1. Introduction to Software Architectures.

*

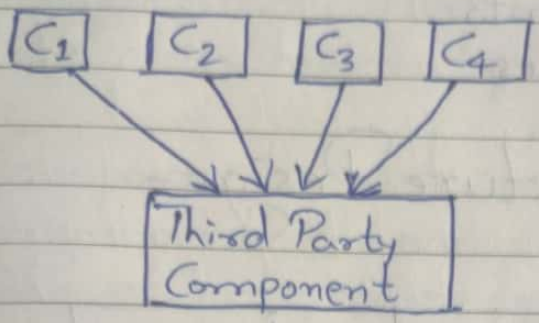


Fig 1.1

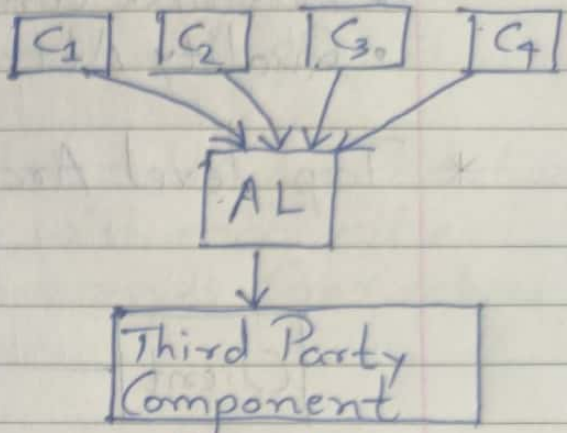


Fig 1.2

In Fig 1.1. Four Components are directly dependent on a third party component. If some changes are made on the interface of the third party component, changes will need to be made on the four components as well.

In Fig 1.2.
Only the abstraction layer (AL) is directly dependent on the third party component. If changes are made in third party component Only AL will be replaced.

* Non-functional Requirements:
  - Technical Constraints
  - Business Constraints
  - Quality Attributes

* Top level Architecture Design-

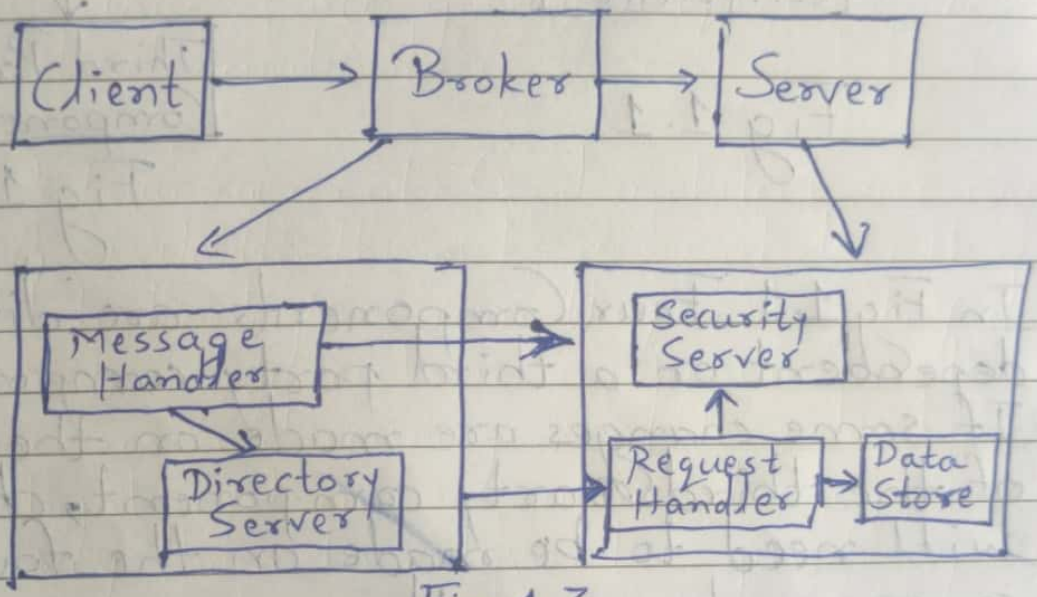


Fig. 1.3.

