Software Engineering

Chapter 1: Introduction to Software Engineering

What is Software? :

Software is nothing but a collection of computer programs and related documents that are intended to provide desired features, functionalities and better performance.

There are two types of Software products:

- ***Generic***
  - Means developed to be sold to a range of different customers
- ***Custom***
  - Means developed for a single customer according to their specification

Software Applications:

Seven broad categories of computer software present continuing challenges for software engineers:

- **System software**
- **Application software**
- **Engineering/scientific software**
- **Embedded software**
- **Product-line software**
- **WebApps (Web applications)**
- **AI software**

Software—New Categories:

- **Open world computing**.- Distributed Computing
- **Netsourcing -** renting or paying to use
- **Open source -** developed as public collaboration or made free available
- Also …
  - *Data mining – examining large pre-existing databases*
  - *Grid computing – distributed comp resources*
  - *Cognitive machines – AI technologies-ML,DL*
  - *Software for nanotechnologies*

What is Software Engineering?

A formal definition of software engineering is,

> ***"An organized, analytical approach to the design, development, use, and maintenance of software."***

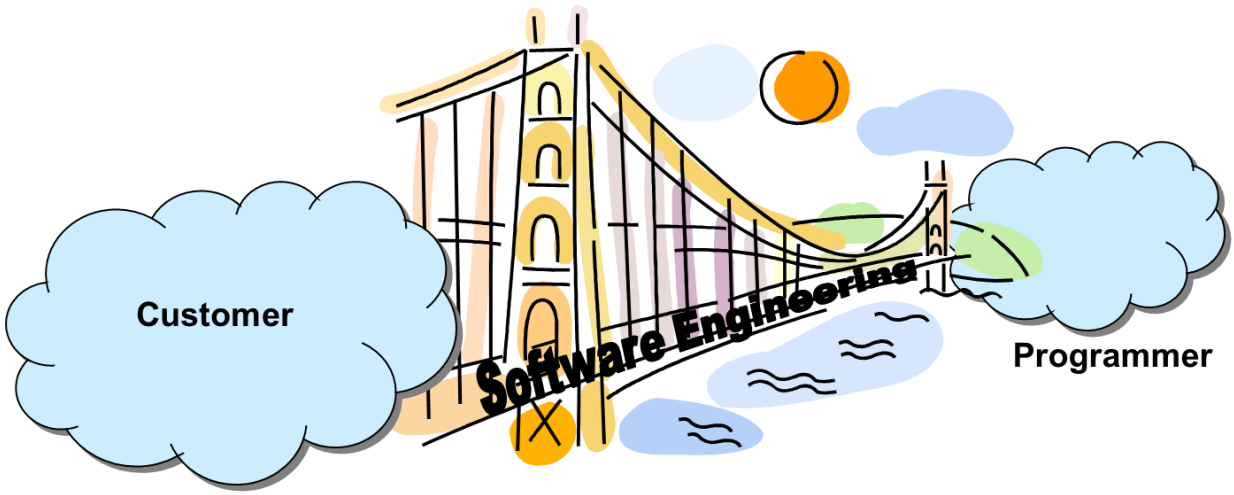
Software engineering is everything which need to do to produce successful software.

It includes the steps that

- take a raw, possibly unclear idea and
- turn it into a powerful and adaptive application
  - that can be enhanced to meet changing customer needs for years to come.

