# UNIT-1

## INTRODUCTION

- Software Engineering Process paradigms
- Project management
- Process and Project Metrics
- Software estimation
- Empirical estimation models
- Planning
- Risk analysis
- Software project scheduling.

SOFTWARE ENGINEERING:-

- Software engineering is an engineering discipline that is concerned with all aspects of software production from early stages of system design specification through to maintaining the system after it has gone into use.
- Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production.

SOFTWARE PROCESS:-

A systematic approach that is used in software engineering is called a **software process**. A software process is a sequence of activities that leads to the production of a software product. There are four fundamental activities that are common to all software processes.

1. Software Specification:- customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software Development:- software is designed and programmed.
3. Software Validation:- software is checked to ensure that it is what the customer requires.
4. Software Evolution:- the software is modified to reflect changing customer and market requirements.

**SOFTWARE ENGINEERING PROCESS PARADIGMS:**

A process framework establishes the foundation for a complete software engineering process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size and complexity.

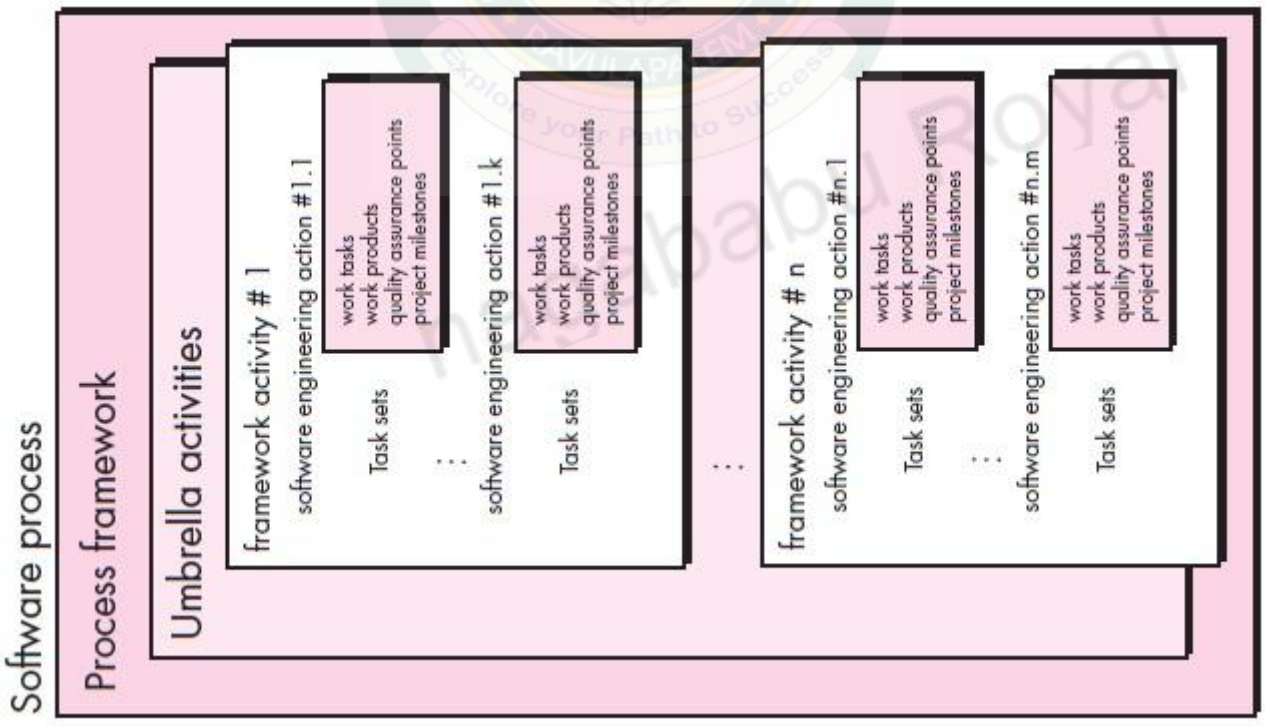
A generic process framework for software engineering has five activities.

1. **Communication**:- Before commencement of any technical work it is important to communicate with customer to understand objectives for the project and gather requirements that help define software features and functions.
2. **Planning:-** Software project plan defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely to be raised, the resources that will be required, the work products to be produced and a work schedule.
3. **Modeling:-** Creating models that better understand d requirements and the design that will achieve those requirements.
4. **Construction:-** This activity combines code generation and the testing that is required to handle errors.

5. **Deployment:-** The software is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

These five generic framework activities can be used during the development of small, simple programs, the creation of large web applications and for engineering of large and complex computer-based systems.

- Software engineering process framework activities are complemented by a number of umbrella activities.
- These activities are applied throughout a software project and help software team manage and control progress, control quality, change and risk.
  1. ***Software project tracking and control:-*** Allows the software team to evaluate progress against the project plan and take any necessary action to maintain the schedule.
  2. ***Risk Management:-*** Evaluate risks that may affect the outcome of the project or the quality of the product.
  3. ***Software Quality Assurance:-*** Defines and conducts the activities required to ensure software quality.
  4. ***Technical Reviews:-*** Evaluates software engineering work products in an effort to uncover and remove errors before they moved to the next activity.
  5. ***Measurement:-*** Defines and collects process, project and product measures that assist the team in delivering software that meet customer needs.
  6. ***Software Configuration Management:-*** Manages the effects of change throughout the software process.
  7. ***Reusability Management:-*** Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
  8. ***Work Product Preparation and Production:-*** Encompasses the activities required to create work products such as models, documents, logs, forms and lists.



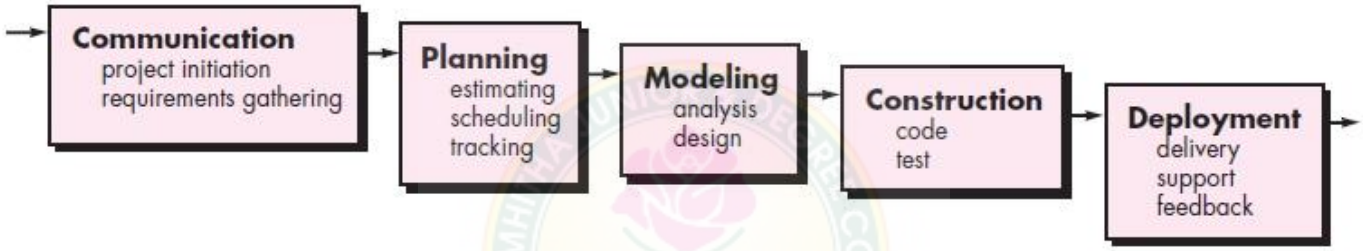
**SOFTWARE PROCESS MODEL:**

The development strategy that includes the process, methods and tools to solve actual problem in an industry is referred as a process model or a software engineering paradigm.

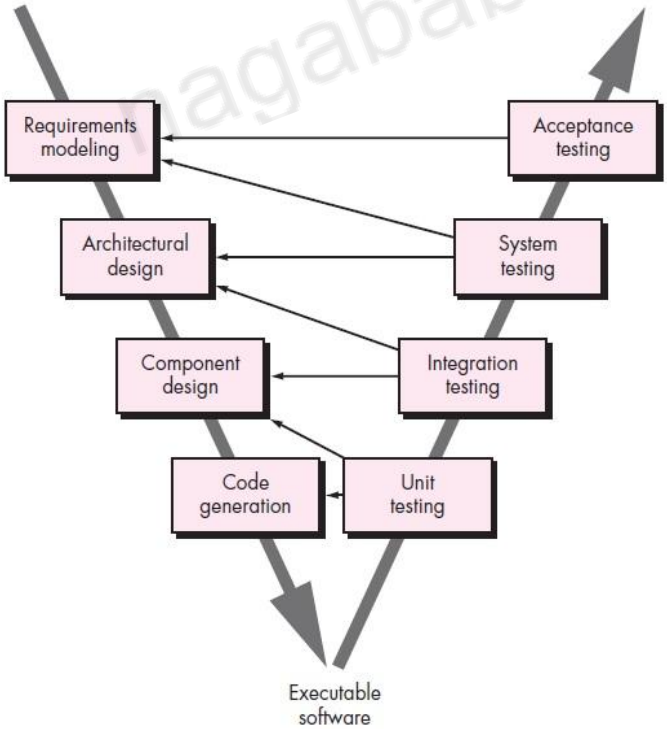
A process model for software engineering depends on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required.

**WATERFALL MODEL (or) LINEAR SEQUENTIAL MODEL (or) CLASSICAL LIFE CYCLE MODEL:**

1. The waterfall model is the oldest paradigm for software engineering.
2. The waterfall model suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction and deployment, finally provides support for the completed software.
3. In waterfall model each phase must be completed fully before the next phase can begin.
4. At the end of each phase, a review takes place to determine whether the project is on the right path and whether to continue or discard the project.




5. A variation in the representation of the waterfall model is called the V-Model.
6. The V-Model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early construction activities.
7. There is no fundamental difference between the waterfall model and V-Model.
8. The V-Model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.



**Advantages:-**

1. Simple to implement and manage.
2. Enforces a disciplined software development approach.
3. Phases are processed and completed one at a time hence phases do not overlap.
4. Works well for smaller projects where requirements are very well understood.

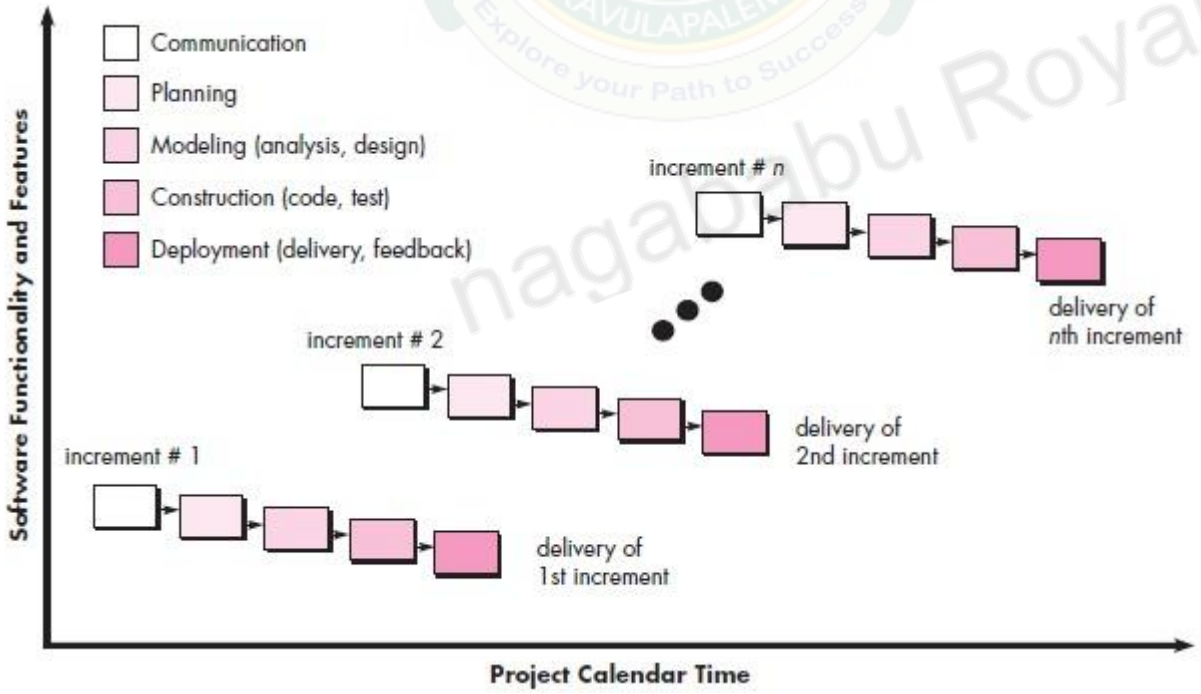
**Disadvantages:-**

1. Unable to perform changes at a later stage.
2. A working version of the software is not available during development.
3. Unable to perform iteration.
4. Deadlock can occur due to any delay in step.

**INCREMENTAL PROCESS MODEL:**

- ❖ The incremental process model combines elements of linear and parallel process flows.
- ❖ In incremental process model, initially a partial implementation of a total system is constructed so that it will be in a deliverable state.
- ❖ The first increment is the ***core product*** when an incremental process model is applied.
- ❖ New functionality is added.
- ❖ If there are any defects in previous delivery, they are fixed and the working product is delivered.
- ❖ The repetitions of these processes are called iterations.

  Consider the example:
  - → Word processing software is developed using the incremental paradigm might deliver basic file management, editing and document production functions in the first increment.
  - → Editing and document production capabilities in second increment.
  - → Spelling and grammar checking in the third increment.




**Advantages:-**

1. We can develop prioritized requirements first.
2. Initial product delivery is faster and customers get important functionality early.

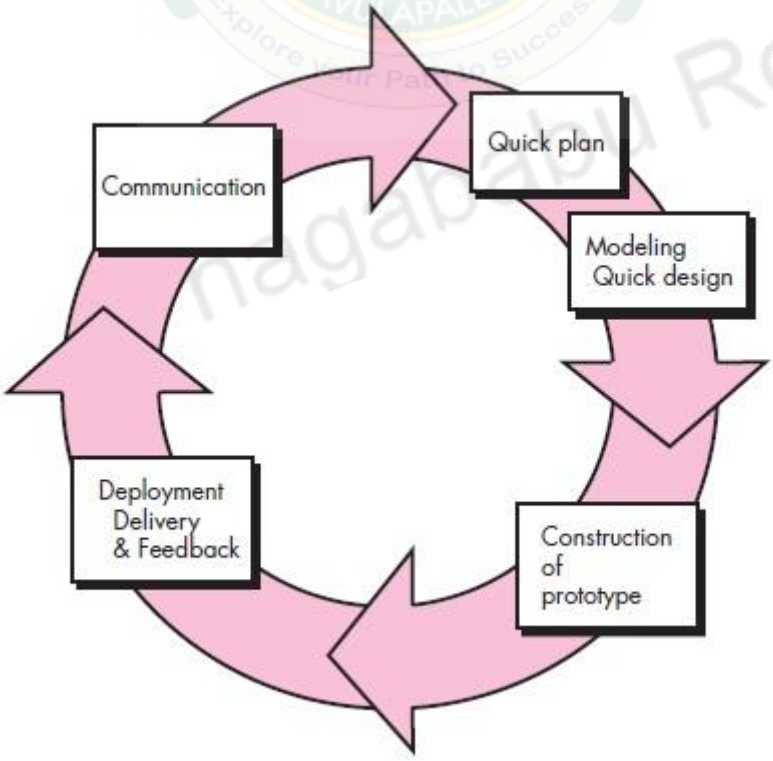
3. Reduces the initial delivery cost.
4. Each release is a product increment, so that the customer will have a working product all the time.
5. Requirement changes can easily accommodate.