* Architecture Views:

- Logical views - Structure of the application.
- Process view - Concurrency and communication elements of architecture. Synchronous/Asynchronous or multithreaded/replicant
- Physical view - Depicts how major processes and components are mapped to hardware.
- Development view - Internal organization of software components. Class hierarchy/nested packages.

Views and Beyond Approach:

- Module: Structural view of architecture. Like class hierarchy, nested packages, inheritance
- Components and connectors: How architectural components communicate with each other. Like objects, threads or processes. Connectors like sockets, middleware.
- Allocation: View shows how architectural processes are connected to hardware.

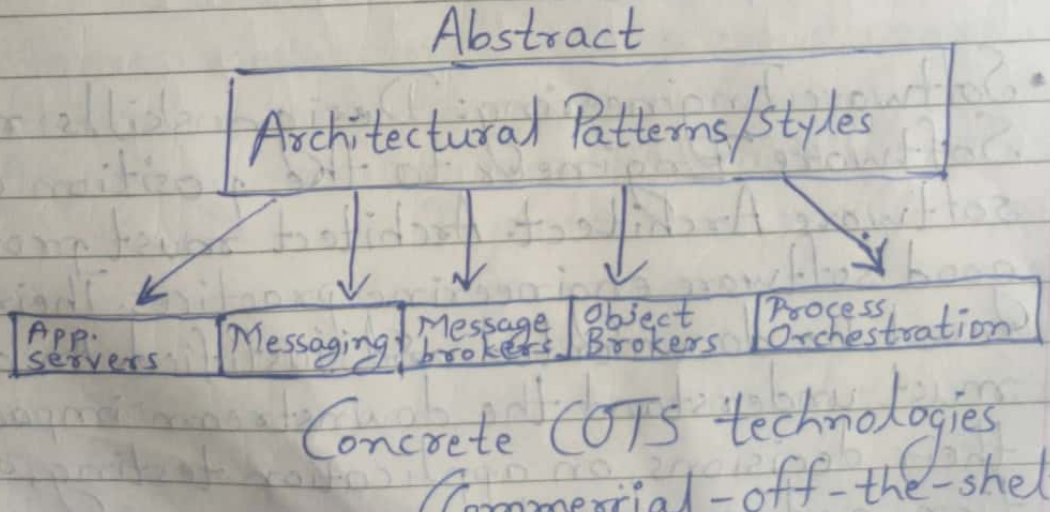
* Four essential skills of a Software Architect:

- Liaison: Architect acts as a liaise between customers or clients of the application. They communicate with various engineering teams in a project. They liaise with salesforce, and help to promote a system to potential purchasers/investors.
- Software Engineering: Design skills make a Software Engineer to the position of a software Architect. Architect must promote good software engineering practices. Their designs must be documented and justified. An architect must understand the downstream impact of their decisions, on application testing, documentation and release teams.

- Technology Knowledge: Architects have a deep understanding on the applications they work on. They are influential in evaluating and understanding third party components. They track technology developments and products and see to that how it can be exploited in their projects. They know what they don't know and ask others with greater expertise when they need information.

- Risk Management: Good architects are cautious. They evaluate the risks associated with design and technology choices they make. They document and manage risks in conjunction with project sponsors and management. They develop and instigate risk minimization strategies and convey them to relevant engineering teams.

* Mapping between logical architectural patterns and concrete technologies:



* Architect Title Soup:

- Chief Architect: A senior position who manages a team of architects. Very experienced and deep understanding of technical knowledge.
- Solutions Architect: Manages and oversees the design of a particular system/application.
- Enterprise Architect: Less technical, more business-focused. They use business methods to understand, document and design major systems of the enterprise.

Software Quality Attributes.

* Quality Attributes-
Quality Attribute requirements are part of an application's nonfunctional requirements which capture the many facets of how the functional requirements of an application are achieved.

* Performance-
A performance quality requirement defines a metric that states the amount of work an application must perform in a given time/ deadlines that must be met for correct operation.
Performance usually manifests in the following measures:

1) Throughput:
Throughput is the measure of the amount of work that should be performed in unit time. Work is typically measured in transaction per second (tps) or messages per second (mps)

2) Response Time-
Response time is the measure of time an application takes to respond to an input.

3) Deadlines-
Any application that has a limited time of window to complete has a performance deadline requirement.

