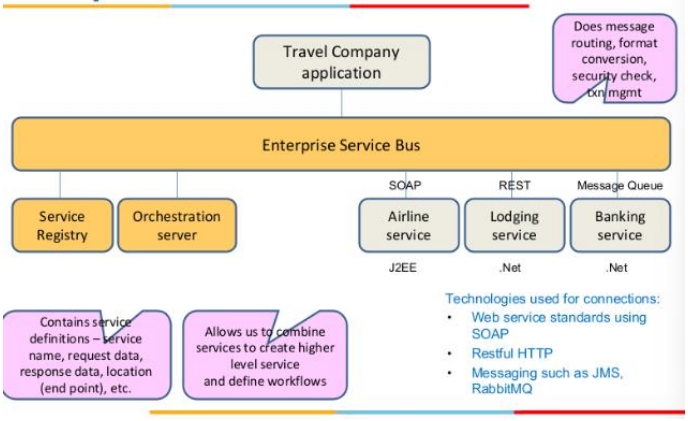


Service Oriented Architecture – Started as a means to integrate disparate applications of vendors and partners.

SoA consists of :- Business services deployed in a network. Application components that make us these services. Infrastructure to define services and communicate with services.



SoA Travel Company – SoA Components:-



SOA Features :-

Different services are provided by different service providers. Services run on different platforms developed using different technologies – J2EE, .Net , etc.

Connection types can be varied :- Web services using SOAP, REST ,Messaging. The SoA concepts have now been used inside enterprise. Different applications within an enterprise can be integrated using SOA. For example – a

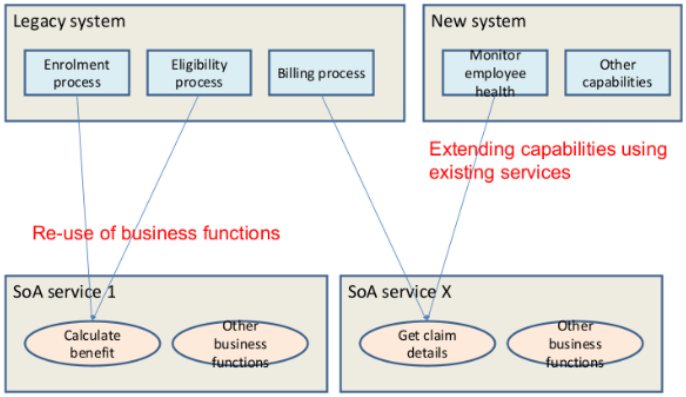
manufacturing application can be integrated with HR application, marketing application and finance application using SOA.

Case Study :-

1. How did CIGNA go about identifying common business services?

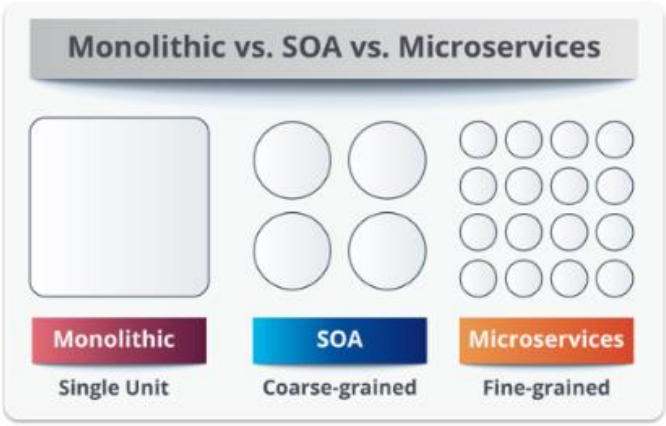
Cigna went through the different business processes such as Enrolment, Eligibility & Billing to identify the granular level common business functions such as 'Calculate benefit for an individual' which are used by multiple processes. Then they grouped related business functions into business services.

Cigna SoA Architecture:-



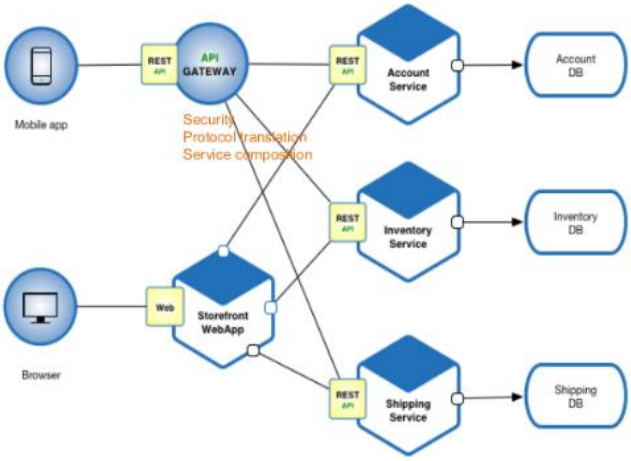
3. What benefit did CIGNA get from this architectural approach?

Increased reuse: Legacy Business processes call newly built reusable
business services instead of the existing Legacy functions. This reduces maintenance effort. (Earlier these functions existed in multiple business processes).Agility in creating new functionality: Example: Predict health issues,
by analysing the past history and patient related data. This makes use of the newly built services. Such extensions would have been difficult if they did not have common business functions, identified and packaged as services.



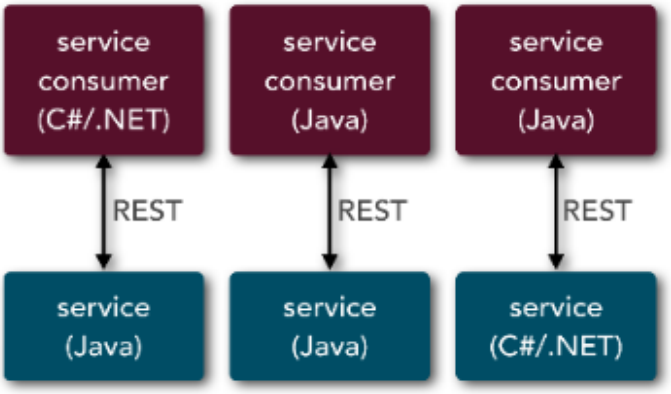
Key features: More granular, Minimal sharing, no ESB

Micro-Services architecture : Retail Application:-

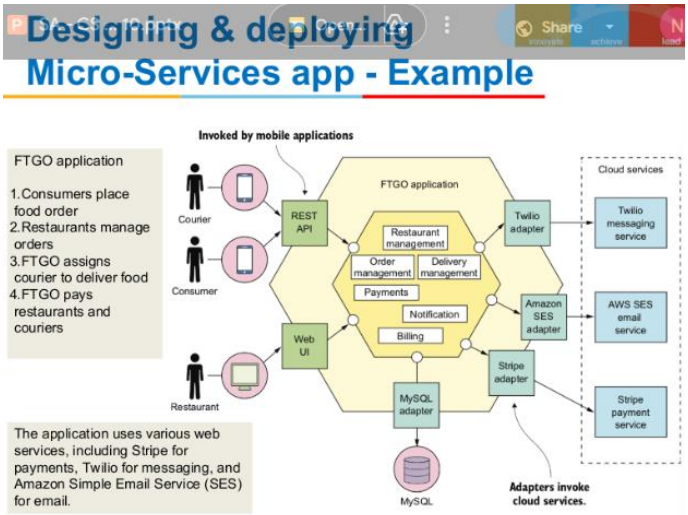


API Gateway:- The granularity of APIS provided by microservices is often different than what a service client needs. API gateway implements facades and they provide additional services like proxying, protocol translation, security.

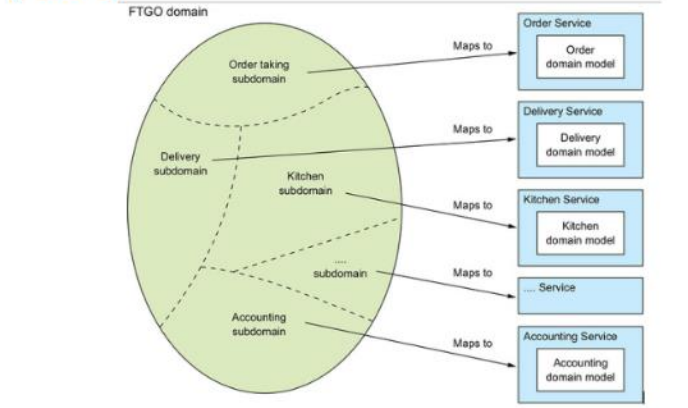
Micro-Services (Communication method – most commonly REST)



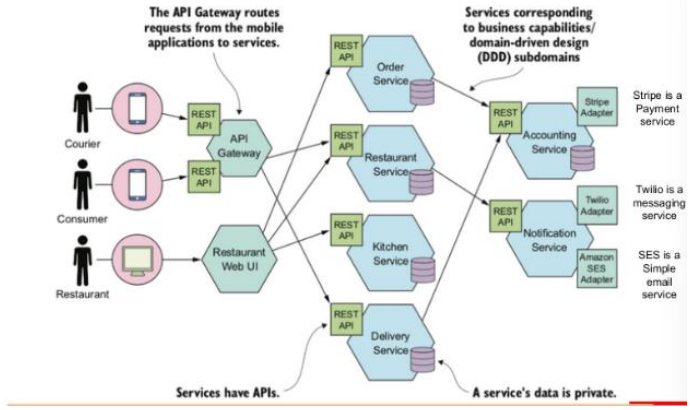
Other communication methods – Messages. gRPC



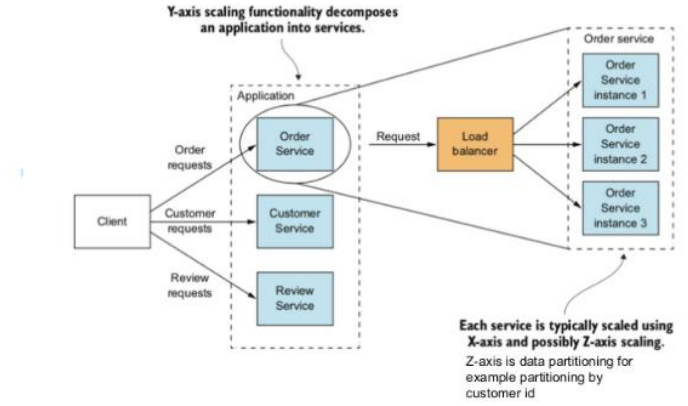
Identifying services using Domain driven design (DDD)



Micro-Services architecture example



Scaling Micro-Services application



Case Study – Danske Bank – Case Study.

What are the micro-services in Danske bank FX application? – Linecheck, Trading, Data sync, logging, monitoring, etc

How do they communicate with each other?

Internally, the microservices integrate only via messaging on RabbitMQ

What is the benefit of such as communication?

Results in very low coupling and no interfaces to violate

What tools are used for logging & monitoring?

LoggingService, ElasticSearch, and Kibana

MonitoringService, Icinga, and cAdvisor,

What new issues had to be handled when micro-services were introduced?

Aspects like fault-tolerance mechanisms, concurrency handling, and monitoring

SOA	Micro-Services
• Course grained services	• Fine grained services
• Feature rich – Lots of meta data about services, message conversion, protocol conversion, txn. mgmt. across services., etc.	• Simple – usually uses a one protocol among several available ones, no txn mgmt across services
• Complicated	• Easy

Key benefits and challenges of Micro-services architecture

Key benefits – Fster delivery, improved scalability and greater autonomy.