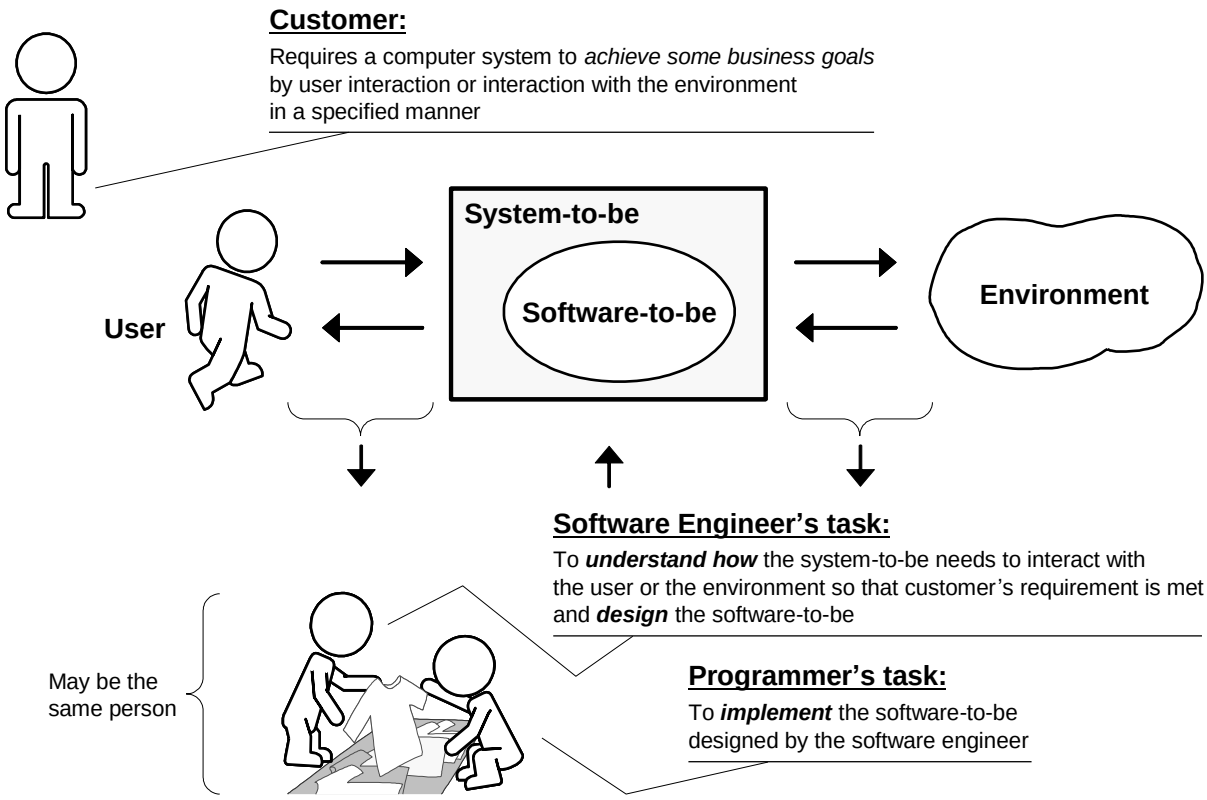
Role of Software Engineering:

A bridge from customer needs to programming implementation



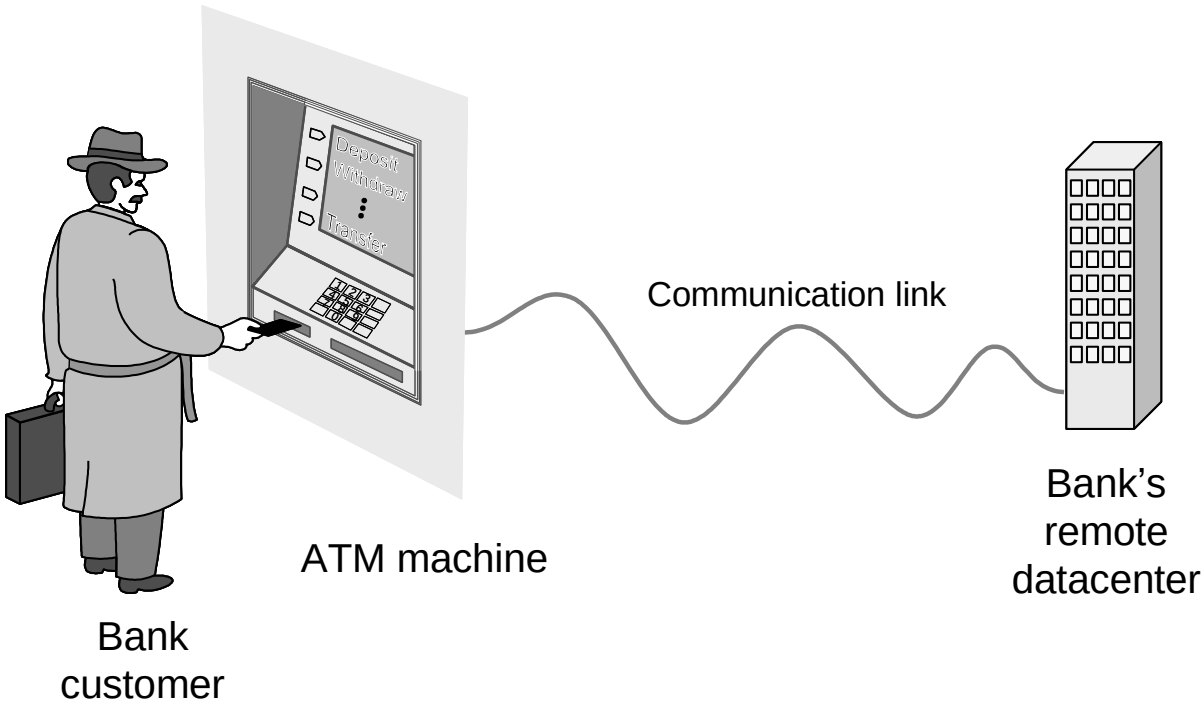
First law of software engineering

Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)



Example: ATM Machine

Understanding the money-machine problem:



Goal of Software Engineering:

The primary goals of software engineering are:

- To improve the quality of the software products.
- To increase the productivity
- To give job satisfaction to the software engineers.

Why is Software Engineering important?

- It includes techniques to avoid difficulties.
- It ensures the final application is effective, usable, and maintainable.
- It helps to produce a finished project on time and within budget.
- It gives the flexibility to make changes to meet unexpected demands.

Characteristics of Good Software:

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational:

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Functionality
- Dependability
- Security & Safety

Transitional:

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability

- Adaptability

Maintenance:

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

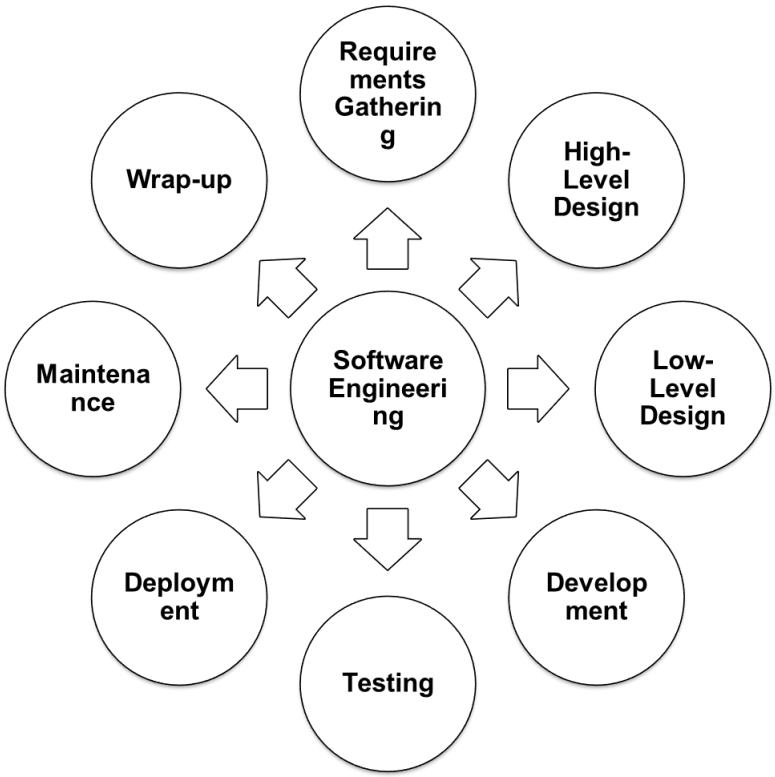
- Modularity
- Maintainability
- Flexibility
- Scalability

Chapter 2: Overview of Software Engineering phases

Steps/ Phases of Software Engineering:

There are various software development approaches defined and designed which are used/employed during development process of software.

Following are the steps/ phases applied during software engineering process.