* Scalability-
It can be defined as how well a solution to the problem will work if load increases.

1) Request Load:-
In a perfect world without additional hardware, as load increases application throughput should remain constant and response time per request should increase linearly.

2) Simultaneous Connections:-
An architecture must support concurrent users and scale up as more number of users grow.

3) Data Size:
Architecture should be able to upscale if the size of data/messages grows significantly.

4) Deployment:
An ideal solution would provide automated mechanisms that can dynamically deploy and configure an application.

* Modifiability -
It is a measure of how easy it is to make change to an application to cater for new functional and non-functional requirements.

* Security -

- Authentication: Identify users and applications
- Authorization: Users have defined access rights like read/write.
- Encryption: The messages sent to/from applications are encrypted.
- Integrity: Contents of message are not altered.
- Non-repudiation: Sender should have proof of delivery and receiver should know sender's identity.

* Availability:
Availability is related to an application's reliability. If an application is not available when required, then its functional requirements are not used.

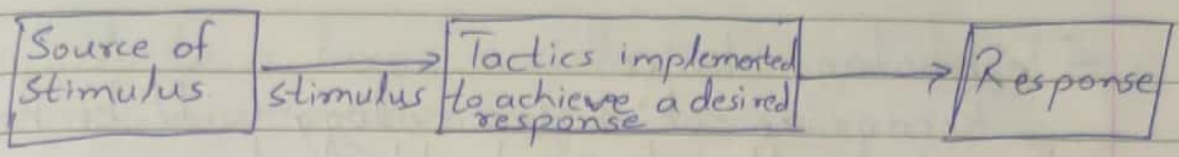
* Interoperability:
It refers to an application's/system's ability to seamlessly communicate between different ~~users~~ processes without any need of end-user

* Testability:
Degree to which module/subsystem can be verified as satisfactory or not.

* Usability:
Usability refers to how easily a user can perform the tasks safely, efficiently and effectively on the system.

* Quality Attribute Scenario:

* Architectural Tactics-
To achieve a quality one needs to take a design decision - called Tactic.
Collection of such tactics is called as Architectural Strategy



* To address a quality following 7 decisions need to be taken-

- Allocation of responsibilities:
Identify features necessary for the quality attributes. Identify modules and runtime elements necessary for quality requirements

- Coordination-
stateless/stateful, lossless/concurrent, communication of elements.

- Data Model -
Data Structure, creation, it's use, persistence destruction mechanism, metadata, data organization.

- Resource Management-
Identify resources (I/o, memory, cpu, battery)
Arbitration policy
Impact of what happens when threshold increases.

- Binding time decision
Parametrized files, runtime protocol negotiation, Runtime binding of new devices, plugins/apps.

- Technology choice -
Recognize which technology will be helpful to achieve the desired attribute.

* Business qualities -

1) Time to Market
2) Cost and Benefit
3) Projected lifetime of the system
4) Targeted Market
5) Rollout Schedule
6) Integration with Legacy System.

* Architectural qualities -
1) Conceptual Integrity - Unify design at all levels
2) Correctness and completes - System and runtime constraints should be met
3) Build ability - Open to changes/modification.

* Usability Tactics:

- Cancel - When the user cancels, system must clean memory, release resources.

- Undo - System should maintain a history of earlier states which can be restored.

- Pause/resume - Should implement a mechanism to stop a running activity

- Aggregate - An operation should be able to

apply to a large number of objects (like changing font of a paragraph)

Quality Attribute Seenario-
(Pls refer the fig. on page 8).

QAS has six parts-

Stimulus- Describes an event that arrives at a system or project. For e.g. a user operation or a security attack.

Stimulus source- Stimulus must come from a source (A human, computer system)

Response- Response is the activity that occurs as the result of the arrival of stimulus.

Response measure- When response occurs, it should be measurable so that it can be tested.

Environment- Environment is a set of circumstances in which the scenario takes place. Like a runtime operation, overload condition or a normal operation.

Artifact- The stimulus arrives at a target. It may be a project, system or a subsystem.

* Availability General Scenario:

Source – Internal/external, people, hardware, software, etc.