**Disadvantages:-**

1. Well-defined module interfaces are required as some are developed long before others are developed.
2. Total cost of the complete system is too high.
3. Requires effective planning of increments.

**PROTOTYPE MODEL**

- A prototype is an initial version of the software system that is used to
  - → Demonstrate concepts
  - → Tryout design options
  - → Find out more about problem and its possible solutions
- A software prototype can be used in software development process to help expect changes that may be required
  - → In the requirements engineering process
  - → In the system design process
- Software prototype allows users to see how well the system supports their work.
- They may get new ideas for requirements and find areas of strengths and weakness in the software.
- As the prototype is developed, it may reveal errors and omissions in the requirements that have been proposed.
- The Prototypes are usually not complete systems and many of the details are not built in the prototype.
- Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining requirements.



**Advantages:-**

1. Customers can see steady progress.
2. This is useful when requirements are changing rapidly.
3. Errors can be detected much earlier.
4. Quicker user feedback is available leading to better solutions.
5. Missing functionality can be identified easily.

**Disadvantages:-**

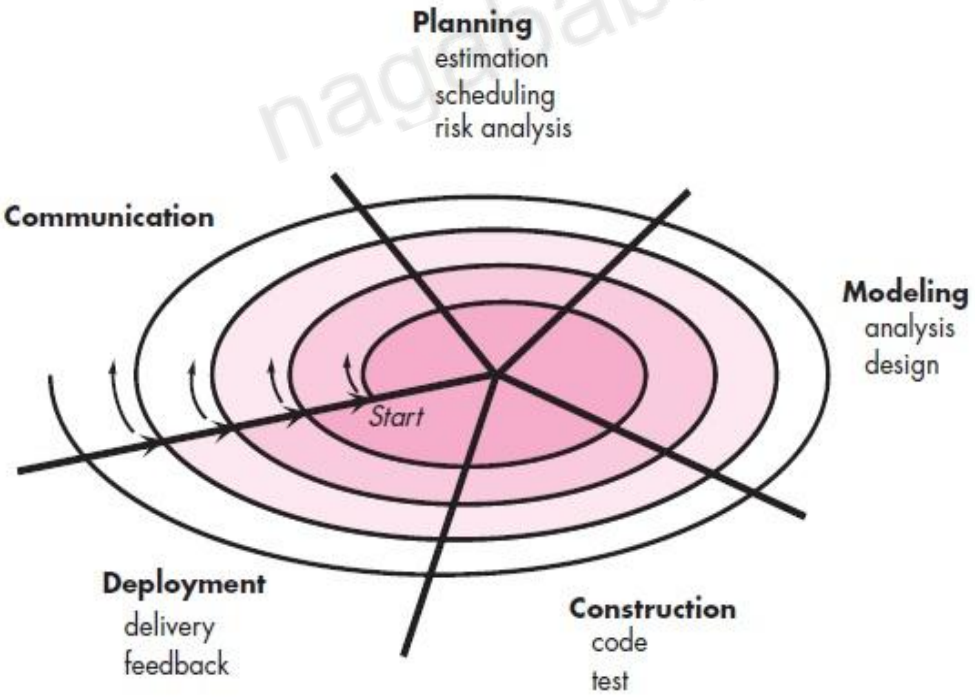
1. It is impossible to know how long it will take to complete.
2. There is no way to know the number of iterations required.
3. This methodology may increase the complexity of the system as scope of the system may expand.

**SPIRAL MODEL**

- Spiral model was originally proposed by Boehm.
- Rather than represent the software process as a sequence of activities with some back tracking from one activity to another, the process is represented as a spiral.
- Each loop in spiral model represents a phase of the software process.
- It is a combination of waterfall model and iterative process model.
- Each phase in spiral model begins with design goals and ends with client reviewing.

When to use spiral model:-

1. When project is too large
2. When releases are required to be frequent
3. When risk and cost evaluation is important
4. When requirements are unclear and complex
5. When changes may require at any time
6. For medium to high risk projects



**Advantages:-**

1. Additional functionality or changes can be done at later stages
2. Cost estimation becomes easy.
3. Development is fast and features are added in a systematic way.
4. There is always space for customer feedback.

**Disadvantages:-**

1. Management is more complex.
2. End of the project may not be known early.
3. Large number of intermediate stages requires excessive documentation.
4. Expensive for small projects.
5. Not suitable for small or low risk projects.

**RAD MODEL:**

- RAD (Rapid Application Development) is a type of incremental model.
- In RAD model the components or functions are developed in parallel as if they were in mini projects.
- The developments are time boxed, delivered and then assembled into a working prototype.

When to use RAD model:

1. When there is a need to create a system that can be modularized in 2 to 3 months of time.
2. If there is high availability of designers for modeling
3. The budget is high enough to afford their cost along with the cost of automated code generating tools.
4. If resources with high business knowledge are available.
5. Need to produce the system in a short span of time.

Phases in RAD model: RAD model contains five phases.

1. Business Modeling:- The information flow is identified between various business functions.
2. Data Modeling:- Information gathered from business modeling is used to define data objects that are needed for the business.
3. Process Modeling:- Data objects defined in data modeling are converted to achieve the business information flow to achieve some specific business objectives.
4. Application Generation:- Automated tools are used to convert process models into code and the actual system.
5. Testing and Turnover:- New components and all interfaces are tested.

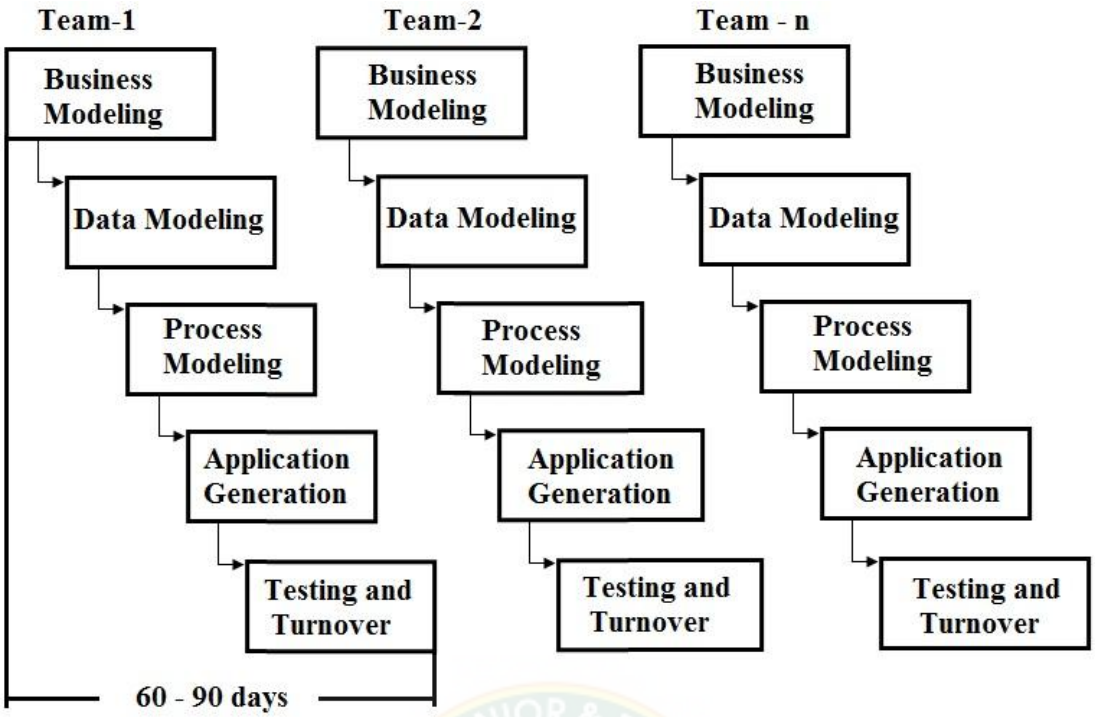
**Advantages:-**

1. Reduced development time.
2. Increases reusability of components.
3. Quick initial reviews occurred.
4. Integration from beginning solves a lot of integration issues.

**Disadvantages:-**

1. Requires highly skilled developers.
2. High dependency on modeling skills.

3. Not applicable to cheaper projects as cost of the modeling and automated code generation is very high.




## PROJECT MANAGEMENT

1. Software project management is an essential part of software engineering.
2. Projects need to be managed because professional software engineering is always subject to organizational budget and schedule constraints.
3. The project manager's job is to ensure that the software project meets and overcomes these constraints as well as delivering high-quality software.
4. Good management cannot guarantee project success and bad management usually results in project failure.

Important Goals of Software Project Management:

- ↪ Deliver the software to the customer at the agreed time.
- ↪ Keep overall costs within budget.
- ↪ Deliver software that meets the customer's expectations
- ↪ Maintain a happy and well-functioning development team.

Most managers take responsibility for some or all of the following activities:

1. ***Project Planning:-*** Project managers are responsible for
    a. Planning
    b. Estimating
    c. Scheduling
    d. Project development
    e. Assigning people to tasks
2. ***Reporting:-*** Project managers are usually responsible for reporting on the progress of a project to customers and to managers of the company developing the software.
3. ***Risk Management:-*** Project managers have to evaluate the risks that may affect a project, monitor these risks, and take action when problems arise.

4. ***People Management:-*** Project managers are responsible for managing a team of people. They have to choose people for their team and establish ways of working that lead to effective team performance.
5. ***Proposal Writing:-*** The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work. The proposal describes the objectives of the project and how it will be implemented.

**Risk Management:**

Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed and then taking actions to avoid these risks.

There are three types of risks.

1. Project Risk:- Risks that affect project schedule or resources.
2. Product Risk:- Risks that affect the quality of the software being developed.
3. Business Risk:- Risks that affect the organization development.

The process of risk management involves several stages:

a. Risk Identification:- You should identify possible project, product and business risks.
b. Risk Analysis:- You should analyses the consequences of the risks identified in risk identification stage.
c. Risk Planning:- You should make plans to address the risk either by avoiding it or minimizing its effect on the project.
d. Risk Monitoring:- You should regularly evaluate the risks.

The risk management process is an iterative process that continues throughout the project.

**Managing People:**

A project manager should be aware of the potential problems of people management and should try to develop people management skills.

There are four critical factors in people management:

1. Consistency:
   → People in a project team should all be treated in a same way.
   → People should not feel that their contribution to the organization is undervalued.
2. Respect:
   → Different people have different skills and managers should respect these differences.
   → All members of the team should be given an opportunity to make a contribution.
3. Inclusion:
   → People contribute effectively when they feel that others listen to them and take account of their proposals.
4. Honesty:
   → A manager should always be honest about what is going well and what is going badly in the team.

# PROCESS AND PROJECT METRICS

Software metrics provides measures for various aspects of software process and software product. There are various types of metrics.

→Process Metrics → Project Metrics → Product Metrics → Organizational Metrics

**Measure:-**A quantitative indication of the extent, amount, dimension, capacity or size of some attribute of a product or process.

**Measurement:-**The act of determining a measure.

Measurement is a management tool that assists the project manager and the software team in making decisions that will lead to a successful project.

**Metric:-**A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

**Indicator:-**A Metric or combination of metrics that provides insight into the software process, a software project, or the product itself.

A software engineer collects measures and develops metrics, so that indicators will be obtained.

**PROCESS METRICS:**

1. Process metrics are collected across all projects and over long period of time.
2. Their intent is to provide a set of process indicators that lead to long term software process improvement.

Process Metrics are as follows:

→ Statistical Software Process Improvement(SSPI):
→ Direct Removal Efficiency(DRE)

**Statistical Software Process Improvement:**

It is an approach and it uses software failure analysis to collect information about all errors and defects encountered as an application is developed and used.
The information is as follows:

1. All errors and defects are categorized by origin.
2. The cost to correct each error and defect is recorded.
3. The number of errors and defects in each category is computed.
4. Plans are developed to modify the process.

**Direct Removal Efficiency:**

A quality metric that provides benefit at both project and process level is direct removal efficiency.
Direct Removal Efficiency is a measure of the filtering ability of quality assurance and control actions.
DRE is defined as follows:

$$DRE = \frac{E}{E + D}$$

E → Number of errors found before delivery of software.

D → Number of defects found after the delivery of software.

The Ideal value for DRE is 1, i.e., no defects found in the software.

**PROJECT METRICS:**

These are metrics that pertain to project quality. They are used to quantify defects, cost, schedule, productivity and estimation of various project resources.

These metrics used by a project manager and software team to adapt project work flow and technical activity. The purpose of project metrics as follows:

- → Minimize the development schedule by making the necessary adjustment to avoid delivery and mitigate problems.
- → Evaluate product quality on an ongoing basis.

The following are project metrics:

1. Effort or time per software engineering task
2. Error uncovered per review hour
3. Scheduled versus actual milestone dates
4. Number of changes and their characteristics
5. Distribution of effort on software engineering task

Schedule Variance:

Any difference between the scheduled completion of an activity and actual completion of an activity is known as schedule variance.

$$schedule\ variance = \frac{((actual\ calander\ days - planned\ calander\ days) + start\ variance)}{planned\ calander\ days} X100$$

Effort Variance:

Difference between the planned outlined efforts and the efforts required to actually undertake the task is called effort variance.

$$Effort\ Variance = \frac{(Actual\ effort - Planned\ effort)}{Planned\ effort} X100$$

# SOFTWARE ESTIMATION

To achieve reliable cost and effort estimates, a number of options arise:

1. Delay estimation until late in the project
2. Base estimates on similar projects that have already been completed.
3. Use relatively simple decomposition techniques to generate project cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

The following are various estimation techniques:

- Past (similar ) project experience
- Conventional estimation techniques
  a. Task breakdown and effort estimates
  b. Size estimates
- Empirical models
- Automated tools

**Decomposition techniques:**

***Software Sizing:***

The accuracy of a software project estimate is predicated on a number of things:

1. The degree to which the planner has properly estimated the size of the product to be built.
2. The ability to translate the size estimate into human effort, calendar time and dollars.
3. The degree to which the project plan reflects the abilities of the software team.
4. The ability of product requirements and the environment that supports the software engineering effort.

NOTE:- If a direct approach is chosen, size can be measured in lines of code (LOC) , if an indirect approach is chosen, size can be measured in function points(FP).