1. Requirements Gathering:

- First steps in a software project is figuring out the requirements.
- Find out what customers want and what customers need.
- Depending on how well defined the user's needs are
    - This can be time-consuming.
- After determining the customers' wants and need,
    - Turn them into requirements documents.

2. High Level Design:

- After the project's requirements, start working on the high-level design.
- It includes about what platform to use

- What data design to use and
- Interfaces with other systems.
- It also include information about the project architecture
    - Break the project into modules
    - Handle the project's major areas of functionality.

3. Low Level Design:

- After high-level design breaks the project into pieces,
    - Assign those pieces to groups within the project
    - So that they can work on low-level designs.
- The low-level design includes
    - Information about *how* that piece of the project should work

4. Development:

- Development is the part that most programmers enjoy the most
- Programmers continue refining the low-level designs
    - Until they know how to implement those designs in code.
- As the programmers write the code,
    - They test it to make sure it doesn't contain any bugs.
    - They test it to find and remove as many bugs as they reasonably can

5. Testing:

- First developers test their own code.
- Then testers who didn't write the code test it.
- After a piece of code seems to work properly,
    - It is integrated into the rest of the project, and
    - The whole thing is tested to see if the new code broke anything.
- Any time a test fails,
    - The programmers dive back into the code to figure out what's going wrong and
    - How to fix it.
- After any repairs, the code goes back into the queue for retesting

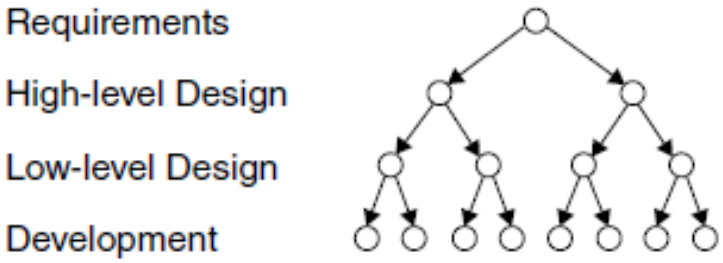


FIGURE: The circles represent possible mistakes at different stages of development. One early mistake can lead to lots of later mistakes.

6. Deployment:

- Deployment includes all the process required for preparing a software application to run and operate in a specific environment.
- It involves installation, configuration, testing and making changes to optimize the performance of the software.

7. Maintenance:

It is the process of modifying a software product after it has been delivered to the customer. Main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.

Need for Maintenance –

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.

8. Wrap-up:

- At this point in the process, you're probably ready for a break.
- You've put in long hours of planning, design, development, and testing.

- You've found bugs you didn't expect, and the users are keeping you busy with bug reports and change requests.
- You want nothing more than a nice, long vacation.

Chapter 3: Document Management in Software Development

What is Document Management?

Document management is the process of storing, locating, updating, and sharing data for the purpose of workflow progression and business outcomes.

A software engineering project uses a lot of documents.

It includes:

- Requirements documents,
- Use cases,
- Design documents,
- Test plans,
- User training material,
- Lunch menus for team-building exercises,
- Resumes if the project doesn't go well, and much more.

Importance of Document Management:

- In some projects
    - Requirements are allowed to change as the project progresses.
- As the project progresses
    - Customers will also get a better understanding of the system.
- During development
    - It's important to check the documentation.
- Easily find the most recent version of the requirements to see what the application should do.
- Need to find the most recent high-level and low-level designs to see if we're following the plan correctly.
- Need to find older versions of the documentation, to find out what changes were made, why they were made, and who made them.
- To prevent the conflict, we need a document control system that prevents two people from making changes to the same document at the same time.

- To handle all these issues, we need a good document management system.

Operations in Document Management:

Ideally, the system should support at least the following operations:

- People can share documents so that they can view and edit them.
- Only one person can edit a document at a given time.
- Fetch the most recent version of a document.
- Fetch a specific version of a document by specifying either a date or version number.
- Search documents for tags, keywords, and anything else in the documents.
- Compare two versions of a document to see what changed, who changed it, and when the change occurred.