Key challenges – Resource monitoring and management , Failure , recovery and self-repair.

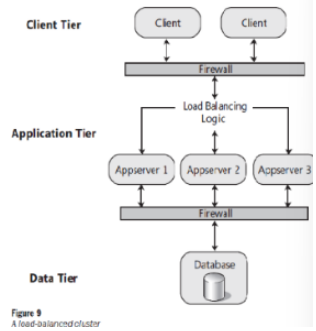
Broker – is an intermediary software that provides certain services. Eg – Load Balancer, EJB Server.

Load balancer
•Distributes requests to different servers in a server farm
•Takes care of non-availability issues by switching to another server

Some algorithms used to distribute load are:

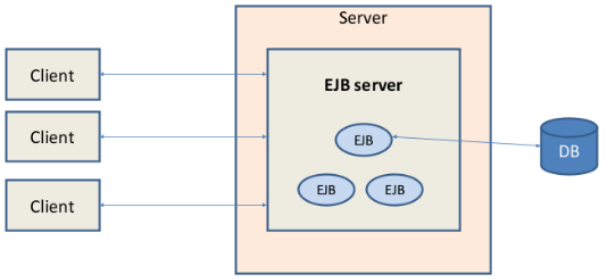
•Round robin
•Least connections

Load balancer vendors
•F5
•Citrix



Brokern Patter - EJB Server / Application server

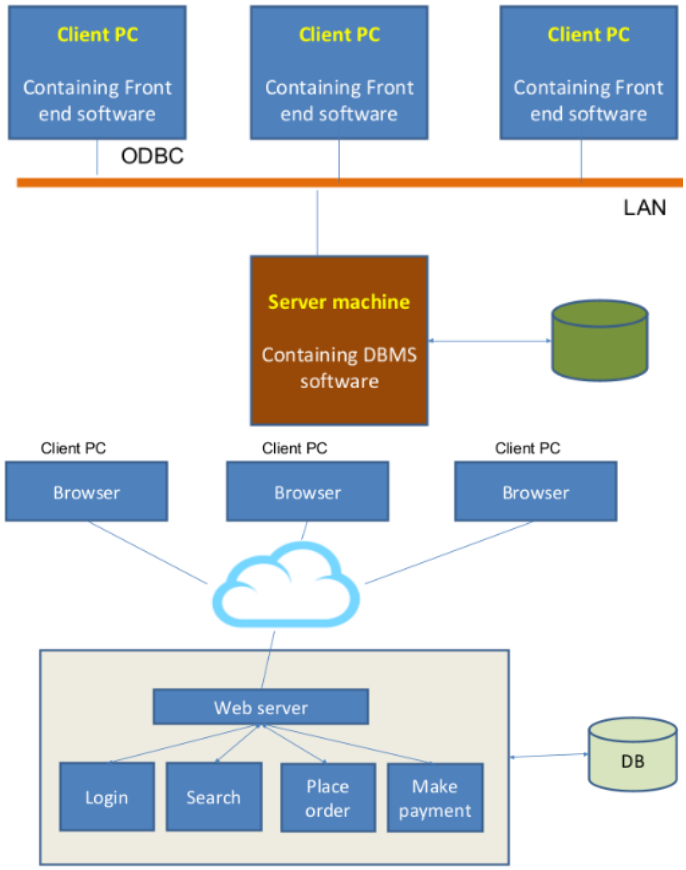
- EJB server provides services pertaining to transaction management, authenticating clients, persistence, session management, etc.
- It intercepts requests to EJB to perform these tasks



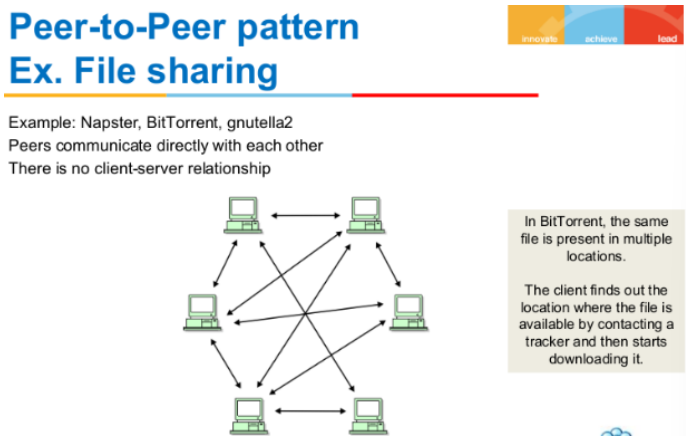
Broker pattern meditate between client and server to provide different kind of services.It adds a layer of indirection that may cause performance issues. A broker can be a single point of failure and hence needs to be designed for fault tolerance. It can also be a target for security attacks.

Client-Server Pattern - Useful when several clients want to access the services provided by the server.Examples Database server (accessed using ODBC),Mail server (MS Outlook),Web server (accessed using HTTP request)

Client Server Pattern:-



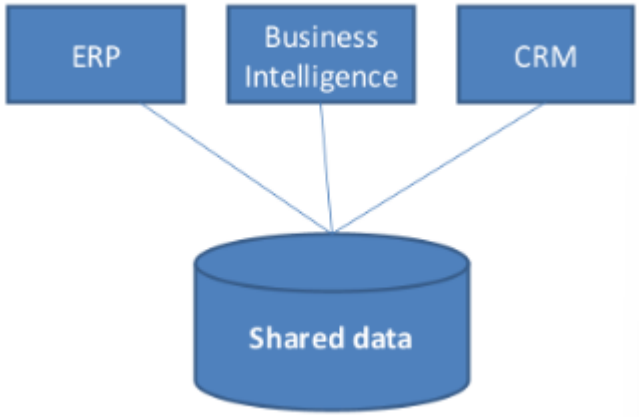
Client – Server Pattern:-
Variants of client – server:- Earlier client server communication was synchronous
Now we have - Asynchronous communication (using AJAX), Clients providing call back functions (ex. Javascript). Eg. Updating Live cricket scores on a Cricket website or updating Live stock prices in a stock exchange
Demerits- Server can be a single point of failure, Server can be a bottleneck.



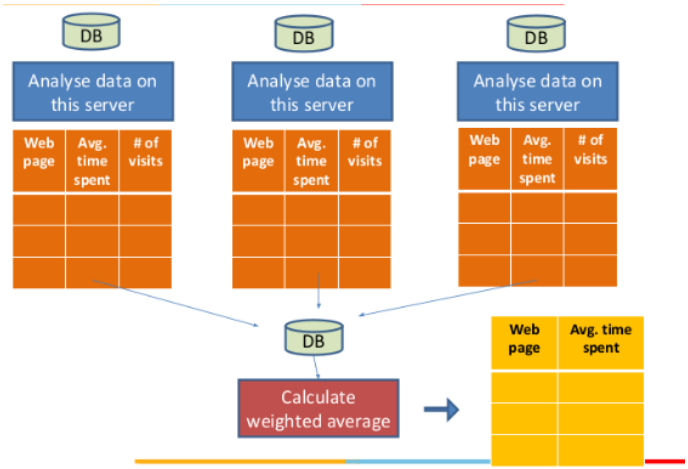
Peer-to-Peer Pattern:- There is no client-server relationship. At one time one peer may be a client, at another time, it may act like a server.
Challenges –
Data consistency – copies of data in different nodes be out of sync, Data and service availability – no central authority to ensure this. Backup and recovery – May be more involved, since there is no single source from where to recover.

Wireless Adhoc Networks - Army uses Wireless Ad-hoc networks to share information between different entities in the battle field. The entities are on the move and there is no fixed network consisting of links and routers. Every node gathers information about the enemy and shares it with other nodes,

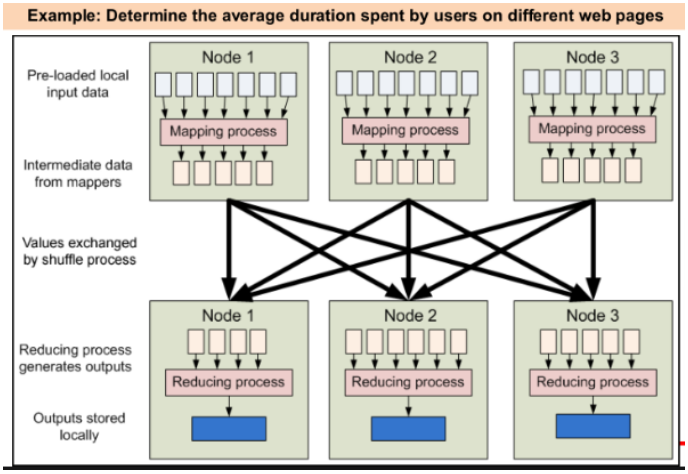
Shared Data Pattern – Database is the central piece. Different components share the database. The database supports persistence, concurrent access, fault tolerance, access control, distribution, and caching.



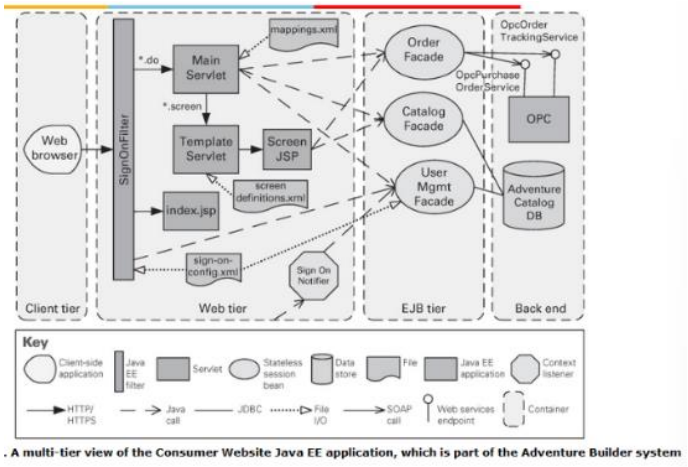
Case Study – SaleForce.com :-
What are the performance improvement techniques used ?- Load balancer, caching, separate storage for BLOBs.
What is the architecture pattern used here ? – Shared data, client server.
What kind of customization can be performed ?- Workflow, custom fields, security settings, service APIs, UI look and feel.
Map Reduce Pattern:-
Used to analyse vast amounts of data. Let us say we want to find out the top 3 web pages visited by users of a website every month.Top web page is a page where users spend maximum time.We will have to capture data about different pages visited by each user and record the duration for which they stayed on that page.Then we need to determine the average time spent by users on every page during the 30 days of the month.How can we speed up this analysis?



Map-Reduce Pattern:- Data is distributed across a cluster servers. Processing is performed in parallel. This leads to low latency and high availability. Map is an algorithm that is executed on each server resulting in <key, value>instances. Reduce is an algorithm that takes uses output of Map process and performs further processing to produce another list of <key , value >pairs.



Multi-tier pattern:-
Tier is a physical unit, where the code / process runs. E.g.: client, application server, database server;Layer is a logical unit, how to organize the code. ... Layers are the logical groupings of the software components that make up the application or service.All layers may reside on one server or can be distributed across multiple servers. Also all software components in a layer may run on server or can be distributed across multiple servers.Tier makes it easier to ensure security & optimize performance.