There are four different approaches to the sizing problem:

a. Fuzzy Logic sizing:-This approach uses the approximate reasoning techniques that are the cornerstones of fuzzy logic.
b. Function Point Sizing:-The planner develops estimates of the information domain characteristics.
c. Standard Component Sizing:-Software is composed of a number of different standard component that are generic to a particular application area.
d. Change Sizing:-This approach is used when a project encompasses the use of existing software that must be modified in some way as part of a project.

***Problem Based Estimation:***

- ✓ LOC and FP data are used in two ways during software project estimation.
  → As estimation variables to size
  → As baseline metrics collected from past projects
- ✓ LOC and FP estimation are distinct estimation techniques but both have a number of characteristics in common.
- ✓ LOC or FP is estimated for each function.
- ✓ The LOC and FP estimation techniques differ in the level of detail required for decomposition and the target of the partitioning.

- ✓ The greater the degree of partitioning, the more accurate estimates of LOC can be developed.
- ✓ The expected value for the estimation variable size (S) can be computed as a weighted average of the optimistic ($S_{opt}$), most likely ($S_m$) and pessimistic ($S_{pess}$) estimates.

$$S = \frac{S_{opt} + 4S_m + S_{pess}}{6}$$

***Process-Based Estimation:***

- ✓ Process-Based estimation begins with a delineation of software functions obtained from the project scope.
- ✓ A series of framework activities must be performed for each function.
- ✓ Once problem functions and process activities are melded, you estimate the effort that will be required to accomplish each software process activity for each software function.
- ✓ Average labor rates are then applied to the effort estimated for each process activity.
- ✓ Costs and effort for each function and framework activity are computed as the last step.

## EMPIRICAL ESTIMATION MODELS

a. An estimation model for computer software uses empirically derived formulas to predict effort as a function of Lines Of Code or Function Points.
b. The empirical data that support most estimation models are derived from a limited samples of projects.
c. No estimation model is appropriate for all classes of software and in all development environments.
d. The model should be tested by applying data collected from completed projects, and then comparing actual to predicted results.
e. A typical estimation model is derived using regression analysis on data collected from past software projects. The structure of such models is as follows:

$$E = A + BX(e_v)^C$$

Where A,B,C are empirically derived constants.
E is effort in person-month
$e_v$ is estimation variable

### COCOMO II Model

1. Barry Boehm introduced a hierarchy of software estimation models named COCOMO, for COnstructive COst MOdel.
2. COCOMO is the most widely used and discussed software cost estimation model in the industry.
3. It has evolved into a more comprehensive estimation model COCOMO II and is a hierarchy of estimation models that address the following areas:
   a. ***Application Composition Model:*** used during
      - ↪ Prototyping of user interfaces
      - ↪ Consideration of software and system interaction
      - ↪ Assessment of performance
      - ↪ Evaluation of technology
   b. ***Early Design Stage Model:*** This model is used once requirements have been stabilized and basic software architecture has been established.
   c. ***Post-Architecture Stage Model:*** Used during the construction of the software.
4. The COCOMO II models require sizing information.
5. There are three different sizing options are available as part of the model hierarchy:

a. Object points
b. Function points
c. Lines of source code

6. The COCOMO II application composition model uses object points.
7. The object point is an indirect software measure that is computed using
   a. Number of screens
   b. Reports
   c. Components likely to be required to build the application.
8. When software reuse is to be applied then percent of reuse is estimated.
9. The object point count is determined as
   NOP = (object points) X [(100 - % reuse)/100]
   Where NOP is new object points.
10. To derive an estimate of effort based on the NOP value, a "productivity rate"(PROD) must be derived.
11. The productive rate is determined as
    PROD = NOP / person-month
12. Now the estimate of project effort is computed as
    Estimated effort = NOP / PROD

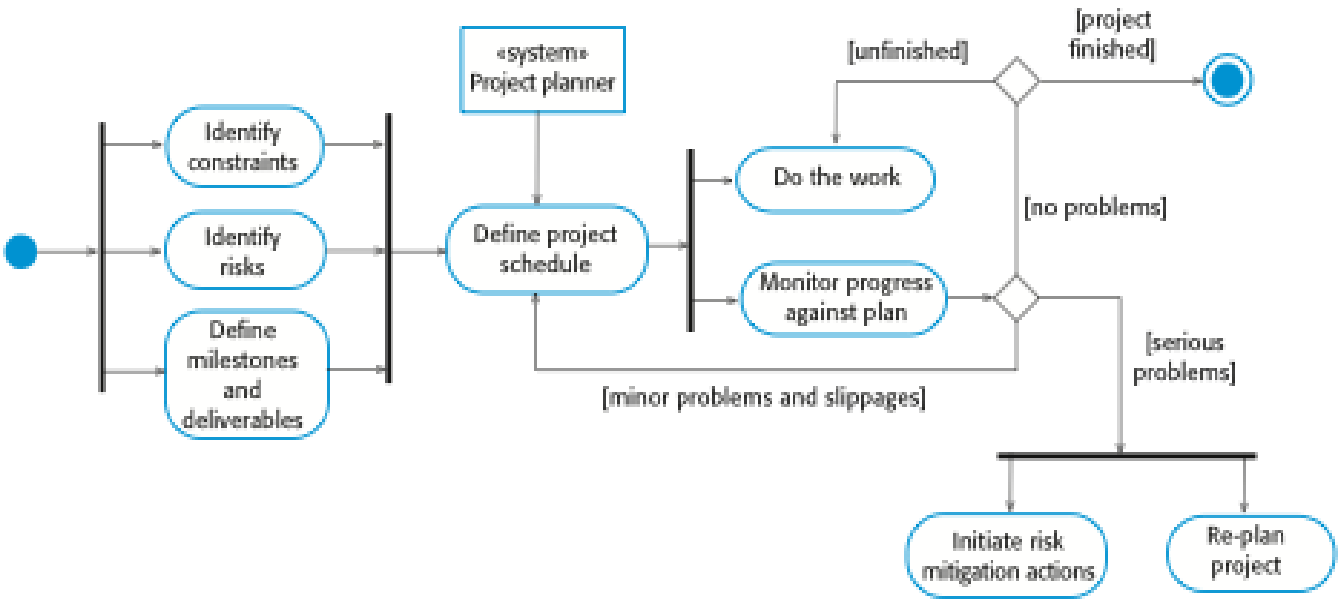
## PLANNING

1. A project plan sets out the resources available to the project, the work breakdown, and a schedule for carrying out the work.
2. The plan should identify risks to the project and software under development, and the approach that is taken to risk management.
3. Project plans vary depending on the type of project and organization.

Plans normally include the following sections:

- Introduction:-Describes the objectives of the project and sets out the constraints such as budget and time which affect the management of the project.
- Project Organization:-Describes the way in which the development team is organized, the people involved and their responsibilities in the team.
- Risk Analysis:-Describes possible project risks and risk reduction strategies.
- Hardware and Software resource requirements:-Specifies the hardware and support software required to carry out the development.
- Work Breakdown:-Sets out the breakdown of the project into activities and identifies the milestones and deliverables associated with each activity.
- Project Schedule:-The estimated time required to reach each milestone, and allocation of people to activities.
- Monitoring and Reporting:-Management reports that should be produced and the project monitoring mechanisms to be used.

4. Project planning is an iterative process that starts when you create an initial project plan during the project startup phase.
5. At the beginning, you should identify the constraints affecting the project, risks in the project and the milestones and deliverables.
6. The process then enters a loop. An estimated schedule for the project and the activities defined in the schedule are initiated.




7. If there are serious risks with the development, you need to initiate risk mitigation actions to reduce the risks or you can replan the project.
8. If there are minor problems with the development, you need to perform scheduling again.
9. If there are no problems with the development and if the task is unfinished then perform development and if the task is completed then check the milestones and deliverable.

## SOFTWARE PROJECT SCHEDULING

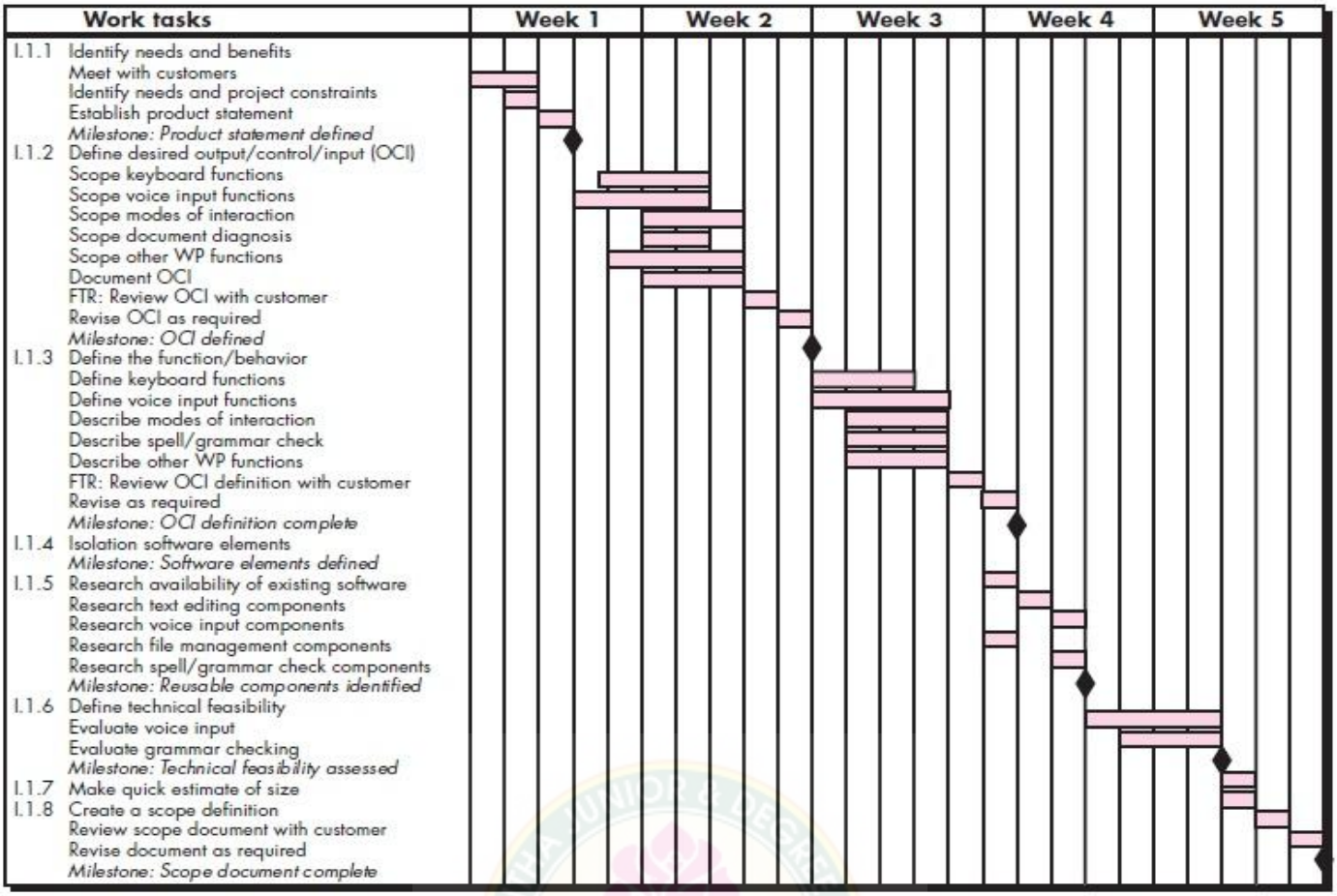
1. *Program Evaluation and Review Technique* (PERT) and the *Critical Path Method* (CPM) are two project scheduling methods that can be applied to software development.
2. Both PERT and CPM provide quantitative tools that allow you determine the critical path that determines the duration of the project and establish most likely time estimates for individual tasks and calculate boundary times.

### TIME LINE CHARTS:

a. A time line chart also called as Gantt chart.
b. When creating a software project schedule, you began with a set of tasks.
c. Effort, duration, and start date are then input for each task.
d. A time line chart can be developed for the entire project or separate charts can be developed for each project function.
e. All project tasks are listed in the left hand column.
f. The horizontal bars represents the duration of each task.
g. When multiple bars occur at the same time on the calendar, task concurrency is implied.
h. The diamonds indicate milestones.
i. Once the time line chart is generated then software project scheduling tools produce project tables.

### TRACKING THE SCHEDULE:

1. The project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds.
2. Tracking can be accomplished in a number of different ways.




| Work tasks | Planned start | Actual start | Planned complete | Actual complete | Assigned person | Effort allocated | Notes |
|---|---|---|---|---|---|---|---|
| I.1.1 Identify needs and benefits | | | | | | | Scoping will require more effort/time |
| Meet with customers | wk1, d1 | wk1, d1 | wk1, d2 | wk1, d2 | BLS | 2 p-d | |
| Identify needs and project constraints | wk1, d2 | wk1, d2 | wk1, d2 | wk1, d2 | JPP | 1 p-d | |
| Establish product statement | wk1, d3 | wk1, d3 | wk1, d3 | wk1, d3 | BLS/JPP | 1 p-d | |
| *Milestone: Product statement defined* | wk1, d3 | wk1, d3 | wk1, d3 | wk1, d3 | | | |
| I.1.2 Define desired output/control/input (OCI) | | | | | | | |
| Scope keyboard functions | wk1, d4 | wk1, d4 | wk2, d2 | | BLS | 1.5 p-d | |
| Scope voice input functions | wk1, d3 | wk1, d3 | wk2, d2 | | JPP | 2 p-d | |
| Scope modes of interaction | wk2, d1 | | wk2, d3 | | MLL | 1 p-d | |
| Scope document diagnostics | wk2, d1 | | wk2, d2 | | BLS | 1.5 p-d | |
| Scope other WP functions | wk1, d4 | wk1, d4 | wk2, d3 | | JPP | 2 p-d | |
| Document OCI | wk2, d1 | | wk2, d3 | | MLL | 3 p-d | |
| FTR: Review OCI with customer | wk2, d3 | | wk2, d3 | | all | 3 p-d | |
| Revise OCI as required | wk2, d4 | | wk2, d4 | | all | 3 p-d | |
| *Milestone: OCI defined* | wk2, d5 | | wk2, d5 | | | | |
| I.1.3 Define the function/behavior | | | | | | | |

a. Conducting periodic project status meetings in which each team member reports progress and problems
b. Evaluating the results of all reviews conducted throughout the software engineering process.
c. Determining whether formal project milestones have been accomplished by the schedule date.
d. Comparing the actual start date to the planned start date for each project task listed in the resource table.

