
# Software Engineering
# Unit - II

**Compiled by**
**M S Anand**

Department of Computer Science

# Software Engineering

## Introduction

**Text Book(s):**

1. “Software Engineering: Principles and Practice", Hans van Vliet, Wiley India, 3rd Edition, 2010.
2. “Software Testing – Principles and Practices”, Srinivasan Desikan and Gopalaswamy Ramesh, Pearson, 2006.

**Reference Book(s):**

1. “Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling” by By Jennifer Davis, Ryn Daniels, O’ Reilly Publications, 2018
2. “Software Engineering: A Practitioner’s Approach”, Roger S Pressman, McGraw Hill, 6th Edition 2005
3. “Software Engineering”, International Computer Science Series, Ian Somerville, Pearson Education, 9th Edition, 2009.
4. “Foundations of Software Testing ", Aditya Mathur, Pearson, 2008
5. “Software Testing, A Craftsman’s Approach ", Paul C. Jorgensen, Auerbach, 2008.
6. IEEE SWEBOK, PMBOK, BABOK and Other Sources from Internet.

# Software Engineering

## Software Project Management

All projects are a temporary effort to create value through a unique product, service or result. All projects have a beginning and an end. They have a team, a budget, a schedule and a set of expectations the team needs to meet.

Project management is the use of specific knowledge, skills, tools and techniques to deliver something of value to people.

**Software project management** is an art and science of planning and leading software projects. It is a sub-discipline of project management in which software projects are planned, implemented, monitored and controlled.

**Software Project Manager**

A software project manager

1. defines the requirements of the project,
2. builds the project team,
3. lays out a blue print for the whole project including the project scope and parameters,
4. clearly communicates the goals of the project to the team; the targets to be achieved,
5. allots budget to the various tasks to be completed, and
6. ensures that the expectations of the Board of Directors and Stakeholders are met through timely completion of the project.

# Software Engineering

## Software Project Manager – Different roles

**Leader**

- A software project manager must lead his/her team towards success.
- He/She should provide them direction and make them understand what is expected of them.
- Clearly explain the roles of each member of the team.
- He/She must build a team comprising of individuals with different skills so that each member contributes effectively to the best of their abilities.

**Liaison**

- The project manager is a link between his/her clients, his/her team and his/her own supervisors.
- He/She must coordinate and transfer all the relevant information from the clients to his/her team and report to the upper management.
- He/She should work closely with analysts, software designers and other staff members and communicate the goals of the project.
- He/She monitors the progress of the project, taking action accordingly.

# Software Project Manager – Different roles

**Mentor**

- He/She must be there to guide his/her team at every step and ensure that the team has cohesion.
- He/She provides advice to his/her team wherever they need it and points them in the right direction.

# Fundamentals of Software Project Management

Software Engineering is very relevant for large projects which involve

- Several people
- Several months
- Planning and Controlling
- Coordination
- Efficient Project Management

Good project management cannot guarantee success, but poor management on significant projects always leads to failure.

# Software Engineering

## Quality of Projects

**Main objective: i**s to achieve project goals and targets while keeping in mind the project scope, time, quality, and cost.

# Software Engineering

## Quality of Projects

The project management triangle visualizes the problem of “triple constraints”—the need to balance scope, cost, and time in order to maintain a high-quality final product.

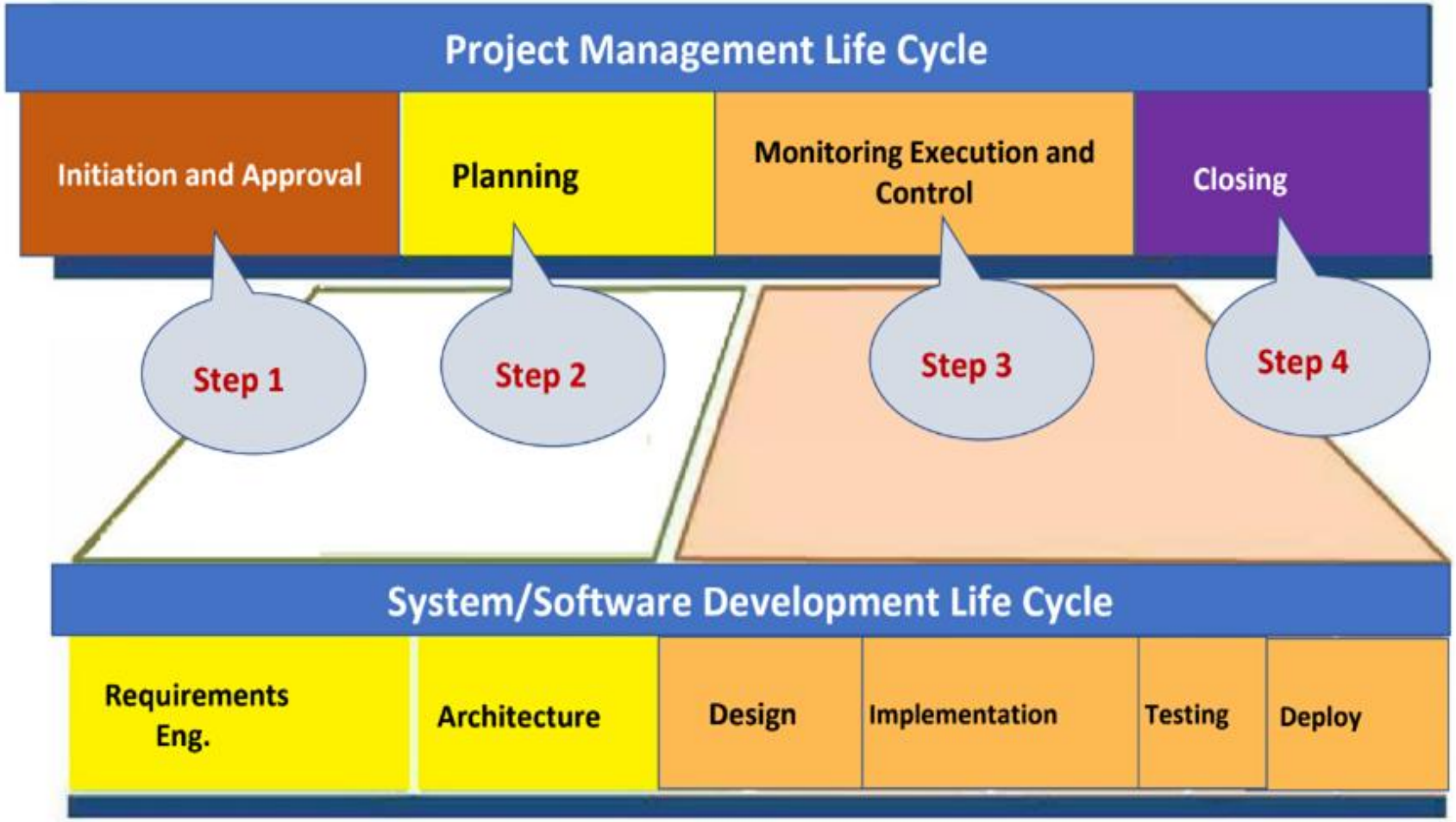
Every project manager who has known the frustration of trying to keep their team productive with too little time, not enough budget, and an impossibly large scope has experienced the project management triangle in action.

No single variable of the project triangle can be changed without making tradeoffs with the other two points of the triangle.

It’s the project manager’s job to balance all three elements in order to keep their project within budget and on deadline while still fulfilling the specifications of the project’s scope.

# Software Engineering

## Software Project Management lifecycle



# Software Engineering

## Step 1 – Initiation and Approval

**What is Project Initiation & Approval?**

It is at this point where the opportunity or reason for the project has been identified and the project is being kicked-off to take advantage of that opportunity.

**When is a project initiated?**

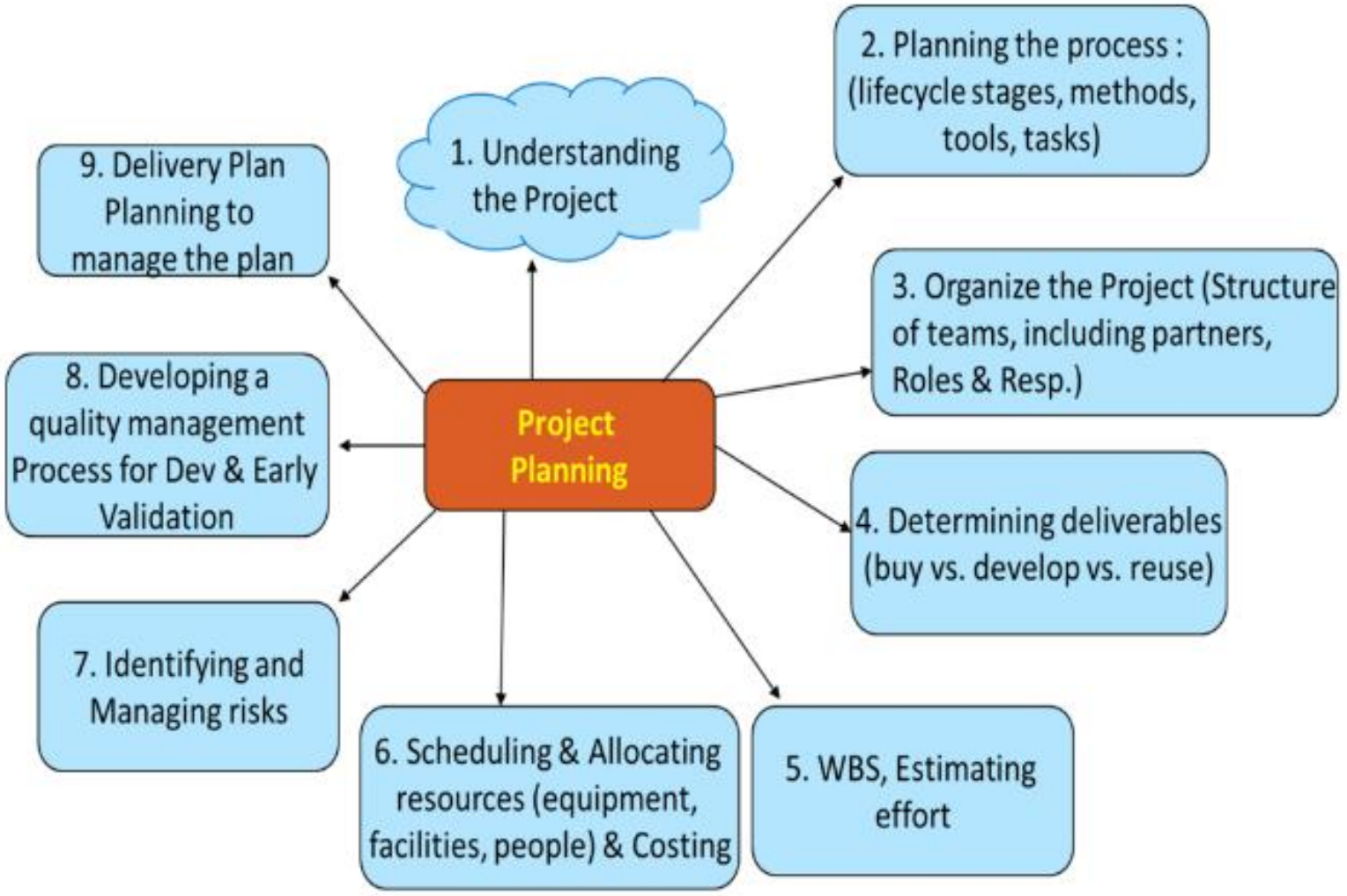
Initiation of the project is at the approval or a “go” post the *feasibility study*.

**What is done during this phase?**

- Project charter (consisting of vision, objectives, high level scope, detailed deliverables) is created
- Project charter outlines purpose, structure and execution of project.
- Detailing out the responsibilities of the project teams and the stakeholders
- Initial project owner/manager is identified
- Initial budget is identified
- Identification of resources for a more detailed plan

# Software Engineering

## Step 2 – Project Planning



# Step 3 – Project monitoring and control

**What is monitoring & control?**

The Monitoring and Controlling process is continuously performed and encompasses all the tasks and uses all the measures and metrics necessary to ensure that the project is on track

What is Project monitoring?

It involves using *quantitative data* which is continuously collected all along the project & may include events like checkpoints/milestones/toll-gates

What is Project control?

It involves making decisions or adjustments in dimensions like time, cost etc.

PROJECT MONITORING AND CONTROL BEGIN ONCE PROJECT PLAN IS CREATED

# Software Engineering

## Step 4 – Project Closure

Project Closure Phase will formally close the project and then report its overall level of success to the sponsor.

1. Hand over deliverables + User Acceptance Test sign off

2. Complete documentation

3. Post Implementation Review
   (Post Mortem)

4. Release staff and equipment + inform stakeholders of closure


PES
UNIVERSITY
ONLINE

# Software Project Planning

**This is the first task in any project**

Project Manager focuses on **what** and **how**.

Outcome – Project Plan, which depends on nature of project

Project Plan is a live document – can change over the course of the project.

# Software Engineering

## Project Planning – Different perspectives

Sponsor's perspective

Customer's perspective

Execution Stakeholder's perspective

**Execution Stakeholder's perspective**

- What Lifecycle are we going to follow
- Criteria for prioritizing of requirements
- Project Organization:
  - Relationships to other parts of the organization (Upstream/Downstream)
  - Roles of people
  - User involvement
- Standards-Guidelines-Procedures to be followed
- Communication Mechanisms
- Schedule with Detailed Work Breakdown and Ownership

# Steps in Project Planning

**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

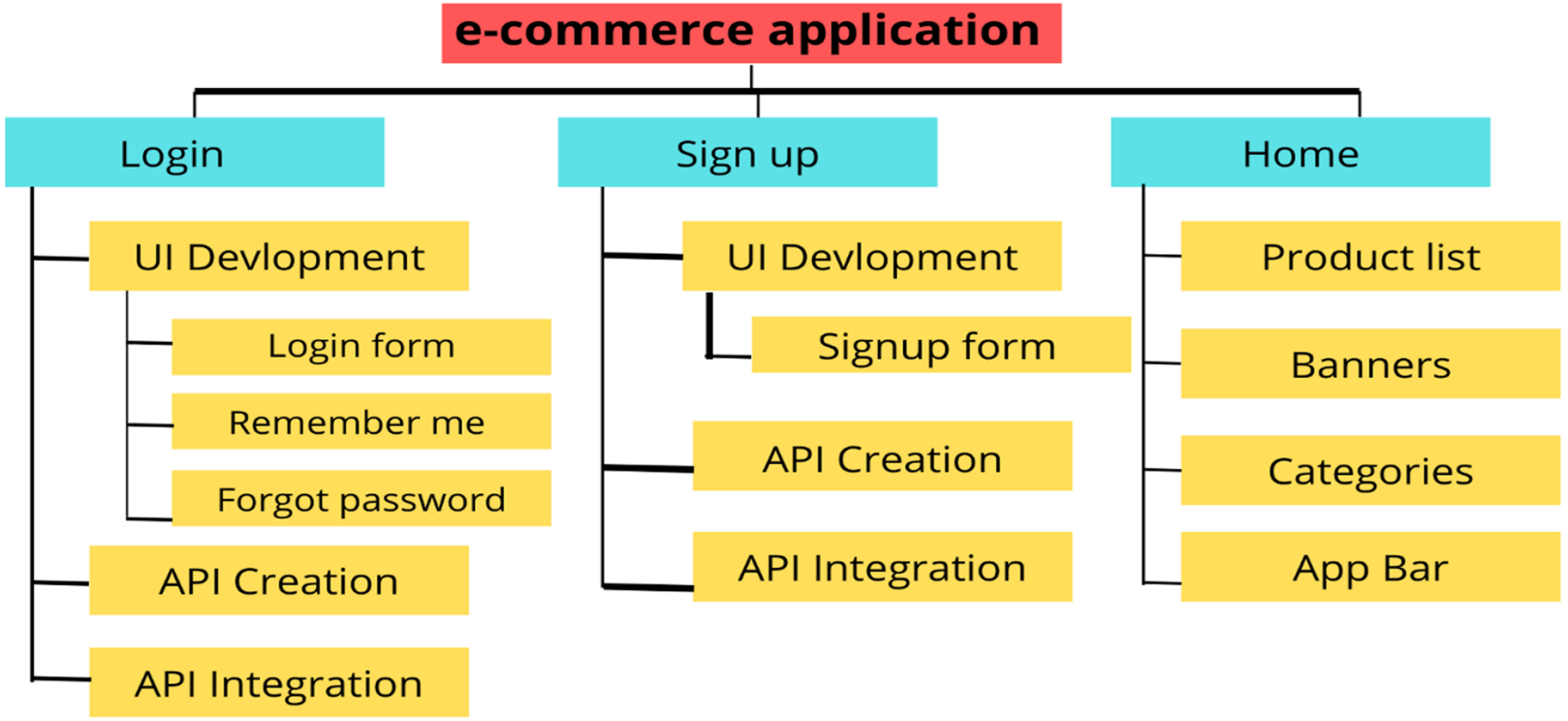
**Plans for tracking Project Plan & Delivery plan**

- Plan for management of Project Plan
- Procedures for release of product to customer
- Staff and people management
- Performance management
- Compensation and benefits management

# Software Engineering

## More about Work Breakdown Structure (WBS)

**An example WBS**



# Software Engineering

## More about WBS – Estimation approaches

CoCoMo (Constructive Cost Model)
It is a regression model based on LOC, i.e **number of Lines of Code**.