. A multi-tier view of the Consumer Website Java EE application, which is part of the Adventure Builder system

Relationship between Tactics and Patterns:- Patterns are a higher level organization of modules. Tactics are specific techniques to address different quality attributes. For example, we may use a Layered pattern in an application. Within the business layer we may use parallel threads to improve performance. So we can say that Patterns are like molecules and Tactics are like atoms within a molecule. Analogy: Macro – micro
Case Study – TripAdvisor
What was the basis for defining / designing services

Each functional area is packaged as a Service - media, members, reviews, travel
lists, etc.

Are the services 'Course' grained or 'Fine' grained. Justify.

Course grained.

Each service is high level, business oriented API

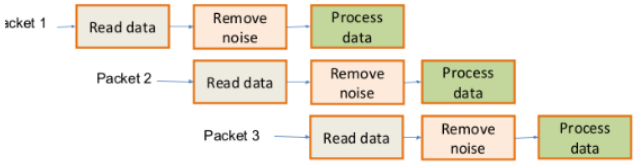
What techniques are used to improve performance?

Content Delivery Network (CDN)

Memcached for caching

Sharding of the largest table (1 Billion rows) into 12 partitions, to address
Input-Output bottleneck issue

How does Pipe & Filter pattern improve performance?



Case study: 3D-P Publish Subscribe

Questions & Answers:

1. Who is the publisher and who are the subscribers of GPS correction data?
   - Publisher: GPS correction data provider
   - Subscribers: Earth Moving Equipment (EME)

2. How was the network traffic reduced?
   - Unicast instead of broadcast. That is sending data only to relevant EME and not to all EMEs. Thus reducing data transmission
   - Maintain list of active EME. Send GPS correction data only if the EME is active. Thus further reducing data transmission

architecture patterns

For the following situations, identify the arch pattern that can be used:

1. Let us say, MakeMyTrip.com provides APIs for
   a. Enquiring about availability of tickets on flights to a destination from an origin city
   b. Booking ticket on a certain flight
   c. Cancelling ticket

   SoA

2. The city traffic control center shows a map of the city roads along with information about the level of congestion on different roads: Red for heavy congestion, Orange for medium and Green for no congestion. This is enabled by sensors sensing the traffic density on important roads. The sensor data is stored in a database which is then analysed to display the level of congestion on the road map
   Pipe & Filter, MVC, Shared data

3. A cooking gas company wants to implement a system to provide different services to consumers – services such as booking of gas cylinder, requesting for a technician visit, ordering a tube, etc.. Users should be able to access this system via browser, smart phone or SMS. Backend database is Oracle.
   Layered, Micro-services

Mention the architecture patterns used in the architecture described below of an ERP system:

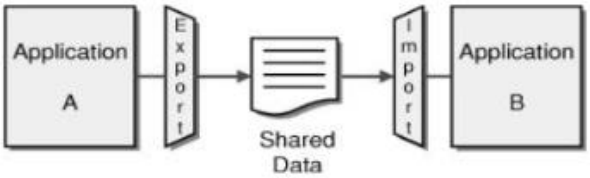
a. Sending 'Request for Quote' to Vendors who have registered for supplying the item
   - Publish – Subscribe

b. Instruction to bank at end of month to pay salary to employees by calling a service supported by the banking IT system
   - SoA

c. Automatic tracking of inventory received & dispatched from warehouses using RFID
   - Pipe & Filter

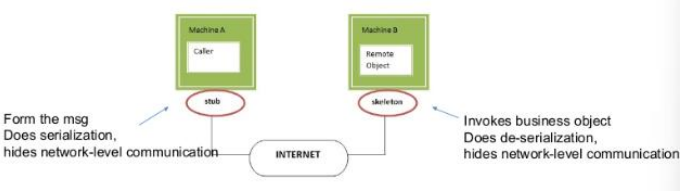
d. Send transport request to logistics partners
   - SoA, Pub-Sub

File Transfer:- Good for low frequency data synchronization between applications. It is simple, need to agree on file naming convention and format and frequency / timing of transfer.
Use Cases – Change un customer addressin customer master needs to be propagated to say Loyalty system. Technologies – FTP, SFTP :-



File transfer – In what situation have you used File transfer ?
Shared Database:-
Pro :- Timely availability of data, Common format, Consistency of data.
Con – Difficult to integrate with 3rd party packages.
Use Case – Production module sharing data with Finance module.

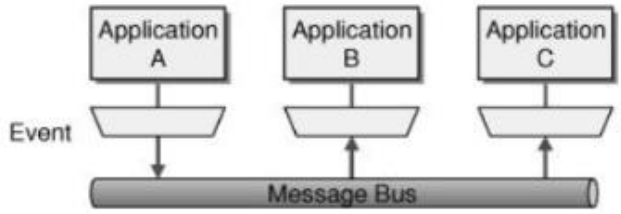
Remote Method Invocation – Good for quick data access from a disparate application.
Pro – Encapsulates applications, Con – Tightly coupled
Technologies – CORBA, COM, .Net Remoting, Java RMI, SOAP, gRPC



Serialization -converts data structure in different programming language to a common format. XML. JSON and YAML are example of common formats.

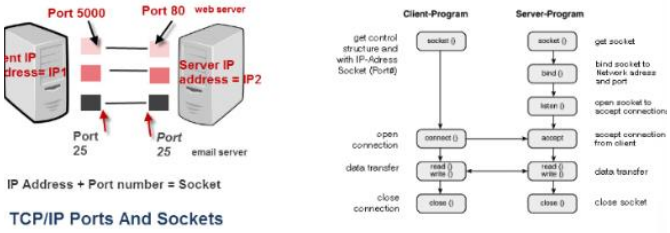
Remote Method Invocation -In what situation have you used RMI ?

Messaging -Good for notifying of events. Loose coupling , asynchronous, works even if the other systems is down, not as fast as RPC. Eg – when an insurance claim is filled , send message to fraud detection application. Technologies – MQ Series, RabbitMQ, Kafka, MQTT, Simple Notification Service, SQS, TIBCO Enterprise Service Bus.

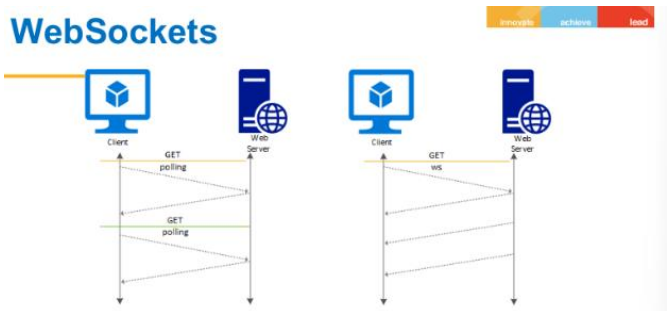


MMessaging : MQ Series
MMQ supports point-to-point and Publish-Subscribe messaging.
APIS supported in C, C++, Java,.net.
Features – One time delivery (No duplications), Data trtransformation, Trigger applications and Persistent message (((Resilient to server failure)

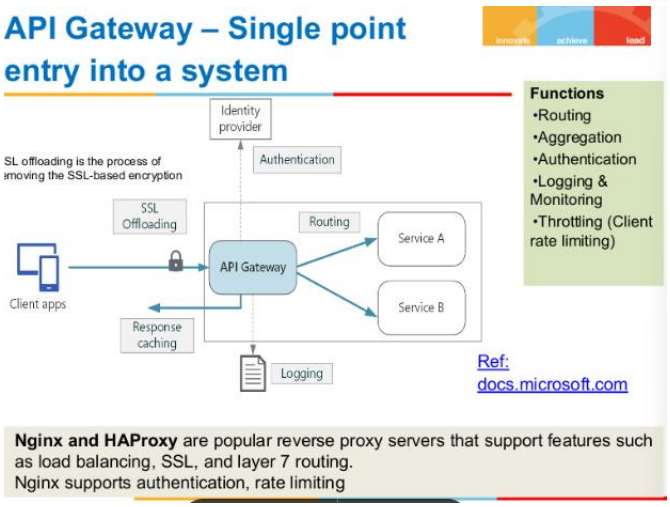
T TCP/IP Sockets



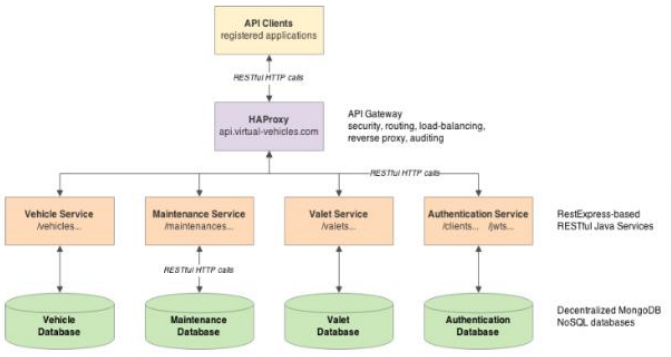
ockets do not have much communication overhead, compared to underlying rotocols such as HTTP/DHCP/SMTP etc.



- Usually Webserver responds only when browser sends request
- ***WebSocket*** is a protocol that allows the webserver to send things to the browser without the browser having to request it.
- **Use case:** Notifify browser when stock price changes.
- The connection remains open till closed by client or server



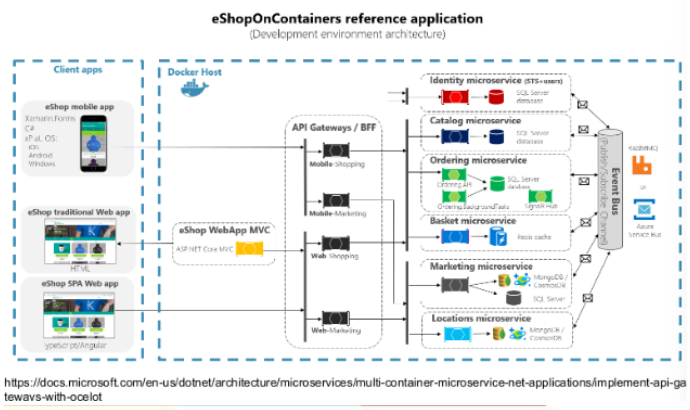
Nginx and HAProxy are popular reverse proxy servers that support features such as load balancing, SSL, and layer 7 routing.
Nginx supports authentication, rate limiting



Benefits and Drawbacks of an API gateway

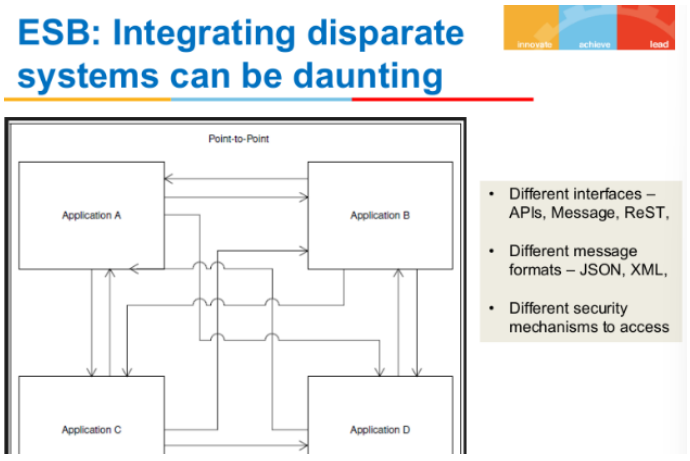
Benefits – Client simply talk to the gateway, rather than having to invoke specific services. Reduces the number of round trip between the client and application. Simplifies the client code.

