UNIT-2

Software Project Management

Software projects are temporary individual or collaborative enterprises that are carefully planned to achieve a particular aim/or to create a unique product or service

- Large projects have lots of people working together for prolonged time
- Coordination to ensure integration and interoperability
- Business and Environment change frequently and rapidly

Project Management will need to maintain this triangle in equilibrium

Characteristics of a project

- Made up of unique activities which do not repeat
- Goal specific
- Sequence of activities to deliver end-product
- Time bound
- Inter-related activities
- Need adequate resources (time, manpower ,finance, knowledge-bank)
- Intangible; can claim 90% completion without visible outcomes

SOFTWARE PROJECT MANAGEMENT	PROJECT MANAGEMENT
Art and discipline of planning and supervising software projects	It facilitates the project workflow with team collaboration on a single project
Includes activities towards planning, execution of these plans, monitoring and controlling the project	Includes planning, organizing, motivating, controlling the resources to achieve specific project goals, monitoring, risk management, managing quality & managing people performance

Project Management Lifecycle

Initiation:

- **Purpose:** This is the starting point of the project where the idea is conceived, and the project is defined at a broad level.
- **Activities:**
  - **Project Charter:** Creating a document that formally authorizes the existence of the project and provides the project manager with authority.
  - **Stakeholder Identification:** Identifying all individuals or groups that may be affected by or have an impact on the project.
  - **Feasibility Study:** Assessing the viability of the project in terms of technical, financial, legal, and operational aspects.
- **Outcome:** The project is officially initiated with a clear understanding of its purpose, objectives, and stakeholders.

Planning:

- **Purpose:** In this phase, detailed planning is conducted to define the scope, time, cost, quality, communication, risk, and resource management plans.
- **Outcome:** A comprehensive project plan that serves as a roadmap for the project team.

Project Perspective

- Sponser's perspective
- Stakeholder's perspective
- customer's perspective

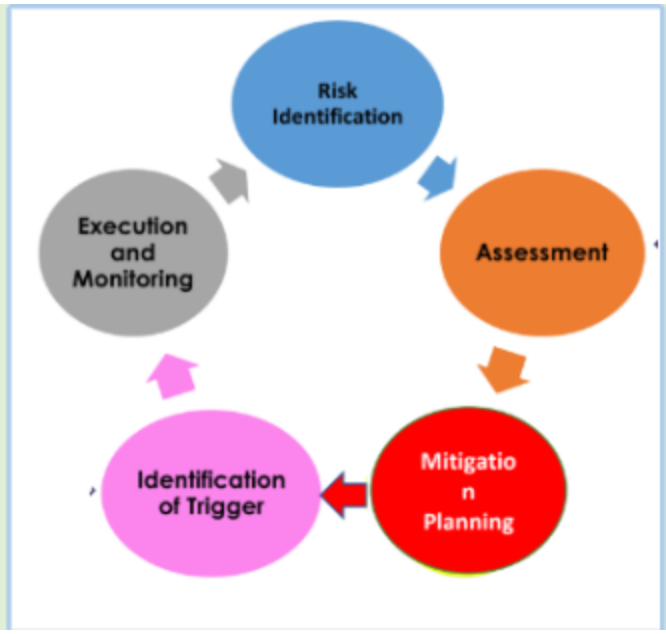
Sponsors	Customer	Execution Stakeholder
Projects role in organisation	Team understands problem?	Software Lifecycle to follow
Address customer requirement	Time to develop	Project Organisation (roles)
Investment vs Revenue	Cost of Solution	Standards, Guidelines, Procedures
Time expenditure and Risks	Delivery plans	Communication Mechanism
Responsibility and Progress Tracking	Metrics to indicate quality	Criteria to Prioritise requirements
Resources and Deliverables	Exit Criteria	Work breakdown and Ownership
Exit Criteria	Project support	
	Interaction/Review plans	

Steps in Project Planning

1. **Developing a quality management process**
   - Plan for progress tracking of the project
   - Communication plans
   - Quality Assurance plans
   - Test completion criteria
   - Early verification or customer validation

2. **Plans for tracking Project Plan & Delivery plan**
   - Plan for management of Project Plan
   - Procedures for release of product to customer
   - Staff and people management
   - Performance management
   - Compensation and benefits management

3. **Risk management**

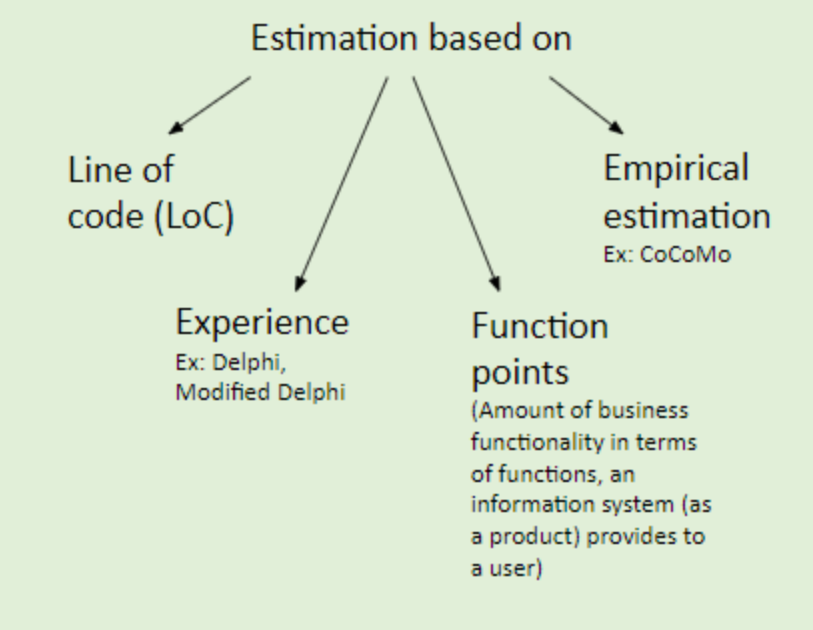


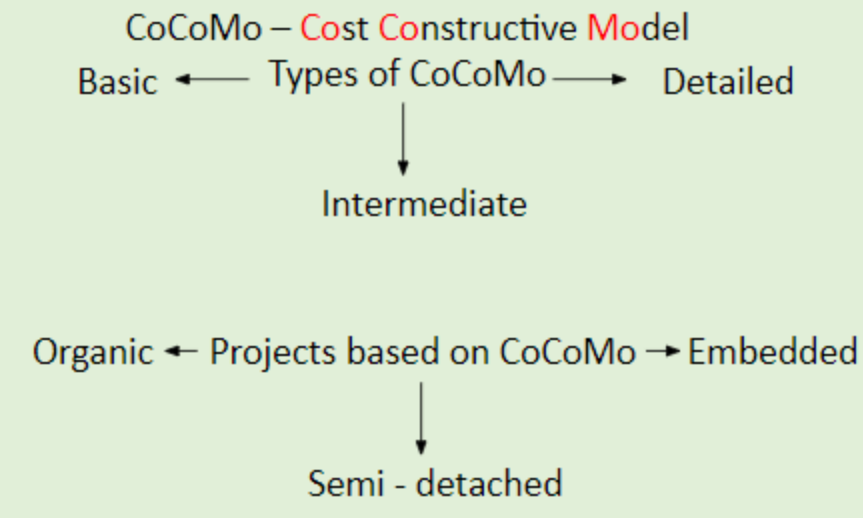
4. **Scheduling & allocating resources**

   Calendarization of work activities:

   - Bring concerned individuals to participate in forming the schedule
   - Identification and allocation of resources according to WBS
   - Validation of upstream and downstream dependencies for tasks
   - Organizing tasks concurrently
   - Usage of tools such as Gantt chart for visualization
   - Identification of risks
   - Minimize task dependencies
   - Account for working conditions
   - Costing

5. **Estimation of the Tasks/Activities**





Estimation could be

•

Experience based: expert judgement, comparative studies

- Delphi, Modified Delphi
  • Empirical estimation: Formula derived from past projects (size, process char, experience)
- CoCoMo (Constructive Cost Model) formula based
  • Three categories of projects

    - Organic: Small team, problem well understood, experienced people
    - Embedded: Large team, complex problem, need people with sufficient experience
    - Semi-detached: In between
  - Types of CoCoMo
    - Basic: rough calculations
    - Intermediate: considers cost drivers
    - Detailed: factors dependencies

6. **Organize the project**
   - Organizational Structure in terms of people, team & responsibilities
   - Identification of eco-system partners for the Project
   - Upstream and Downstream partners

7. **Work Break Down – Leads to WBS**

   What is Work Break Down?