It is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality.

The key parameters which define the quality of any software products, which are also an outcome of the Cocomo are primarily Effort & Schedule:

**Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.

**Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, months.

# Software Engineering

## More about WBS – Estimation approaches

Different models of CoCoMo have been proposed to predict the cost estimation at different levels, based on the amount of accuracy and correctness required.

All of these models can be applied to a variety of projects, whose characteristics determine the value of constant to be used in subsequent calculations. These characteristics pertaining to different system types are mentioned below.

**Organic –** A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
**Semi-detached –** A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, knowledge of the various programming environment lie in between that of organic and Embedded.

## More about WBS – Estimation approaches

**Embedded –** A software project requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

There are 3 different types of CoCoMo models

1. Basic
2. Intermediate
3. Detailed

# Software Engineering

## More about WBS – Estimation approaches

Basic model

$$E = a\,(KLOC)^b$$

$$time = c\,(Effort)^d$$

$$Personsrequired = Effort / time$$

The constant values a,b,c and d for the Basic Model for the different categories of system:

| Software Project | a | b | c | d |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi-detached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

# Software Engineering

## More about WBS – Estimation approaches


What are the outcomes of Project Planning

Project Plan

Work Breakdown Structure

Schedule

Resource Management

Plans for communication, metrics, etc

Risk management plan

Different roles

# Software Engineering

## Contents of project plan

- Introduction
- Deliverables of the project
- Process model
- Organization of project
- Standards, guidelines, procedures
- Management activities
- Risks
- Staffing
- Methods and techniques
- Quality criteria/assurance
- Work packages
- Resources
- Budget and schedule
- Change control process
- Delivery means

A good template is here.

# Software Engineering

## Project monitoring and control

**What is Project Monitoring and Control?**

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.

# Software Engineering

## Dimension - Time

In terms of effort (number of man-months) and the schedule

- Measuring progress is hard ("we spent half the money, so we must be halfway")
- Development models serve to manage time
- More people $\Rightarrow$ less time?

  Brooks' law: adding people to a late project makes it later

# Software Engineering

## Dimension - Information

Includes availability, propagation, communication and documentation

- Documentation
  - Technical documentation
  - Current state of projects
  - Changes agreed upon
  - ...
- Agile projects: less attention to explicit documentation, more on tacit knowledge held by people

# Software Engineering

## Dimension - Organization

Involves structure, roles & responsibilities from people and team aspects

- Building a team
- Reorganizing structures
- Clarifying, managing expectations and reorienting roles and responsibilities
- Coordination of work

# Software Engineering

## Dimension - Quality

Quality is not an add-on feature; it has to be built in.

A leading indicator informs business leaders of how to produce desired results.
A lagging indicator measures current production and performance.

- Quality has to be designed in
- Quality is not an afterthought
- Quality requirements often conflict with each other
- Requires frequent interaction with stakeholders

# Software Engineering

## Dimension – Cost and Infrastructure

Includes personnel, capital and expenses

- Which factors influence cost?
- What influences productivity?
- Relation between cost and schedule

# Software Engineering

## Project monitoring and control

Project monitoring and control process could include:

- Monitoring and controlling project work
  1. Collects, measures and disseminates performance information, assess trends to identify potential items requiring corrective action or monitoring for risks.
  2. This can result in corrective or preventive actions and request for changes
  3. Periodically do **Critical Path Analysis**

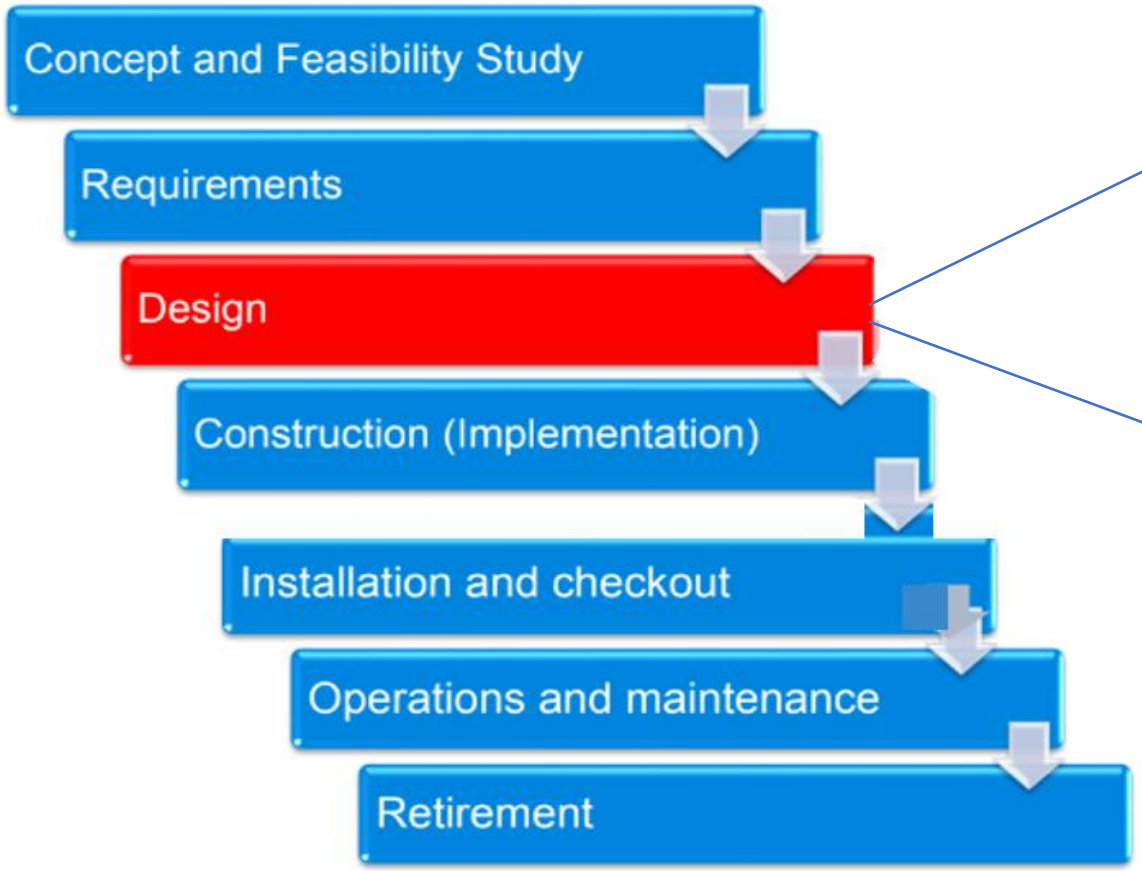
- Ensuring that all of the change controls are being meticulously followed

- 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

## Context of Software Design




**Architectural Design**
Architectural design describes how software is decomposed and organized into components (the software architecture) [IEEE 1471-00]

**Detailed Design**
Detailed design describes the specific behavior of components identified during architectural design

# Software Engineering

## Food for thought

What might be the similarities between mastering chess and mastering the software development? Can you think of some similarities?

**Learning the rules**

**Learning the principles** (relative weight of certain chess pieces; generic/object oriented programming)

**Study game patterns of other masters; Study the architectural and design patterns of other masters**

**The patterns studied must be understood and applied repeatedly**

# Software Architecture

**What is Software Architecture?**

It is the top level decomposition of the system being developed, 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 which happen during architecture

# Importance of Software Architecture

***Architecture manifests the earliest set of design decisions***

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

***Supports reuse at architectural system level***

***Helps in work breakdown and structures the development***

- Reduces risk and can enable cost reduction during product development

***Changes to architecture is expensive in the later phases of SDLC***

# Software Engineering

## Characteristics of Software Architecture

Addresses variety of stakeholder perspectives

Realizes all of the use cases and scenarios envisaged for the problem

Supports separation of concerns

Quality driven

Recurring styles

Conceptual integrity

## Factors that influence Software Architecture

1. Functional requirements
2. Data Profile
3. Audience
4. Usage characteristics
5. Business priority
6. Regulatory and Legal obligations
7. Architectural standards
8. Dependencies and Integration
9. Cost constraints
10. Initial state
11. Architect, staff background and the skill level
12. Technical and organizational environment
13. Technology constraints
14. Type of user experience planned

# Software Engineering

## Architect

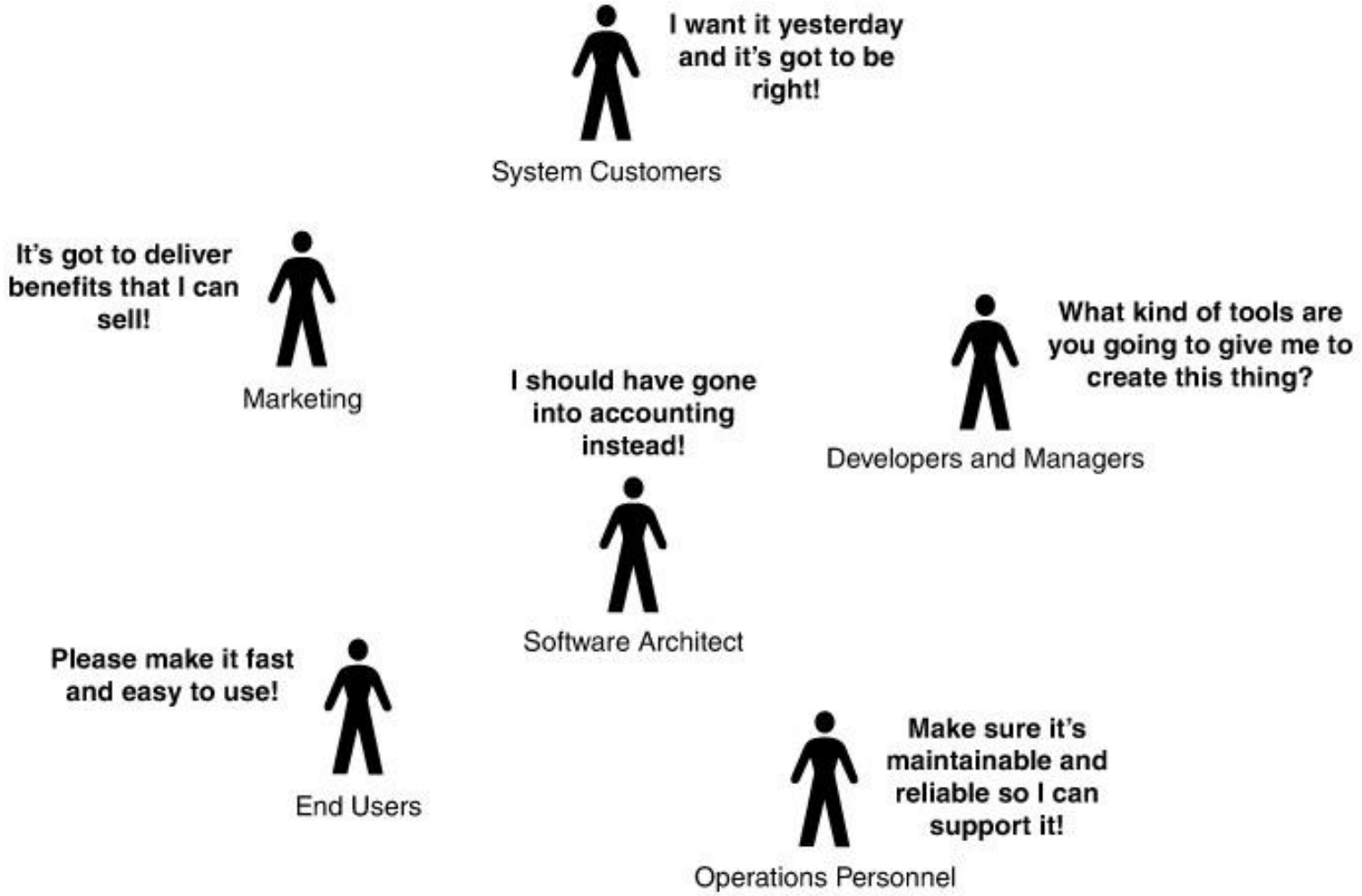
Software Architect is a role within the software development organization structure, who makes high level design choices and dictates technical standards, including software coding standards, tools and platforms.