Features of Document Management:

Following are some other features that are less common but still useful:

- The ability to access documents over the Internet or on mobile devices.
- The ability for multiple people to collaborate on documents.
- Integration into other tools such as Microsoft Office or project management software.
- Document branches so that it can split a document into two paths for future changes.
- User roles and restricted access lists.
- E-mail change notification.
- Workflow support and document routing.

Types of Documentation:

The following are the different kinds of documentation:

- Historical Documents
- E-Mail
- Code
- Code Documentation
- Application Documentation

Historical Documents:

- Every design decision, requirements change, and memo is a Historical Documents
    - Keep it in safe for later use.
- Save the information which is available in an electronic form.
- Save an e-mail about the project.
- Save the change request.
- Put all new created document in document repository, or at least e-mail it to yourself for the record.
- Record meetings and phone calls if required:
    - Make a quick summary and e-mail it to all the participants.
    - Follow-up e-mail should be send to the disagree participant that can also go into the historical documents.

E-Mail:

- E-mail is the best medium to communicate with each other.
- Through e-mail, Memos, discussions about possible change requests, meeting notes, and lunch orders are all easy to distribute.
- Storing those e-mails for historical purposes is also easy:
    - Simply CC a selected e-mail address for every project e-mail.

We can invent following message classes which are useful and handy.

- Admin —Administration
- Rqts —Requirements
- HLDesign —High-level design
- LLDesign —Low-level design
- Dvt —Development
- Test —Testing
- Deploy —Deployment
- Doc —Documentation
- Train —Training
- Maint —Maintenance
- Wrap —Wrap-up

Code:

- Program source code is different from a project's other kinds of documents.
- Code changes continually, up to and sometimes even beyond the project's official ending date.
- Source code control systems a slightly different than other kinds of document control systems.
- Requirements document might go through a dozen or so versions, but a code module might include hundreds or even thousands of changes.
- A source code control system enables all the developers to use the code.
- Some source code control systems are integrated into the development environment.
    - They make code management so easy.

Code Documentation:

- Code documentation should include high- and low-level design documents that can store in the document repository with other kinds of documentation.
- These provide an overview of the methods the code is using to do.
- Code documentation should also include comments in the code to help and understand what the code is actually doing and how it works.
- Don't need to comment every line of code but it should provide a fairly detailed explanation.
    - So that a new user can also understand.

Application Documentation:

All the documentation described so far deals with building the application.

- It includes such items as
    - Requirements and design documents,
    - Code documentation and comments,
    - Meeting and phone call notes, and memos.
- Application Documentation also includes
    - User manuals (for end users, managers, database administrators, and more)
    - Quick start guides
    - Cheat sheets
    - User interface maps

- Training materials, and
- Marketing materials.

Chapter 4: Project Management

What is Project?:

A ***project*** is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery).

Characteristics of Project:

A Project can be characterized as:

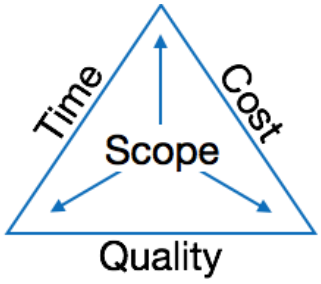
- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

What is Project Management?:

In Software Engineering, ***Project Management*** dedicated to the planning, scheduling, resource allocation, execution, tracking and delivery of software and web projects.

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Need of Project Management:



- The image above shows triple constraints for software projects
- It is an essential part of software organization to deliver quality product, keeping the cost within client's budget constrain and deliver the project as per scheduled.
- Any of three factor can severely impact the other two.

- Therefore, software project management is essential to incorporate user requirements along with budget and time constraints.

Project Manager:

- A ***project manager*** is generally the highest-ranking member of the project team.
- He works with the team through all stages of development, starting with requirements gathering, moving through development and testing, and continuing until application rollout.
- Project manager monitors the project's progress to ensure that work is heading in the right direction at an acceptable pace.
- He meets with customers and other stakeholders to verify that the finished product meets their requirements.