Drawbacks – Needs to be highly available component. Can become a development bottleneck – developers must update the API gateway in order to expose each microservice's endpoints.

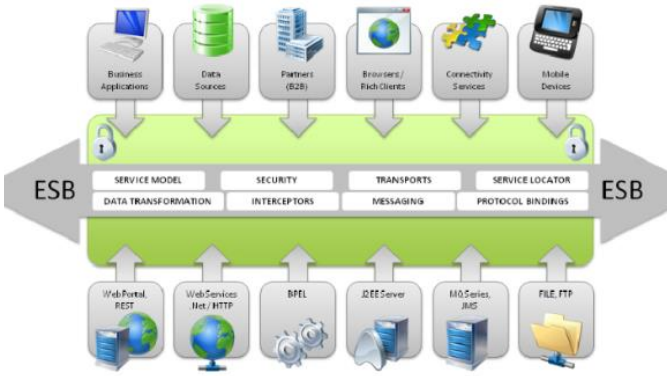


https://docs.microsoft.com/en-us/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gatewavs-with-ocelot

Supports traffic management, authorization and access control, throttling, monitoring, and API version management



Enterprise Service Bus is a standardized integration platform that combines messaging, web services, data transformation and intelligent routing. It is used in legacy applications.



E - ESB – Core Functionalities:-

. Transport Protocol Conversion – eg – HTTPS to MQ Series

. Message Transformation – eg – XML to JSON, XML to Java Objects

.Security – Protects services from unauthorized access.

.Service Orchestration /Choreography – Manages complex business services by coordination with different services.

. Transaction Management – Provides atomicity of transaction spanning multiple disparate system.

ESB implementation:- Jboss ESB, Apache Servicemix, Open ESB, Mule ESB, JBoss Fuse

Case Study – SWIFT

SWIFT stands for Society for worldwide interbank financial telecommunication. It is a messaging network that financial institutions use to securely transmit information and instructions through a standardized system of codes. SWIFT is used by Banks, Brokerage Institutes and Trading Houses, Securities Dealers and Foreign Exchange and Money Broers.

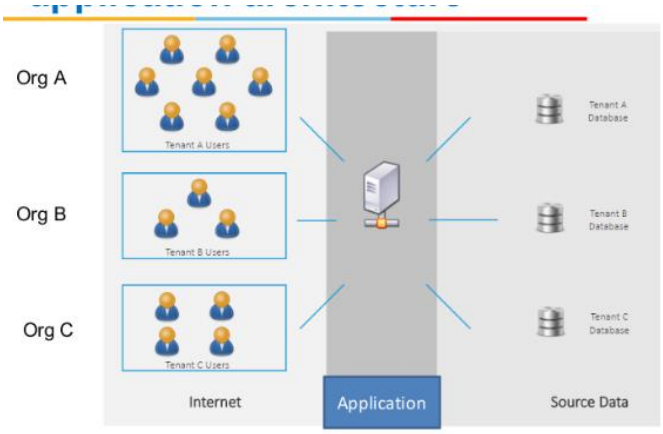
Cloud Based Apps – SaaS

Some example of SaaS apps are , SaleForce, SAP and Intuit. SaaS applications are becoming popular . because it is pay per use, easy scaling and reduced cost.

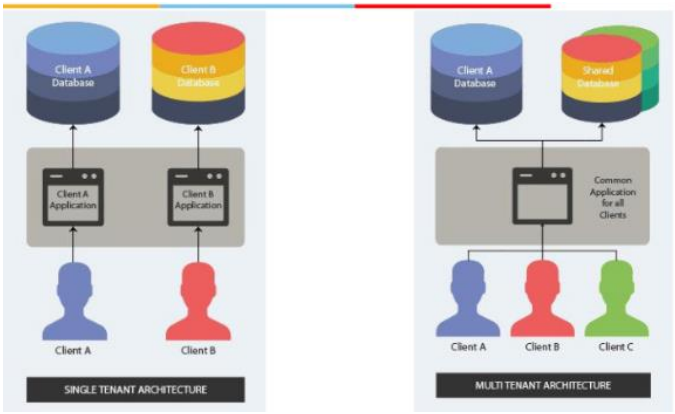
Cloud Apps:- Application is shared by many users, Reduced cost of upgrades, high volumes of users/organizations, enables procuring resources at lower costs.

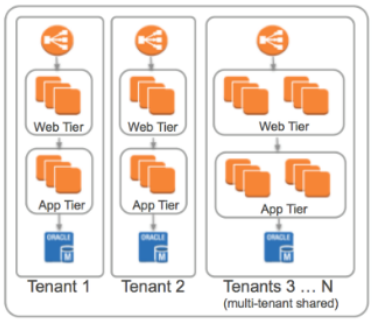
SaaS appa are shared by multiple users/organization.Such and application is called a multi-tenant application. In - premise app is usually a single tenant app.

Example of multi-tenant architecture:-



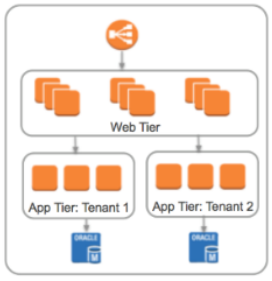
Multi – tenancy is a key characteristics of SaaS which refers to a software architecture in which single instance of software serves multiple tenants ot client organizations. You can think of this as multiple organizations sharing the same office space.





Pure
Each tier is single tenant

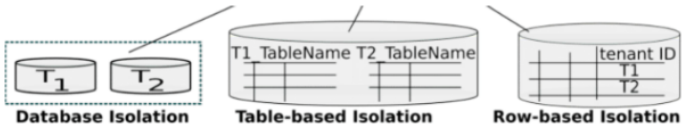
Shared
Each tier is shared



Hybrid
Web tier is multi-tenant
App tier is single tenant

Data isolation options

Options



Legend:

T1 = Tenant 1
T2 = Tenant 2

Pros and cons of multi-tenancy

Pros	Cons
• Lower cost • Better resource usage • Easier updates / upgrades	• Lower security • Malfunctioning of one may affect another

Multitenant applications are expected to provide adequate, isolation (security), robustness and performance.

Multi-tenant example – Public survey software

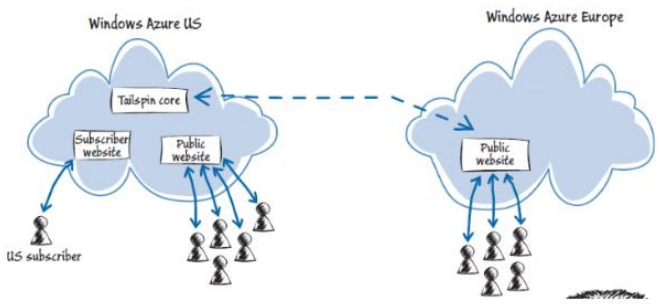


FIGURE 3
A U.S. based subscriber running a survey in Europe

Sample Architecture – Component level

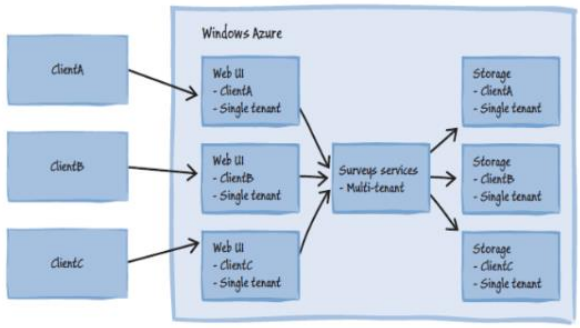
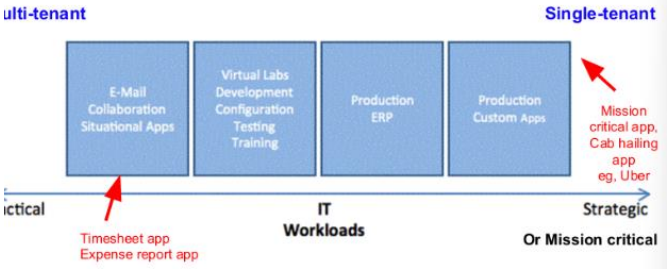


FIGURE 2
Sample architecture for Windows Azure

How to choose degree of multi-tenancy?

Consider
- Nature of workload – Are we using Cloud for Production, development or testing?
- Security need – Do we require high security?



What is the degree of multi-tenancy you would recommend for the following applications? Why?

.Customer service – Complaint, FAQ, Assigning engineer
Multi-tenant - because not much confidentiality involved

.HR – Employee salary, Performance
Data tier - Single tenant, because data is highly confidential.
Web-tier and App tier - can be multi-tenant

.Attendance system with Face recognition
Web tier – Multi-tenant (not much processing load)
App tier – Single tenant (machine learning tool with high processing)
Data tier – Single tenant (Confidential data)

Trends in Cloud Based Service:-

- Language
  - Python, Go language (from Google – simple & supports concurrent pgmming)
- Database: NoSQL database for large data
- Integration methods
  - Queues
  - Publish-subscribe
  - REST based services
- Micro-services deployed in Containers
- Engineering practices
  - Continuous integration / Continuous Deployment (CI / CD)
  - Measure performance of applications

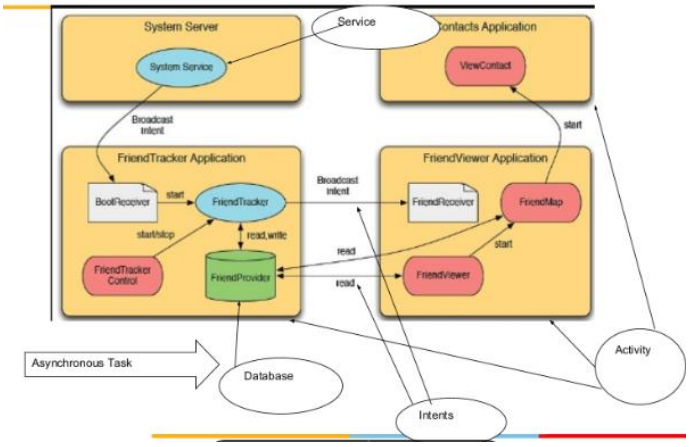
Amazon Web Services Tools

- Load balancer - Elastic Load Balancer
- Edge caching (CDN) – Cloud Front
- Monitoring - Cloud-Watch : CPU, data transfer rate (Net), disk usage
- NoSQL DB – Dynamo DB
- Open source tools
  - Logging – Log Stash
  - Indexing – Elastic Search
  - Analytics & Visualization – Kibana

Popular services of cloud vendors

Google App Engine	Microsoft Azure	Amazon Web Services
• Python & Java development environment • Auto scaling • Database replication	• .Net environment • Auto scaling • Load balancing • Failure detection & auto replication of services • Database replication	• Java development • Auto scaling • Load balancing • Failure detection & auto replication of services • Database replication • Data caching (Memcached) • Service discovery (SoA) • Notification of events (SNS) • Message queue (SQS) • CDN (Content Delivery Network) – YouTube

Mobile app: Example



Amazon's approach to handling issues in distributed systems

To scale you have to partition, so you are left with choosing either high consistency or high availability for a particular system. You must find the right overlap of availability and consistency.