All these tracking techniques are used by experienced project managers.

## UNIT-II

## REQUIREMENTS ANALYSIS

- Requirement Engineering Processes
- Feasibility Study
- Problem of Requirements
- Software Requirement Analysis
- Analysis Concepts and Principles
- Analysis Process
- Analysis Model

## REQUIREMENT ENGINEERING PROCESS:

The broad spectrum of tasks and techniques that lead to an understanding of requirements is called requirements engineering.

Requirements engineering is a major software engineering action that begins during communication activity and continues into the modeling activity.

Requirements engineering builds a bridge to design and construction. It provides the appropriate mechanisms for understanding

- What the customer wants
- Analyzing needs
- Assessing feasibility
- Negotiating a reasonable solution
- Specifying the solution unambiguously
- Validating the specification
- Managing the requirements as they are transformed into an operational system

Requirements engineering encompasses seven distinct tasks:

1. Inception
2. Elicitation
3. Elaboration
4. Negotiation
5. Specification
6. Validation
7. Management

**INCEPTION:**

a. Most projects begin when a business need is identified or a potential new market or service is discovered.
b. Stake holders define a business case for the idea, try to identify the breadth and depth of the market, do a rough feasibility analysis.
c. In Inception task, you establish a basic understanding of the problem, the people who want a solution, the nature of the solution and the effectiveness of preliminary communication and collaboration between stakeholders and software team.

**ELICITATION:**

a. What are the objectives for the system or product are gathered from customers and users and others.
b. We need to identify how the system or product fits into the needs of the business and it is to be used on a day-to-day basis.
c. The following problems occur while the elicitation process is performed.
    1. ***Problem of Scope:-***The boundary of the system is not properly defined or the customers specify unnecessary technical details that may confuse the software team rather than clarity.
    2. ***Problem of Understanding:-***The customers/users don't have a full understanding of the problem domain, specify requirements that conflict with the needs of other customers, specify requirements that are ambiguous or unstable.
    3. ***Problem of Volatility:-***The requirements are not stable, changed over time.

**ELABORATION:**

a. The information obtained from inception and elicitation is expanded and refined.
b. Used to develop a requirement model that identifies various aspects of the software function, behavior and information.
c. The attributes of each analysis class are defined and the services that are required by each class are identified.
d. The relationships and collaboration between classes are identified, and a variety of diagrams are produced.

**NEGOTIATION:**

a. A common communication problem is customers or users propose conflicting requirements and these are essential for their special need.
b. The software team has to reconcile these conflicts through a process of negotiation. We will perform the following tasks here.
    1. Prioritize requirements using an iterative approach
    2. Assesses their cost and risk
    3. Address internal conflicts
    4. Unnecessary requirements are eliminated, combined and/or modified.

**SPECIFICATION:**

a. Specification means different things to different people.
b. A specification can be a
    1. written document
    2. set of graphical models
    3. formal mathematical model
    4. collection of usage scenarios
    5. prototype or any combination of these.

**VALIDATION:**

a. The work products produced are assessed for quality during validation step.
b. Requirements validation examines the specification to ensure that all software requirements
    1. Have been stated unambiguously
    2. The inconsistencies and errors have been detected and corrected
    3. Work products conform to the standards established for the process, the project and the product.
c. The primary requirements validation mechanism is the technical review.

**MANAGEMNET:**

a. Requirements management is a set of activities that help the project team identify, control and track requirements and changes to requirements at any time as the project proceeds.
b. Many of these activities are identical to the software configuration management techniques.

## FEASIBILITY STUDY

- Feasibility is defined as the practical extent to which a project can be performed successfully.
- Feasibility study determines whether the solution considered to accomplish the requirements is practical and workable in the software.
- Information such as resource availability, cost estimation for software development, benefits of the software to the organization after it is developed and cost to be incurred on its maintenance are considered during the feasibility study.
- The objectives of the feasibility study is
  1. To analyze whether the software will meet organizational requirements
  2. To determine whether the software can be implemented using the current technology and within the specified budget and schedule
  3. To determine whether the software can be integrated with other existing software
- The following are various types of feasibility that are commonly considered
  → Technical feasibility
  → Operational feasibility
  → Economic feasibility

**TECHNICAL FEASIBILITY:**

Technical feasibility assesses the current resources and technology which are required to accomplish user requirements in the software within allocated time and budget.

Technical feasibility performs the following tasks:

1. The software development determines whether the current resources and technology can be upgraded or added in the software to accomplish specified user requirements.
2. Analyzes the technical skills and capabilities of the software development team members.
3. Determines whether the relevant technology is stable and established.
4. Determines that the technology chosen for software development has a large number of users so that they can be consulted when problems arise.

**OPERATIONAL FEASIBILITY:**

Operational feasibility is depend on human resources and involves visualizing whether the software will operate after it is developed and be operative once it is installed.

Operational feasibility performs the following tasks:

1. Determines whether the problems anticipated in user requirements are of high priority.
2. Determines whether the solution suggested by the software development team is acceptable.
3. Analyzes whether users will adapt to a new software
4. Determines whether the organization is satisfied by the alternative solution proposed by the software development team.

**ECONOMIC FEASIBILITY:**

Economic feasibility determines whether the required software is capable of generating financial gains for an organization.

Software is said to be economically feasible if it focuses on the issues:

1. Cost incurred on software development to produce long term gains for an organization
2. Cost required to conduct full software investigation such as requirements elicitation and analysis
3. Cost of hardware, software and development and training.

## PROBLEM OF REQUIREMENTS

Problems in the requirements can cause delays and knock-on errors in the rest of the process. The following are the problems which are commonly occurred in requirements.

1. **Customers don't know what they want:**
   - → The most common problem in the requirements analysis phase is that customers have only a vague idea of what they need.

     **Solution:**
   - → It is our responsibility to ask right questions and perform the analysis.
   - → You have to spend sufficient time at the start of the project on understanding the objectives, deliverables and scope of the project.
   - → Critically evaluate both the likely end-user benefits and risks of the project.
   - → Get your customer to read, think about and sign off on the completed software requirements specification, to have a clear understanding of the deliverables.
2. **Requirements change during the course of the project**
   - → The requirements defined in the first phase change as the project progresses.
   - → As the development progresses and prototypes are developed, customers are able to more clearly see problems with the original plan and make necessary course corrections.
   - → It may also occur because changes in the external environment require reshaping of the original business problem and hence necessitates a different solution than the one originally proposed.

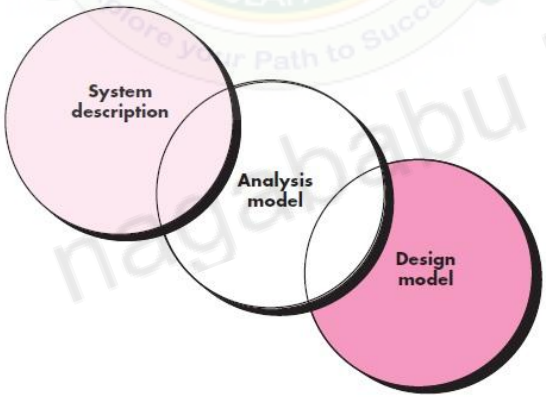
     **Solution:**
   - → Have a clearly defined process for receiving, analyzing and incorporating change requests.
   - → Set milestones for each development phase beyond which certain changes are not permissible.
   - → Ensure that change requests are clearly communicated to all stakeholders and the master project plan is updated accordingly.
3. **Customers have unreasonable timelines**
   - → It's quite common a customer say something like "it is an emergency job and we need this project completed in X weeks".
   - → A common mistake is to agree to such timelines before actually performing a detailed analysis and understanding both of the scope of the project and the resources necessary to execute it.
   - → In accepting such timelines without discussion, the project will either get delayed or suffer form quality defects.

     **Solution:**
   - → Convert the software requirements specification into a project plan, detailing tasks and resources needed.
   - → The project plan takes account of available resource constraints and keeps sufficient time for testing and quality inspection.

4. **Communication gaps exist between customers, engineers and project managers:**
   → Customers and engineers fail to communicate clearly with each other because they come from different worlds and do not understand technical terms in the same way.
   → An important task of a project manager is to ensure that both parties have a precise understanding of the deliverable and the tasks needed to achieve it.
   **Solution:**
   → Take notes at every meeting and disseminate these throughout the project team.

## SOFTWARE REQUIREMENT ANALYSIS

1. Requirements analysis indicates software's interface with other system elements, and establishes constraints that software must meet.
2. Requirements analysis allows you to elaborate on basic requirements established during the inception, elicitation and negotiation tasks that are part of requirements engineering.
3. The requirements modeling action results in one or more of the following types of models:
   → Scenario-based models
   → Data models
   → Class-Oriented models
   → Flow-Oriented models
   → Behavioral models
4. These models provide a software designer with information that can be translated to architectural, interface and component level design.
5. The requirement model must achieve three primary objectives:
   → To describe what the customer requires
   → To establish a basis for the creation of a software design
   → To define a set of requirements that can be validated once the software is built



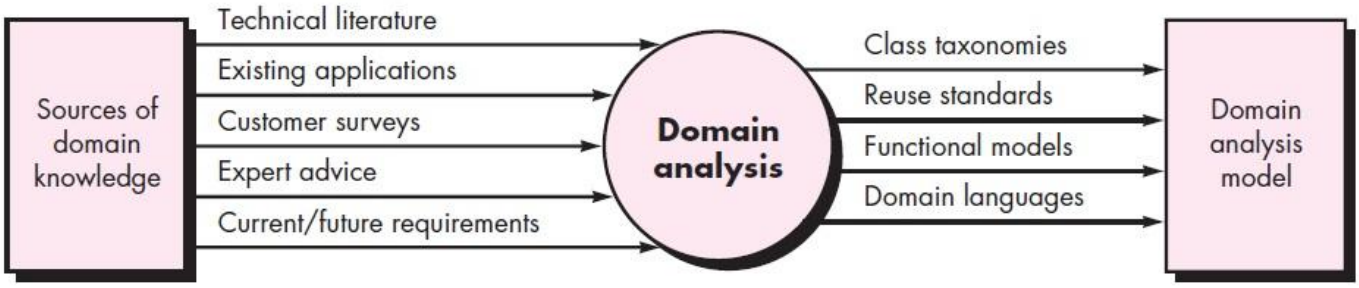



**ANALYSIS RULES OFTHUMB:**

1. The model should focus on requirements that are visible within the problem or business domain.
2. Each element of the requirement model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
3. Delay consideration of infrastructure and other non-functional models until design.
4. Minimize coupling throughout the system.
5. Be certain that the requirements model provides value to all stakeholders.
6. Keep the model as simple as it can be.

**DOMAIN ANALYSIS:**

1. Software domain analysis is the identification, analysis and specification of common requirements from a specific application domain, typically for reuse of multiple projects within the application domain.
2. The goal of domain analysis is to find or create those analysis classes and/or analysis patterns that are broadly applicable so that they may be reused.
3. Domain analysis is an ongoing software engineering activity that is not connected to any one software project.
4. The following diagram illustrates key inputs and outputs for the domain analysis process:

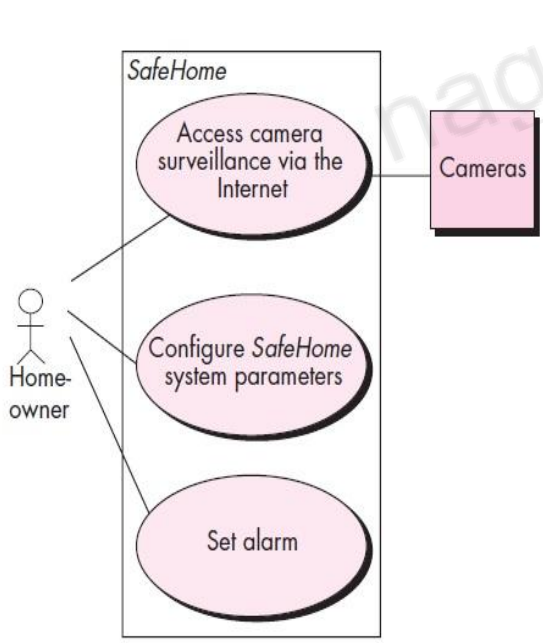


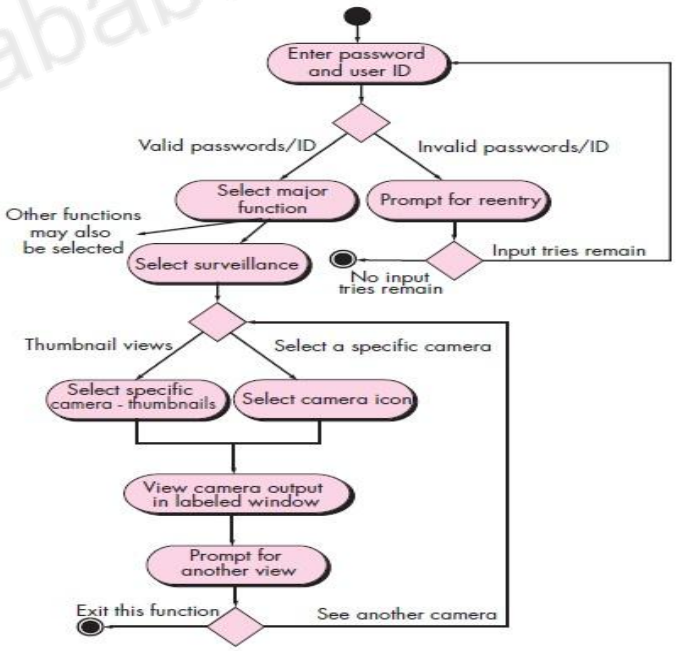

**REQUIREMENTS ANALYSIS MODELS:**

We have five types of models to analyze the requirements. They are

***SCENARIO BASED MODELS:***

1. Scenario-Based elements depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.
2. In scenario-based modeling we use use-cases, activity diagrams, and swimlane diagrams.
3. To develop a set of use cases, list the functions or activities performed by a specific actor.
4. Here actors are the different people (or devices) that use the system.
5. Consider the function access camera surveillance via the internet display camera views (ACS-DCV) in safe home example.
6. The scenario based modeling for this function by using use cases and activity diagrams is as follows.



***DATA MODELING***

1. Data modeling is used when we need to create a software with a database or if complex data structures must be constructed.
2. The data modeling consists of three elements.