An Architect has:

- A Distinct role in project
- Broad training and extensive experience
- Deep understanding of domain

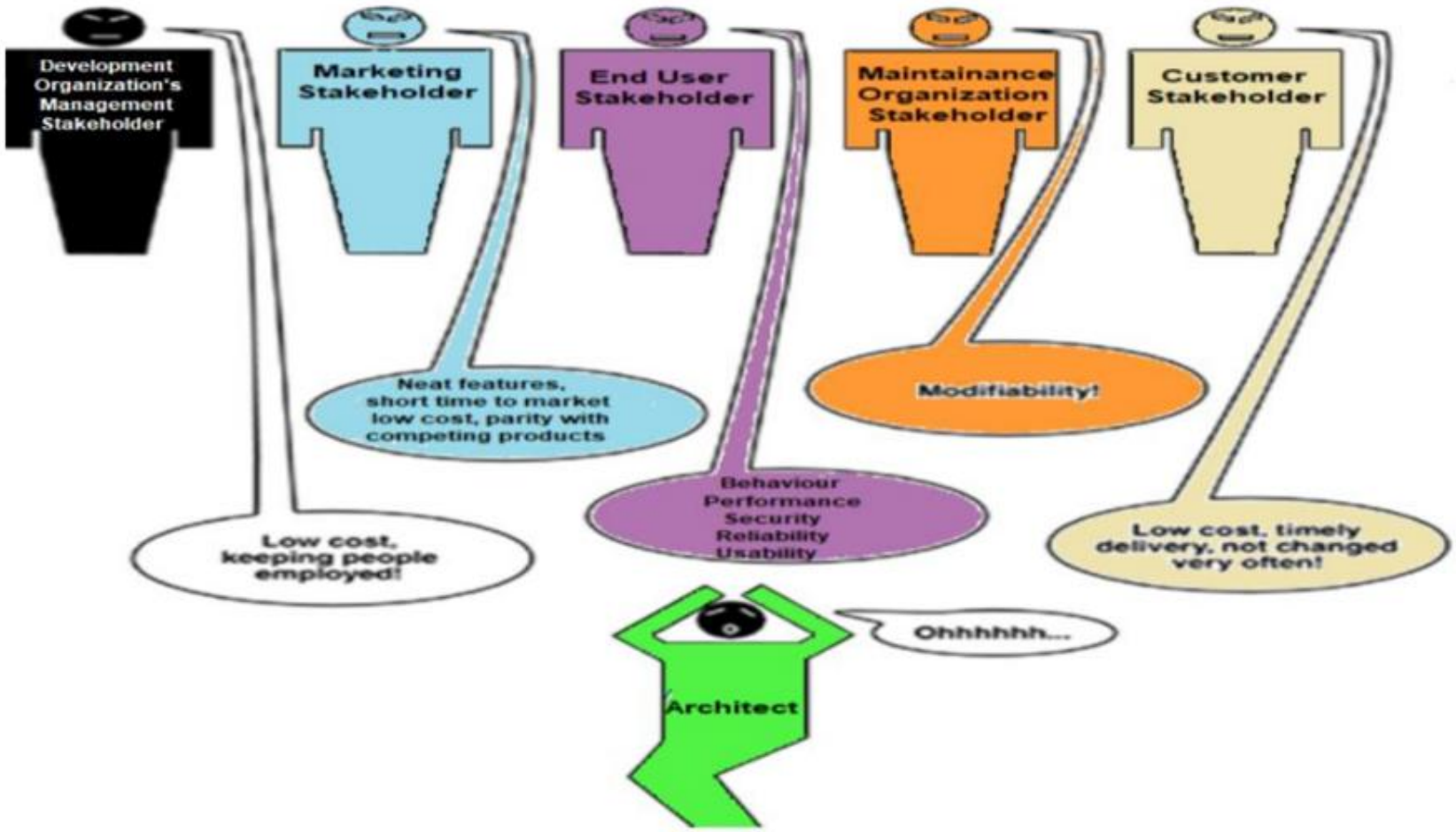
# Software Engineering

## Role of Software Architect



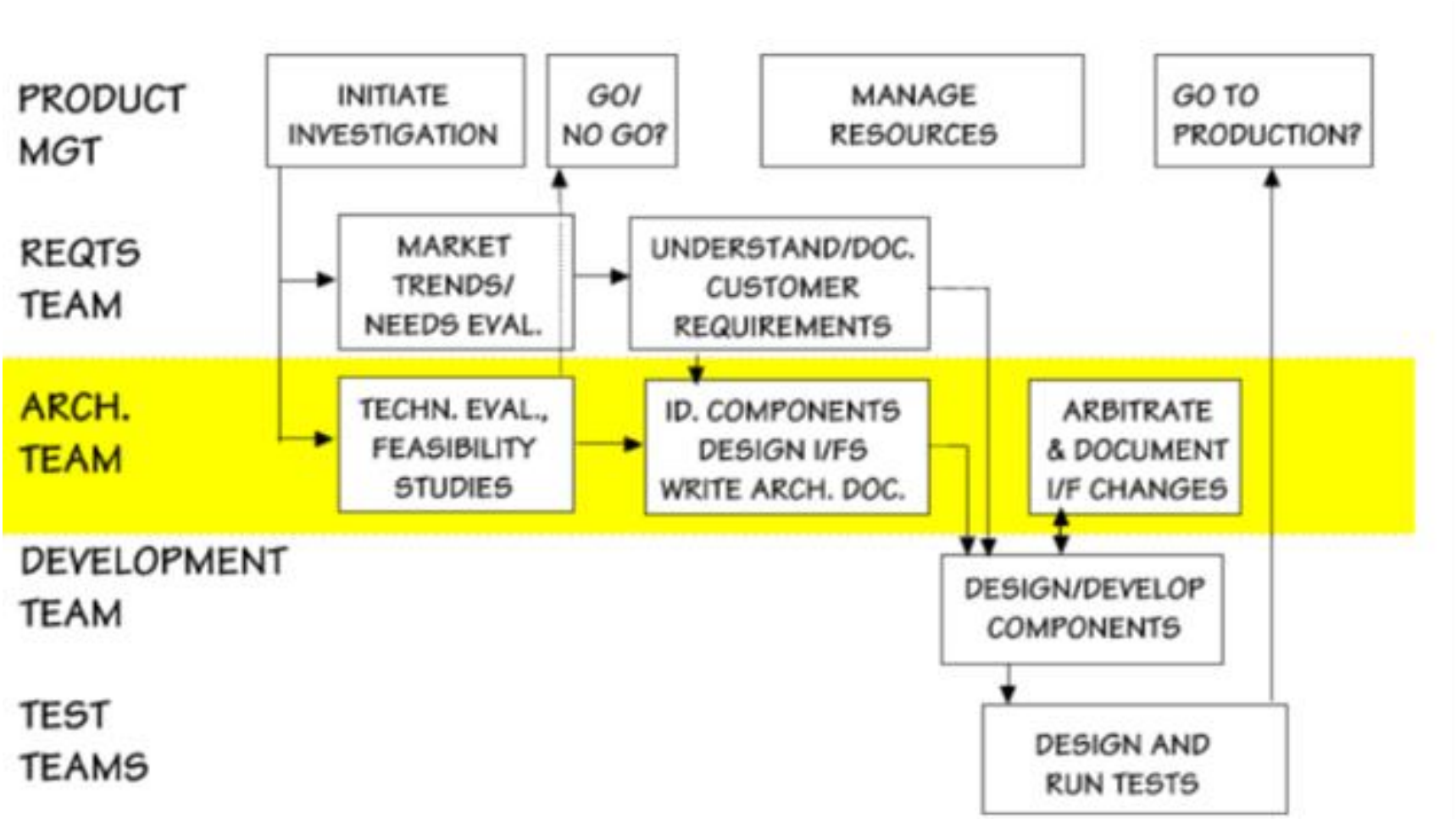
# Software Engineering

## Role of Software Architect



# Software Engineering

## Components of different teams



# Software Engineering

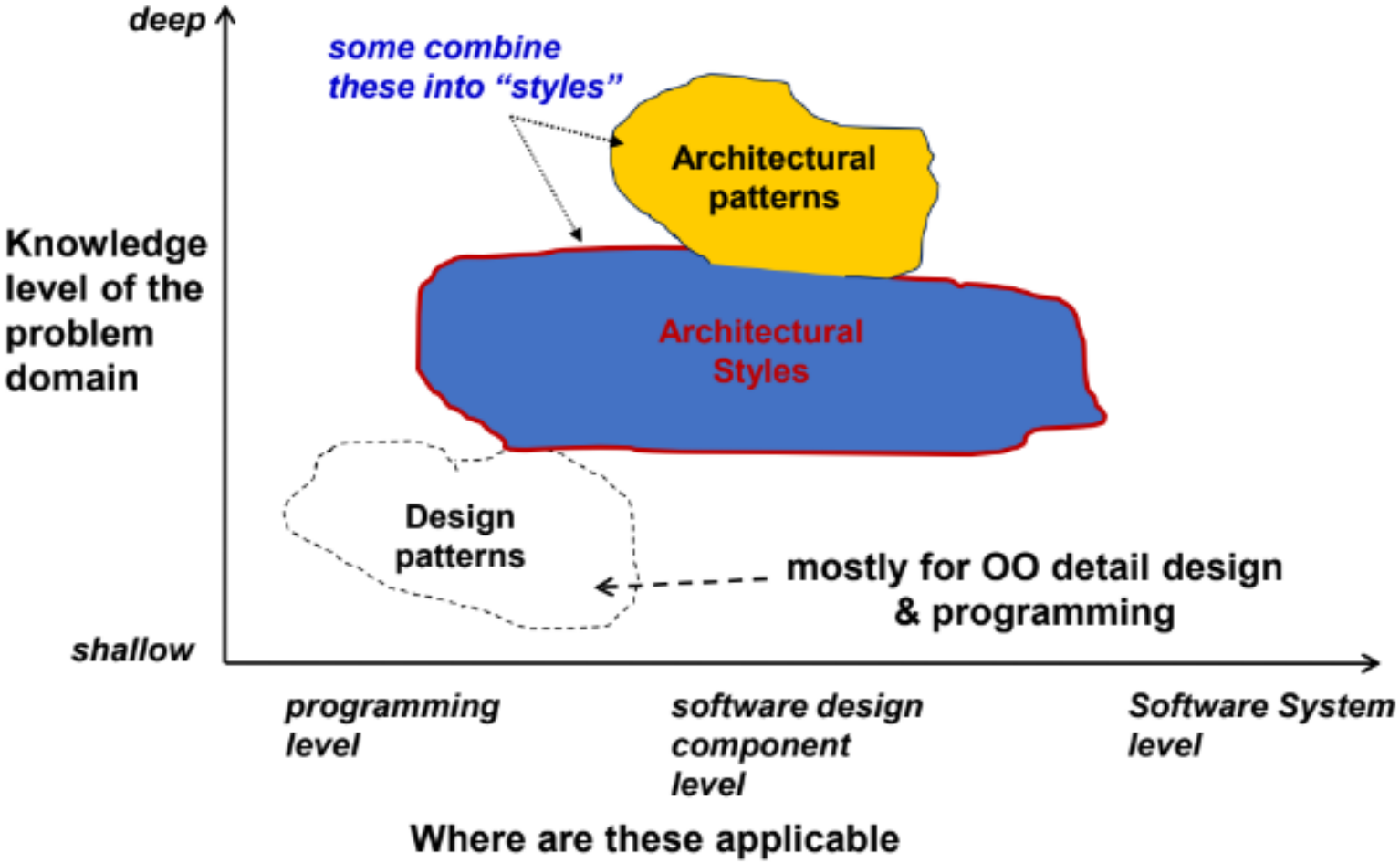
## Architectural views, styles and patterns

| Architectural View | Architectural Styles | Architectural Pattern |
| --- | --- | --- |
| Views represent ways of describing the Software Architecture, enabling the system to be viewed by different stakeholders in perspectives of their interest.<br>IEEE standard 1471<br><br>*Ex: UI View, Process View* | Demonstrates how the subsystems or elements are organized or structured<br>It is a way of organizing code<br><br>*Ex: Pipe & filters, Client-Server, Peer-to-Peer* | 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 known to work for the problem scenario<br><br>*Ex: MVC separating UI from the rest* |

A good sample SA document is here.

# Software Engineering

## Combined view



# Software Engineering

## Comparison

Consider a telephone used in the earlier days and a telecommunication application such as Skype

What are the characteristics and architectures of these two?

**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

# Software Engineering

## Food for thought

You are a service provider offering mail service
- Customers call in with their problems
- You need to review mail server logs to diagnose their problems

What are the challenges and the solution choices?

Challenges
- Have hundreds of servers
- GBs of logs are generated every day
- Collecting logs takes time
- Searching logs takes time.

Solution choices
- Local log files
- Centralize collection of logs
- Distribute searching and collection of logs
- Can you pre-process logs to speed up queries?

# Software Engineering

## Quality attribute trade-off

| Trade-off | Version 1 | Version 2 | Version 3 |
|---|---|---|---|
| Data freshness | Queries run on current data | Queries run on 10 minutes old data | Queries run on 10-20 minutes old data |
| Scalability | Email server slows down | MySQL speed / stability problem | No problems yet |
| Ad hoc query ease | Regular expression | SQL expression | Map-Reduce program |

## Architectural conflicts

Using large-grain components improves performance but reduces maintainability.

Introducing redundant data improves availability but makes security / data integrity more difficult
(Recall normalization and de – normalization in DB)

Localizing safety-related features usually means more communication so degraded performance.

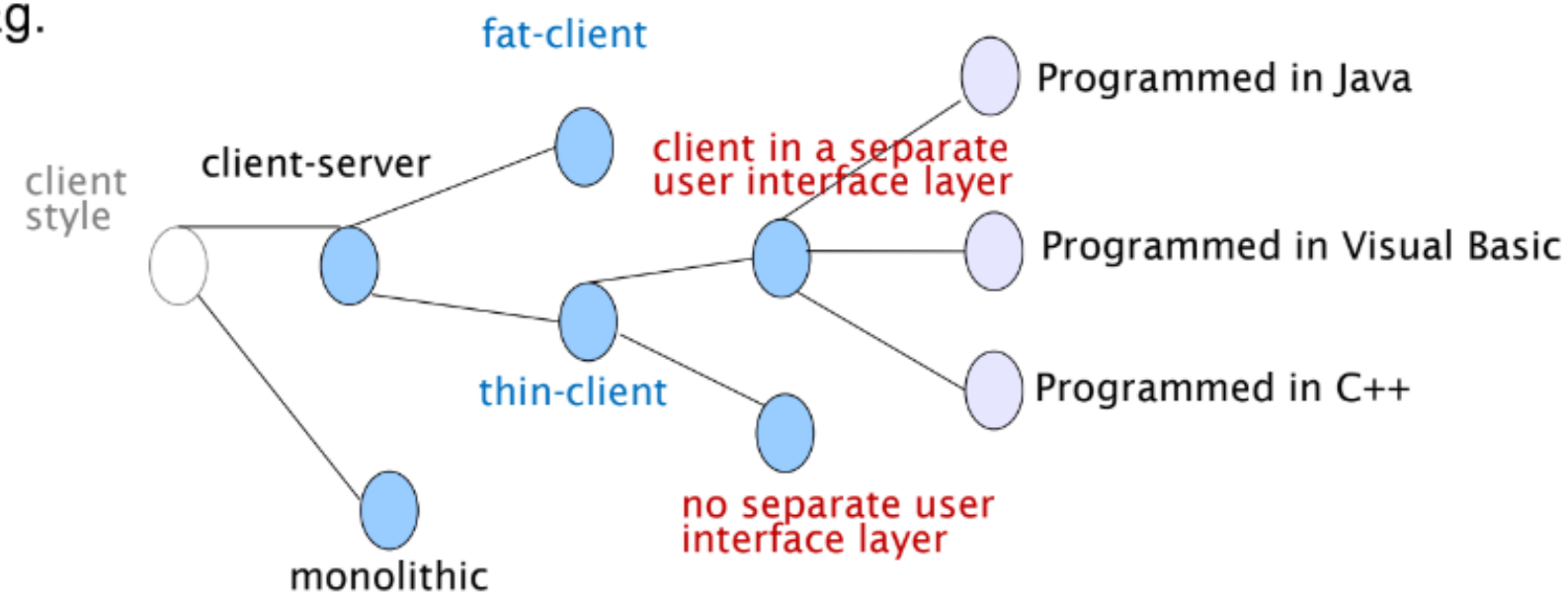
# Software Engineering

## Design issues and decisions

Designer makes a *design decision* to *resolve* design issues faced.