   Decomposition of the project activities into deliverable smaller components & is an iterative process

   Split project into tasks and activities/ subprojects (Level 1). Break these down further into detailed tasks (Level 2). Now, each task can be meaningfully tracked.

   These levels are part of the *Work Breakdown Structure* (hierarchical structure).

   Tasks → Phases → Work packages for product

8. **Understanding the expected deliverables of the project**

- Expectations of stakeholders in terms of the Deliverables
- The market forces driving the project

- Looking at the high level decisions of Make-Buy-Reuse
- Supported by the results of the feasibility study and the requirements elicited

9) Planning the process

- Choice of lifecycle is based on: activities, goals, time and so on
- Degree of certainty (high or low) of product, process, resource

ROLES

- Project Management Office (PMO)
- Project Manager

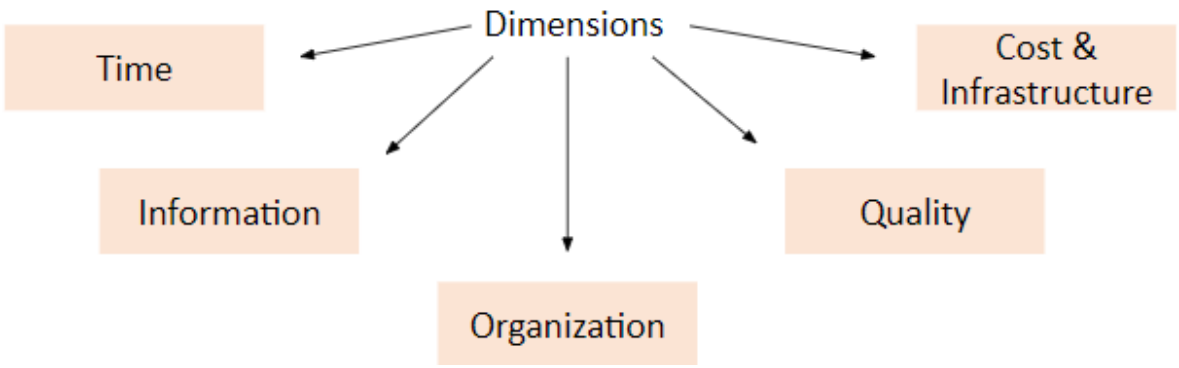
Monitoring and Execution:

Project Monitoring and Control includes those processes performed to observe project execution so that potential problems can be identified in a timely manner and corrective action can be taken, when necessary, to control the execution of the project

Project control activities involves making decisions or adjustments in dimensions like time, cost, organizational structure, scope etc. for the project under execution

- Monitoring and controlling project work
- Collects, measures and disseminates performance information, assess trends to identify potential items requiring corrective action or monitoring for risks.
- This can result in corrective or preventive actions and request for changes
- Periodically do Critical Path Analysis
- Ensuring that all of the change controls are being meticulously followed
- Ensuring that all of the scope, deliverables, documents are periodically updated with the project plans.
- Controlling the Quality triangle

- Managing the project team, performance management and communication management



💡 Quality → **Quality is not an add-on feature; it has to be built in**

💡 Organization → **Involves structure, roles & responsibilities from people and team aspects**

💡 Information → **Includes availability, propagation, communication and documentation**

💡 Time → **In terms of effort (number of man-months) and the schedule**

Brooks law: adding people to a late project delays it further

💡 Infrastructure → **Includes personnel, capital and expenses**

Closing:

- **Purpose:** The final phase involves formally closing the project, completing any remaining deliverables, obtaining customer or stakeholder acceptance, and

releasing project resources.

Close project and report success to sponsor

- Handover deliverables to customer and obtain project/UAT (User acceptance testing) sign-off from client
- Complete and pass on documentation; cancel supplier contracts
- Release staf, equipment; inform stakeholders
- Post mortem: determine projects success and lessons learned
- **Outcome:** A formal closure of the project with a complete handover of deliverables and documentation.

Software Architecture

It is the top level decomposition into major components together with a characterization of how these components interact.

Software architecture of a computing system can be:

- Used for communication among stakeholders
- a blueprint for the system and project under development
- Used for negotiations and balancing of functional and quality goals happen during architecture

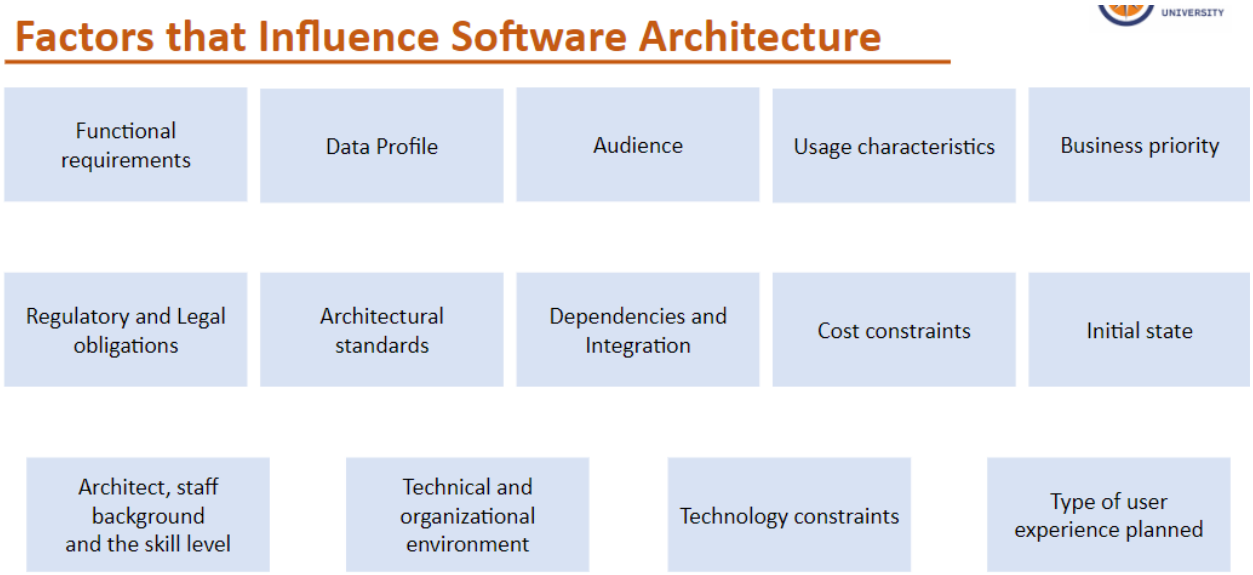
Architecture manifests the earliest set of design decisions

- Constraints on implementation
- Dictates organizational structure
- Inhibits or enable quality attributes and supports WBS

Characterstics

- Address variety of stakeholder perspective
- Realizes all of the use cases and scenarios envisaged for the problem
- Supports seperation of concerns

- Quality Driven
- Recurring Styles
- Conceptual integrity



Software Architect

Software Architect is a role within the software development organization structure, who makes high level design choices and dictates technical standards, including software coding standards, tools and platforms

He should have

- Distinct role in Project
- Broad training & Extensive experience
- Deep understanding of domain

Architectural View

Views represent ways of describing the Software Architecture, enabling the system to be viewed by different stakeholders in perspectives of their interest. *Ex: UI View, Process View*