Stimulus – Fault: Omission, crash

Artifact – Processors, storage, comm. channels

Environment – Normal operation, startup/shutdown

Response – log, recover, ~~operate~~ the fault, operate in degraded mode.

Response Measure – Availability percentage, Time to recover the fault, etc.

Tactics for Availability:

When a failure occurs, the service is no longer available. These availability tactics will help prevent system faults:

1) Detect faults: Monitor the state of health of the system. Use ping/echo to determine the fault. Detect any exceptions and check with multiple duplicate systems to compare failure results with the failed system.

2) Fault recovery: Handle exceptions, use duplicate components if the primary component fails, rollback to a good state, try upgrading your

software or downgrade to achieve the working state, provide a restart to the failed component.

3) Prevent faults:
Remove the failed component from service. Design the system to resolve faults by introducing exception classes. Keep a predictive model and monitor the system for any faults.

* Modifiability General Scenario:

Source - End-user, developer, system administrator,

Stimulus - To add/delete/modify functionality, product, service

Artifact - Code data, interfaces, components, configuration, test cases.

Environment - Runtime, Compile time, design time.

Response - Make, test and deploy modification

Response measure - Effort, Complexity, cost, network traffic.

Tactics:
1) Increase Cohesion: Cohesion refers to how strongly the responsibilities of a module are related. High cohesion is like a class that does a

well-defined job. low cohesion is when a class does a lot of jobs but has nothing in common. Group common responsibilities in a module for increased cohesion.

2) Reduce Coupling:
Use encapsulation, abstraction and interfaces to reduce coupling. A loosely-coupled system is good for modifiability as a change in one module will lower the chances of making a change in another module.

3) Defer binding: An automated way of modification rather than hand-coding will reduce the number of human errors and will increase system flexibility.

* Performance General Scenario:

Source-
User request, Request from external system

Stimulus-
Periodic, stochastic or sporadic event occurs to a system.

Artifact-
Whole System, component within a system.

Environment- Runtime, normal, emergency, error correction mode.

Response - System returns a response, error, consumes resources.

Response Measure - Latency, number of satisfied/unsatisfied requests.

Tactics for Performance -

The goal of performance tactics is to generate a response to events arriving at the system under time-based or resource-based constraint. Unavailability of a resource, dependency on other computation can increase latency. Reducing processing time by underutilizing resources can increase performance.

Control Resource Demand:

One way to increase performance is to control the demand of resources.

- Manage Work requests - Reduce number of requests coming to the system. Ways to do that include:

1) Manage event arrival - Using SLA (Service Level Agreements). SLA means system/component will process X events arriving per unit time with a response time of Y.

2) Manage sampling rate -

When system cannot maintain adequate response levels, reduce the sampling frequency of the stimuli - rate at which data is received from sensor/video frames per second processed.

- Limit event response -
  When discrete events arrive at a system too rapidly to be processed, then the events must be queued. If the events are discarded, make sure to log the events if necessary.

- Prioritize events -
  Rank events according to the importance. Drop events that are low priority if required. For e.g. for a building management system. Life-threatening and fire alarms are more important than informational alarms like room is too cold.

- Reduce indirection - Use of interfaces/intermediaries (which is important for modifiability) increases computing power requirement, so removing intermediaries improves latency. This is a classic modifiability/performance tradeoff.

- Co-locating communicating resources -
  Hosting cooperating components on the same processor to avoid the time delay of network communication. It also means putting the resources in same runtime software component.

- Periodic clearing - A periodic cleanup of inefficient resources reduces computational overhead.

- Bound execution times - Place a limit on how much execution time is used to respond to an event. Limit the number of iterations is one method.

- Increase efficiency - Improve efficiency of algorithms to improve throughput and resource consumption.

Introducing concurrency (process parallel work requests), maintain multiple copies of computation and data can be some of the tactics used for managing resources.

* Security General Scenario:
Security is the measure of system's ability to protect data from unauthorized access while providing access to people and systems that are authorized.

Source - Attack from human, system, internal external, unknown.

Stimulus - Unauthorized attempt to display, capture, change, delete data. Change system's behaviour, reduce availability.

Artifact - System services, components/resources

Environment - System maybe online/offline, behind a firewall or open network, operational

Response - Data is protected from unauthorized access. Services will be available only for legitimate use, recording attempts to modify/ access data.