Design decision: Process involves choosing the best option from among the alternatives

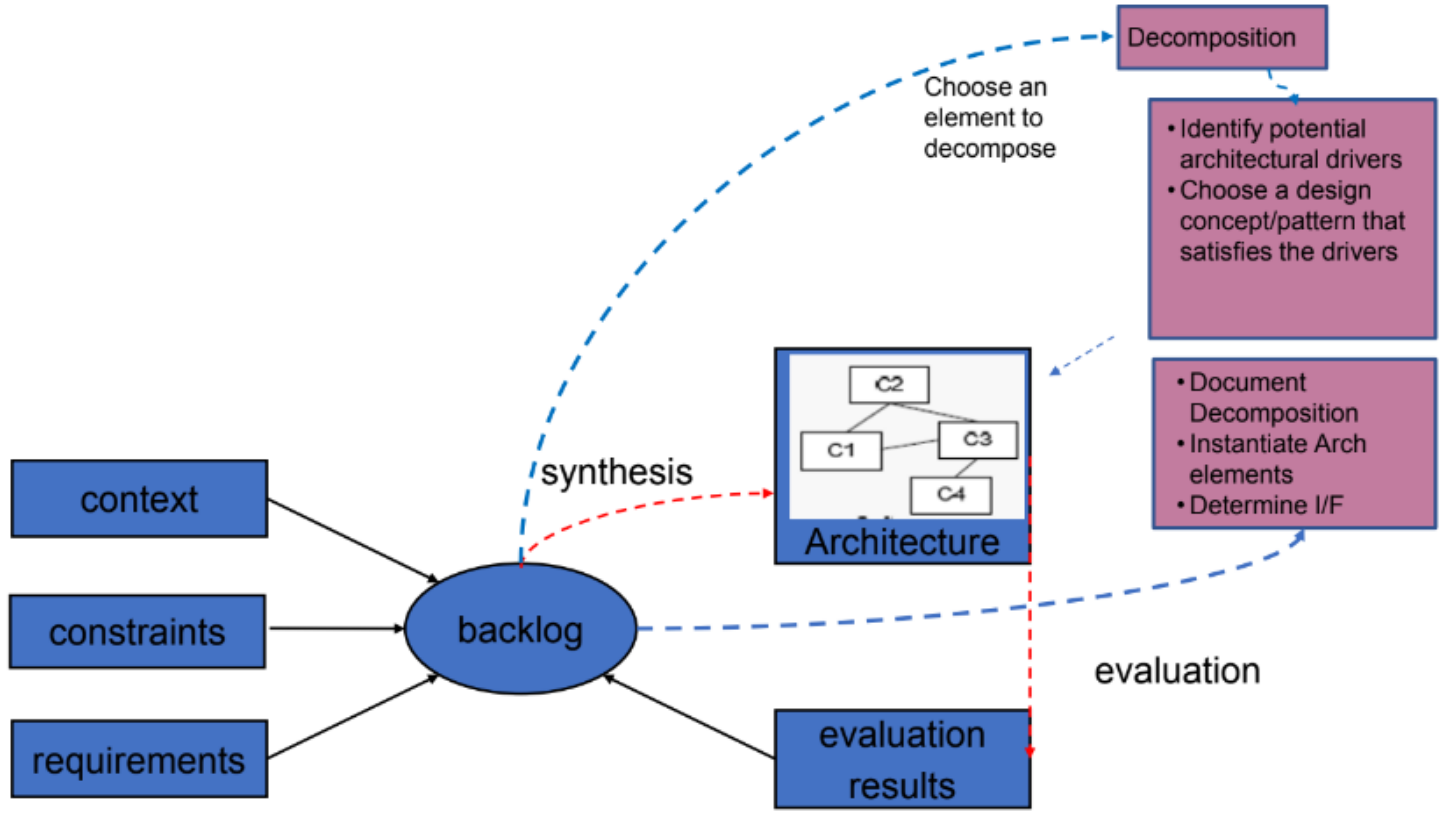
Eg.



# Software Engineering

## Generalized model for architecting

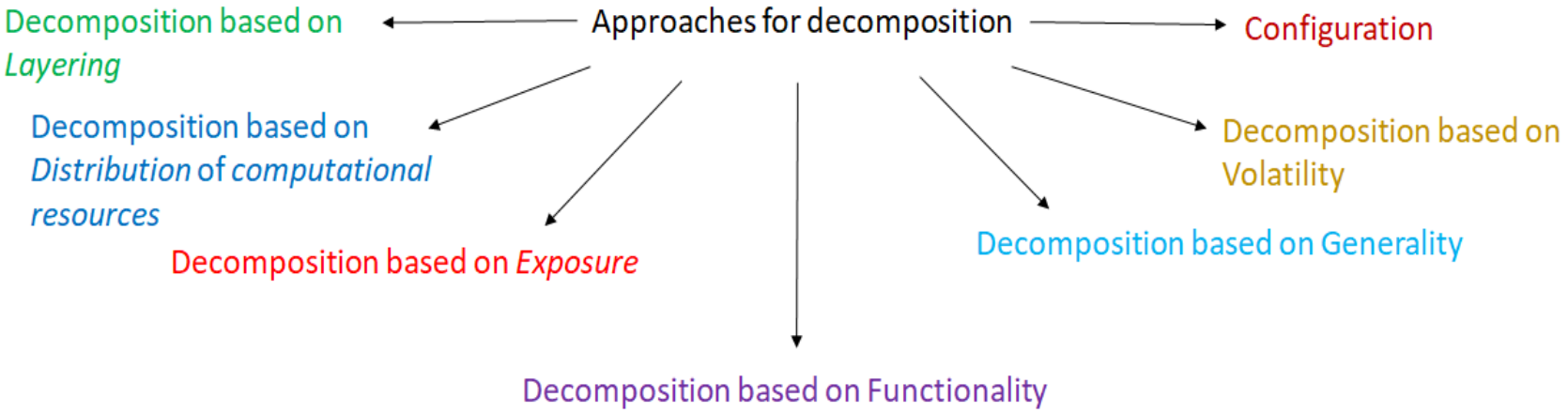
Understand problem → Solve it → Evaluate solution



# Software Engineering

## Theme of Architecture - Decomposition

The first step typically would involve decomposition of the problem to individual Modules (a set of code or data units)




Keep things together (Coupling) that work together (Cohesion)
Look for **low coupling** and **high cohesion**.

# Software Engineering

## Theme of Architecture – Decomposition …

Other ways of looking at approaches for decomposition include

- Divide & Conquer
- Stepwise Refinement
- Top Down approach
- Bottom Up approach
- Information Hiding

# Software Engineering

## Gantt Chart

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

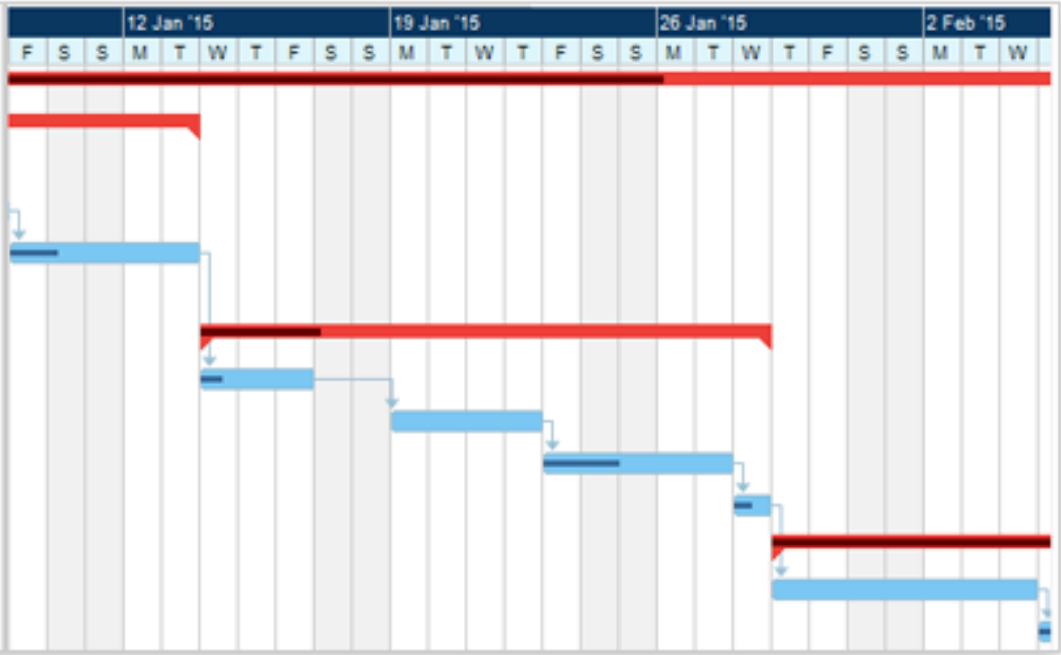
- What the various activities are
- When each activity begins and ends
- How long each activity is scheduled to last
- Where activities overlap with other activities, and by how much
- The start and end date of the whole project

To summarize, a Gantt chart shows you what has to be done (the activities) and when (the schedule).

# Software Engineering

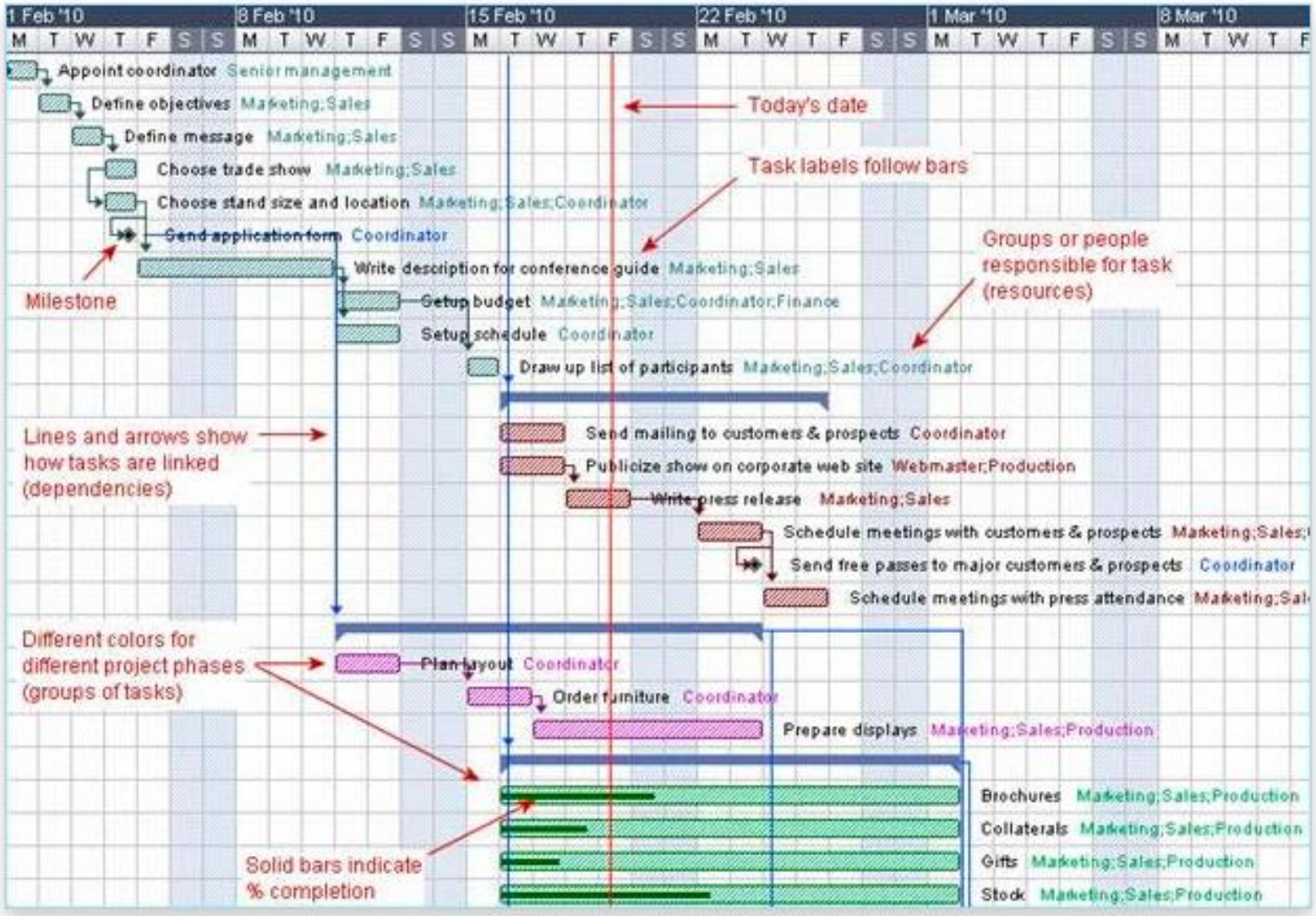
## Gantt Chart …

| | | | Tas… | Task Name | Duration | Start | End |
|---|---|---|---|---|---|---|---|
| 1 | | | | Business Plan | 48 days? | 1/1/2015 | 3/9/2015 |
| 2 | | | | ⊟ Phase 1 - Str... | 9 days | 1/1/2015 | 1/13/201 |
| 3 | | | | Define the... | 2 days | 1/1/2015 | 1/2/2015 |
| 4 | | | | Revenue P... | 4 days | 1/5/2015 | 1/8/2015 |
| 5 | | | | Evaluate B... | 3 days | 1/9/2015 | 1/13/201 |
| 6 | | | | Helpful Links | 0 days | 1/1/2015 | 1/1/2015 |
| 7 | | | | ⊟ Phase 2 - De... | 11 days? | 1/14/2015 | 1/28/201 |
| 8 | | | | Define the... | 3 days | 1/14/2015 | 1/16/201 |
| 9 | | | | Identify Ne... | 4 days | 1/19/2015 | 1/22/201 |
| 10 | | | | Evaluate P... | 3 days | 1/23/2015 | 1/27/201 |
| 11 | | | | Confirm de... | 1 day? | 1/28/2015 | 1/28/201 |
| 12 | | | | ⊟ Phase 3 - Pla... | 15 days | 1/29/2015 | 2/18/201 |
| 13 | | | | Develop D... | 5 days | 1/29/2015 | 2/4/2015 |
| 14 | | | | Describe th... | 2 days | 2/5/2015 | 2/6/2015 |



# Software Engineering

## Gantt Chart …



## Gantt chart …

What Is a Critical Path in a Gantt Chart?
A *critical path* in a Gantt chart refers to the phenomenon in which any individual task causes a delay in the related sequence of tasks, thus pushing back a project's overall end date.

In project management, the "critical path" refers to the longest sequence of dependent or floating tasks that must be completed in order to get the project done on time.

Activities that have a total slack time of zero are considered to be on the critical path.