Architectural Styles

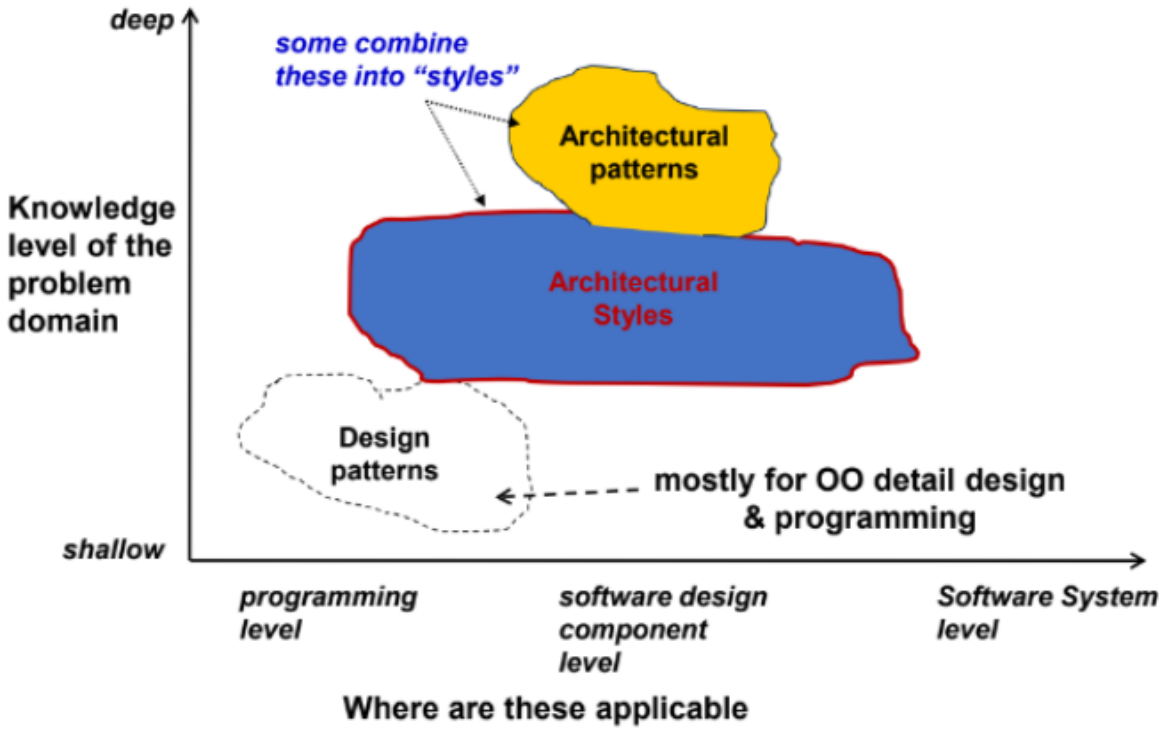
Demonstrates how the subsystems or elements are organized or structured.

It is a way of organizing code. *Ex: Pipe & filters, Client-Server, Peer-to-Peer*

Architectural Pattern

Is a known or a proven solution to the architectural problem of structuring and functioning of the subsystems which has been used earlier and is know to work for the problem scenario. *Ex: MVC separating UI from the rest*

Combined View



Types of Architecture

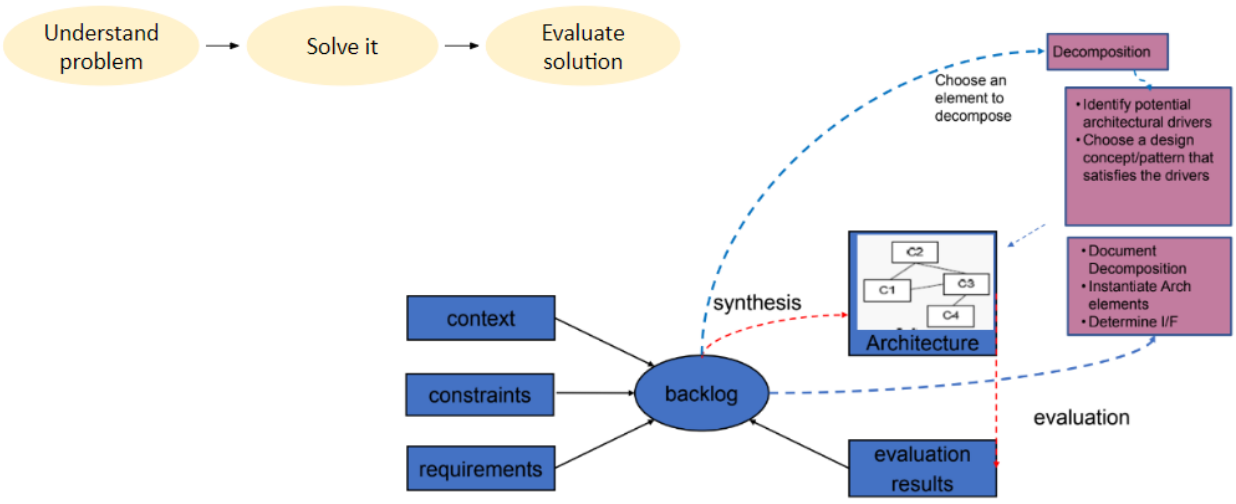
CENTRALIZED ARCHITECTURE

Central office hardware (PSTN switch)
Benefits:
- Works through power outage
- Reliability

PEER – TO – PEER ARCHITECTURE

Peer to peer
Benefits:
- Scales without changes
- Features can be added easily

Generalized Model of Architecture

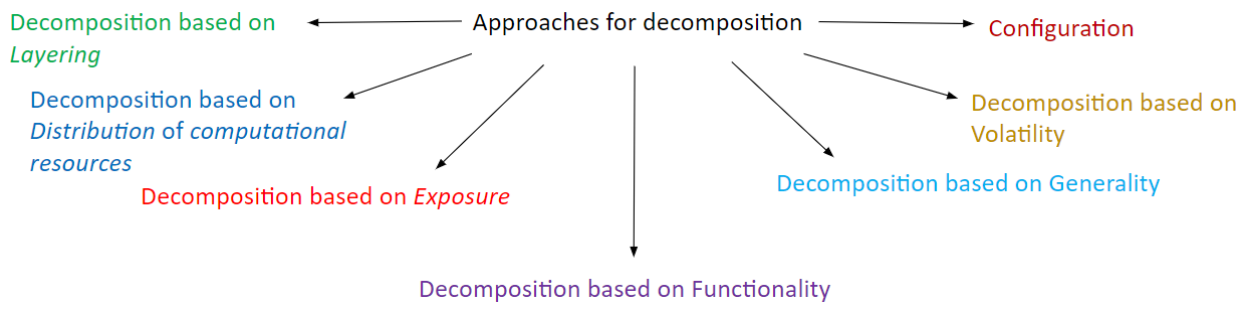


Thin Client : Depends on server to do all the work

Fat client : High performance expensive and powerful hardware

Architectural Decomposition

Architectural decomposition is a process in software engineering and system design where a complex system or software application is broken down into smaller, more manageable components or modules



Based on layering

- Order system into layers; each layer consumers service from lower layer and provides service to higher layers
- Ordering of abstractions
- Eg: TCP/IP model

Based on distribution (computational resource)

- Dedicated task owns thread of control; process need not wait
- Many clients need access
- Greater fault isolation

Based on exposure

- How is the component exposed and consumes other components
- Service offered, logic and integration

Based on functionality

- Grouping with problem domain and separate based on functions
  - Eg: login module, customer module, so on
- Mindset of operational process

Based on generality

- Components which can be used in other places as well

Based on Volatility

- Identify parts which may change and keep them together (UI)

Based on Configuration

- Look at target for features needing to support different configurations
- Eg: security, performance, usability

Based on Coupling:

- keeping things together; Cohesion: things that work together
- Low coupling and high cohesion (good software always obeys this