**Data Objects:**

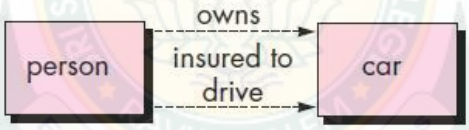
a. A data object is a representation of composite information that must be understood by software.
b. Composite information means something that has a number of different properties or attributes.
Example: Dimensions (incorporating height, width and depth)

**Data Attributes:**

a. Data Attributes define the property of a data object.
b. They can be used to
    1. Name an instance of a data object
    2. Describe the instance
    3. Make reference to another instance in another table

**Relationships:**

1. Data Objects are connected to one another in different ways. These are called relationships.
2. Consider two objects 'person' and 'car'. The relationships are as follows.
    a. A person owns a car
    b. A person is insured to drive a car.



Relationships between data objects

***CLASS-BASED MODELING***

1. Class-based modeling represents the objects that the system will manipulate, the operations that will be applied to the objects and responsibilities between objects.
2. The elements of a class-based modeling are classes, objects, attributes, operations.
3. In class-based modeling we use class diagrams and collaboration diagrams.

Identifying Analysis Classes:

a. Classes are identified in one of the following ways.
b. Things (eg: reports, letters, signals) that are part of the information domain for the problem.
c. Roles (eg: manager, engineer) played by people who interact with the system.
d. Places (eg: manufacturing floor) that establish the context of the problem and overall function of the system.

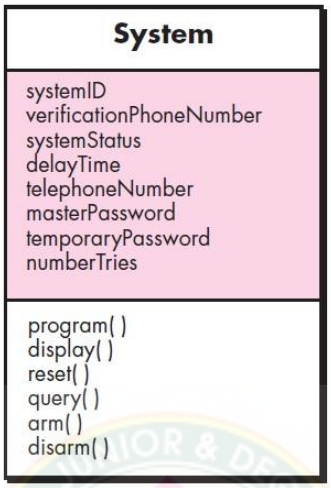
Specifying Attributes:

a. Attributes describe a class that has been selected for inclusion in the requirements model.
b. To develop a meaningful set of attributes for an analysis class, you should study each use case and select those things that reasonably belong to the class.

Defining Operations:

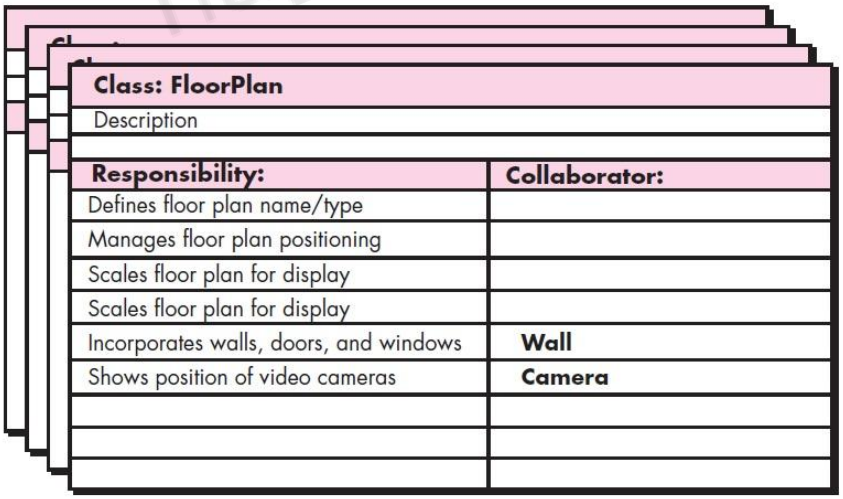
a. Operations define the behavior of an object and are classified into four categories.
   1. Operations that manipulate data
   2. Operations that perform a computation
   3. Operations that inquire about the state of an object
   4. Operations that monitor an object for the occurrence of a controlling event

The system class defined for safe-home as follows:




**Class-Responsibility-Collaborator (CRC) Modeling**

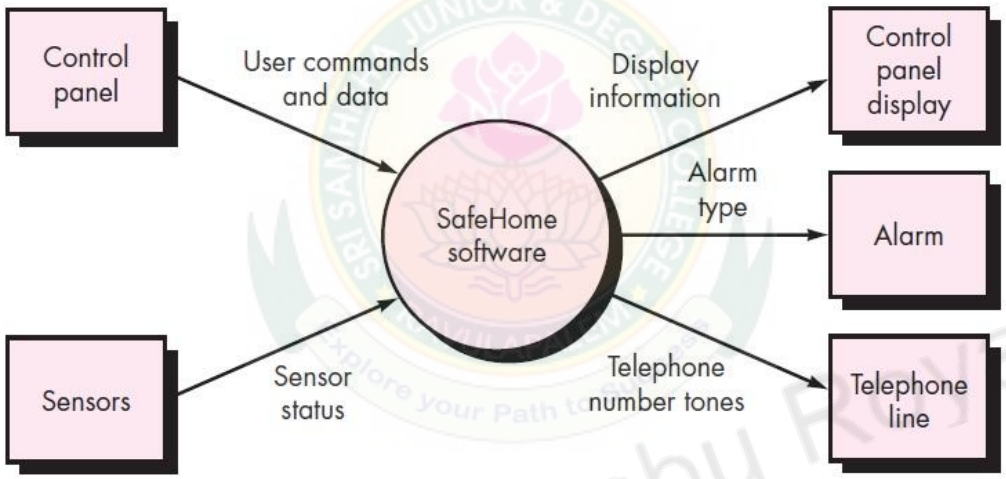
1. Class-Responsibility-Collaborator modeling provides a simple means for identifying and organizing the classes that are relevant to system or product requirements.
2. CRC model is really a collection of standard index cards that represent classes.
3. The cards are divided into three sections.
   a. You write the name of the class on top of the card.
   b. You list the class responsibilities on the left of the body of the card.
   c. You list the collaborators on the right of the body of the card.
4. The intent of the CRC model is to develop an organized representation of classes.
5. Responsibilities are the attributes and operations that are relevant for the class.
6. Collaborators are those classes that are required to provide a class with the information needed to complete a responsibility.



| Class: FloorPlan | |
|---|---|
| Description | |
| **Responsibility:** | **Collaborator:** |
| Defines floor plan name/type | |
| Manages floor plan positioning | |
| Scales floor plan for display | |
| Scales floor plan for display | |
| Incorporates walls, doors, and windows | **Wall** |
| Shows position of video cameras | **Camera** |
| | |
| | |
| | |

## FLOW – ORIENTED MODELING

1. Flow-oriented modeling is one of the most widely used requirements analysis notation.
2. Data Flow Diagram (DFD) and related diagrams can be used to insight into system requirements and flow.
3. The DFD takes an input-process-output view.
4. Data objects flow into the software, are transformed by processing elements and resultant data objects flow out of the software.
5. Data objects represented by labeled arrows and transformations are represented by circles.
6. The DFD is presented in a hierarchical fashion i.e., the first data flow model represents the system as a whole and the subsequent data flow diagrams provide increasing details with each subsequent level.
7. The data flow diagram enables you to develop models of the information domain and functional domain.
8. The data model and the data flow diagram are all that is necessary to obtain meaningful insight into the software requirements.
9. A control specification (CSPEC) represents the behavior the system.
10. A process specification (PSPEC) is used to describe all flow model processes that appear at the final level of refinement.



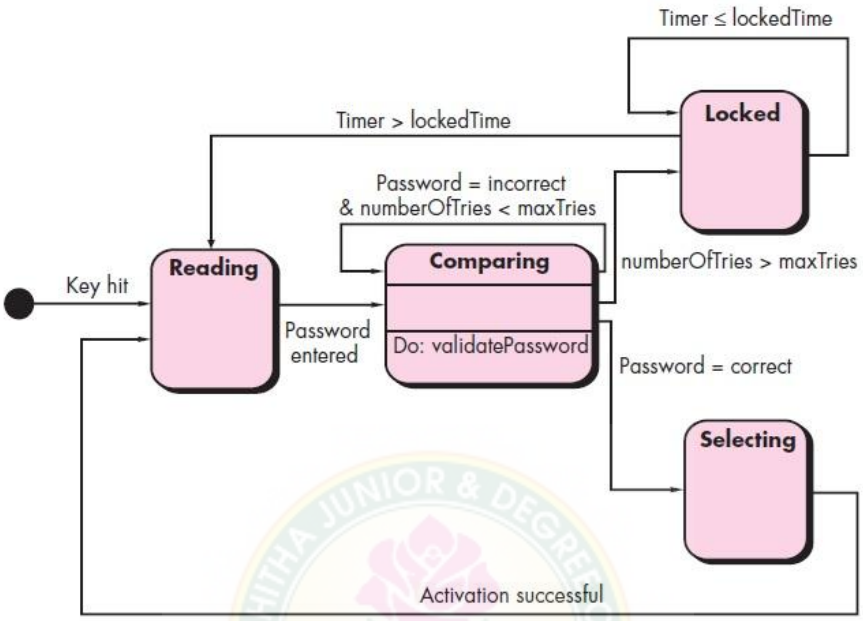

## BEHAVIORAL MODEL

1. The Behavioral Model indicates how software will respond to external events or stimuli.
2. You can represent the behavior of the system as a function of specific events and time.
3. To create behavioral model you should perform the following steps:
   a. Evaluate all use-cases to fully understand the sequence of interaction within the system.
   b. Identify events that drive the interaction sequence and understand how these events relate to specific objects.
   c. Create a sequence for each use case.
   d. Build a state chart diagram for the system.
   e. Review the behavioral model to verify accuracy and consistency.
4. Behavioral model is represented using
   a. State chart diagram
   b. Sequence diagram

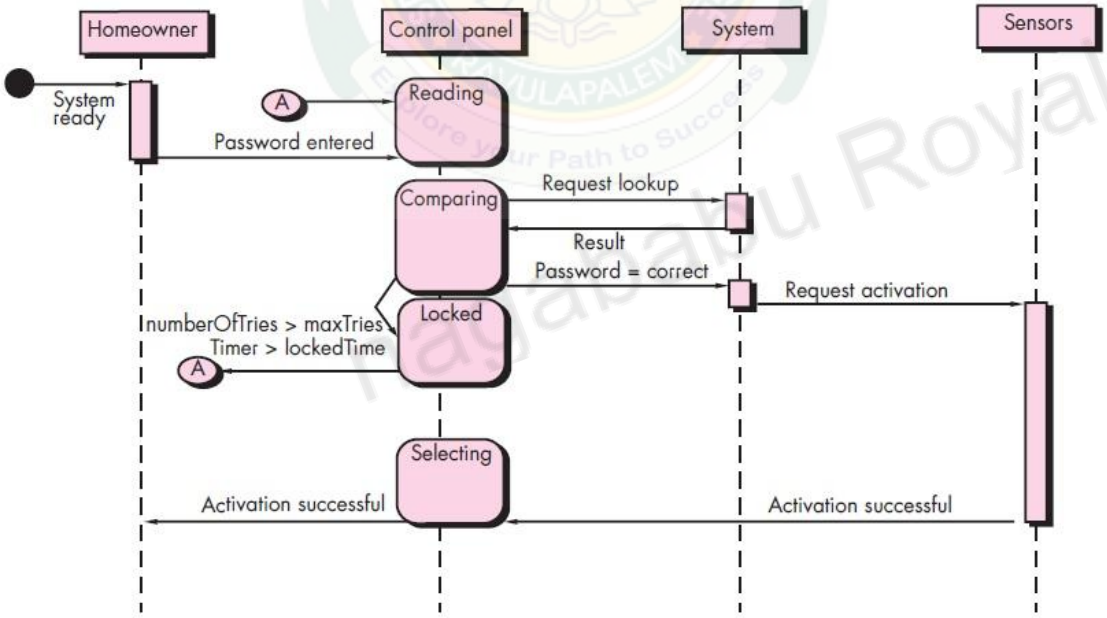
- A use case represents a sequence of activities that involves actors and the system.
- An event is occurred whenever the system and an actor exchange information.

→A State chart diagram represents active states for each class and the events that cause changes between these active states.

→A Sequence diagram indicates how events cause transitions from object to object. Once events have been identified by examining a use case, the modeler creates sequence diagrams.




Sequence Diagram:



# UNIT-III

# SOFTWARE DESIGN

- Software design
- Abstraction
- Modularity
- Software Architecture
- Effective modular design: Cohesion and Coupling
- Architectural design and Procedural design
- Data flow oriented design

## SOFTWARE DESIGN

- Software design encompasses the set of principles, concepts and practices that lead to the development of a high quality system or product.
- Design creates a representation or model of the software, but unlike the requirements model, the design model provides details about software architecture, data structures, interfaces and components that are necessary to implement the system.
- Software design is an iterative process through which requirements are translated into a ***"blue print"*** for constructing software.

## DESIGN PRINCIPLES:

1. **Software design should correspond to the analysis model:** Design element correspond to many requirements, the design model satisfies all the requirements represented by the analysis model.
2. **Choose the right programming paradigm:** A program paradigm describes the structure of the software system. The paradigm should be chosen keeping constraints in mind such as time, availability of resources and nature of user's requirements.
3. **Software design should be flexible:** Software design should be flexible enough to adapt changes easily. To achieve the flexibility, the basic design concepts such as abstraction and modularity should be applied effectively.
4. **Software design should represent correspondence between the software and real-world problem:** The software design should be structured in such a way that it always relates with the real-world problem.
5. **Software reuse:** The software components should be designed in such a way that they can be effectively reused to increase the productivity.

**DESIGN CONCEPTS:** The design concepts help you to answer the following questions:

1. What criteria can be used to partition software into individual components?
2. How is function or data structure detail separated from a conceptual representation of the software?
3. What uniform criteria define the technical quality of a software design?

Software design concepts that span both traditional and object-oriented software development are as follows:

## ***ABSTRCTION:***

- When you consider a modular solution to any problem, many levels of abstraction can be presented.
- At the highest level of abstraction, a solution is stated in broad terms and at lower levels of abstraction, a more detailed description of the solution is provided.

- ❖ As different levels of abstraction are developed, we have to create both procedural and data abstractions.

Procedural Abstraction:

- ↪ A Procedural Abstraction refers to a sequence of instructions that have a specific and limited function.
- ↪ The name of a procedural abstraction implies these functions, but specific details are suppressed.
  Example: "Open for a door" – here open implies a long sequence of procedural steps.

Data Abstraction:

- ↪ A Data Abstraction is a named collection of data that describes a data object.
- ↪ In the context of procedural abstraction ***open***, we can define a data abstraction called ***door***.
- ↪ The data abstraction for door would encompass a set of attributes that describes the door.
  Example: door type, swing direction, opening mechanism etc.