One of the most commonly used tools for project management is **Jira**. This software is used for **bug tracking, issue tracking and project management.**

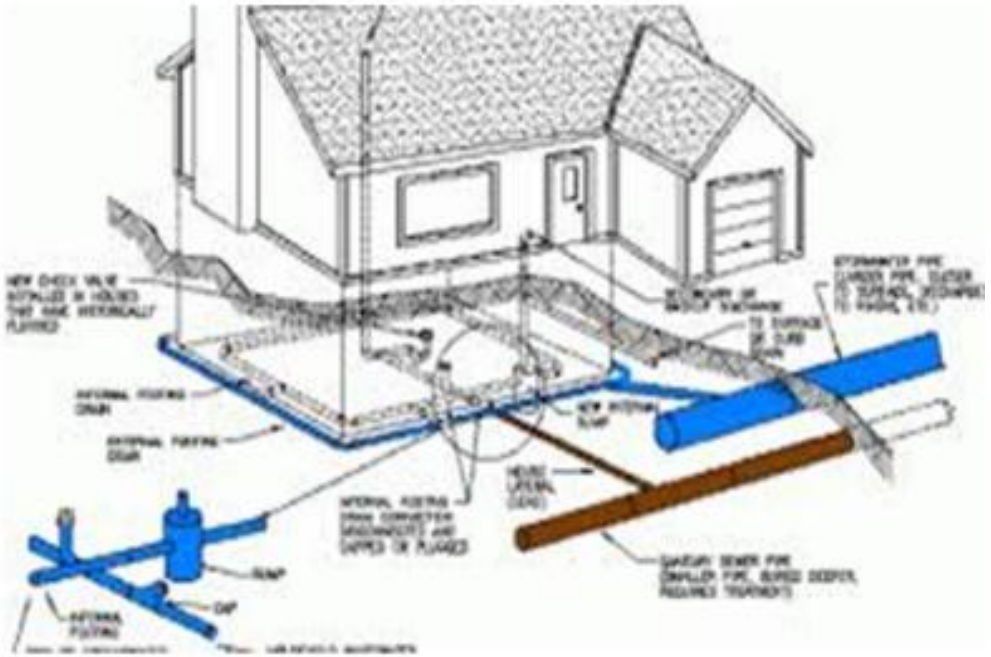
# Software Engineering

## Architectural views – An example

Elevation

Plan

Plumbing View



# Architectural views

## Different views

1. Module view point/Structure of modules
2. Component-and-Connector structure/view point
3. Allocation structure/view point
4. Krutchens 4+1 View/Model

View point should be chosen based on

- Who are the stakeholders and what are their concerns?
- Which viewpoints address these concerns?
- Prioritize and possibly combine viewpoints

# Module view point

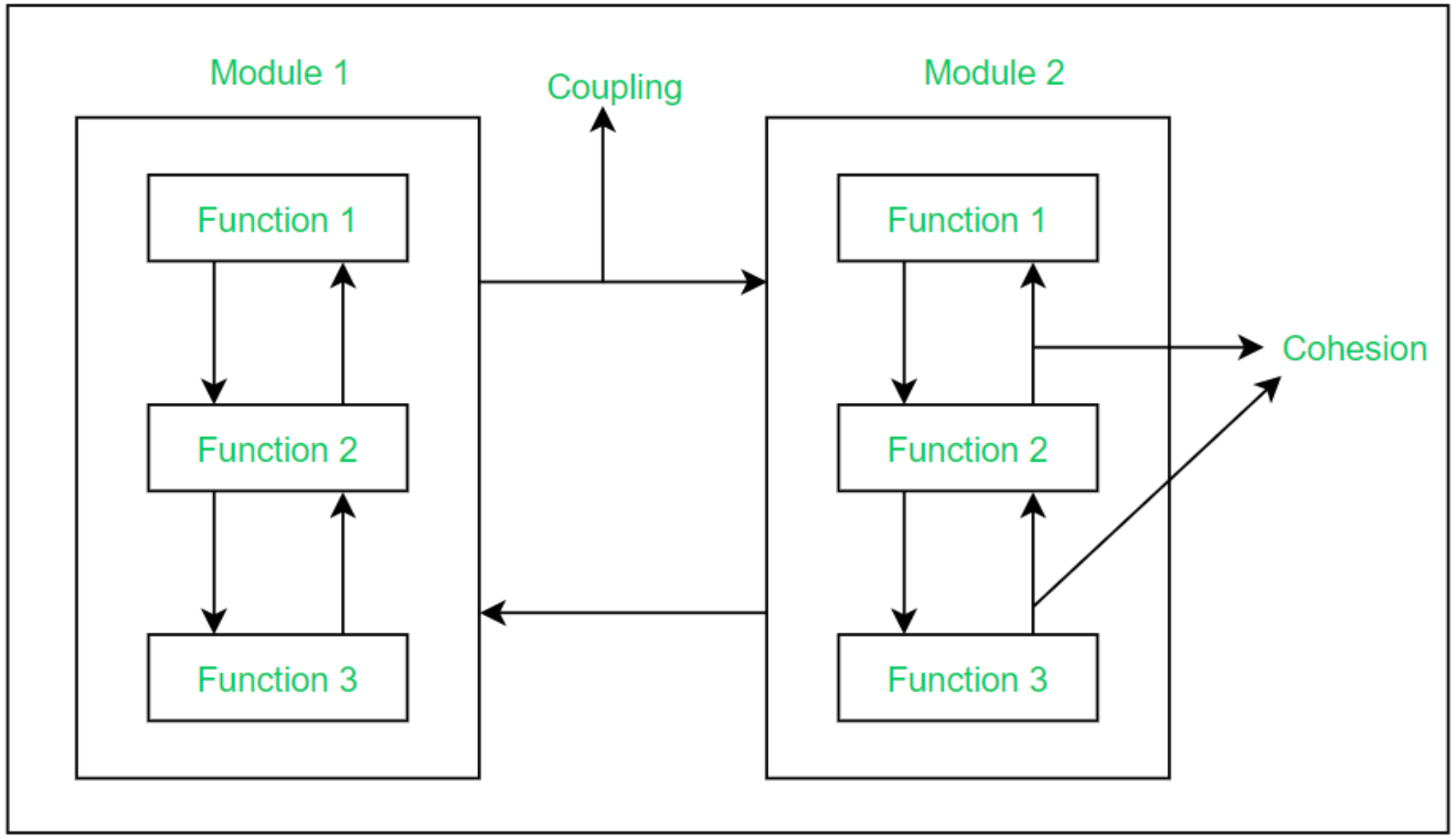
**Modules are units of (code) Implementation with some functional responsibility**

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.

# Software Engineering

## Module view point



# Component-and-Connector viewpoint

Component and Connector (C&C) architecture view of a system has two main elements—components and connectors.

Components are usually computational elements or data stores that have some presence during the system execution. Connectors define the means of interaction between these components.

A C&C view describes a runtime structure of the system—what components exist when the system is executing and how they interact during the execution.

## Component-and-Connector viewpoint …

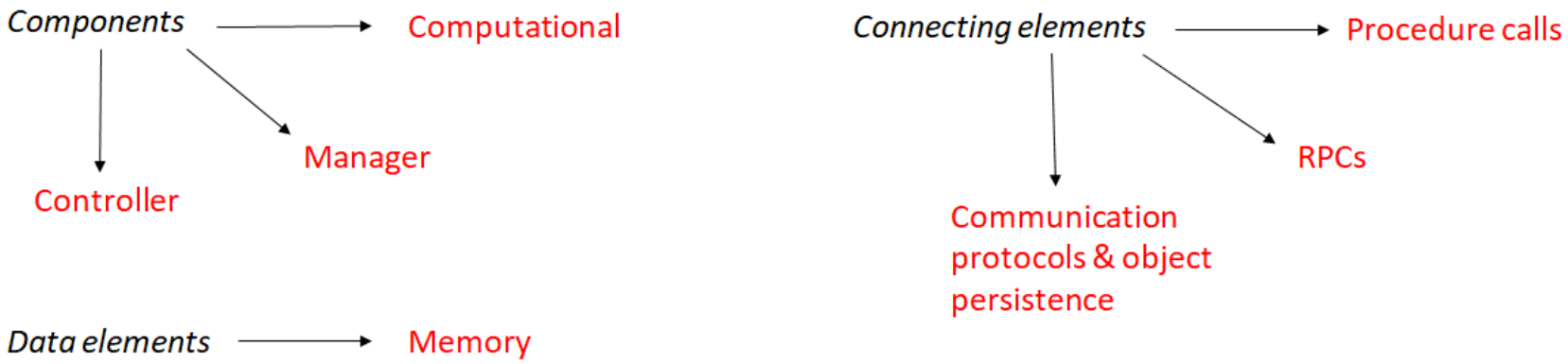
**Connectors: -** The different components of a system are likely to interact while the system is in operation to provide the services expected of the system. After all, components exist to provide parts of the services and features of the system, and these must be combined to deliver the overall system functionality. For composing a system from its components, information about the interaction between components is necessary.

Interaction between components may be through a simple means supported by the underlying process execution infrastructure of the operating system.

For example, a component may interact with another using the procedure call mechanism (a connector,) which is provided by the runtime environment for the programming language. However, the interaction may involve more complex mechanisms as well. Examples of such mechanisms are remote procedure call, HTTP, etc.

# Software Engineering

## Components-and-Connectors ..



# Allocation view point

**Deployment structure** which shows how software is assigned to the hardware elements and which communication paths are used. This view allows an engineer to reason about performance, data integrity, availability, and security. It is of particular interest in distributed or parallel Systems.

**Implementation structure** which indicates how software is mapped onto file structures in the system's development, integration, or configuration control environments.

**Work Assignment structure** which shows who is doing what and helps to determine which knowledge is needed where.

# Software Engineering

## Krutchens view point

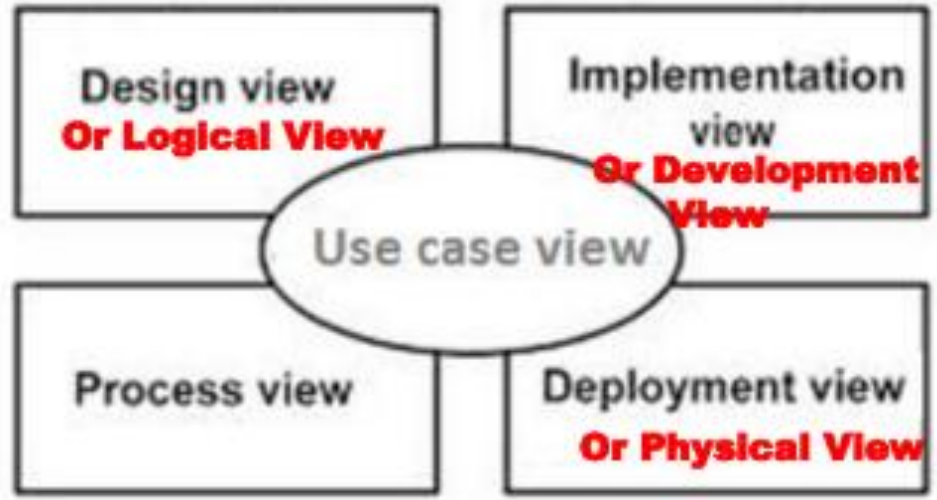
**Use case view** (exposing the requirements of the system or the scenarios)

**Design view** (exposes vocabulary of the problem space and the solution space) Class Diagrams, Sequence diagrams etc.

**Process view** encompasses the dynamic aspects or the runtime behavior of the system. Threads and processes that form the systems. Addresses performance, concurrency etc.

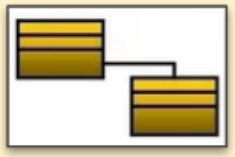
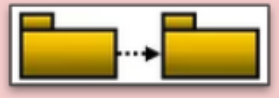
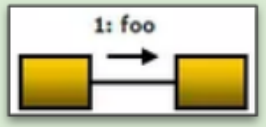
**Implementation view** (addresses the realization of the system. UML diagrams like package diagrams are used)

**Deployment view** (focuses on system engineering issues)



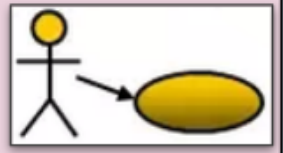
# Software Engineering

## Krutchens view point

| | Conceptual / Logical | Physical / Operational |
|---|---|---|
| Functional | **Logical / Structural view**<br>**Perspective**: Analysts, Designers<br>**Stage**: Requirement analysis<br>**Focus**: Object oriented decomposition<br>**Concerns**: Functionality<br>**Artefacts**:<br>• Class diagram<br>• Object diagram<br>• Composite structure diagram<br>  | **Implementation / Developer view**<br>  <br>**Perspective**: Developers, Proj. mngs.<br>**Stage**: Design<br>**Focus**: Subsystem decomposition<br>**Concerns**: Software management<br>**Artefacts**:<br>• Component diagram<br>• Package diagram |
| Non-functional | **Process / Behaviour view**<br>**Perspective**: System Integrators<br>**Stage**: Design<br>**Focus**: Process decomposition<br>**Concerns**: Performance, scalability, throughput<br>**Artefacts**:<br>• Sequence diagram<br>• Communication diagram<br>• Activity diagram<br>• State (machine) diagram<br>• Interaction overview diagram<br>• Timing diagram<br>  | **Deployment / Physical view**<br>**Perspective**: System Engineers<br>**Stage**: Design<br>**Focus**: Map software to hardware<br>**Concerns**: System topology, delivery, installation, communication<br>**Artefacts**:<br>• Deployment diagram<br>• Network topology (not UML)<br>  |

**Use Case/Scenario view**

**Perspective**: End users
**Stage**: Putting it alltogether
**Concerns**: Understandability, usability
**Focus**: Feature decomposition
**Artefacts**:
- Use-case diagram
- User stories



# Software Engineering

## Architectural style

# Architectural Styles

**What are architectural styles?**

Architectural style is a way of organization of the components in an architecture, characterized by the features that make it notable. It can be looked at as a tool box containing tools of architecture.

It addresses the structure and behavior of the system & is a way of organizing modules.

Architectural styles provide:

1. **Vocabulary** – A set of design elements
2. **Design rules** – A set of design constraints
3. **Semantic interpretation** - Well-defined meaning of the connected design elements)
4. **Analysis** - Analysis that can be performed on systems built in that style

# Software Engineering

## Architectural Styles – Main program with subroutines

**Generic:** Traditional Language-Influenced Style

**Problem:** The system can be described as a hierarchy of functions; This is the natural outcome of the functional decomposition of the system. The top level module acts as a main program and invokes the other modules in the right order. There is a usually a single thread of control

**Context:** Language with nested procedures

**Solution:**

- System model: Procedures and modules are defined in a hierarchy. Higher level module calls and lower level modules. Hierarchy may be strict (n can only call n-1) or week (where n can call n-i) may be weak or strong, coupling/cohesion arguments
- Components: Procedures which can be viewed as residing in the main program, and have their own local and global data
- Connectors: procedure call and shared access to global data
- Control structure: single centralized thread of control: main program pulls the strings

# Architectural Patterns

**What is an Architectural pattern?**

An **architectural pattern** is a general, reusable solution to a commonly occurring problem in software architecture within a given context.

Following traditional building architecture, a 'software architectural style' is a specific method of construction, characterized by the features that make it notable.

An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined.