Other approaches

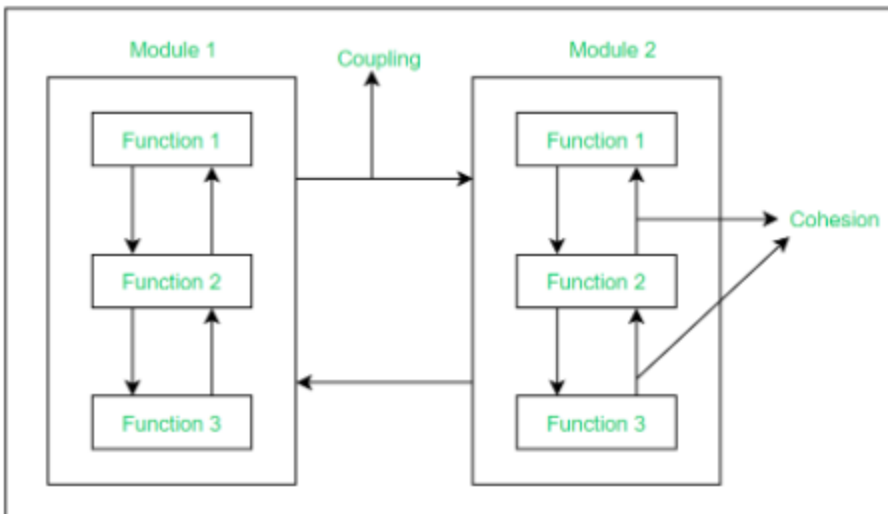
- Divide and conquer
- Stepwise refinement: simple solution and enhance
- Top-down approach: overview of system, detail the subsystems
- Bottom-up approach: specify individual elements and compose
- Information hiding

Decomposition Types

- Functional
- Structural
- Data
- Object Oriented
- Process
- Module
- Temporal
- Hirearchical

Architectural Views

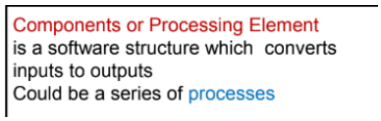
1. Module Viewpoint



- Structure the system as a set of code units (modules)
- An Architect enumerates what the units of software will have to do and assigns each item to a module.
- The larger modules may be decomposed to sub-modules of the acceptable fine level.
- It is often used as the basis for the development project's organization and deliverables like documentation.

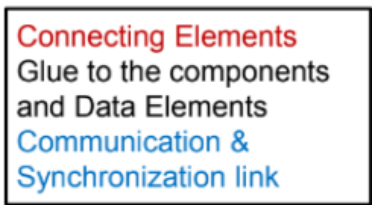
2. Component – and – Connector View Point

Component 1

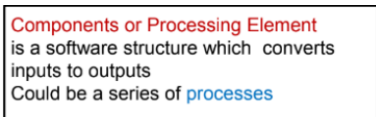


- A component is a modular unit or building block within the system.

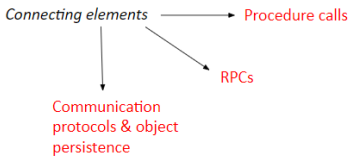
Connector



Component 2

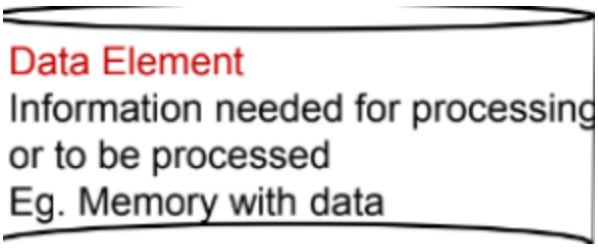


- It encapsulates a set of related functionalities and may have well-defined interfaces through which it interacts with other components.
- Components can be implemented in various ways, such as classes in object-oriented programming or services in a service-oriented architecture.



- A connector represents the means by which components interact or communicate with each other.
- Connectors define the relationships and communication patterns between components.
- Examples of connectors include method calls, message passing, shared memory, or network protocols.

Data Element

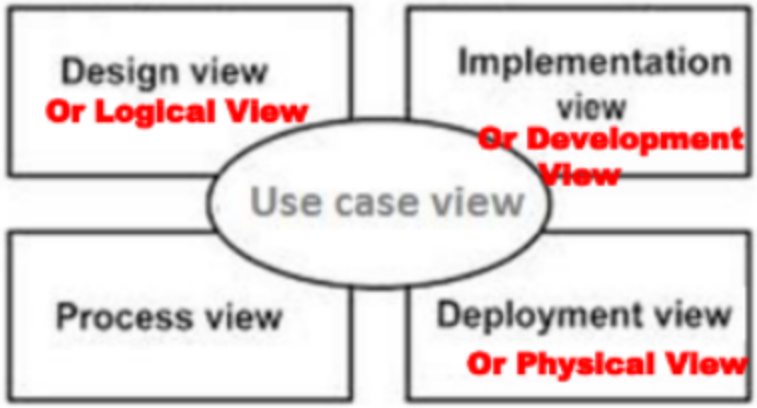


3. Allocation View Point

The "Allocation" viewpoint is a perspective in system architecture that focuses on the mapping or assignment of software components or modules to various elements within the system.

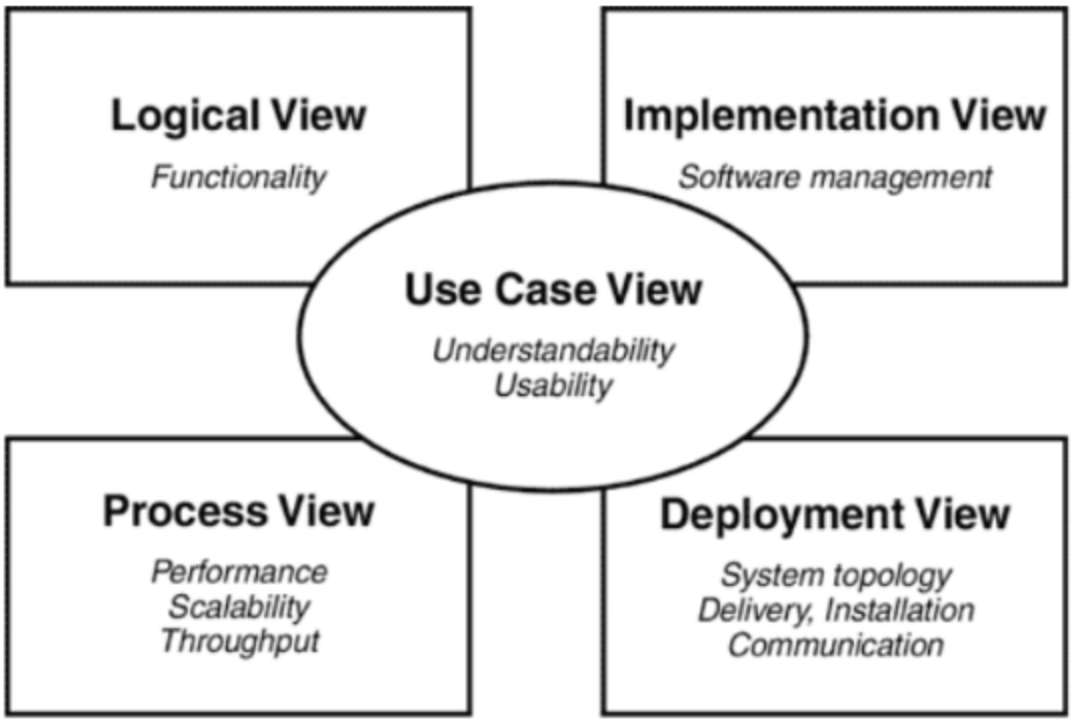
1. **Deployment Structure:**
    - **Objective:** This view shows how software components are assigned to hardware elements and the communication paths used between them.
    - **Relevance:** It is particularly important in distributed or parallel systems where software may run on multiple servers or nodes.
    - **Considerations:** Helps engineers reason about performance, data integrity, availability, and security in the context of the system's deployment.
2. **Implementation Structure:**
    - **Objective:** Indicates how software is mapped onto file structures in the system's development, integration, or configuration control environments.
    - **Relevance:** Useful during the development and integration phases to understand how source code, binaries, and other development artifacts are organized in the file system.
    - **Considerations:** Helps in managing version control, configuration management, and the organization of code and related files.
3. **Work Assignment Structure:**
    - **Objective:** Shows who is responsible for performing specific tasks or activities in the project.
    - **Relevance:** Essential for project management, helping to allocate tasks to individuals or teams and ensuring that the right expertise is available where needed.
    - **Considerations:** Facilitates effective project planning, coordination, and monitoring of progress.