***ARCHITECTURE:***

- ❖ Architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components.
- ❖ A set of architectural patterns enables a software engineer to solve common design problems.

***PATTERNS:***

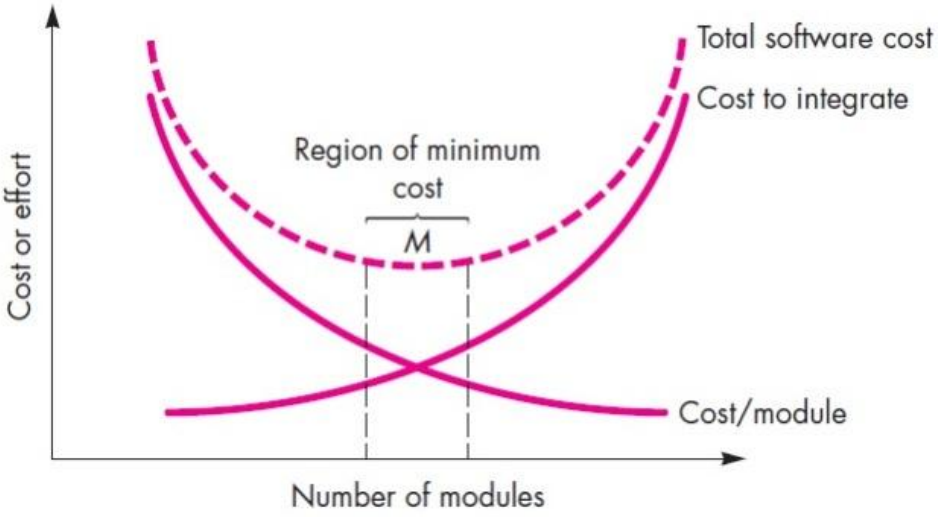
- ❖ A design pattern describes a design structure that solves a particular design problem within a specific context.
- ❖ The design patterns are used to determine
  a. Whether the pattern is applicable to the current work
  b. Whether the pattern can be reused
  c. Whether the pattern can be serve as a guide for developing a similar but functionally or structurally different pattern.

***SEPARATION OF CONCERNS:***

- ❖ Separation of concerns is a design concept that suggests that any complex problem can be more easily handled if it is subdivided into pieces that can each be solved independently.
- ❖ A concern is a feature or behavior that is specified as part of the requirements model for the software.

***MODULARITY:***

- ❖ Modularity is the most common manifestation of separation of concerns.
- ❖ Modularity is the single attribute of software that allows a program to be intellectually manageable.
- ❖ Almost all instances, you should break the design into many modules, to make understanding easier and reduce the cost required to build the software.
- ❖ The cost to develop an individual software module does decrease as the total number of modules increases.
- ❖ However, as the number of modules increases, the cost associated with integrating the modules also increases.
- ❖ There is a number M of modules that would result in minimum development cost, but we do not have the necessary sophistication to predict M with assurance.
- ❖ You should modularize, but care should be taken to stay in vicinity of M.




**EFFECTIVE MODULAR DESIGN**

- When a software program is modularized, its tasks are divided into several modules based on some characteristics.
- Modules are set of instructions put together in order to achieve some tasks.
- There are measures by which the quality of a design of modules and their interaction among them can be measured.
- These measures are called ***coupling*** and ***cohesion***.

**COHESION:**

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

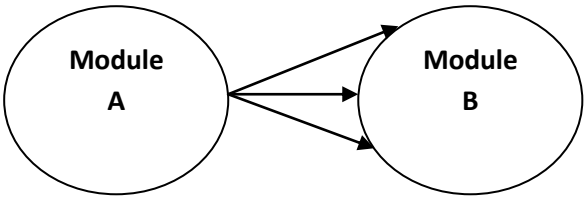
- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

**COUPLING:**

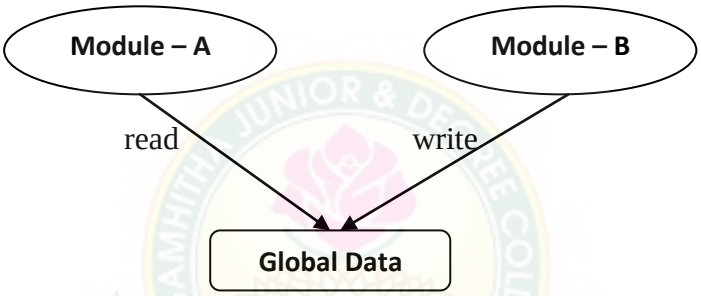
Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

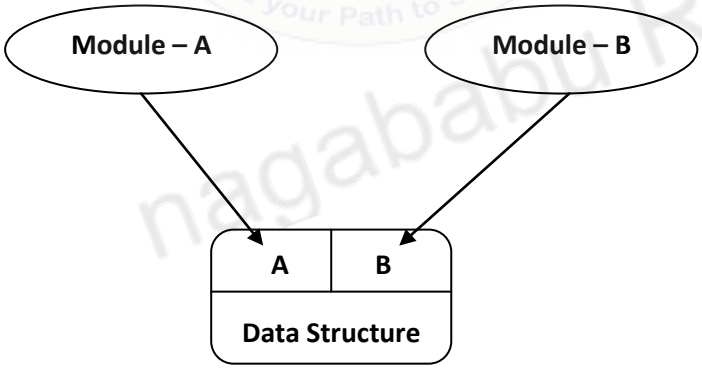
- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.




- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.




- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.




- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

## ARCHITECTURAL DESIGN

Architectural design represents the structure of data and program components that are required to build a computer-based system.

Architectural design begins with data design and then proceeds to the derivation of one or more representations of the architectural structure of the system.

The architectural design can be represented using one or more of an number of different models:

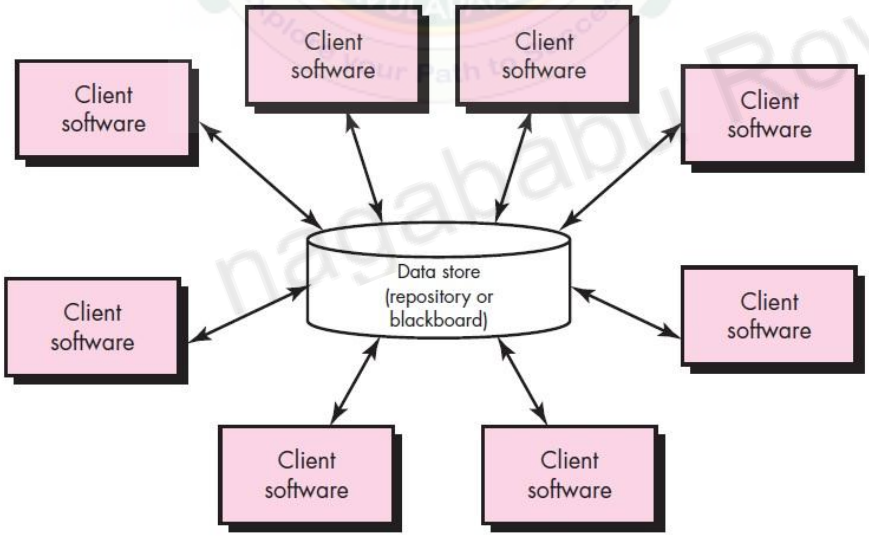
1. ***Structural Models***:-represent architecture as an organized collection of program components.
2. ***Framework Models***:-identify repeatable architectural design frameworks that are encountered in similar types of applications.
3. ***Dynamic Models***:-address the behavioral aspects of the program architecture.
4. ***Process Models***:-focus on the design of the business or technical process.
5. ***Functional Models***:-represents the functional hierarchy of a system.

A number of ***Architectural Description Languages*** (ADLs) have been developed to represent these models.

Software Architecture represents a structure in which some collection of components is connected by a set of defined relationships.

**Data-Centered Architecture:**

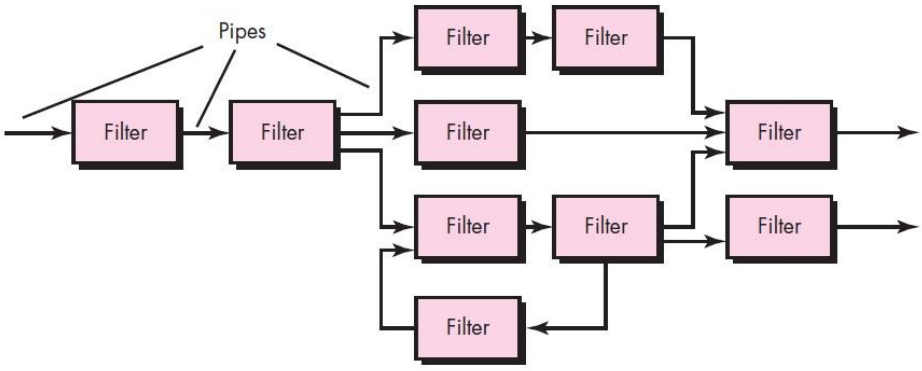
- A data store resides at the center of this architecture and is accessed frequently by other components that update, add, delete and modify data within the data store.
- Data-centered architecture promotes integrability, that is existing components can be changed and new components added to the architecture without concern about other clients.
- Client components independently execute processes.




**Data-Flow Architecture:**

- Data-flow architecture is applied when input data are to be transformed through a series of computational components into output data.
- A pipe and filter pattern has a set of components called filters, connected by pipes that transmit data from one component to the next.

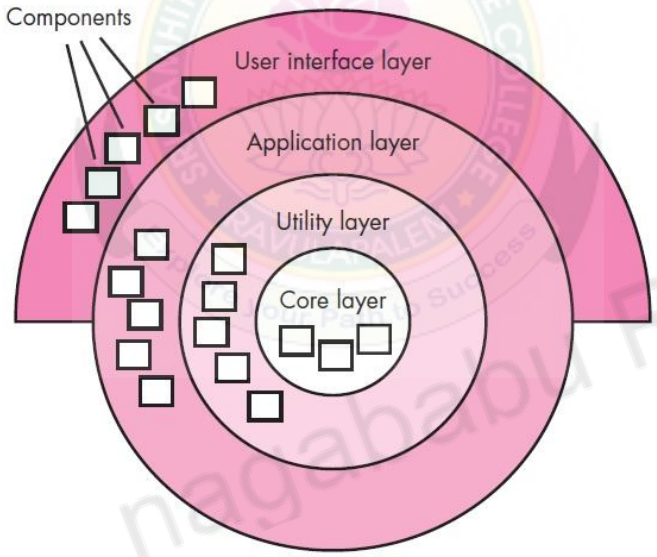
- Each filter works independently on these components and the filter doesn't require knowledge of the working of its neighboring filters.




Pipes and filters

**Layered Architecture:**

- A number of different layers are defined, each accomplishing operations that progressively becomes closer to the machine instruction set.
- At the outer layer, components serve user interface operations.
- At the inner layer, components perform operating system interfacing.
- Intermediate layers provide utility services and application software functions.




→ The software to be developed must be put into context, that is the design should define external entities like other systems, devices, people that the software interact with and the nature of interaction.

→ Once the context is modeled and all external software interfaces have been described, you can identify a set of architectural archetypes.

→ An archetype is an abstraction that represents one element of the system behavior.

**Representing the System in Context:**

a. A software architect uses Architectural Context Diagram (ACD) to model the manner in which software interacts with entities external to its boundaries.

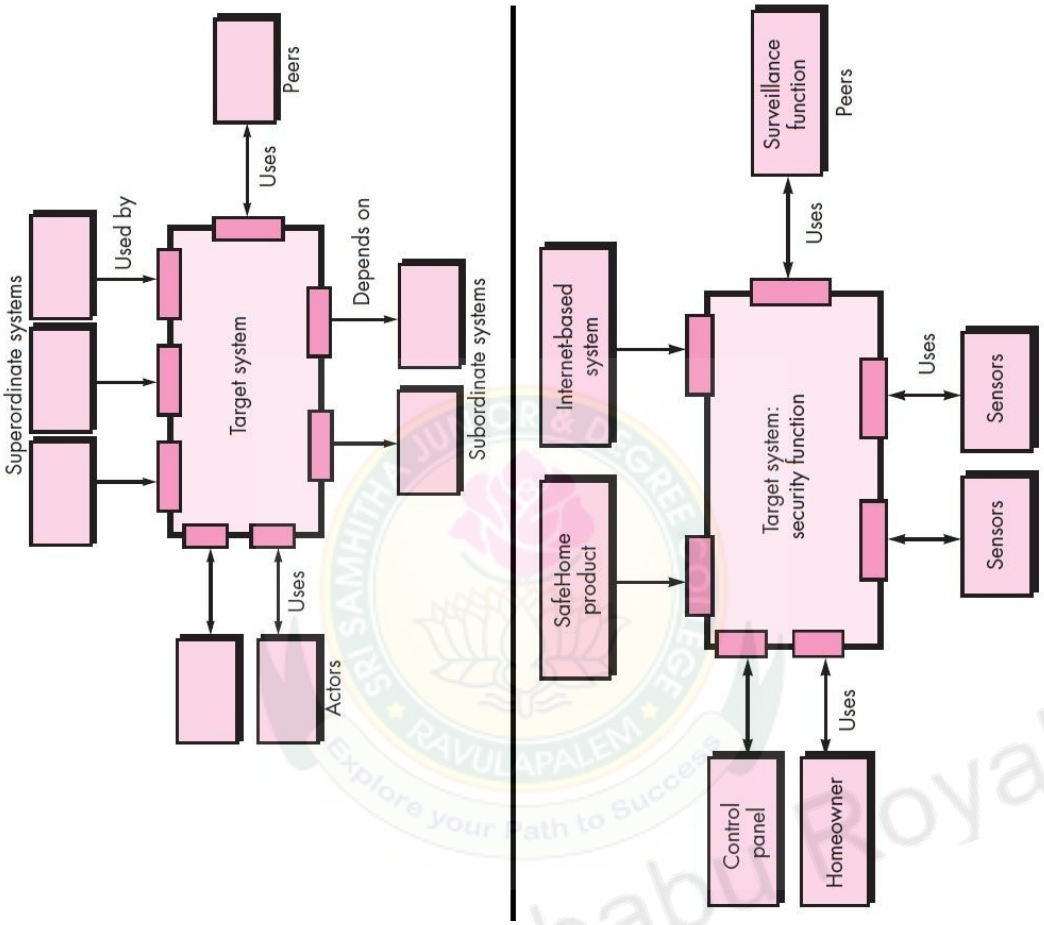
b. The generic structure of the architectural context diagram contains

→ Super-ordinate System:- Those systems that uses the target system as part of some higher level processing scheme.