Choose a specific approach based on the needs of the service.

For the checkout process you always want to honor requests to add items to a shopping cart because it's revenue producing. In this case you choose high availability. Errors are hidden from the customer and sorted out later.

When a customer submits an order you favor consistency because several services--credit card processing, shipping and handling, reporting--are simultaneously accessing the data.

Is. Issues in Cloud Based Systems:-

A.Availability – Coud vendors promise high availability ex- 99.95%

b.but still not 100%. Need to design for the 0.5%. One approach – S. store same data in different geographica; zones as done by Netflix.

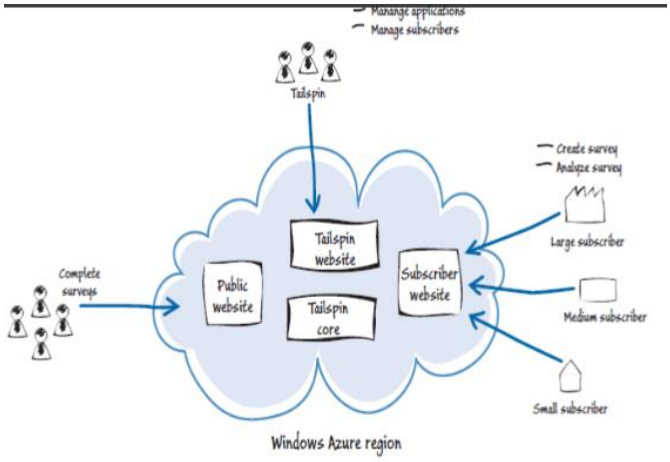
.

..- Scenario

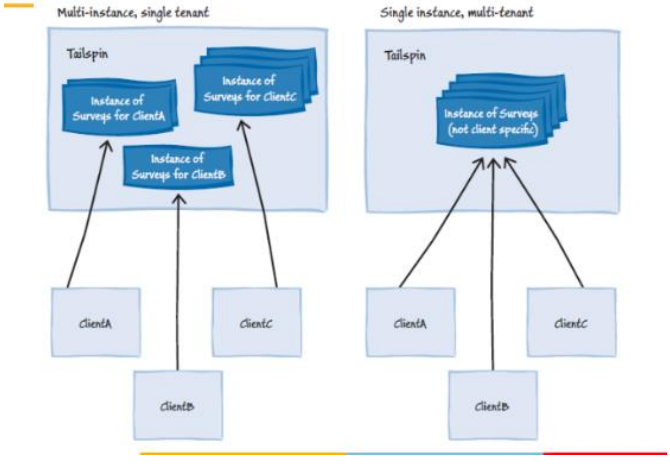
start-up company is developing a GST tax returns filing system. This software will be deployed on the Cloud and offered to small and medium businesses as a SaaS.

he clients will have to input their data or upload data using an Excel file. They also need to provide other details such as GST #, etc. The software will process the data and file the returns into the Government's GST system on behalf of the client.

fter developing the software, what options exist for deploying it in the cloud?



Architecture Options – High level

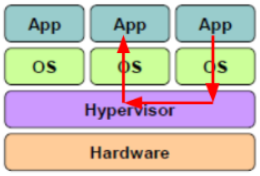


Issues in Cloud based systems

- Security issue due to multi-tenancy
  - Poor design leading to inadvertent sharing of information
  - Virtual machine 'Escape'

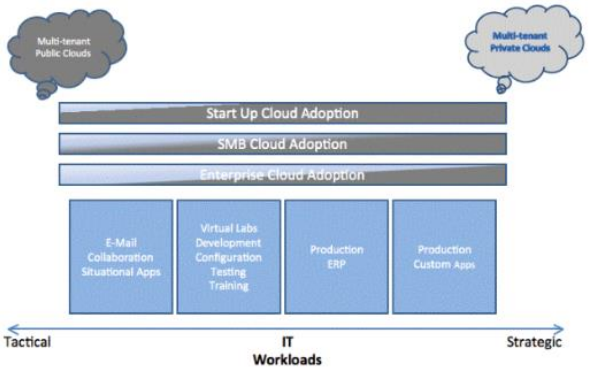
Other fields of the table					OU_ID
					Org1
					Org1
					Org1
					Org2
					Org2

Same table contains data of different organizations



Virtual machine 'Escape'

How to choose your multi-tenancy degree?



How to choose your multi-tenancy degree ?

The characteristics of the workload in question have to be carefully studied first, including the workload's utilitarian versus strategic value, volatility, security etc. Higher degree of multi – tenancy are best suited for cross-inductry utilitarian versus strategic value, volatility, security etc. Higher degrees multi-tenancy are best suited for cross-industry utilitarian workloads such as e-mail, expense reporting, travel authorization and sales force management. These applications can very easily share the same schema.

Degress of multi-tenancy:- Highest degree : IaaS and PaaS are multi-tenant. SaaS Is fully multi-tenant also.

Middle degree – IaaS and PaaS are multi-tenant. Small SaaS clusters are multi – tenant.

Lowest degree – IaaS and PaaS are multi-tenant. SaaS is single tenant.

Cloud Native Applications:- Cloud native computing takes advantages of many modern techniques including:- PaaS, multicloud, microservices, agile methodology, container, CI/CD and devops.

Architecting in a Cloud Environment

1. Security | 2. Performance | 3. Availability

Security Attacks in Multi-tenancy:

1. **Inadvertent information sharing**: *It is possible that information remaining on a physical resource from one tenant may "leak" to another tenant.*
2. **A virtual machine "escape"**: *An attacker can exploit software errors in the hypervisor to access information they are not entitled to.*
3. **Side-channel attacks** : *A malicious attacker to deduce information about keys and other sensitive information by monitoring the timing activity of the cache.*
4. **Denial-of-service attacks** : *Other tenants may use sufficient resources on the host computer so that your application is not able to provide service*

rchitecting in a Cloud Environment

1. Security | 2. Performance | 3. Availability

Computational capacity of any virtual machine:

1. One virtue of the cloud is that it provides an ***elastic host***.
2. **Elasticity** means that additional resources can be acquired as needed.
3. The application *should be self-aware of both its current resource usage and its projected resource usage* (Instead of depending on cloud provider auto allocation of additional virtual machine or based on requests).
4. An application should have a better model of its own behaviour and be better equipped to do its own allocation or freeing of resources.
5. The freeing of resources may not be instantaneously reflected in the charging algorithm used by the provider.

rchitecting in a Cloud Environment NETFLIX

1. Security | 2. Performance | 3. Availability

A virtual machine, hosted on a physical machine can fail.

The service-level agreement that Amazon provides for its EC2 cloud service provides a 99.95 percent guarantee of service, and as an architect need to plan for that .05 percent.

Netflix is a company that streams videos to home television sets, hosts much of its operation on Amazon EC2. On April 21, 2011, Amazon EC2 suffered a four-day sporadic outage. Netflix customers, however, were unaware of any problem.

Architecting in a Cloud Environment NETFLIX

1. Security | 2. Performance | 3. Availability

Stateless services.

Netflix services are designed as "service instances" : If a server fails, requests can be routed to another service instance.

Data stored across zones.

Amazon provides "availability zones," which are distinct data centres. Netflix ensured that there were multiple redundant hot copies of the data spread across zones.

Graceful degradation.

Fail fast: Set aggressive timeouts such that failing components don't make the entire system crawl to a halt.

Fall-backs: Each feature is designed to degrade or fall back to a lower quality representation.

Feature removal: If a feature is noncritical, then if it is slow it may be removed from any given page.

When would you use following options:-
App1: Web tier – Multi-tenant, AppTier – Single Tenant Data - Multi- tenant
Ans – Web tier processing is light, App processing is heavy, data is not confidential.

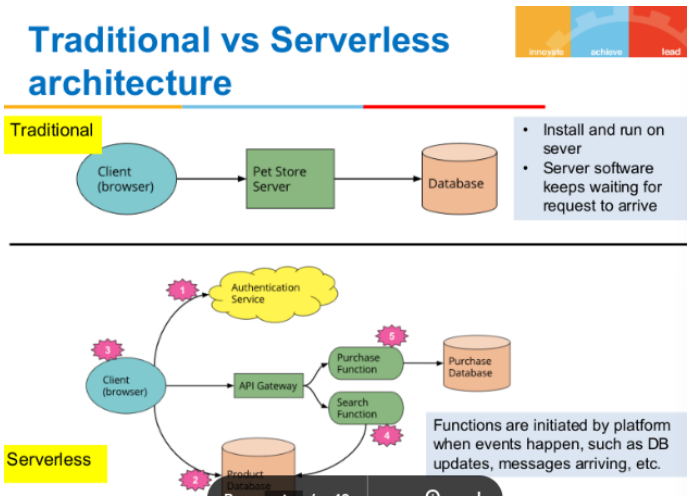
App2:- Web tier – Single Tenant, App Tier – Multi-tenant, Data Tier – Single Tenant
Ans – Web tier processing is heavy, App processing is light, data is confidential.

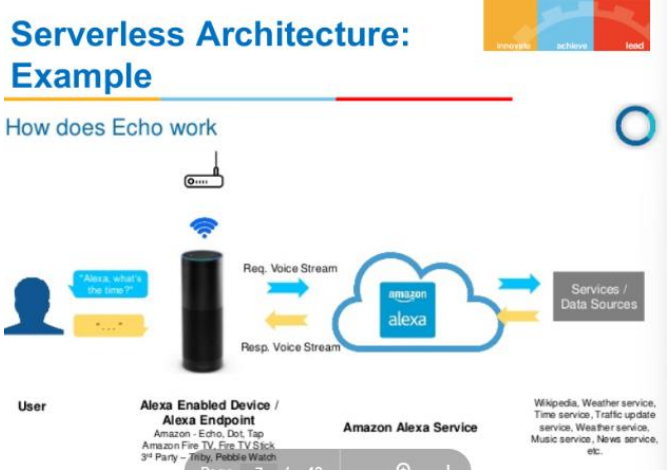
Answer:

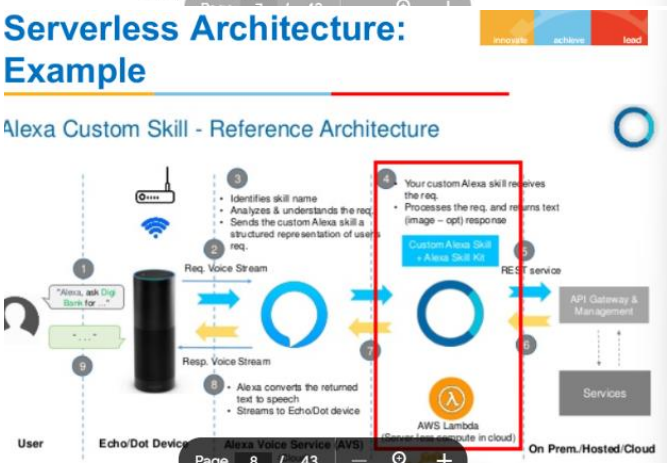
- 1 could be used for software solutions where multiple organizations or users need access to a standardized application with shared functionality but require separate instances, For example, a SaaS product offering accounting or project management services to multiple businesses would benefit from this
- architecture2 might be suitable for applications where data isolation and security are major concerns, such as healthcare or financial systems, where compliance requires strict control over data access and storage. Each client requires their own dedicated resources to ensure compliance and mitigate security risks.