4. Krutchens View Point



1. **Use Case View:**
    - **Objective:** Exposes the functional requirements of the system and scenarios that describe how users interact with the system.
    - **Representation:** Typically uses use case diagrams to show actors, use cases, and their relationships.
    - **Relevance:** Provides a high-level understanding of system functionality from a user's perspective.
2. **Design View:**
    - **Objective:** Exposes the vocabulary of both the problem space (requirements) and the solution space (design).
    - **Representation:** Utilizes various UML diagrams, including class diagrams, sequence diagrams, and other design artifacts.
    - **Relevance:** Helps in understanding the system's static structure and dynamic behavior, capturing how components interact and collaborate.
3. **Process View:**
    - **Objective:** Encompasses the dynamic aspects and runtime behavior of the system, focusing on threads and processes.
    - **Representation:** Involves process diagrams, showing the flow of control and communication between different system processes.
    - **Relevance:** Addresses performance, concurrency, and other runtime characteristics.

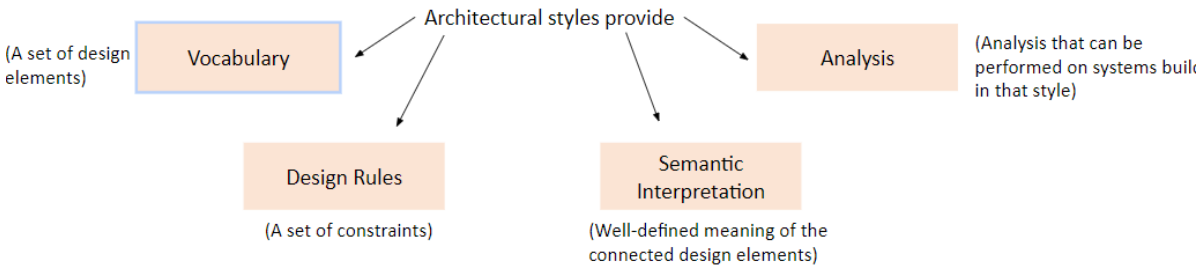
4. **Implementation View:**
    - **Objective:** Addresses the realization of the system in terms of code and software components.
    - **Representation:** Uses UML diagrams like package diagrams, class diagrams, and other artifacts to show how the system is structured in terms of implementation.
    - **Relevance:** Helps developers understand how the design is translated into actual code and components.
5. **Deployment View:**
    - **Objective:** Focuses on system engineering issues related to deployment, including hardware infrastructure and network configuration.
    - **Representation:** Involves deployment diagrams, illustrating the physical arrangement of hardware and software components.
    - **Relevance:** Aids in understanding how the system is physically distributed, addressing issues related to scalability, performance, and reliability.



Architectural Style

Way of organisation of components, characterised by features that make it notable

- Used to construct software modules
- Structure and behaviour of the system



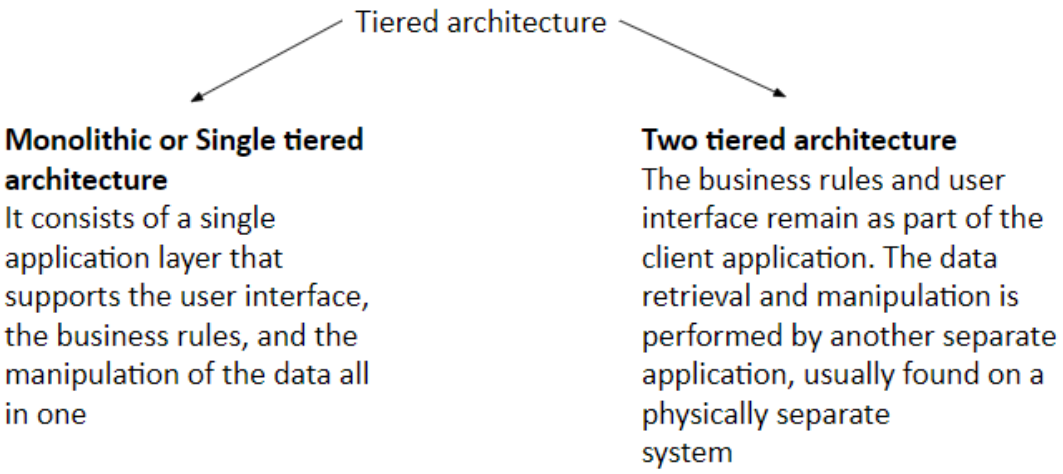
- Vocabulary: set of design elements (Pipes, filters, client, severs,)
- Design rules: constraints that dictates how processing elements would be connected
- Semantic interpretation: meaning of connected design elements
- Analysis: performed on the system (Deadlock detection, scheduling)

Architectural Pattern

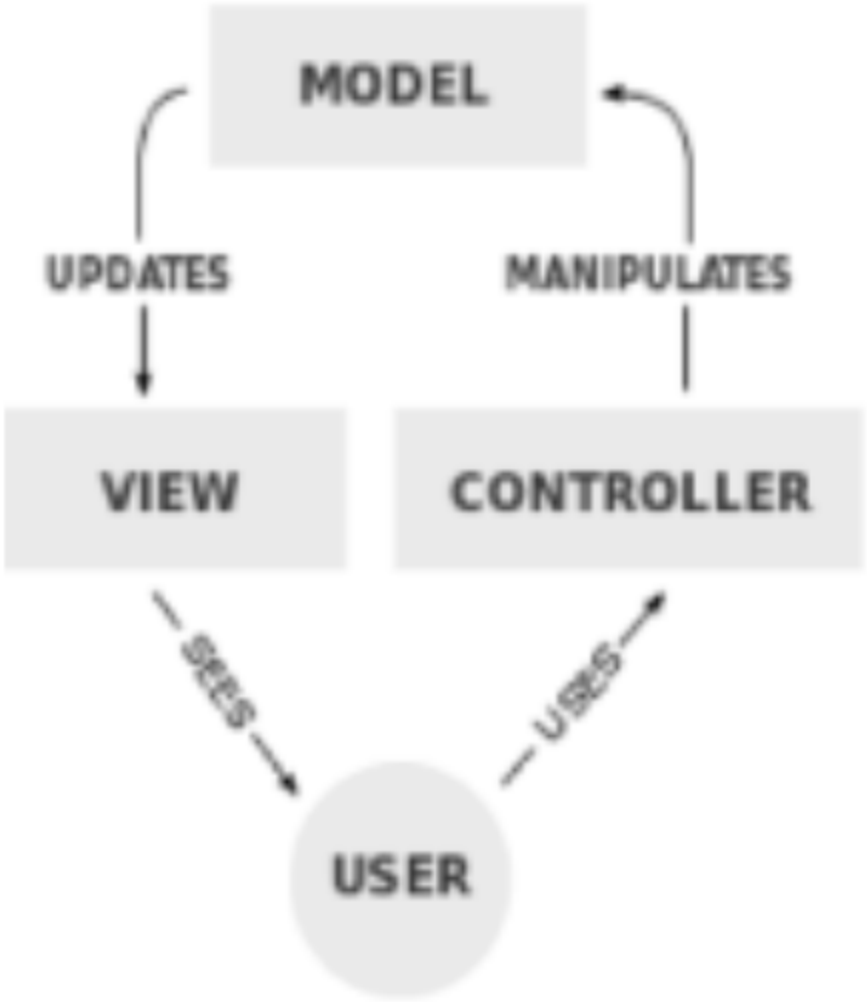
An Architectural Pattern is a proven potential solution of structuring, functioning to a recurring architectural problem. They are typically discovered.