Response measure - Resource compromised, Accuracy of detection of attack, time passed after detection, amount of data vulnerable to attack.

Security tactics -

- Ensure physical security - limit access to resources like servers, cpus at the stored location.
- Detect attacks - Lookout for intrusion, identify actors, authorize them, Encrypt data, limit exposure and access, validate input and seperate entities.
- Recover from attack -
  Revoke access of attacker, revoke login and inform affected actors, keep a record of system and data.
- Non-repudiation - This tactic guarantees that sender sent the message and receiver received it. None of the acting parties can deny their action.

* Designing and documenting architecture:

What is Architecturally Significant Requirements (ASR)?
ASR is a requirement that will have a profound effect on architecture.

When does the architect start?
The architect doesn't wait until the requirements finish before starting the work. The architect begins while the requirements are still in flux.

Sniffing out ASRs from a requirements document -

- Usage - User roles
- Physical elements
- Security
- Technologies.

Gathering ASRs by interviewing stakeholders-
Interviewing relevant stakeholders is the surest way to learn what they know and need. It behooves a project to capture critical information in a systematic, clear and repeatable way. One method is the QAW (Quality Attribute Workshop)

The Quality Attribute Workshop -
The QAW is a facilitated, stakeholder focused method to generate, prioritize and

refine QA scenarios. The QAW is keenly dependent on the participation of system stakeholders.

The QAW involves the following elements:

- Business/mission presentation - The stakeholder represents the business concern behind the system.
- Architectural plan presentation - While a detailed system/software architecture does not exist, broad system descriptions, context diagrams can describe system's technical details.
- Identification of architectural drivers - The facilatators will share a key list of architectural drivers that they assembled.
- Scenario brainstorming - Each stakeholder represents a scenario expressing his/her concerns with respect to a system.
- Scenario consolidation - After the scenario brainstorming, similar scenarios are consolidated.
- Scenario prioritization - Prioritization is accomplished by allocating each stakeholders a number of votes to any scenario.
- Scenario refinement - After prioritization, top scenarios are refined and elaborated.

* Gathering ASRs by understanding business goals -
  - Business goals often lead to quality attribute requirements.
  - Business goals may affect the architecture without inducing a quality attribute requirement.
  - No influence of a business goal on architecture.