Serverless Architecture – Serverless does not mean we do not use servers. A serverless architecture is a way build and run applications and services without having to manage infrastructure. Serverless architecture reduces the operational complexity of running and managing applications. It removes the need for the traditional "always on" server system sitting behind an application. In serverless architecture, we focus on the application and not bother about provisioning (configuring) servers, deploying the application on servers, managing the servers etc. AWS Lambda is an example of a serverless architecture. Scales up the number of instances to handle high volumes of usage.

Deploying and managing applications:-
Steps:- Estimate the load on the application, Determine the server configuration and number of servers needed, Load software onto a server, Start the application and monitor application health and scale up/ scale down as appropriate.

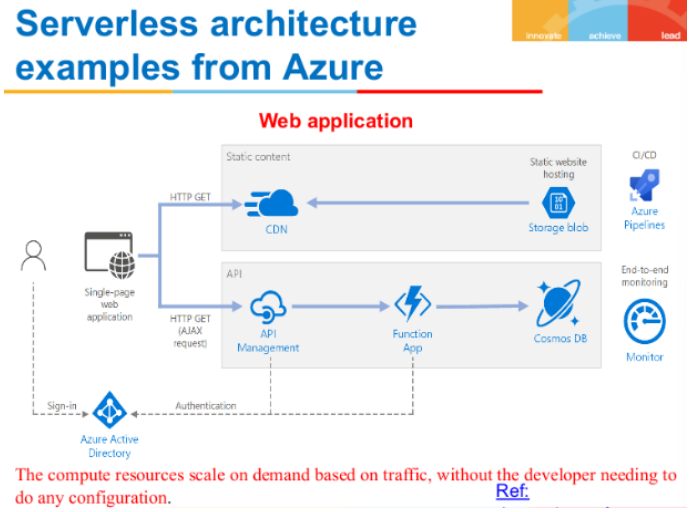


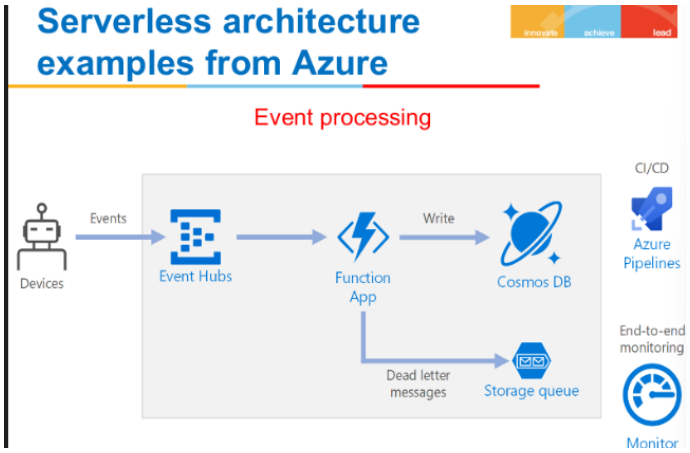




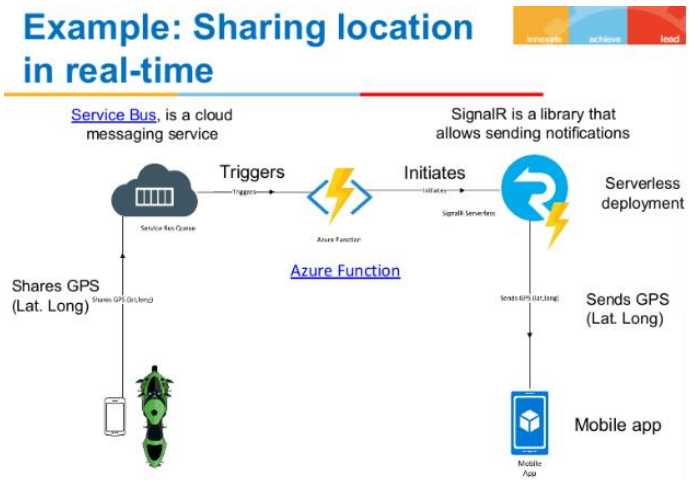
C. Creating a serverless application in AWS Lambda:-
.. . creates a Lambda function to process an event one at a time. Use one supported languages – Python, Java, Go, C#, Node.js.
....Configure the Lambda function – Timeout, Concurrent Instances etc. You can log execution flow and performance.
C. Creating a serverless application in AWS Lambda :- Events that can trigger the function/call the function are:- An HTTP request for ttthe function. File creation in S3, database update, Steam events in Kinesis, Customer services. Lamda functions can make use of libraries , custom runtime, and other functions. It can also make use of services such as DynamoDB, S3 Buckets.

..Some use cases for Lambda functions:- In a photo sharing applications, uploading a photo can trigger a Lambda function to create a thumbnail. Lambda function can be invoked by resource usage monitoring tools such as Cloudwatch ,when a threshold is crossed, and the Lambda function can notify system administrative via email or SMS.
.
...Serverless Architecture is also called as Function as a Service (FaaS). Characteristics – Functions are stateless, Start -up latency m may by high.



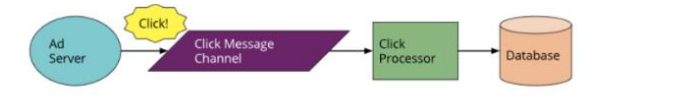


. Use cases for serverless architecture:- Sharing real-time location of fleet vehicles with client devices, Pushing notifications to users upon occurrence of an event. Updating project timelines upon completion of a project task. Responding to chat rooms messages.



Cloud vendors offering serverless technology:- Amazon AWS Lambda functions, Microsoft Azure functions, Google Cloud functions, IBM Cloud functions, Oracle Cloud – Fn project.

Another example of serverless architecture:-



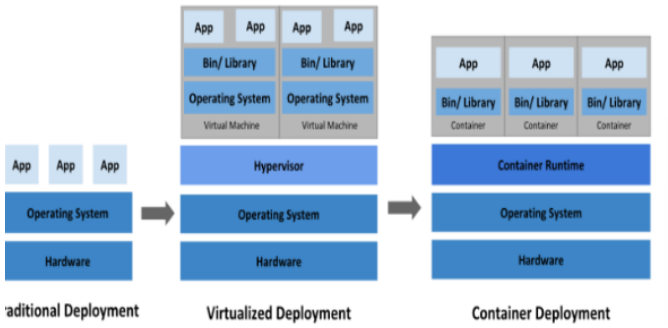
In the Serverless world this looks as follows:

....Functions are invoked only when required.

. ..Container Technology – Container is a lightweight, standalone, e executable package that includes everything needed to runan application: - code, runtime, system tools, system libraries and s. settings. Amazon Web Services , Microsoft Azure and Google Cloud Platform have embraced container technology. Examples of container technology – Docker, Apache Mesos, rkt , and k.kubernates.

.

...container deployment:-



C

.....Containers vs Virtual Machines:- Containers are similar to vivirtual machines but are much lighter.With a virtual machine, it requires several minutes to boot up, just like the PC on your disk g..getting started up at the beginning of the day. With container t._technology , as the OS is already running on the server, a container can be started in a few seconds. This allows containers to to be started and stopped as needed, to flex up at a time of p.peak demand, and to flex down when not needed.

C Why use containers ? – Have you had a situation where a program runs perfectly on development machine, but not on QA M_ machine? With containers one can package a code along with li _ libraries, etc. into a container and ship it to QA / production Eeenviornment..

__Benefits of containers

Application becomes modular, consistent run time environment, easy to scale, changes can be deployed rapidly in CI/CD model, services can be monitored.

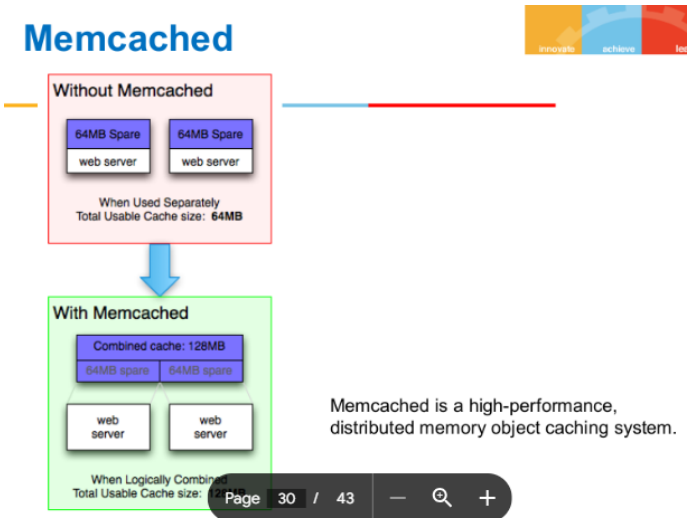
Use of containers to run micro-services:- Containers are used to run microservices. If a particular microservice is invoked more often, we can create more instances of that container. Each container can be maintained independent of others.

Docker :- Docker is a platform where you make an image of your application along with the environment setup. Steps – Create docker image, Put image in Docker container and Create Docker instance. Docker instance acts like separate machine and has an IP address and ecosystem of its own. Docker creates additional instances depending on the load.

Container deployment tools – Docker Swarm and Kubernates :-
Features of container deployment tools – Service discovery, Scaling, Load Balancing, Self healing – Restart failed containers, Kill containers not responding to health check.

Caching – Keeping most frequently used data in memory for faster access. Use Least Recently Used (LRU) algorithm to manage space available in cache.
Example of data that are cache – Product data, Airfare, Country codes, State codes.
When should the cache be populated ?- At start up :- if we know upfront what data needs to be cached, for example list of countries or list of states in a country , we can populate the cache at the start of the application.
Dynamic – Else the contents can be built dynamically . eg- product details.
How to keep the cache up-to-date when contents in database changes?
Periodic update: If we can afford to have some old / stale data (eg. Average price of stock / share), then we can refresh at fixed
intervals, say every 5 minutes

Update when changed: If not, we can set up in such a way that the database calls a function whenever data changes and this function then updates the data in cache

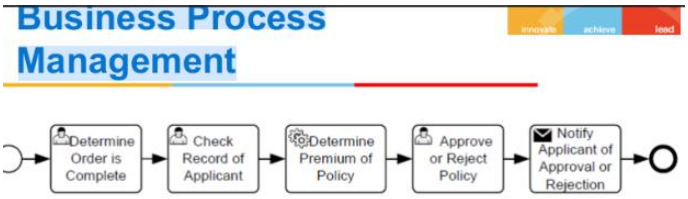


Memcached : How does it work ?
To store : Client determines a server to store using a hashing algo. Sends data to that server. Server uses another hashing algo to store the object in a table. If memory is full, it uses LRU algo to release space and store new data.
To retrieve – Client determines the server where the object is likely to be stored using a hashing algo. Sends request to the server. Server does a lookup in its hash table to see if the object exists and returns if found. If not found, the client will retrieve from DB.