# Software Engineering

## Architectural Patterns

An architectural style is a named collection of architectural design decisions

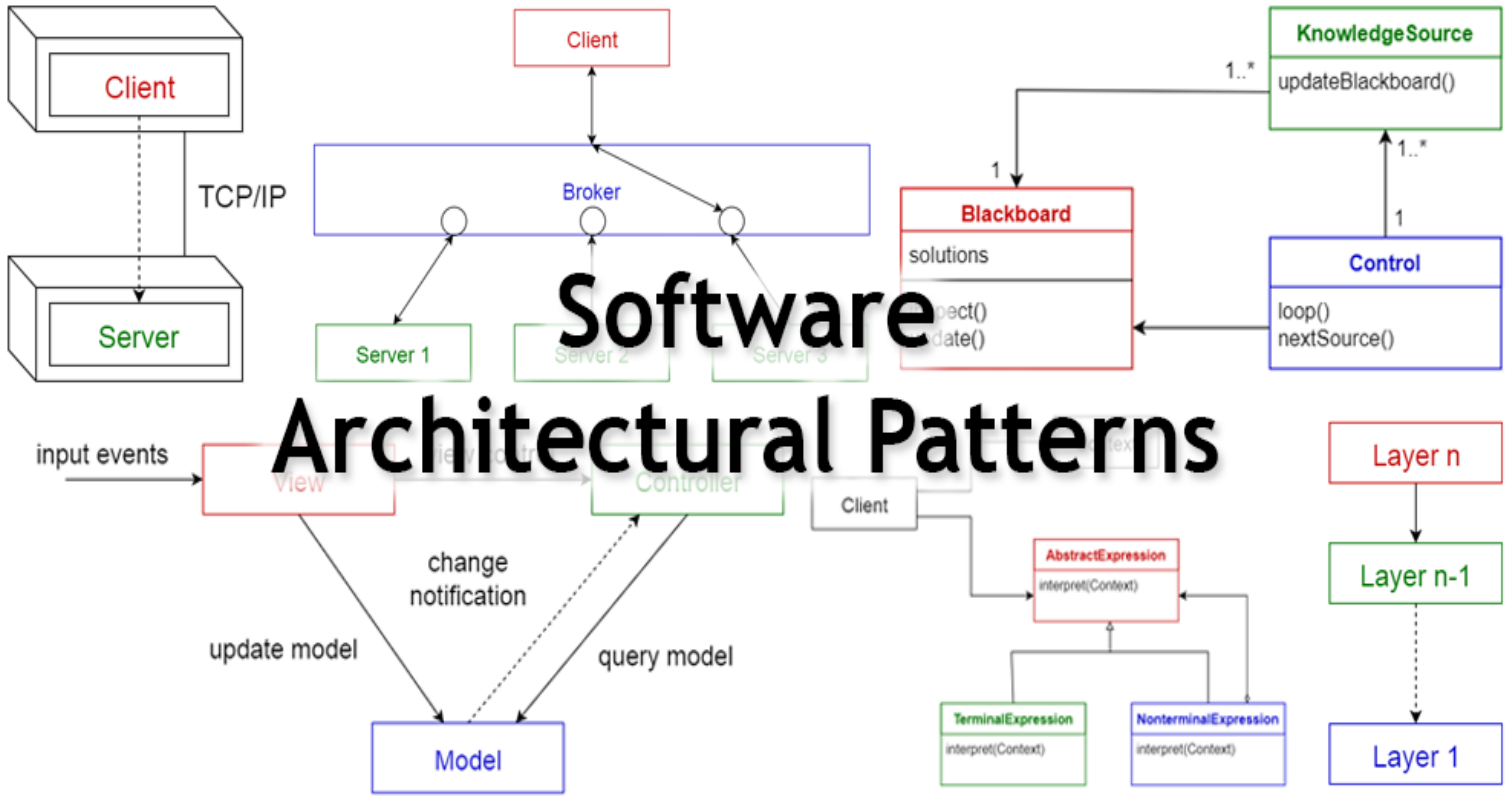
- that are applicable in a given development context,
- constrain architectural design decisions that are specific to a particular system within that context, and
- elicit beneficial qualities in each resulting system.

Some treat architectural patterns and architectural styles as the same, some treat styles as specializations of patterns. What they have in common is both patterns and styles are idioms for architects to use, they "provide a common language" or "vocabulary" with which to describe classes of systems.

The main difference is that a pattern can be seen as a solution to a problem, while a style is more general and does not require a problem to solve for its appearance.

# Software Engineering

## Architectural Patterns



# Software Engineering

## Some of the most popular architectural patterns

1. Layered pattern
2. Client-server pattern
3. Master-slave pattern
4. Pipe-filter pattern
5. Broker pattern
6. Peer-to-peer pattern
7. Event-bus pattern
8. Model-view-controller pattern
9. Blackboard pattern
10. Interpreter pattern

**Self Study**

More details can be found at
https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013

# Importance of Architectural Styles and Patterns


Benefits

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

Represents earliest design decisions

Hardest to modify
Most critical to get right
Communication between various stakeholders

First design artefact

Performance, Reliability, Modifiability, Security

Key to systematic reuse

Transferable, reusable abstraction

# Software Engineering

## An example of Architecture document

1. ***Introduction***
   1. Purpose
   2. Scope
   3. Definitions, Acronyms and Abbreviations
   4. References
2. ***Architectural Representation***
3. ***Architectural Goals and Constraints***
4. ***Use-Case View***
   1. Architecturally-Significant Use Cases
5. ***Logical View***
   1. *Architecture Overview* – Package and Subsystem Layering
   2. *Process View*
      1. Processes
      2. Process to Design Elements
      3. Process Model to Design
      4. Model Dependencies
      5. Processes to the Implementation
6. ***Deployment View***
   1. External Desktop PC
   2. Desktop PC
   3. Registration Server
   4. Course Catalog
   5. Billing System
7. ***Performance***
8. ***Quality***

# Software Engineering

## Software Design

## Software Design principles

1. Further decomposition (post architecture) of the components being developed if necessary.

2. Description of the behavior of components/sub-systems identified as part of Architectural design

3. Description of how the interfaces will actually be realized using the appropriate algorithms and data structures

4. Description on how the system will facilitate interaction with the user through the user interface

5. The use of appropriate structural and behavioral design patterns & consideration of maintenance and reuse as couple of its goals

# Software Engineering

## Techniques that enable design

Abstraction

Focus on essential properties

Modularity, Cohesion and Coupling

Modularity is the degree or the extent to which the larger module can be decomposed

Cohesion is the glue that keeps the components together. Cohesion which is the extent to which the component/modules are dependent on/fit into or are related to each other, when trying to address a specific responsibility (Strong cohesion is good)

There are seven levels of cohesion of increasing strength:

## Techniques that enable design …

**Coincidental cohesion** With coincidental cohesion, elements are grouped into modules in a haphazard way. There is no significant relation between the elements.

**Logical cohesion** With logical cohesion, the elements realize tasks that are logically related. One example is a module that contains all input routines. These routines do not call one another and they do not pass information to each other. Their function is just very similar.

**Temporal cohesion** A typical example of this type of cohesion is an initialization module. The various elements of it are independent but they are activated at about the same point in time.

**Procedural cohesion** A module exhibits procedural cohesion if it consists of a number of elements that have to be executed in some given order. For instance, a module may have to first read some datum, then search a table, and finally print a result.

## Techniques that enable design …

**Communicational cohesion** This type of cohesion occurs if the elements of a module operate on the same (external) data. For instance, a module may read some data from a disk, perform certain computations on those data, and print the result.

**Sequential cohesion** Sequential cohesion occurs if the module consists of a sequence of elements where the output of one element serves as input to the next element.

**Functional cohesion** In a module exhibiting functional cohesion all elements contribute to the one single function of that module. Such a module often transforms a single input datum into a single output datum. The well-known mathematical subroutines are a typical example of this. Less trivial examples are modules like ‘execute the next edit command’ and ‘translate the program given’.

# Software Engineering

## Techniques that enable design

Coupling indicates how strongly the modules are connected to other modules. (Loose coupling is good)

The following types of coupling can be identified (from tightest to loosest):

**Content coupling** With content coupling, one module directly affects the working of another module. Content coupling occurs when a module changes another module's data or when control is passed from one module to the middle of another (as in a jump). This type of coupling can, and should, always be avoided.

**Common coupling** With common coupling, two modules have shared data. The name originates from the use of COMMON blocks in FORTRAN. Its equivalent in block-structured languages is the use of global variables.

## Techniques that enable design

**External coupling** With external coupling, modules communicate through an external medium, such as a file.

**Control coupling** With control coupling, one module directs the execution of another module by passing the necessary control information. This is usually accomplished by means of flags that are set by one module and reacted upon by the dependent module.

**Stamp coupling** Stamp coupling occurs when complete data structures are passed from one module to another. With stamp coupling, the precise format of the data structures is a common property of those modules.

**Data coupling** With data coupling, only simple data is passed between modules.

# Software Engineering

## Food for thought

What is the type of cohesion for Input routines?
Logical

What is the type of cohesion for Initialization sequence?
Temporal

What is the type of cohesion for Read and print?
Procedural

# Techniques that enable design

**Information hiding**

Design involves a series of decisions and for each such decision, need to consider who needs to know & who can be kept in the dark. This is done through:

**Encapsulation**

An 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

# Techniques that enable design

## LIMITING COMPLEXITY

Complexity refers to the amount of effort required for building its solution

Can be Intra – modular (within a module) or
Inter – modular (between modules)

Higher value => Higher complexity => Higher effort required (= worse design)

## HIERARCHICAL STRUCTURE

Views whole structure as a hierarchy

# Software Engineering

## Key issues

Concurrency

Non-functional requirements

Data persistence

Event handling

Error, Exception handling, Fault tolerance

Distribution of components

Interaction and Presentation

# Software Engineering

## In-class exercise


Provide a contrast between architecture and design

Software architecture is the structure (or structures) of the system, which comprise of software components, the externally visible properties of those components, and the relationships between them.

while

Software design is problem-solving and planning for a software solution internal to the system.

Architectural decisions are harder to change compared to design decisions which are simpler with lesser impact.

Software architecture has more influence on the non-functional requirements while design has on 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, etc

**Data Flow Diagrams (DFD)**

Developed by Yourdon and Constantine in early 70s

It is a two-step process:

- 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

# Components of a DFD

**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 – represented as two parallel lines  and

**Data Flows**
Paths where data structures travel between processes and entities and data stores – depicted as arrows

**Structured Analysis (SA)** results in a logical model, consisting of a set of DFDs augmented by minispecs & data dictionary

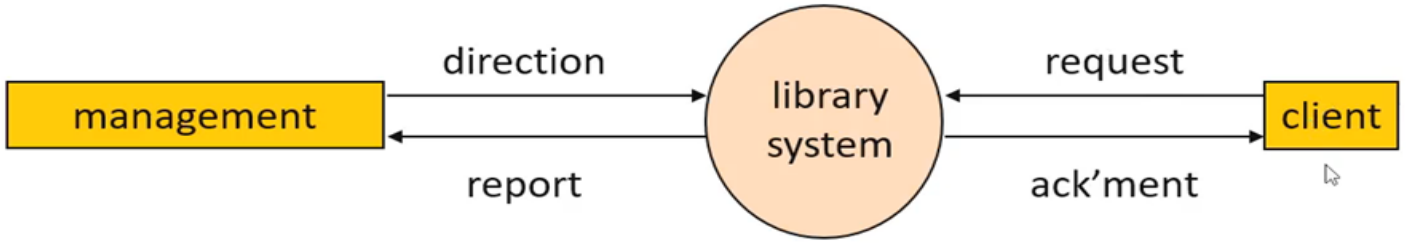
**Structured Design (SD)** results in transition of DFDs to hierarchical structure charts, most often as a transform-centered design.