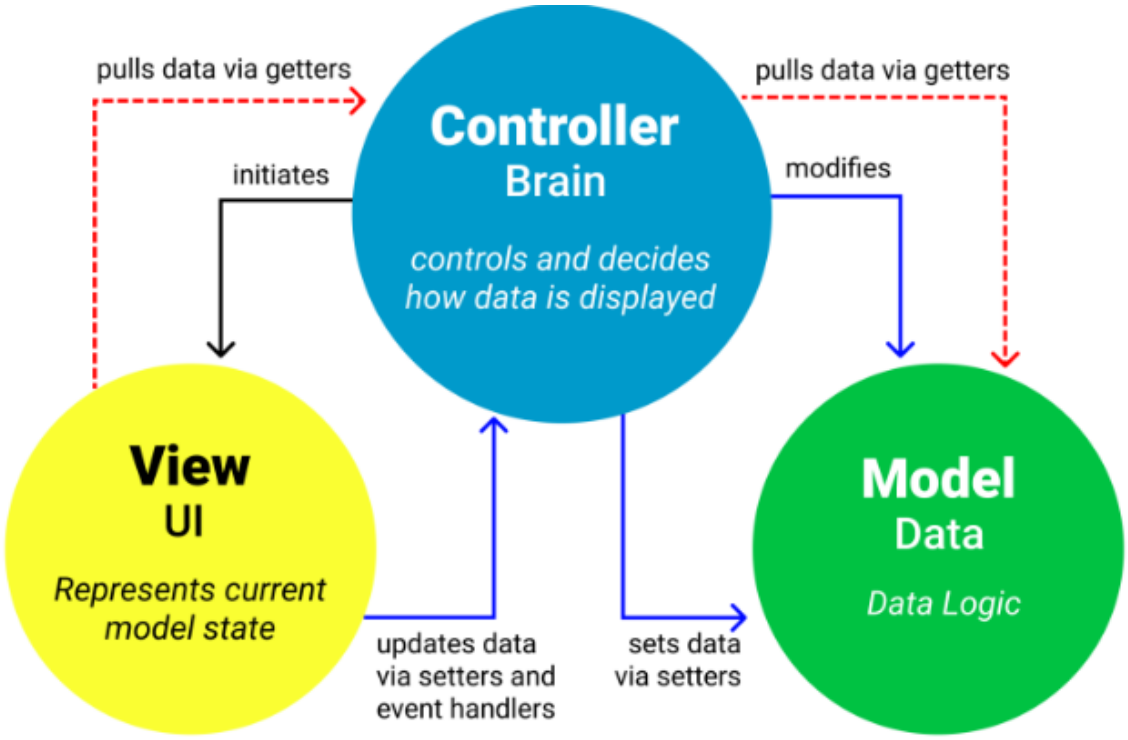
An architectural pattern is a named collection of architectural design decisions that

- Has resulted in a successful solution in a given development context
- Constrain architectural design decisions that are specific to a particular system within that context
- Elicit beneficial qualities in each resulting system
- It is a good starting point solution to the problem



Model View Controller


MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER



Model (CPU)

It is the central component and consists of application data, business rules, logic and functions and is the abstraction layer for features

View (GUI)

View is the graphics design and layout and get information from the model as prompted by the controller

Controller (wire)

Accepts input and converts it to commands for the model or view

Use cases

- UI logic tends to change more frequently than business logic
- App needs to display same data in different ways
- Developing UI and Business logic require different skill sets; separate development teams easily

- UI code is more device-dependent than business logic

Importance of Architertural Style and Pattern

BENEFITS

Reuse of design and code components
Ease of understanding the architecture
Increased interoperability

FIRST DESIGN ARTEFACT

Performance, reliability, modifiability, security

REPRESENTS EARLIEST DESIGN DECISIONS

Hardest to modify
Most critical to get right
Communication between various stake holders

KEY TO SYSTEMATIC REUSE

Transferable, reusable abstraction

Software Design

Design principles:
• Further decomposition post architecture If necessary
• Description of behaviour of components/sub-systems as identified in Architecture design
• How interfaces will be realised (data structures+algorithms)
• System will facilitate interaction with user
• Use of apt structural and behavioural design patterns
• Maintenance and reuse

Techniques in Design

1. **ABSTRACTION : Focus on essential properties**
2. **MODULARITY : Degree or the extent to which the larger module can be decomposed**
3. **COHESION : Extent to which the component/modules are dependent on/fit into or are related to each other, when trying to address a specific responsibility.**
4. **COUPLING : Coupling indicates how strongly the modules are connected to other modules.**

COHESION

- Adhoc (Lowest)
- Logical (input routines)
- Temporal (initialisation sequence)
- Sequential
- Procedural (read and print)
- Functional (contribute to same function) (highest)

COUPLING

- Content: one component directly impacts another (Highest)
- Common: two components share overall constraints
- External: components communicate through external medium
- Control: one component controls the other (passes info)
- Stamp: complete data structures are passed
- Data: only one type of interaction (lowest)

(Strong cohesion is good) & (Loose coupling is good)

5. **INFORMATION HIDING : Design involves a series of decisions and each such decision, need to consider who needs to know & who can be kept in the dark.**
    - **Encapsulation :** A information hiding strategy which hides data and allows access to the data only through specific functions.
    - **Separation of interface and implementation :** An information hiding strategy which involves defining a component by specifying a public interface (known to the clients) but separating the details of how the component is actually realized.

6. **LIMITING COMPLEXITY : Complexity refers to the amount of effort required for building its solution.**

- Intra – modular (within a module)
- Inter – modular (between modules)
- Higher value ⇒ Higher complexity ⇒ Higher effort required (= worse design)

7. **HIERARCHICAL STRUCTURE : Views whole structure as a hierarchy**

Key Issues in Design

Issue	Description
Concurrency	- **Challenge:** Parallel execution of more than one program. - **Concerns:** Deadlocks and race conditions.
Event Handling	- **Challenge:** Managing messages sent between objects.
Distribution of Components	- **Challenge:** Components distributed in a network. - **Considerations:** Middleware support, communication breakdown.
Non-functional Requirement	- **Impact:** May have system-wide implications.
Error, Exception Handling, Fault Tolerance	- **Mistakes:** Divert program execution or create incorrect results. - **Exceptions:** Could lead to termination. - **Faults:** Should not lead to errors causing system failures. - **Approaches:** Fault avoidance, fault detection and removal.
Interaction and Presentation	- **Objective:** React to user input effectively and efficiently.

Data Persistence	- **Objective:** Storage of information between executions.