How long to store in cache ?- You can specify an expiry time after which the data is removed.

Memcached Components – made up of four components :-
Client – Software :- Which is given a list of available Memcached servers.
A client-based hashing algorithm :- Chooses a server based on the "Key".
Server Software – stores values and their keys into an internal hash table.
LRU – Determine when to throw out old data or reuse memory.

How is data stored in cache ?- Data is stored in a key-value pair.
Value can be anything – a string, binary , serialized object.

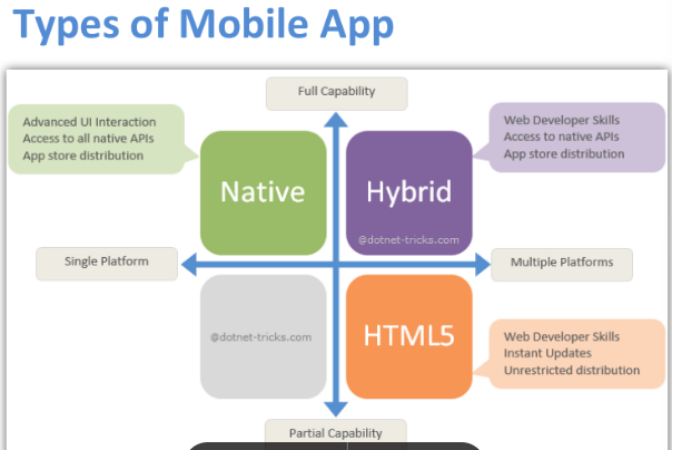


Business Process Management Tools: Business Process Management (BPM) tools are used for automating, measuring and optimizing business processes.

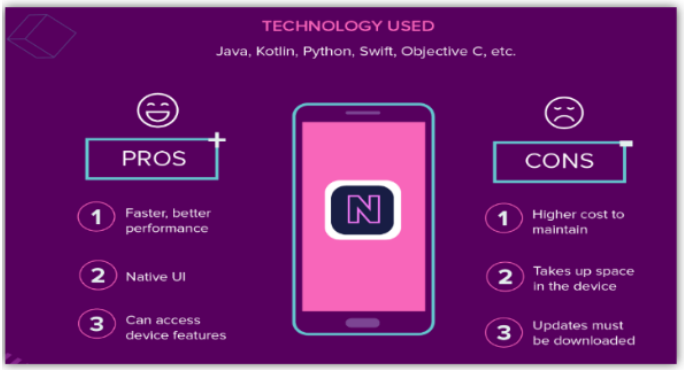
BPM tools use workflow and collaboration to provide meaningful metrics to business leaders

Example: Appian, Zoho

Architecting Mobile Applications:-
Uber, Swiggy, Courier delivery, ecom, banking, spotify, where is my train
Mobile Applications – Design Considerations
Simple User Interface – Easy to type, large buttons , minimal features., menu options, actions.
Responsive design – Adapt to different screen size and orientations.
Compact code – Less usage of CPU, memory, storage.
Few layers to ensure performance
Connectivity – Store data locally and synchronize later if connection is poor.



Native Mobile App

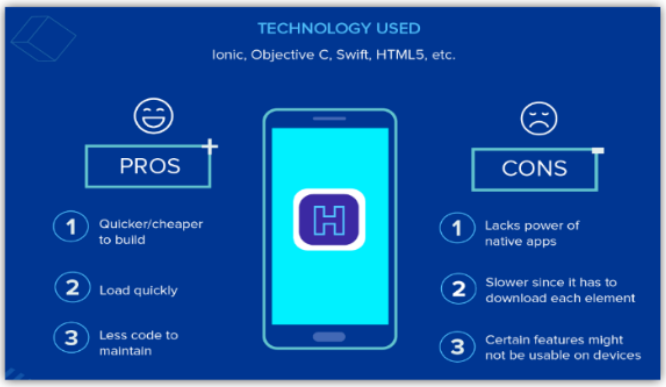


Example : Google Maps, Facebook, LinkedIn - one version for iOS and one for Android

Web Mobile App

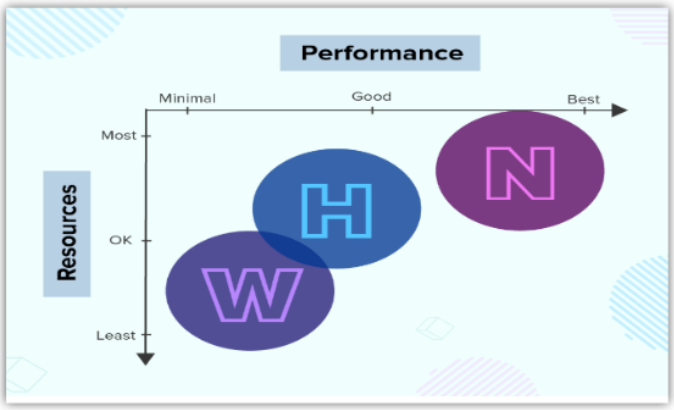
Example : Flip kart, BookMyShow and MakeMyTrip

Hybrid Mobile App



Example : Twitter, Uber and Instagram

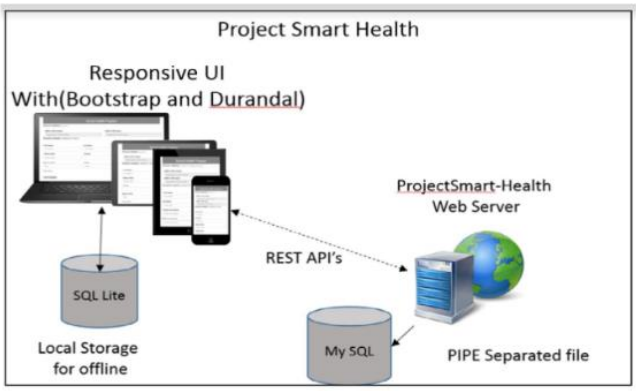
Comparison of Mobile Apps



W - Web Based : H - Hybrid : N - Native

UI Design Patterns:-
Action bars for quick access to frequently used actions
Login using facebook, google etc. instead of separate user id password.
Large buttons for ease of use
Notifications of recent activity.
Discoverable controls – Controls show up only when an item is selected.
Mobile Optimized website (All feature to reduced feature), Responsive design

Store Locally, Sync later
Incase of intermittent connectivity



Android Application Architecture :- 4 types of architecture:
Activity (UI), Service (Background process) – eg – playing music, download
Content provider (Storage) – eg: SQL Lite, files
Broadcast receiver (Acts on events received from OS and other apps) – eg : arrival of SMS
Example of a Mobile App Component in Uber App –
UI – Screen to book a cab
Broadcast receiver – Receive location of cab from backend server and provide to UI for display.
Service – Provide cab location to backend server after journey starts.
Database – Configuration file containing user data.

Big Data Analytics :- as a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.

Big Data Analytics :-
Volume of data –
Processing.

Characteristics of Big Data :- Volume, variety and velocity.

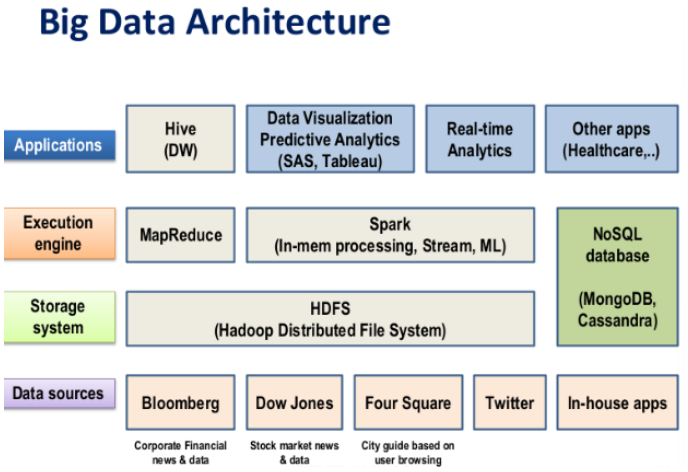
Use of Big Data – Recommend cancer medication, Weather prediction, Predict equipment malfunctioning, credit card fraud detection.

Google Search :- Examples of Big Data
3 Steps:-
Catalog & index – Even before you search, Google goes through various web sites and linked web sites (crawling) and catalogs all the websites. This runs into several Tera bytes.

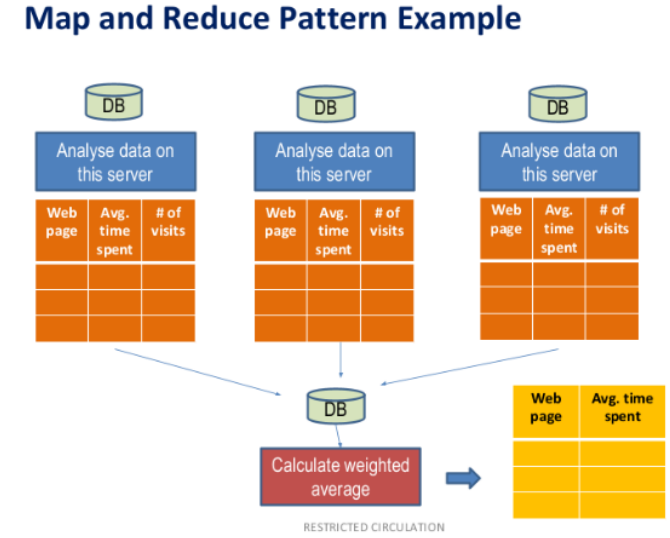
Understand & enrich your query – Correct spelling mistakes, consider synonym etc.
Search – When the search is requested, it uses this catalog to determine which website closely match the requirement. For this it uses a 'Page-Rank" algorithm which considers factors such as which page has max number of occurrences of the search string , how many other websites refer to this website, what is the reputation of the websites that refer to this website.

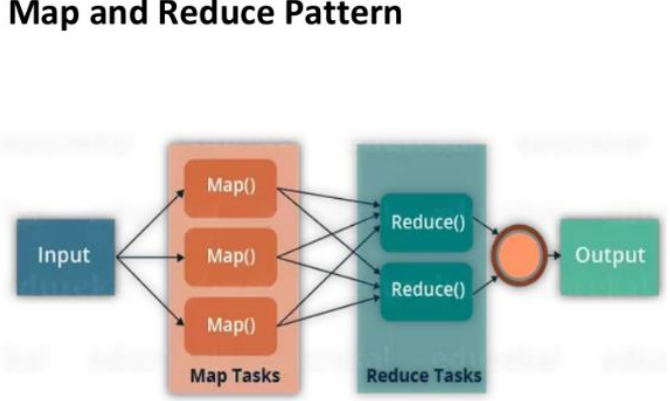
Google Big table is a wide table containing several attributes of a website. Some of the attributes are the contents of the web page, Anchor text, websites referencing the page and time stamp when the data was stored. Google BigTable is built on technologies like Google File System. Google BigTable is used by applications such as Google Maps, Google Analytics, atc.

Use of Big Data - Banking and Financial Services, Retail, Sentimental Analysis, Customer Service, Industrial equipment monitoring and alerting, weather forecasting.