# Software Engineering

## Top level DFD

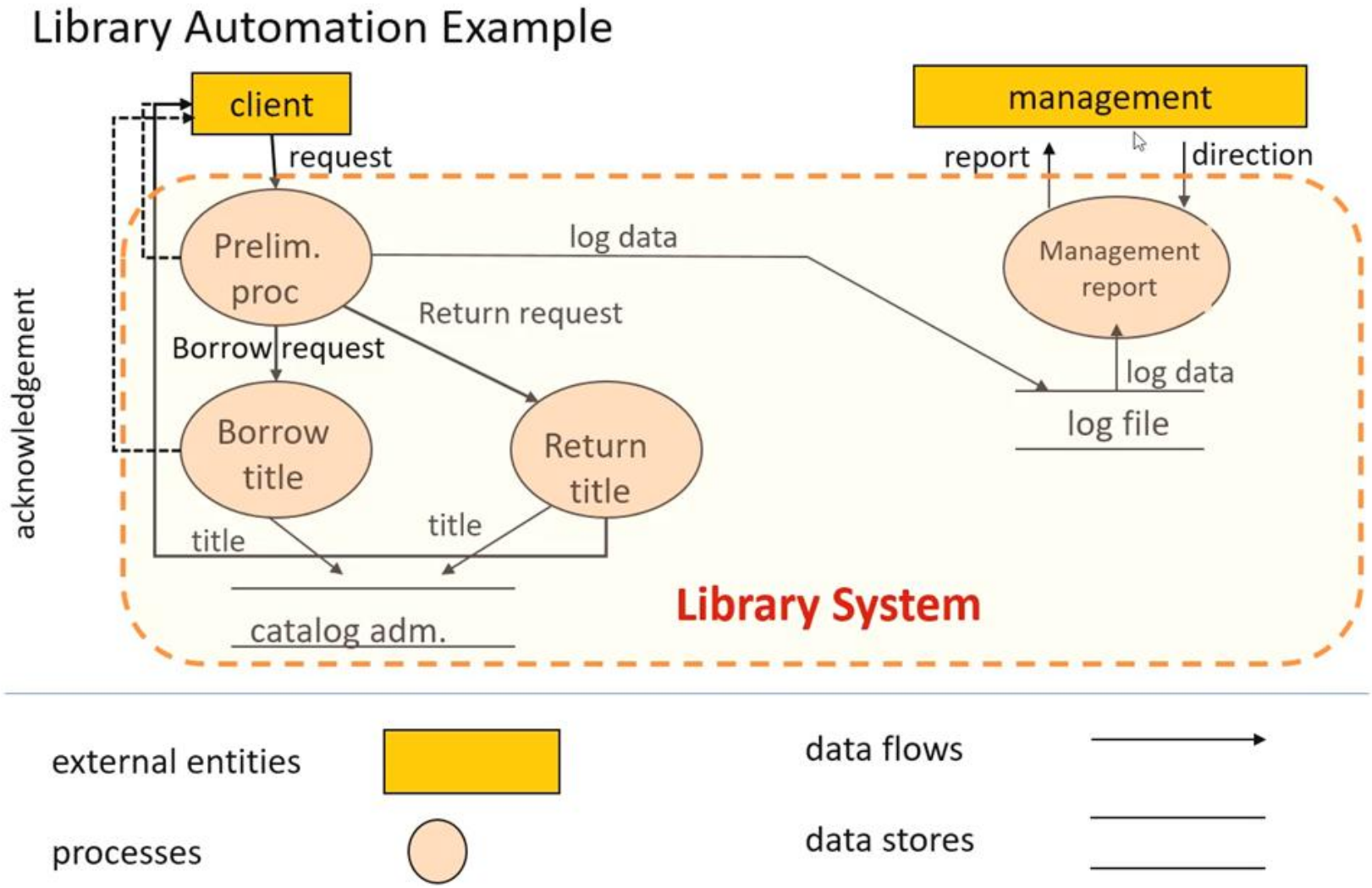
Example – A Library Management System

If we consider an example of Library Automation



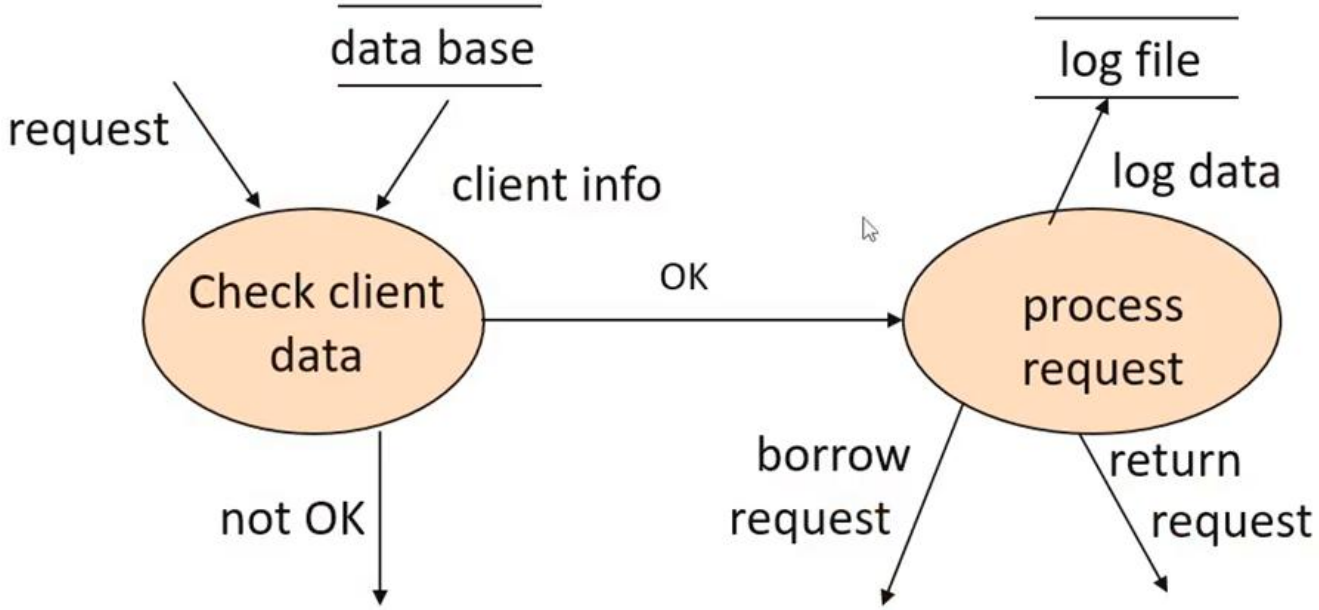

# Software Engineering

## First level decomposition



# Software Engineering

## Second level decomposition



# Software Engineering

## Minispecs

Once the process in DFDs become sufficiently straight forward and does not warrant further decomposition, then these processes are described as “minispecs”

For example,

Identification: Process request for a Book

Description:

1. Enter type of request
    - 1.1 If invalid, issue warning and repeat step 1
    - 1.2 If step 1 repeated 5 times, terminate transaction
2. Enter book identification
    - 2.1 If invalid, issue warning and repeat step 2
    - 2.2 If step 2 repeated 5 times, terminate transaction
3. Log client identification, request type and book identification

# Data dictionary entries

The contents of the DFDs after we are at a logical decomposed state are recorded in a data dictionary. It is a precise description of the structure of data.

A data dictionary is **a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them**. Often a data dictionary is a centralized metadata repository.

# Software Engineering

## Data dictionary entries

borrow-request = client-id + book-id

return-request = client-id + book-id

log-data = client-id + [borrow | return] + book-id

book-id = author-name + title + (isbn) + [proc | series | other]

Conventions:

- [ ]: include one of the enclosed options
- |: separates options
- +: AND
- (): enclosed items are optional

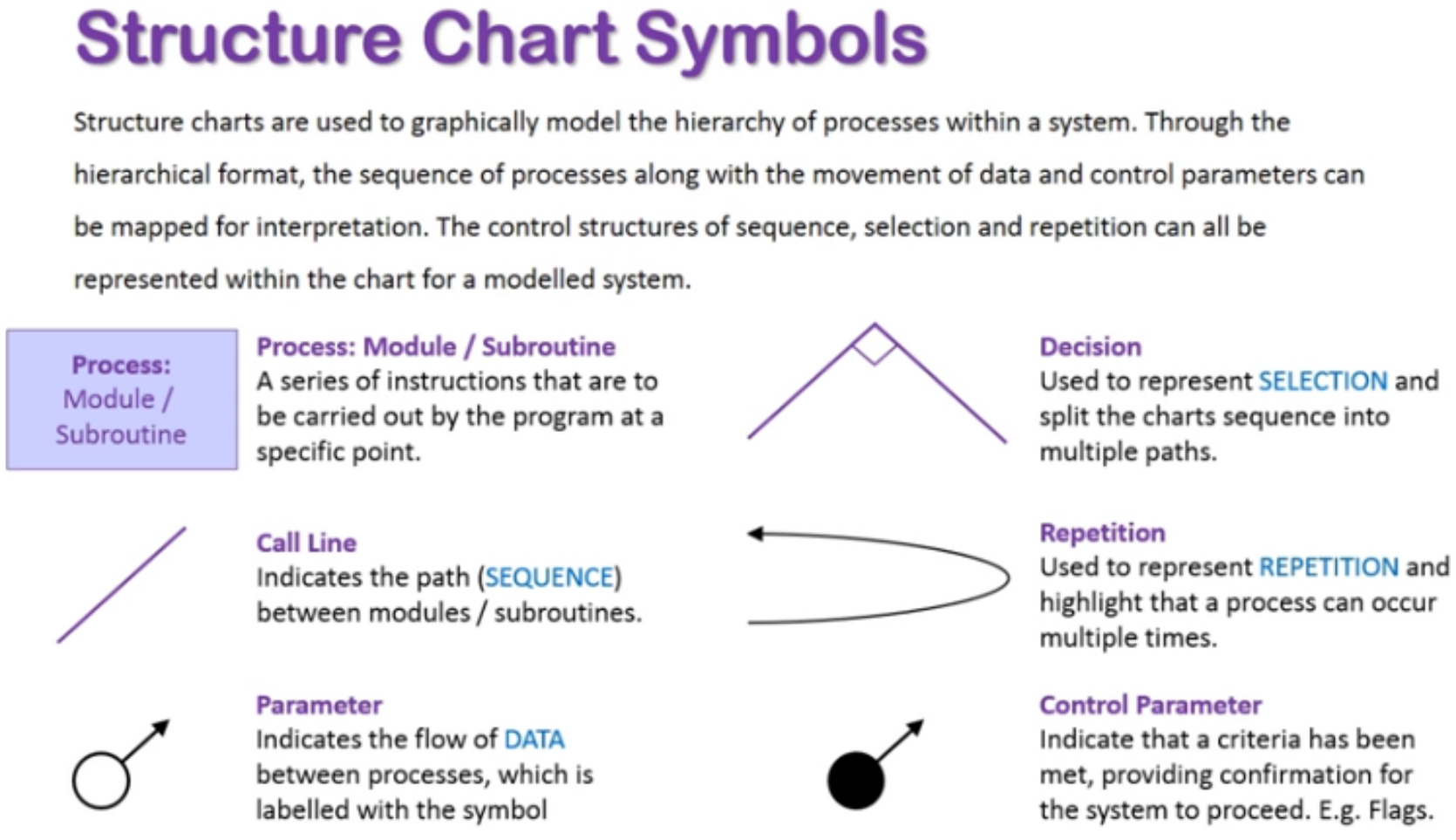
# Software Engineering

## Transform-centred design

Structure chart is **a chart derived from Data Flow Diagram**. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