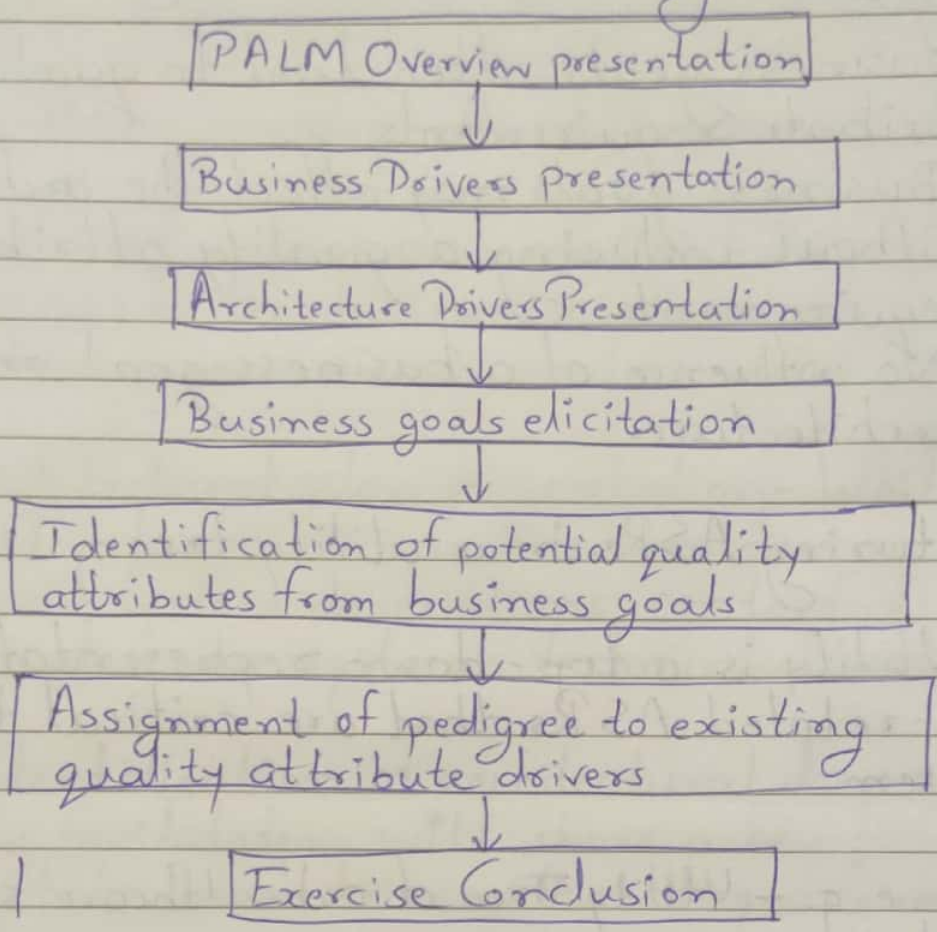
* Capturing ASRs in a utility tree -

Utility is a top-down representation of QA-related ASRs that are critical to the system.

For e.g. - Utility Tree of a healthcare space system

Qualty Attr.	Attr. Refinement	ASR scenario
Performance	• Transaction Response	Transaction completes in less than 0.75 seconds.
	• Throughput	At peak load, system completes 150tps
Usability	• Proficiency training	A new hire can execute system functions in less than 5 seconds
	• Efficiency of operations	Payment officer initiates a payment plan for a patient

* PALM method for eliciting business goals



⊛ Introduction To Agile Methodology:

Agile is an iterative approach to project management and software development that helps team deliver faster to their customers.

Advantages:
- Collaborative
- Interactive and feedback oriented
- Iterative
- Test Driven

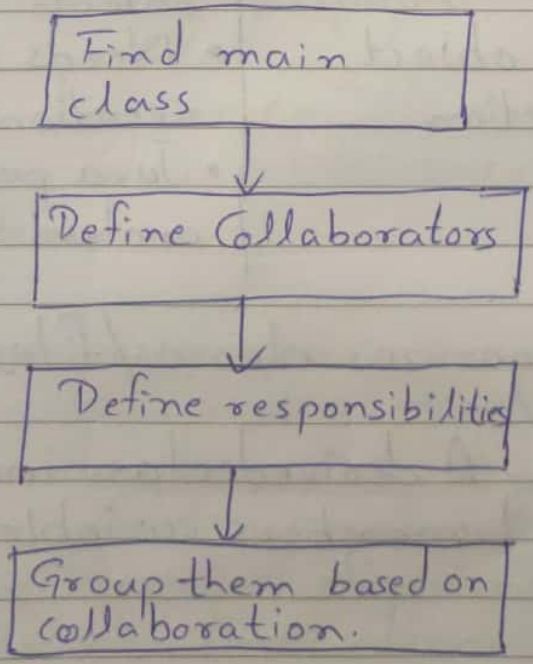
7 Discipline Overview-

- Model
- Implementation
- Test
- Deployment
- Config Management
- Project Management
- Environment

* Class Responsibilities and Collaborators Card (CRC)-

Class Name	Collaborators
Responsibilities assigned to this class	If this class cannot fulfill its functionality on its own then which other classes to collaborate

* How to create CRC model?



Introduction to OO -

Class represents an abstract, user-defined type.
Object is an instance of a class.

Classes are user-defined datatypes

- Primitive types - int, float, char, boolean
- Unified type -
  C# keyword object
  java.lang.object

References and values -

Value	Reference
• Primitive types are accessed by value	• Java allows a variable to have reference to an object.
• No explicit object creation/deletion required	• Needs explicit object creation.
	• Java performs escape analysis for faster allocation.

Modules - Organizes classes/files

Inheritance - A derived class inherits the methods and members variables of the Base Class.

Interfaces-
A published declaration of a set of services. Provides capability for unrelated classes to implement a set of common methods.

⊛ Introduction to UML-
(Unified Modeling Language) Modeling language for specifying, constructing, visualizing and documenting software system and its components.

UML supports 2 types of models
- Static
- Dynamic

UML diagrams can be classified into 2 types-

Structure diagrams emphasize the things that must be present in system - extensively used for documenting purposes.