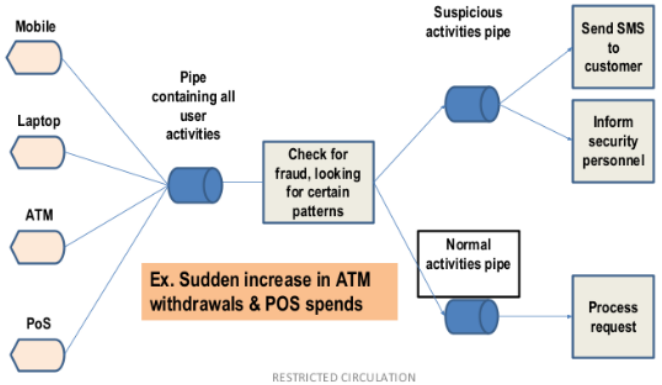


Map and Reduce Pattern:- Used to analyse vast amount of data. Suppose we keep track of every click of the users and store these details in a database. Let us say we want to find out the average time spent by users on each web page of the website, across thousands of users who visited the website in the last 30 days. How can we speed up the analysis ?-



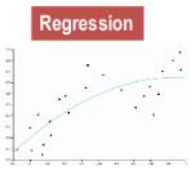


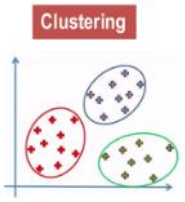
Real Time Analytics:- Continuous monitoring of client's activity to see if there are any potential issues.

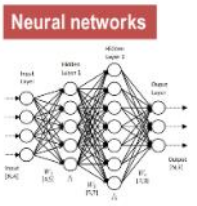


Machine Learning – ML is the ability of machines to detect patterns and perform activities that humans do.
Applications:- Determine treatment for a patient based on their genetic, makeup, demographic and psychographic characteristics.
Pro-active maintenance – of industrial equipment by looking at health parameters such as temperature, vibration, oil level.
Credit Card Fraud Detection – by looking at spend frequence, amount, time of day, place of transaction, product purchased, customer profile.

Types of Machine Learning – Supervised, Unsupervised, Reinforcement.







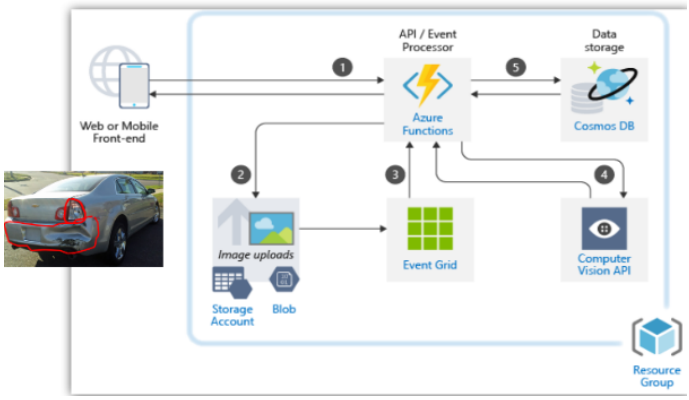
Supervised learning – In this approach, we provided a labelled dataset to the machine. Labelled dataset consists of features and result or class. Using the dataset, the machine builds a model (an equation or structure) to predict.

Unsupervised Learning – Here we have an unlabelled dataset. We do not know what all features will constitute a class. The machine learns by itself and builds a model.

Reinforcement Learning – In this, the machine learns from the environment by interacting with it. The machine is provided a set of allowed actions, rules and potential end states. By exploring different actions and observing resulting reactions the machine learns to exploit the rules to create a desired outcome. Eg – Game of chess, robotics.
Algorithm used – Neutral networks, Learning Automata, Q-Learning, Markov decision process.
Deep Learning – used to understand and analyse image, sound and video. Uses Neural networks. Used to understand human language (NLP)
Neural Networks – These networks can learn and model the relationships between inputs and outputs that are complex.
Example – Detecting rare events such as frauds, help doctors with an opinion.
Steps to build ML Model :-
Identify the problem to be solved. Identify the features , decide on the model , train-test-validate – experiment

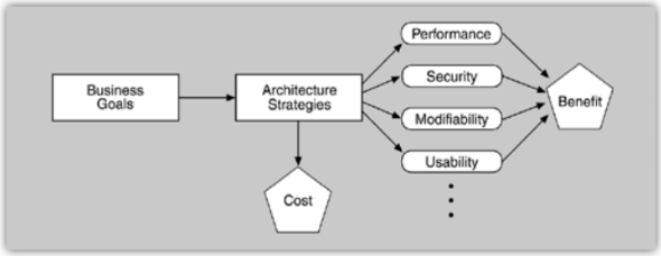
Architecture of ML system:

Image classification for insurance claims



Scenario for ML in Modeical devices Technology:-
CT Scanners and Digital X-Ray Machines use X-Ray tubes , which are backbone for patient scan.
The cost of x-ray tubes is almost 30% of the equipment cost.
Any unprecedental failure of X-ray tube will make the complete machine out of order.
ML can help in monitoring specific parameters such Current Inflow, oil Levels, Temperature generated , number of scans performed, criticality of the scans etc, and enable predictive maintenance or recommend replacement of x-ray tube that can completely avoid break-downs.

Decision Making Context –

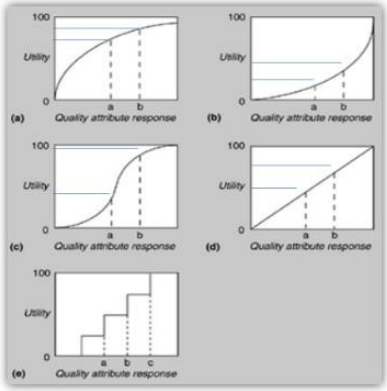


A Business Goal of achieving High Availability may cost more as the infrastructure needs to be made redundant . So, the Quality attributes that are expected to be achieved makes the foundation for the cost analysis.

ARCHITECTURAL STRATEGY = VALUE FOR COST (VFC)

Economic Analysis will help You to evaluate the Viability of Architectural Strategy

Utility - Response Curve



These are various responses for a CHANGE in QA Vs UTILITY

- Curve "a" – For a larger change in QA, a small change in utility value is observed. The curve is flattening. So beyond a point in time, there is no value if you invest money on adding additional QA.
- Curve "b" – shows for a small change in QA, a significant change in UTILITY VALUE is observed. So, VFC is larger. Hence, it is worth in spending more money for QA.

Determining Benefits and Normalization

The overall benefit of an architectural strategy across quality attribute scenarios is the sum of the utility associated with each one, weighted by the importance of the scenario.

For each architectural strategy *i*, its benefit B_i over j scenarios (each with weight W_j) is:

$$B_i = \Sigma_j (b_{i,j} \times W_j)$$

$B_{i,j}$ can be calculated from the **UTILITY VALUE** as below:

$$B_{i,j} = U_{expected} - U_{current}$$

Calculation of Value for Cost (VFC) :- The VFC for each architectural strategy is the ratio of the total benefit to the cost of implementing it. You can use this VFC score to rank-order the architectural strategies under consideration.
CBAM – Cost Benefit Analysis Method
Step 1:- Collate Scenarios

- Give the stakeholders the chance to contribute new scenarios.
- Ask the stakeholders to prioritize the scenarios based on satisfying the business goals of the system.
- This can be an informal prioritization using a simple scheme such as "high, medium, low" to rank the scenarios.
- **Choose the top one-third for further study**

Step2 : Refine Scenarios
Refine the scenarios chosen, focusing on their stimulus – response measures. Elicit the worst-case, current, desired,

and best-case quality attribute response level for each scenario.

Scenario	Worst Case	Current	Desired	Best Case
Scenario #17: Response to user input	12 seconds	1.5 seconds	0.5 seconds	0.1 seconds
...				

Step 3:- Prioritize Scenarios

1. Prioritize the refined scenarios, based on **stakeholder votes**.
2. You give **100 votes to each stakeholder** and have them distribute the votes among the scenarios, where their ***voting is based on the desired response value*** for each scenario.
3. ***Total the votes and choose the top 50 percent of the scenarios*** for further analysis.
4. ***Assign a weight of 1.0 to the highest-rated scenario***; assign the other scenarios a weight relative to the highest rated.
5. This becomes the weighting used in the calculation of a strategy's overall benefit.
6. **Make a list of the quality attributes that concern the stakeholders**

Step 4: Assign Utility

- Determine the utility for each quality attribute response level (***worst-case, current, desired, best-case***) for the scenarios from step 3.
- You can conveniently capture these utility curves in a table (one row for each scenario, one column for each of the four quality attribute response levels).
- This step would assign utility values from 1 to 100 for each of the latency values elicited for this scenario :

Scenario	Worst Case	Current	Desired	Best Case
Scenario #17: Response to user input	12 seconds	1.5 seconds	0.5 seconds	0.1 seconds
	Utility 5	Utility 50	Utility 80	Utility 85

Step 5: Map architectural strategies to scenarios and determine their expected quality attribute response level. – For each architectural strategy under consideration , determine the expected quality attribute response level that will result for each scenario.

Step 6 : Determine the utility of the expected quality attribute response levels by interpolation

- Using the elicited utility values (that form a utility curve), determine the utility of the expected quality attribute response level for the architectural strategy.
- Do this for each relevant quality attribute enumerated in step 3.
- For example, if we are considering a new architectural strategy that would result in a response time of 0.7 seconds, we would assign this a utility proportionately between 50 (which it exceeds) and 80 (which it doesn't exceed).
- The formula for interpolation between two data points (x_a, y_a) and (x_b, y_b) is given by:

$$y = y_a + (y_b - y_a)\frac{(x - x_a)}{(x_b - x_a)}$$

- The *x* values are the quality attribute response levels and the *y* values are the utility values. So, employing this formula, the utility value of a 0.7-second response time is 74

Step 7 : Calculate the total benefit obtained from an architectural strategy

- Subtract the utility value of the "current" level from the expected level and normalize it using the votes elicited in step 3.
- Sum the benefit due to a particular architectural strategy across all scenarios and across all relevant quality attributes.

Step 8 : Choose architectural strategies based on VFC subject to cost and schedule constraints

- Determine the cost and schedule implications of each architectural strategy.
- Calculate the VFC value for each as a ratio of benefit to cost.
- Rank-order the architectural strategies according to the VFC value and choose the top ones until the budget or schedule is exhausted.

Step 9 : Confirm results with intuition

- For the chosen architectural strategies, consider whether these seem to align with the organization's business goals.
- If not, consider issues that may have been overlooked while doing this analysis.
- If there are significant issues, perform another iteration of these steps

North East Observing System:-

The ECS [Environmental Control System] processes an input stream of hundreds of gigabytes of raw environment- related data per day. The computation of 250 standard "products" results in thousands of gigabytes of information that is archived at eight data centers in the United States. The system has important performance and availability requirements. The long-term nature of the project also makes modifiability important.

The ECS project manager had a limited annual budget to maintain and enhance this current system. From a prior analysis - in this case an ATAM exercise - a large set of desirable changes to the system was elicited from the system stakeholders, resulting in a large set of architectural strategies. The problem was to choose a (much) smaller subset for implementation, as only 10 to 20 percent of what was being proposed could actually be funded. The manager used the CBAM to make a rational decision based on the economic criterion of return on investment.