→ Sub-ordinate System:- Those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality.

→ Peer-level Systems:- Those systems that interact on a peer to peer basis to produce or consume information from target system

→ Actor:- Entities like people, deices that interact with the target system to produce or consume information that is necessary for processing.




a. The overall safeHome product controller and the internet based system both are super-ordinate to the security function.
b. The surveillance function is a peer system and uses security function.
c. The home-owner and control panel are actors that are both consumers and producers of information used or produced by the security function.
d. Sensors are used by the security function.

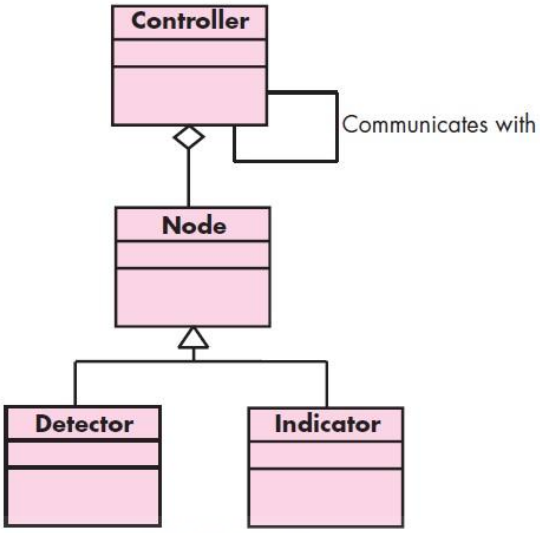
**Defining Archetypes:**

a. An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.
b. The target system architecture is composed of these archetypes, which represents stable elements of the architecture.
c. Archetypes can be derived by examining the analysis classes defined as part of the requirements model.
   NODE :- Represents a cohesive collection of input and output elements.

DETECTOR:- An abstraction that encompasses all sensing equipment that feeds information into the target system.

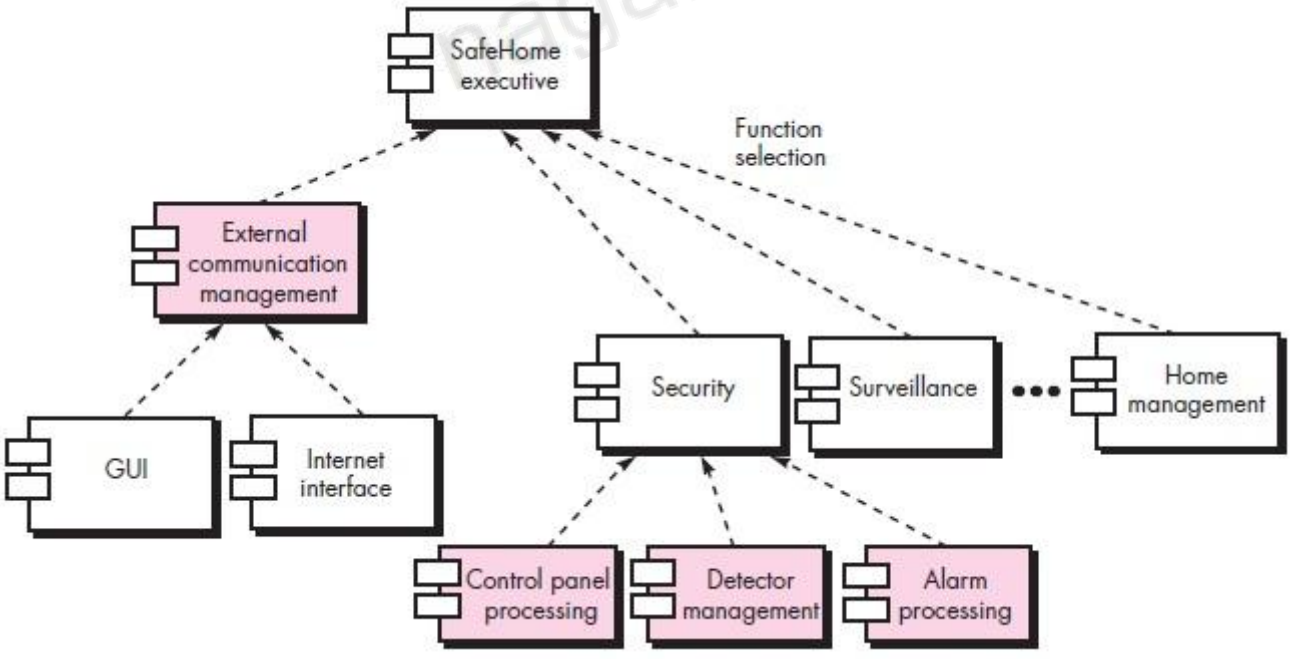
INDICATOR:- An abstraction that represents all mechanisms.

CONTROLLER:- An abstraction that depicts the mechanisms that allows arming or disarming of a node. If controllers reside on a network they have the ability to communicate with one another.



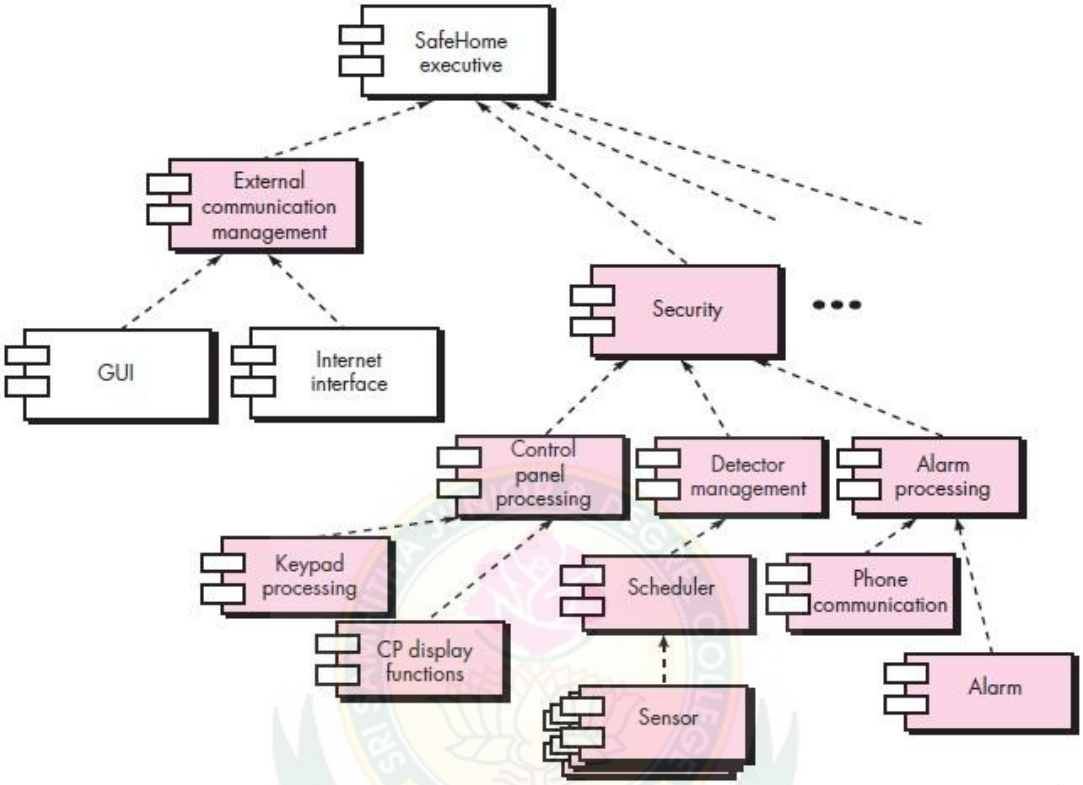

**Refining the Architecture into Components:**

a. The software architecture is refined into components, the structure of the system begins to merge.
b. The interfaces depicted in the architecture context diagram imply one or more specialized components that process the data that flows across the interface.
c. The safehome home security function might define the set of top-level components that address the following functionalities.
   - → External communication management:- coordinates communication of the security function with external entities.
   - → Control Panel processing:- manages all control panel functionality.
   - → Detector management:- coordinates access to all detectors attached to the system.
   - → Alarm processing:- verifies and acts on all alarm conditions.



**Describing the instantiation of the System:**

a. The architectural design that has been modeled is still relatively high level.
b. The context of the system has been represented, archetypes that indicate the abstractions have been defined, major software components have been identified.
c. Further refinement is still necessary and this can be accomplished by instantiation of the architecture.




**PROCEDURAL DESIGN**

The objective in Procedural Design is to transform structural components into a procedural description of the software.

This step occurs after the data and program structures have been established, i.e. after architectural design.

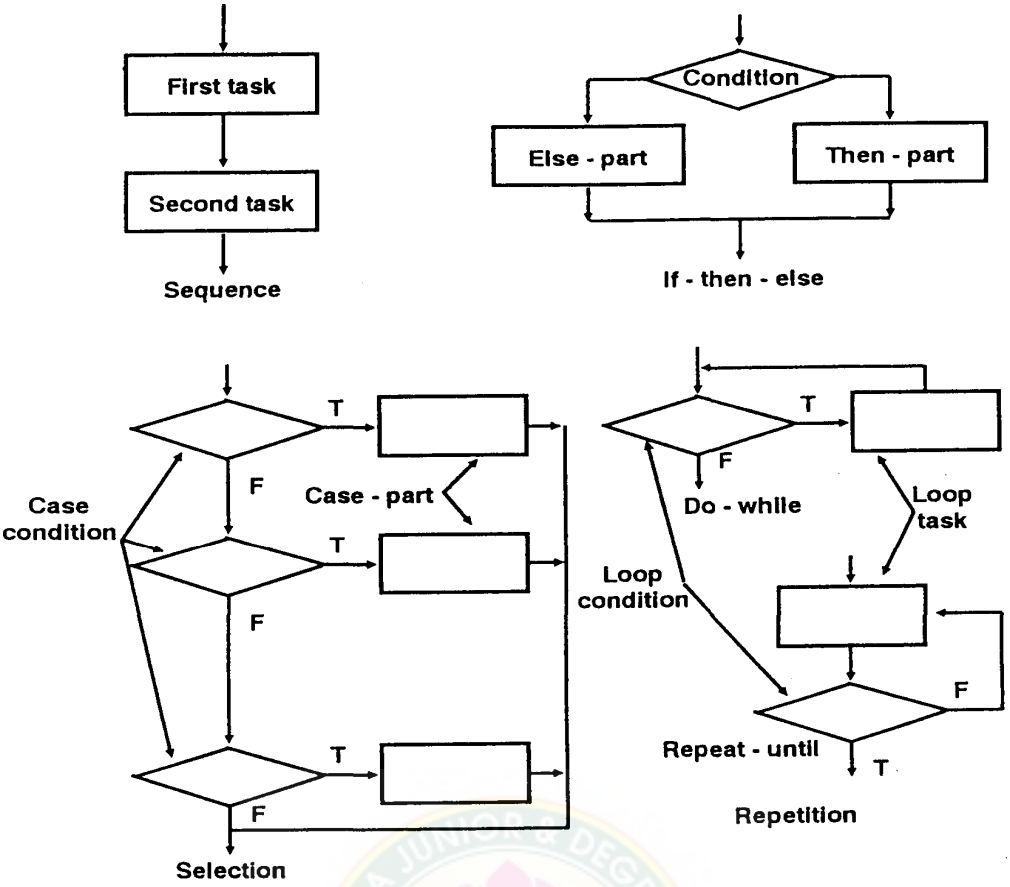
Procedural details can be represented in different ways:

- Graphical Design Notation
- Tabular Design Notation
- Program Design Language (PDL)

**Graphical Design Notation**

The most widely used notation is the flowchart. Some notations used in flowcharts are

1. Boxes to indicate processing steps
2. Diamond to indicate logical conditions
3. Arrows to indicate flow of control
4. Two boxes connected by a line of control will indicate a Sequence.




**Tabular Design Notation**

a. Decision tables provide a notation that translates actions and conditions (described in a processing narrative) into a tabular form.
b. The upper left-hand section contains a list of all conditions. The lower left-hand section lists all actions that are possible based on the conditions.
c. The right-hand sections form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.

| | | | | | | |
|---|---|---|---|---|---|---|
| Conditions | Fixed rate account | 1 | 2 | 3 | 4 | 5 |
| | Variable rate account | T | T | F | F | F |
| | Consumption <100KWH | T | F | T | F | |
| | Consumption □100KWH | F | T | F | T | |
| Actions | Minimum monthly charge | X | | | | |
| | Schedule A billing | | X | X | | |
| | Schedule B billing | | | | X | |
| | Other treatment | | | | | X |

**Program Design Language**

1. Program Design Language (PDL) is also called structured English, or Pseudocode.
2. The main difference between PDL and its nearest neighbor, 4th Generation Languages, is that in PDL, the use of narrative text (e.g. English) is embedded directly within PDL statements.

**PDL have the following characteristics:**

a. A fixed syntax of keywords that provide for all structured constructs, data declaration, and modularity characteristics
b. A free syntax of natural language that describes processing features

c. Data declaration facilities that should include both simple (scalar, array) and complex (linked list or tree) data structures.

d. Subprogram definition and calling techniques that support various methods of interface description.

# UNIT-IV

# USER INTERFACE DESIGN AND REAL TIME SYSTEMS

- User interface design
- Human factors
- Human computer interaction
- Human
- Computer Interface design
- Interface design
- Interface standards.

## USER INTERFACE:

1. User interface is the front end application view in which user interacts with the software. User can manipulates and control the software as well as hardware by using user interface.
2. User interface can be a graphical, text based, audio, video based depending on the user interface.
3. The software becomes more popular if its user interface is
   a. Attractive
   b. Simple to use
   c. Responsive in short time
   d. Clear to understand
   e. Consistent on all interfacing screens
4. User interface is divided into two categories
   → Command Line Interface (CLI) (or) Command User Interface (CUI)
   → Graphical User Interface (GUI)

## COMMAND LINE INTERFACE (CLI):

- CLI has been a great tool of interaction with computers until the video display monitors came into existence.
- CLI is first choice of many technical users and programmers.
- CLI provides a command prompt, the place where user types the command.
- The user needs to remember the syntax of command and its use.
- A command is a text based reference which is to be executed by the system.
- CLI uses less amount of resources of computer as compared to GUI.

## Components or Elements of CLI:

1. Command prompt:- It is text based notifier that shows the path in which the user is working.
2. Cursor:- It is a small horizontal line or vertical bar to represent the position of character while typing it is in blinking state.
3. Command:- A command is an executable instruction.