Behaviour diagrams - Illustrate behaviour of system, they are used to describe the functionality of the system.

* Structural Diagrams-
- Class Diagram - System class, attributes
- Component diagram - A system comprising of components and dependencies.
- Composite Structure diagram - decomposition of a class into finer elements.
- Deployment diagram - describes hardware, environments and artifacts.

- Object diagram - a complete or partial view of structure at a specific time.
- Package diagram - describes how a system is broken into logical groupings.
- Profile diagram: operates at metamodel level.

* Behavioural diagrams -

- Activity diagram - describes business and operational flow.
- State Machine diagram - describes states and state transitions.
- Use Case Diagram - Describes functionality provided by system.

* Interactions (Behaviour model)

- Communication diagram: Shows interaction between objects/parts in term of sequenced messages.
- Sequence diagram: Interaction among objects through sequence of messages.

* Different types of association -

1. Strongest - (Composition) Implies total ownership. If owner is destroyed its parts are also destroyed.

2. Aggregation-
Has a part ownership

3. Weakest (Association) -
General form of dependency
Based on usage.

⊛ Rational Unified Process (RAP) -
It is a software-oriented process for object-oriented models.

* Five main phases
- Inception (Define project scope)
- Elaboration (Architecture and Design)
- Construction (Actual Implementation)
- Transition (Initial release)
- Production (Actual deployment)

* Iterative Development-
Business value is delivered incrementally in timebox iterations.

* Feature Driven Design (FDD)
Agile framework that organizes software development around features.

Starts during Inception, Elaboration phase.

* From Feature to Architecture-
- Once set of features are collected
- Similar features are grouped together into a module.

* Use-Case Model-
A model based on user's point of view (POV).

To create use-case-
- Define actors
- Describe scenarios
  - Preconditions
  - Main tasks
  - Exceptions
  - Interaction with users

* Module views-
- External entities (devices that consume info from system)
- Things
  - Report, letters, signals.
- Structure
  - Sensors, four-wheeled car
- Event Occurences
  - Transfer of fund, Completion of job.
- Roles
  - People who interact with system (Manager, Engineer, etc.) Actors.

⊛ Designing and ~~doo~~ documenting Architecture (Part II)

* 3 Uses for Architecture Documentation-
- Education (For new members of team, new architect)

- Communication among stakeholders
- Basis for system analysis and construction.

* Notations-
Informal, formal, Semiformal
~~Formal - general purpose diagrams~~
~~Informal - Rudimentary Analysis, UML~~
Semiformal

Informal - General-purpose diagrams
Semiformal - Rudimentary analysis, UML
Formal - Precise semantics, Architecture Description languages (ADL)

* Views - let us divide software architecture into interesting and manageable representations.

Types of views-
1) Module views
2) Allocation views
3) C&C views
4) Quality views - Security view, Communications view, Performance view, reliability view.

5) Combining view - Minimum 1 module, allocation and C&C

* Method for choosing a view-
Step 1 - Build a Stakeholder view
Step 2 - Combine views
Step 3 - Prioritize and stage.

* Document a view:
1. Primary presentation - A basic view of the system.
2. Element Catalog - Details of elements from primary presentation.
3. Context Diagram
4. Variability guide - Isolate ~~diffe~~ variation points
5. Rationale - Why the particular view was made.

* Document information beyond views:
1. List issuing organization
2. version set
3. Date of issue, etc.
4. System overview
5. Information provided in the document
6. Mapping between views.
7. Rationale
8. Directory - Acronyms, glossary

* Layered Architecture -

Presentation layer - Responsible for handling user interfaces and browser comm. logic

Business layer - For executing specific business rules with the request

~~Persis~~ Service layer - Gives a set of available operation with each request like accessing, updating and deleting data.

Data layer- A layer of your website where all the data from the requests are stored.

* Architecture Evaluation-
It is the process of determining the degree to which an architecture is fit for the purpose it is intended.

Architecture Trade off Analysis Method (ATAM)
The ATAM requires participation of 3-groups-

- The evaluation team
- Project decision makers - Project Managers.
- Architecture stakeholders - Developers, testers, maintainers.