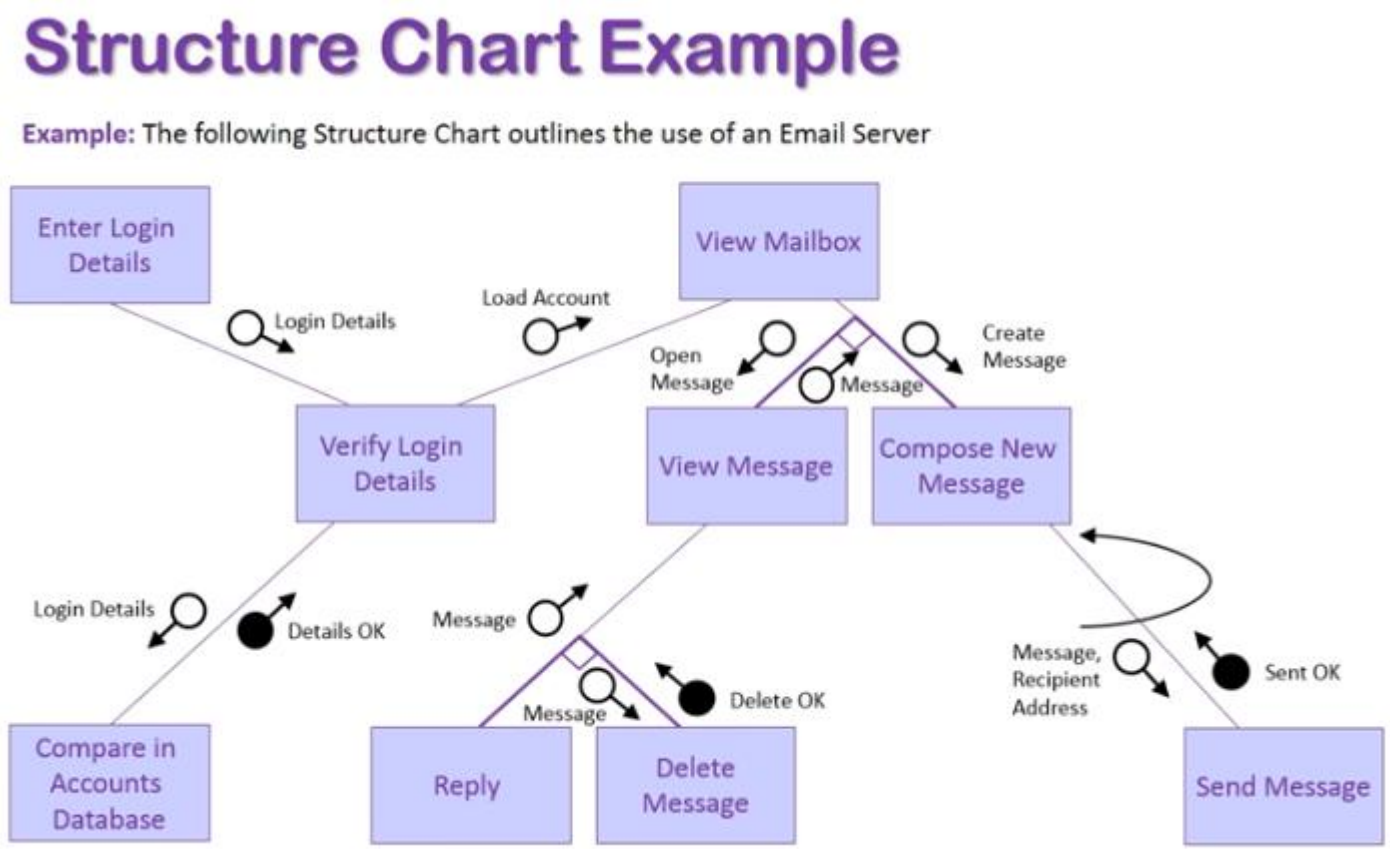
# Software Engineering

## Transform-centred design



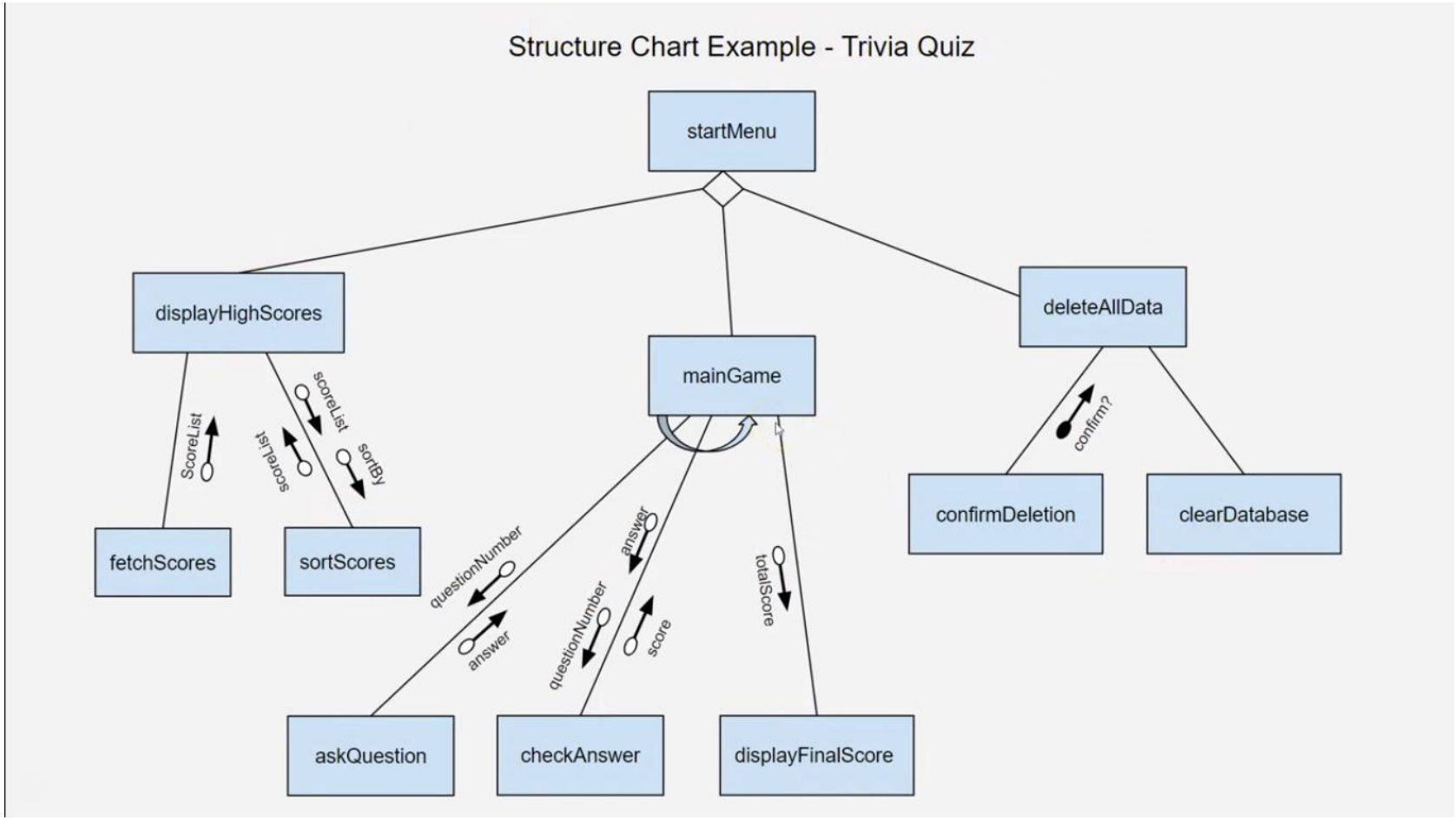
# Software Engineering

## Transform-centred design



# Software Engineering

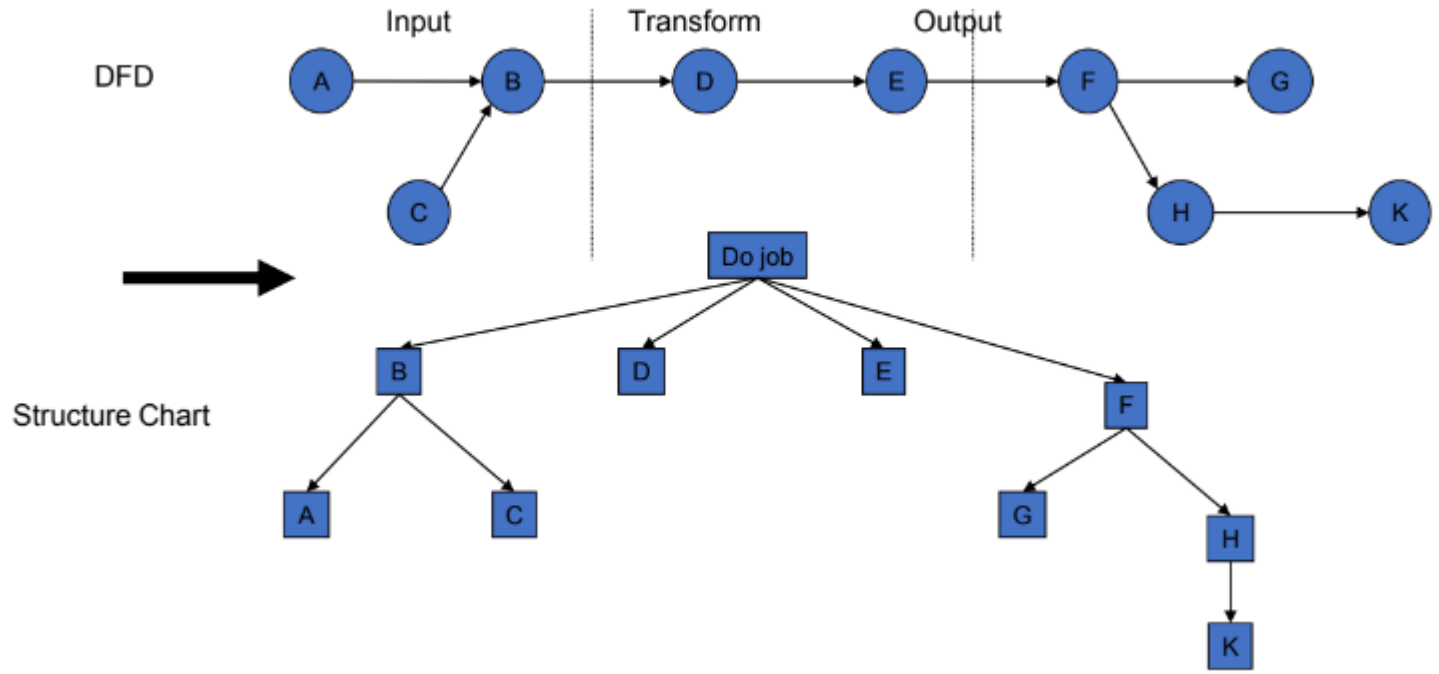
## Transform-centred design



# Software Engineering

## Transform-centred design

Trace the input through the data flow diagram until it can no longer be considered input.



# Software Engineering

## 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.

**Types of design patterns**

| Procedural | Object oriented (Gang of Four) (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) |
|---|---|
| Structural decomposition | Creational |
| Organization of work | Structural |
| Access control | Behavioral |
| Management | Distribution |
| Communication | |

# Software Engineering

## Procedural patterns

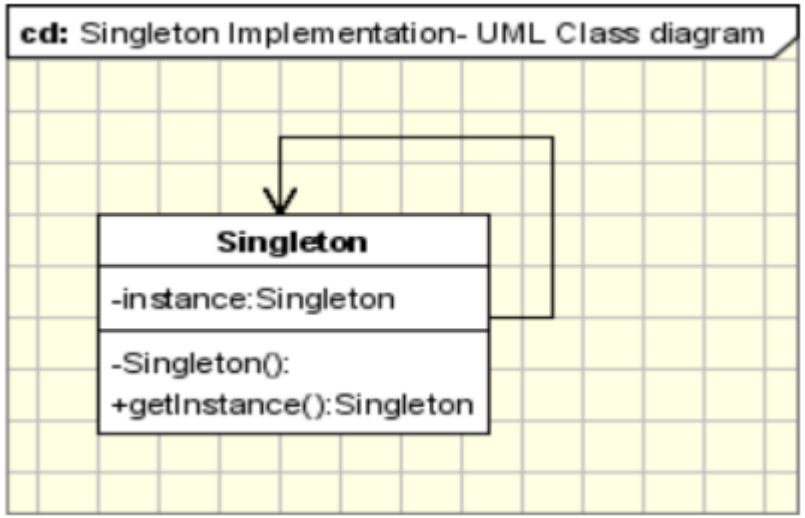
| | |
|---|---|
| **Structural decomposition pattern** | Breaks down a large system into subsystems and complex components into co-operating parts, such as ***a product breakdown structure*** |
| **Organization of work pattern** | defines how components work together to solve a problem, such as ***master-slave 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*** |

# Software Engineering

## Singleton pattern

Ensures that only one instance of a class is created
Provide a global point of access to the object. This is useful when only one object is needed to coordinate actions across the system.
It involves only one class which is responsible to instantiate itself.

The implementation involves a static member in the "Singleton" class, a private constructor and a static public method that returns a reference to the static member




```
class Abc
{
     static Abc obj = new Abc();
      private Abc()
      {

      }
      public static Abc getInstance()
      {
           return obj;
      }
}
```

# Software Engineering

## Anti pattern

An **anti-pattern** is a widespread but ineffective, risky, and/or unproductive approach to solving a class of common problems. In other words, this is a pattern of mistakes (also sometimes called a trap).

Examples

**God object**

A **God object** is an anti-pattern that describes an excessive concentration of all sorts of functions and large amounts of disparate data (an object that the application revolves around).

**Poltergeist**

A **poltergeist** is an anti-pattern involving a pointless class that is used to call methods of another class or simply adds an unnecessary layer of abstraction. This anti-pattern manifests itself as short-lived objects, devoid of state. These objects are often used to initialize other, more permanent objects.

# Software Engineering

## Structured and Object oriented approaches- A comparison

| Comparison Factor | Structural/Function Oriented Approach | Object Oriented Approach |
|---|---|---|
| Abstraction | The basic abstractions, which are given to the user, are real world functions, processes and procedures | The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented. |
| Lifecycles | It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system. | It uses Incremental or Iterative methodology to refine and extend the design. |
| Function | Functions are grouped together by which a higher level function is obtained. | Function are grouped together on the basis of the data they operate since the classes are associated with their methods. |
| State information | In this approach the state information is often represented in a centralized shared memory. | In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system. |
| Approach | It is a top down approach. | It is a bottom up approach. |

# Software Engineering

## Structured and Object oriented approaches- A comparison

| Comparison Factor | Structural/Function Oriented Approach | Object Oriented Approach |
|---|---|---|
| Begins basis | Begins by considering the use case diagrams and the scenarios. | Begins by identifying objects and classes. |
| Decompose | In function oriented design decomposition is in function/procedure level. Stepwise refinement is based on the iterative procedural decomposition and a program is refined as a hierarchy of increasing levels of details. | Decomposition is in class level. It begins with an examination of the real world "things". These things are characteristics individually in terms of their attributes and behavior. |
| Design approaches | Typically would use DFDs (Data Flow Diagram), Structured English, ER (Entity Relationship) diagram, Data Dictionary, Decision table/tree, State transition diagram. | This looks at class diagrams, component diagrams, deployments for static design and uses interaction diagrams, state diagrams for dynamic part of the design |
| Design techniques | Design enabling techniques like abstraction, security, data hiding, abstraction, inheritance etc. will need to be implemented specifically for the benefits | Object Orientation by its approach, constructs and languages support quite a few of those and promotes communication within objects through the means of message passing. |

# Software Engineering

## Structured and Object oriented approaches- A comparison

| Comparison Factor | Structural/Function Oriented Approach | Object Oriented Approach |
|---|---|---|
| Design Implementation | Functions are described and called to perform the specific tasks, wherein the data is not encapsulated with the functions. This was a major problem that comes up with the traditional approach where data is global and not encapsulated within any model object. | Object oriented programming has all the components of the system as a real entity having attributes and functions linked with it. A blueprint or prototype of any entity can be described as a class, and various objects can be created from this. |
| **Ease of development** | Easier although it depends on the size of the software programs | object oriented approach depend on the experience of the development team and complexity of the programs |
| Use | This approach is mainly used for computation sensitive application. | This approach is mainly used for evolving system which mimics a business or business case. |

# Software Engineering

## Service Oriented Architecture (SOA)

Service-Oriented Architecture is an approach for building software that incorporates complete enterprise development.

SOA enables development teams to reuse components from other pre-existing applications across the entire enterprise.

SOA promotes the development of a collection of modular services and can communicate with each other to support a system of applications.

This architecture enables communication between various platforms and languages.

SOA bases its structure on decreasing the coupling between application components, also known as loose coupling or decoupling. So two components can easily communicate with each other even if they are entirely different.

This improves business functionality and makes software development more convenient.

# Software Engineering

## Service Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is an architectural style that supports a way to make software components reusable via service interfaces.

Each service in an SOA embodies the code and data integrations required to execute a complete, discrete business function.

The services are exposed using standard network protocols—such as SOAP (simple object access protocol)/HTTP or JSON/HTTP—to send requests to read or change data.

SOA can also be looked at as a software design methodology based on structured collections of discrete software modules, known as services.

SOA's origin comes from the concept of "separation of concerns" breaking down a large problem into a series of smaller, individual problems or concerns.

## Benefits of SOA

**Greater business agility; faster time to market**

**Ability to leverage legacy functionality in new markets**
SOA enables developers to easily take functionality in one computing platform to another

**Improved collaboration between business and IT**

**Service reusability**

# Services

**What is a service?**

• 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

# Software Engineering

## Characteristics of services

- Adhere to a service contract
- Loosely coupled
- Stateless
- Autonomous
- Abstraction
- Reusable
- Open standards
- Facilitate interoperability
- Can be discovered
- Can be composed to form larger services

## SOA guiding principles

**Standardized service contract:** Specified through one or more service description documents.

**Loose coupling:** Services are designed as self-contained components, maintain relationships that minimize dependencies on other services.

**Abstraction:** A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.

**Reusability:** Designed as components, services can be reused more effectively, thus reducing development time and the associated costs.

## SOA guiding principles

**Autonomy:** Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.

**Discoverability:** Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.

**Composability:** Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

# SOA roles

**Service provider**

A service provider creates web services and provides them to a service registry.

**Service broker or service registry**

A service broker or service registry is responsible for providing information about the service to a requester.

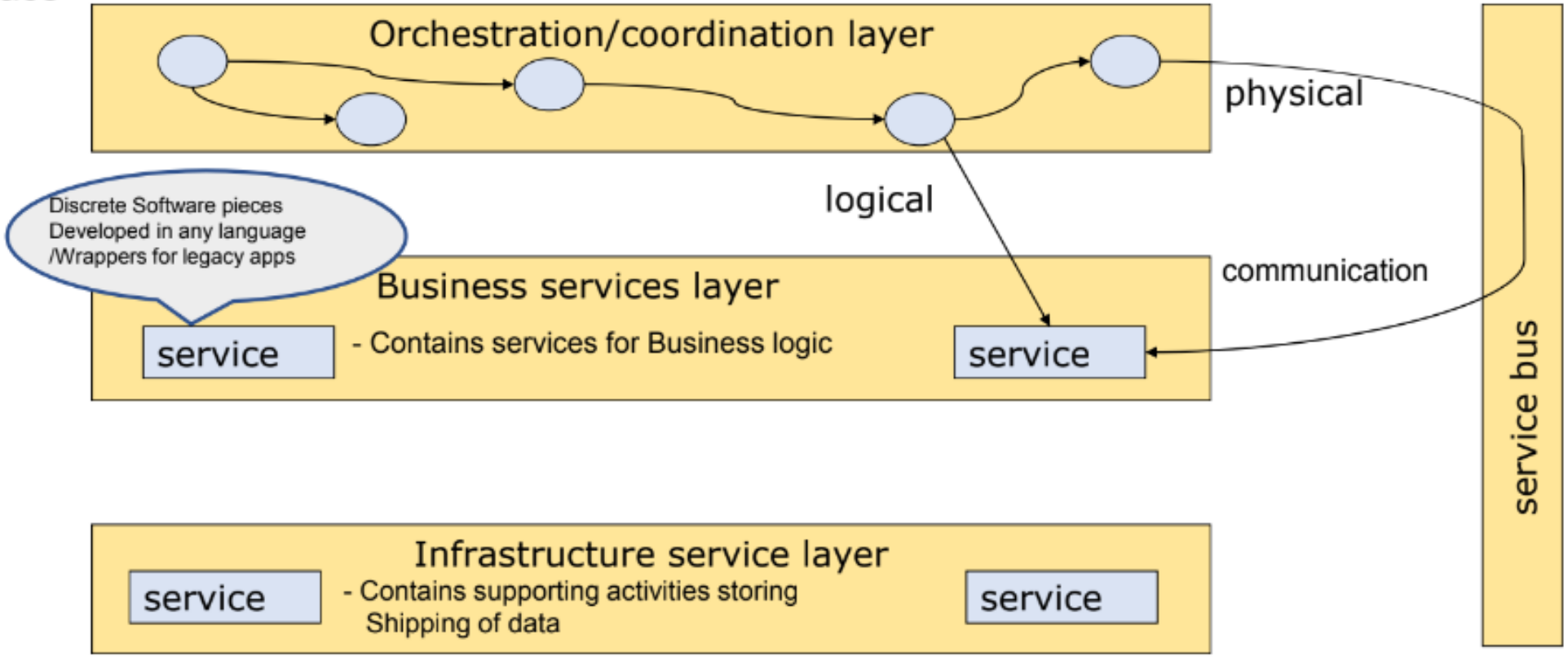
**Service requester or service consumer**

A service requester finds a service in a service broker or service registry and then will connect with the service provider to receive the service.

# Software Engineering

## SOA

Typical architecture with two layers of services, communicating through service bus, with an orchestration layer. There also exists a Service bus through which the communication takes place



# SOA

Example

OpenStack

For more information, visit the following link:
https://scoutapm.com/blog/soa-vs-microservices

# Software Engineering

## In-class exercise

**Draw a contrast between SOA & Microservice architecture**

SOA is an enterprise-wide approach to architecture, while microservices is an implementation strategy within application development teams.

SoA communicates with its respective components using an ESB (Enterprise Service Bus) while microservices can communicate with each other statelessly using language agnostic APIs.

Microservices are more tolerant and flexible as it allows development teams to choose what tools they want to work with as these APIs which are language agnostic.


# THANK YOU

**M S Anand**
Department of Computer Science Engineering

**anandms@yahoo.com**