Architecture vs Design

Architecture	Design
Bigger picture; frameworks, tools, languages, scope, goals, etc	Smaller picture; local constraints, design patterns, programming idioms, code org
Strategy, structure and purpose	Implementation and practice
Software components, visible properties and relationships	Problem solving and planning for internal software
Harder to change	Simpler and have less impact
Influence non-functional requirements	Influence functional requirements

Design Methods

Detailed design methods support us in decomposing the components representing the system requirement into components/subcomponents well. Example: DFD, Booch, Fusion

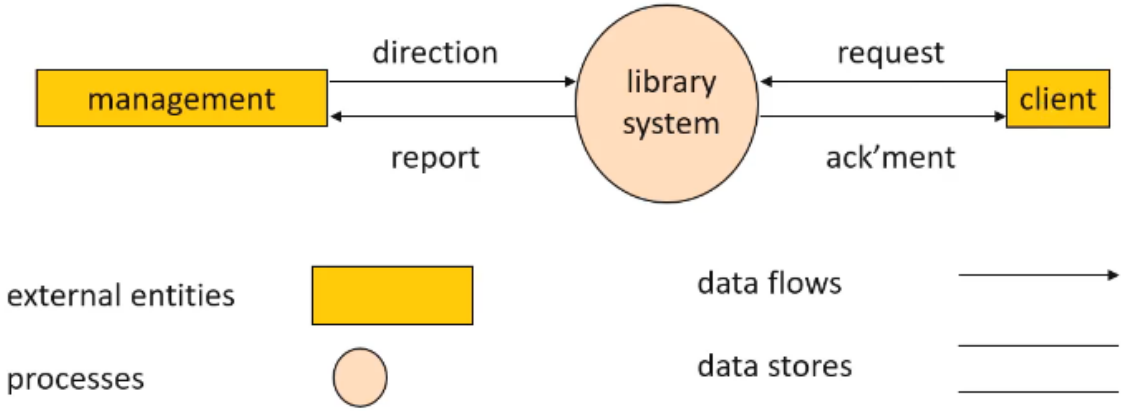
- Structured Analysis (SA), resulting in a logical design, drawn as a set of data flow diagrams
- Structured Design (SD) transforming the logical design into a program structure drawn as a set of structure charts. Heuristics based on coupling and cohesion

Data flow diagram

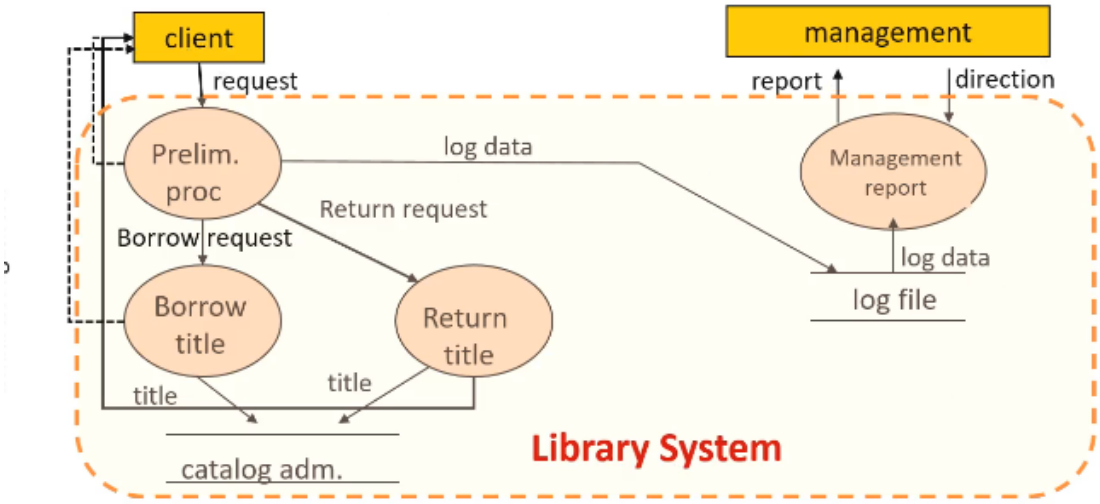
- **External entity - Source and Destination of a transaction - Depicted as squares**
- **Processes - Transforms the data - Depicted as circles**

- **Data Stores - These lie between processes & are places where data structures reside - Depicted as two parallel lines**
- **Data Flows - Paths where data structures travel between processes and entities and data stores - Depicted as arrows**

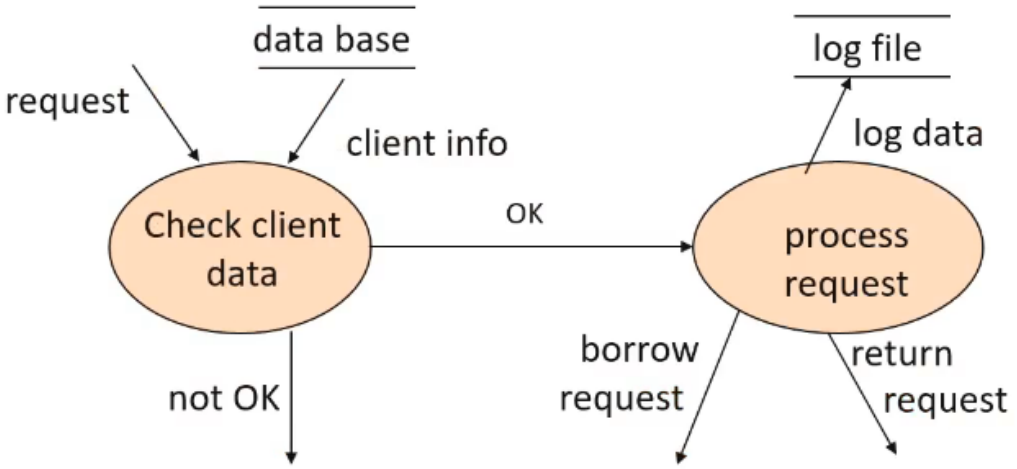
Top Level Decomposition



First Level Decomposition



Second Level Decomposition

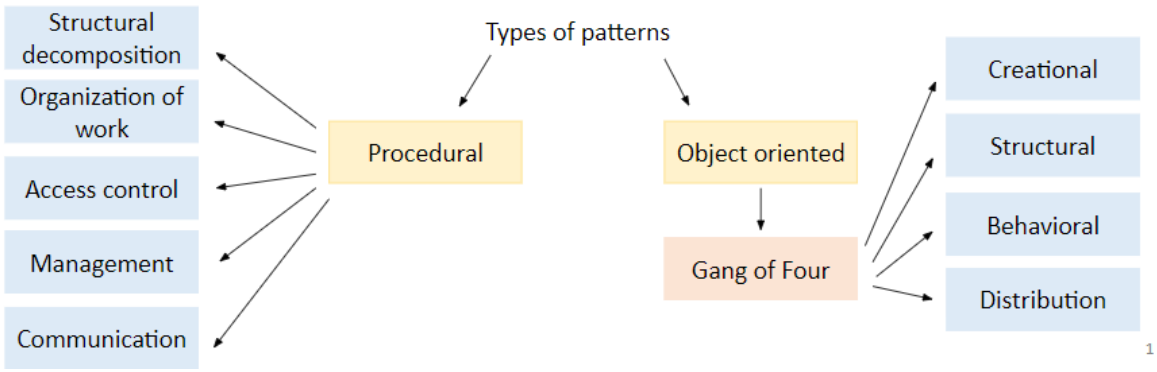


Minispecs: process in DFDs become sufficiently straight forward and doesn't warrant further decompositions

Data dictionary: contents of DFDs after we are at logical decomposed state
• Precise description of the structure of data

Design Pattern

- Design pattern provides solution to a recurring problem
- Provides abstraction above the level of a single component
- Provides common vocabulary and understanding for design principles
- Design patterns are a means of documentation both descriptive and prescriptive



Procedural Pattern (SOAM)

Structural decomposition pattern	Breaks down a large system into subsystems and complex components into co-operating parts, such as ***a product breakdown structure***
Organization of work pattern	defines how components work together to solve a problem, such as ***master-slave and peer-to-peer***
Access control pattern	describes how access to services and components is controlled, such as through a ***proxy***
Management pattern	defines how to handle homogeneous collections in their entirety, such as a ***command processor and view handler***
Communication pattern	defines how to organize communication among components, such as a ***forwarder-receiver, dispatcher-server, and publisher-subscriber***