## GRAPHICAL USER INTERFACE (GUI)

- Graphical user interface provides the user graphical way to interact with the system.
- GUI can be combination of both hardware and software.
- In GUI user can work with the software very easily. There is no need to remember the commands.
- There are so many GUI designs available that work with more efficiently, accurately at speed.

**Components or Elements of GUI:**

Every graphical component provides a way to work with the system.

1. Window:- An area where contents of application are displayed.
2. Tabs:- If an application allows executing multiple instances of itself, then they appear on the screens as separate windows called tabs.
3. Menu:- Menu is a collection of standard commands.
4. Icon:- Icon is a small picture representing an associated software.
5. Cursor:- Cursor is used to select menus, windows and other application features.
6. Dialogue Box:- It is a child window that contains message for the user and request for some action to be taken.
7. Text Box:- It provides an area to enter text.
8. Buttons:- These are used to submit inputs to the software.
9. Radio Button:- Displays available options for selection.
10. Check Box:- Multiple options represented by check boxes.
11. List Box:- Provides list of available items for selection.

## USER INTERFACE DESIGN

- User interface design creates an effective communication medium between a human and a computer.
- A software engineer designs the user interface by applying an iterative process that draws on predefined design principles.
- User interface design begins with the identification of user task, and environmental requirements.
- Once user tasks have been identified, user scenarios are created and analyzed to define a set of interface objects and actions.
- Interface design focuses on three areas.
  a. The design of interfaces between software components.
  b. The design of interfaces between software and other non human producers and consumers of information.
  c. The design of interface between a human and the computer.
- A set of **golden rules** are applied to all human interaction with technology products.
- A set of **interaction mechanisms** were defined to build systems that properly implement golden rules. These interaction mechanisms collectively called the Graphical User Interface.

## Golden Rules:

The following are the golden rules of the interface design:

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent

***Place the User in Control:*** It defines a number of design principles that allows the users to maintain control.

- Define interaction models in a way that does not force a user into unnecessary or undesired actions.
  The user shall be able to enter and exit a mode with little or no effect.
- Provide for flexible interaction.
  The user shall be able to perform the same action via keyboard commands, mouse movement or voice recognition.

- Allow user interaction to be interruptible and undoable.
  The user shall be able to easily interrupt a sequence of action to do.
  The user shall be able to undo any action.

***Reduce User's Memory Load:*** It defines design principles that enable an interface to reduce the user's memory load.

- Reduce demand on short-term memory.
  The interface shall reduce user's requirement to remember past actions and results by providing visual clues of such actions.
- Define shortcuts that are intuitive.
  The user shall be provided mnemonics that tie easily to the action in a way that it is easy to remember such as the first letter like Ctrl+C for copy and Ctrl+P for print.

***Make the Interface Consistent:*** The interface should present and acquire in a consistent fashion. It defines the principles that help to make the interface consistent.

- Allow the users to put the current task into a meaningful context.
  It is important to provide indicators that the user can determine where he has come from.
- Maintain consistency across a family of applications.
  A set of applications should follow the same design rules.

## INTERFACE DESIGN STEPS:

- Once interface design has been completed then all the tasks required by the end-user have been identified in-detail. After that interface design is started.
- Interface design is an iterative process. Each user interface design step occur a number of times.
- The following are the steps that are involved in the user interface design:
  - ✓ By using information from interface analysis and define interface objects and actions.
  - ✓ Define events that will cause the state of the user interface to change. Model this behavior.
  - ✓ Depict each interface state as it will actually look to the end user.
  - ✓ Indicate how the user interprets the state of the system from information provided through the interface.
- Regardless of the sequence of design tasks, the designer must
  - ✓ Always follow the golden rules
  - ✓ Models how the interface will be implement
  - ✓ Consider the environment that will be used

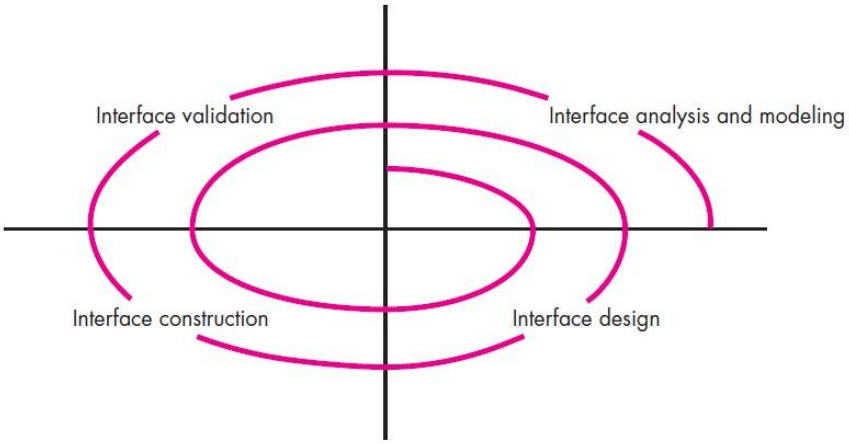
## USER INTERFACE DESIGN PROCESS:

The analysis and design process for user interface is iterative and can be represented using a spiral model.

This process has four distinct framework activities.

1. User, Task and environment analysis and modeling
2. Interface design
3. Interface construction
4. Interface

Each of the above activity will occur more than once. Each pass of the spiral represents the detailed requirements and resultant design.




- Once the requirements have been identified then a more detailed task analysis is conducted. The analysis of the user environment focuses on the physical environment.
- The goal of the interface design is to define interface objects and actions that enable the user to perform all defined tasks.
- Construction activity normally begins with the creation of a prototype.
- Validation focuses on
    - The ability of the interface to implement every user task correctly.
    - The degree to which the interface is easy to use and easy to learn.
    - The user's acceptance of the interface.

## DESIGN ISSUES:

There are some issues occur during the interface design. Those are:

1. Response Time:
   System response time is the primary complaint for many interactive applications. System response time is measured from which the user performs an action until software responds as an output.
2. Help Facility:
   If any problem or issue occurred then the modern software provide online help facilities that enable user to get a question answered without leaving.
3. Error Handling:
   Error messages and warnings are "bad news" delivered to users of interactive system when something has gone wrong.
4. Menu and Command Labeling:
   A number of design issues arise by typed commands or menu labels when interacting with the application.
5. Application Accessibility:
   Interface design should be enable all the users to access the application. Accessibility also provided to the users who may be physically challenged.
6. Internalization:
   Internalization is the process of designing and developing software or web applications so that they can be easily adapted to various linguistic and cultural environment.

## HUMAN FACTORS:

Human Factor is the discipline that tries to establish a relationship between technology and the human.

Human factors deal with the human behavior, abilities, limitations to the use of software, tools and other jobs to make their use easier.

1. Limited Shot-Term Memory:
   People can instantaneously remember about 7-10 items of information. If you present more than this, they are more liable to make mistakes.
2. People Make Mistakes:
   When people make mistakes and systems go wrong inappropriate alarms and messages can increase stress and increase chance to make more mistakes.
3. People have different interaction preferences:
   Some like pictures, some like text based interactions.

Designers and developers must focus on the following:

a. Users and psychology
b. Quality and performance
c. Indentifying end-users and requirements.

Advantages:

- High user satisfaction
- Lower user fatigue
- Lower training time and costs
- Lower product liability
- Lower operator stress
- Lower operating costs
- Lesser operational errors.
- Greater system performance.

## HUMAN COMPUTER INTERACTION

Human Computer Interaction is also known as Man-Machine Interaction (MMI) or Computer Human Interaction (CHI). It is the study of interaction between people (users) and computers. It tells how people and computers work together.

People have trouble with computers in some cases.

- Use of difficult in language and spoken by language
- Complexity
- Trial and error method it may lead to infinite so we will restart
- Different information processing systems process same actions but are named differently on different consoles.
  Ex: shut down or turn off.

**Psychological user responses to poor design:**

Psychological responses affect the user concentration to design an efficient model. It will resulting in higher error rates, low performance and dissatisfaction to the user.

- Attitude – Positive, neutral or negative feeling towards job or system
- Motivation – Low, moderate or high
- Patience – Patience or impatience expected in accomplishing job
- Stress level – High, some or no stress from task performance
- Confusion
- Irritation
- Frustration
- Boring

**Physical User Responses in design:**

Physical characteristics affects the users interest on work. These affects the application.

- Rejection of the system
- Limited or partial use of the system
- Indirect use of the system
- Modification of the task
- Compensatory task
- Misuse of the system

**Important Human Characteristics in Design:**

Human characteristics plays an important role while designing an interface. After the completion of the development of the project, the end user interact with the application. So user characteristics are very important.

- Perception – The people who are interacting with the application should have an awareness with the elements and data objects of an environment.
- Memory – Memory is an essential feature of human, without having memory user unable to interact with the system. Memory consists of two parts.
  Short term memory
  Long term memory
- Visuality – The capacity of eye to resolve the details is called visuality.
- Peripheral Vision – A vision which seems its surroundings.
- Foreal Vision – A vision which is used to direct focus on something is called foreal vision.
- Sensory Memory – It acts as a buffer. It automatically process information.
- Information Processing – The information gathered by senses has to processed in an appropriate way. There are two levels of information.
  Higher-Level (capacity is limited, slow and sequential)
  Lower-Level (unlimited, process quickly)
- Learning Process – Applying skills that allows one situation to be used in another situation.
- Skill – The ultimate goal of human is to perform skill full which can be achieved by linking inputs and responses.
- Individual Differences – Individual differences complicate design because the design must permit people with widely varying characteristics to satisfactorily and comfortably learn the task or job or use website.

**USER INTERFACE:**

User interface is the part of a computer and its software that people can see, hear, touch, talk to. User interface is a subset of a field of a study called Human Computer Interaction (HCI).

User Interface Components:

The user interface has essentially two components:

→ Input → Output

**Input**:- It is how a person communicates his or her needs or desires to the computer. Some common input components are Keyboard, mouse, trackball, one's finger and one's voice.

Output:- It is how the computer conveys the results of its computations and requirements to the user. Some common output devices are monitor, speakers, printers.

**Importance of the User Interface:**

A well-defined interface and screen is important to our users. It is their window to view the capabilities of the system.

- A screen layout and appearance affect a person in a variety of ways. If they are confusing and inefficient the people will have greater difficulty in doing their jobs or will make more mistakes.
- Poor design will make some people away from a system permanently. It can also lead to increases frustration and stress.
- It provides a way through which many critical tasks are presented.

**Principles of User Interface Design**

1. Pleasing appearance
   a. Provide meaningful contrast between screen elements
   b. Align screen elements and groups
2. Clarity
   a. Visual elements
   b. Function
3. Compatibility – provide compatibility with the following
   a. The user
   b. The task and job
   c. The product
4. Comprehensibility – A system should be easily learned and understood. User should know what action to do, how to do, where to do.
5. Consistency – A system should look, act and operate the same throughout.
   a. Similar components should
      a. Have a similar task
      b. Have a similar use
      c. Operate similarly
6. Control
   a. The user must control the interaction
   b. Action should be performed quickly
   c. Action should be capable of interruption or termination

7. Efficiency
    a. Transaction between various system controls should flow easily and freely
    b. Navigation paths should be short as possible
8. Flexibility – System should provide flexibility to manage tasks
9. Recovery
    a. A system should permit
        a. Commands or actions to be reversed
        b. Immediate return to a certain point
10. Responsiveness
    a. System must rapidly respond to the user's requests.
    b. It should provide immediate response for all user's actions
        - ➔ Visually
        - ➔ Textually
        - ➔ Auditory
11. Simplicity – Provide as simple interface as possible.
    a. Present common and necessary functions first
    b. Provide default
    c. Minimize screen alignment points …
12. Transparency
    a. Permit the user to focus on the task or job without concern for the mechanics of the interface

## INTERFACE STANDARDS:

User interface standards are very effective when you are developing, testing or designing any new application or when you are revising existing application.

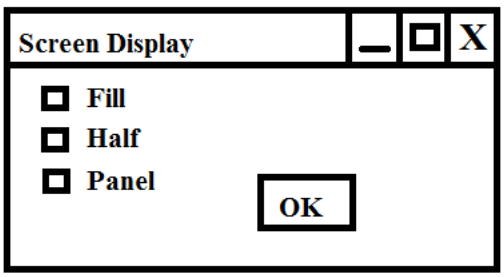
Creating a user interface standards helps you to create user interfaces that are consistent and easy to understand.

Types of Standards: There are three types of standards.

1. Methodological standards – This is a checklist to remind developers of the tasks needed to create usable systems such as interview, task analysis and design etc.
2. Design standards – A set of absolute legal requirements that ensure a consistent look and feel.
3. Design principles – Good design principles ae specific and research based and developers work well within the design standards rules.

Example:

- ↪ Modeling a system which has user controlled display option
- ↪ User can select from one of three choices
- ↪ Choices determine the size of the current window display
- ↪ So the developers come up with schema and present first prototype



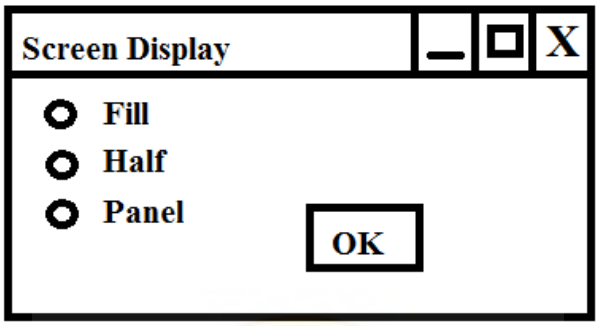
From the above prototype user can select more than one option. But there is only one option enough to perform the task. So it is bad design.

Design has broken guideline regarding selection controls.

**Guidelines for using Selection Controls:**

- ✓ Use radio buttons to allow only one option
- ✓ Use check boxes to allow more than one option

Now extend the specification by first satisfying the guidelines of interface. So the developers extend specification and present revised prototype.




Interface standards divided into four major areas:

1. Navigation – It is needed to develop a standard method for navigating through our application.
2. Forms – It is needed to develop a standard method for presenting information on forms.
3. Reports – It is needed to develop a standard method and layout for reports.
4. Documentation – It is needed standard for both systems and user documentation.

Benefits of Standards:

1. The goal of user interface design is to make the user's interaction as simple and efficient as possible.
2. User or customers see a consistent UI within and between applications.
3. Reduced cost and effort for system maintenance
4. Less time spent evaluating design alternatives.
5. Share system modules more easily
6. Reduced costs for support, user training packages and job aid's.
7. Most important customer satisfaction, reduced errors, training requirement, frustration, time per transaction.