Object Oriented Pattern

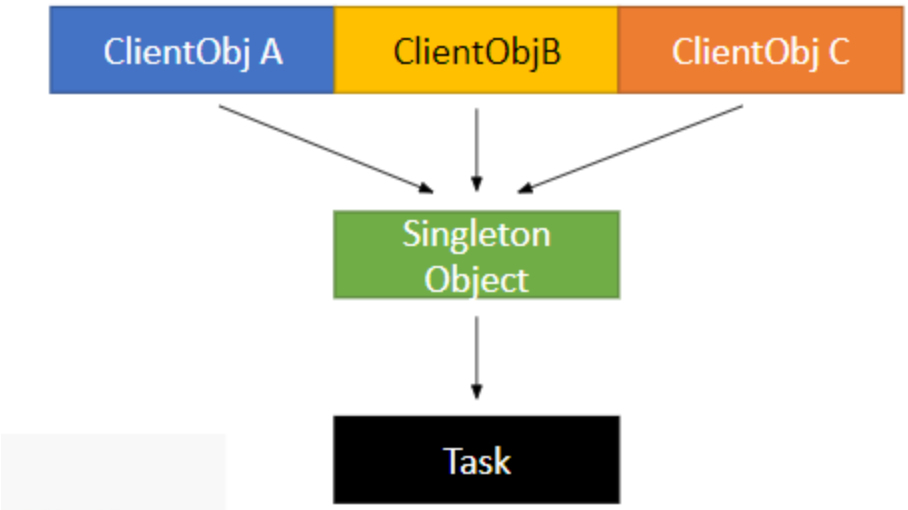
Gang of Four solutions

- Creational patterns that focus on creation of objects (Singleton, builder)
- Structural patterns that deal with composition (Adapter, Bridge)
- Behavioural pattern that describe interaction (Command, interpreter)
- Distribution patterns that deal with interface for distributed systems

Singleton Pattern

Intent: only one instance of class is created; global access point to object
Motivation: one object to coordinate actions across the system

- One class responsible to instantiate itself
- Global point of access to the instance
  - Eg: centralised management of global resources



Anti Pattern

Describes situations a developer should avoid
In agile approaches, refactoring is applied when anti pattern is introduced

God class: class that holds most responsibilities

Lava flow: dead code which gets carried forward indefinitely

Comparison Factor	Structural Approach	Object Oriented approach
Abstraction	Basic abstractions are real world functions, processes and procedures	Basic abstraction are not real world functions but data representing real world entities
Lifecycles	Uses SDLC methodology	Incremental or Iterative methods
Function	Grouped together, hierarchically	Functions grouped based on data
State information	In centralised shared memory	State information distributed among objects
Approach	Top Down	Bottom up approach
Begin basis	Considering use case diagram and scenarios	Begins by identifying objects and classes
Decompose	Function level decomposition	Class level decomposition
Design approaches	Use Data flow diagram, structured English, ER diagram, Data dictionary, Decision tree/table	Class, component and deployment for static design; Interaction and State for dynamic
Design techniques	Design enabling techniques need to be implemented	Communicates with objects via message passing and has design enabling techniques
Design Implementation	Functions are described and called; data isn't encapsulated	Components have attributes and functions; class acts as blueprint
Ease of development	Easier; depends on size	Depends on experience of team and complexity of program
Use	Computation sensitive apps	Evolving systems that mimic a business or business case

Software Oriented Architecture (SOA)

- **Make software components reusable via service interfaces**
- Utilise common communication standards and can be rapidly incorporated into new applications without deep integration
- Each service embodies code and data integrations to execute a complete, discrete business function;
  provide loose coupling

- Exposed using standard network protocols (SOAP - Simple object access protocol/HTTP or JSON/HTTP
) to send requests to read or change data
- Structured collections of discrete software modules that collectively provide
  functionality

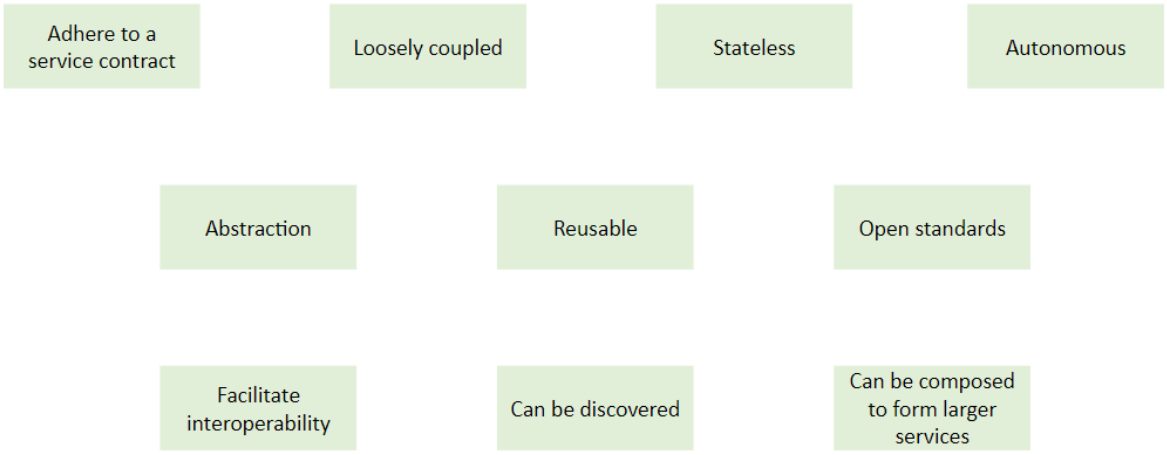
Benefits

- Greater business agility, faster time to market
- Ability to leverage legacy functionality in new markets
- Improved collaboration between business and IT
- Service reusability

Services

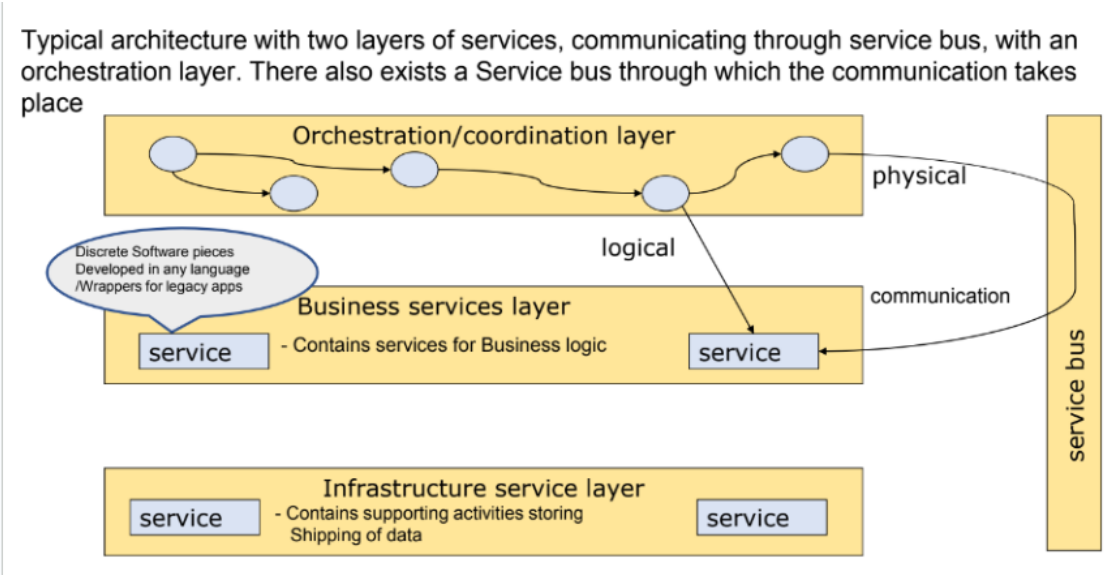
- It is a logical representation of a repeatable business activity that has a specified outcome (ex: check customer credit, provide weather data, consolidate drilling reports)
- These could be implemented as discrete pieces of software or components written in any language capable of performing a task
- These could be implemented as “callable entities” or application functionalities accessed via exchange of messages
- These could also be application functionality with wrappers which can communicate through messages

Service Characterstics



SOA roles

1. **Service provider:**
   - creates web services and provides them to a registry Responsible for terms of service
2. **Service broker/registry:**
   - responsible for providing information about the service
3. **Service requester/consumer:**
   - finds a service in the broker and connects to the provider



SOA vs Microservices

SoA	Microservice
Enterprise-wide approach to architecture	Implementation strategy with app dev teams
Communicates with components using Enterprise service bus (ESB)	Communicate statelessly using APIs
Less tolerant and flexible	More tolerant and flexible