Duties of Project Manager:

Project manager duties include:

- Helping define the project requirements
- Tracking project tasks
- Responding to unexpected problems
- Managing risk
- Keeping users up-to-date on the project's progress
- Providing an interface between customers and developers
- Managing resources such as time, people, budget, hardware, and software tools
- Managing delivery

Project Management Tools:

- The risk and uncertainty rises multi-fold with respect to the size of the project, even when the project is developed according to set methodologies.
- There are tools available, which aid for effective project management.
- A few are described given as below
    - PERT Charts
    - Critical Paths Methods
    - Gantt Charts

PERT Charts:

- A PERT chart is a project management tool that provides a graphical representation of a project's timeline.
- The Program Evaluation Review Technique (PERT) breaks down the individual tasks of a project for analysis.
- PERT charts are considered preferable to Gantt charts because they identify task dependencies, but they're often more difficult to interpret.
- PERT charts were first created by the U.S. Navy's Special Projects Office in 1957 to guide the Polaris nuclear submarine project.
- A PERT chart allows managers to evaluate the time and resources necessary to manage a project.

How do PERT Charts works?:

- A PERT chart uses circles or rectangles called nodes to represent project events or milestones.
- These nodes are linked by vectors or lines that represent various tasks.
- Dependent tasks are items that must be performed in a specific manner.
- For example, if an arrow is drawn from Task No. 1 to Task No. 2 on a PERT chart, Task No. 1 must be completed before work on Task No. 2 begins.
- Items at the same stage of production but on different task lines within a project are referred to as parallel tasks.
- They're independent of each other, but they're planned to occur at the same time.

- A well-constructed PERT chart looks like this:

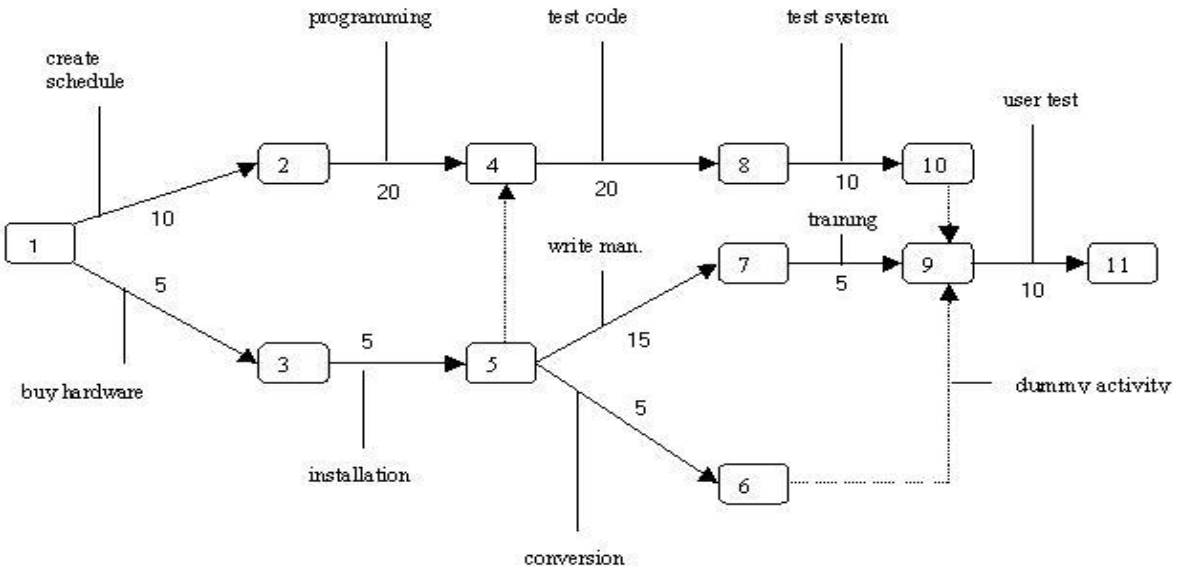


Fig. 1:
PERT Chart

* Numbered rectangles are nodes and represent events or milestones.
* Directional arrows represent dependent tasks that must be completed sequentially.
* Diverging arrow directions (e.g. 1-2 & 1-3) indicate possibly concurrent tasks
* Dotted lines indicate dependent tasks that do not require resources.

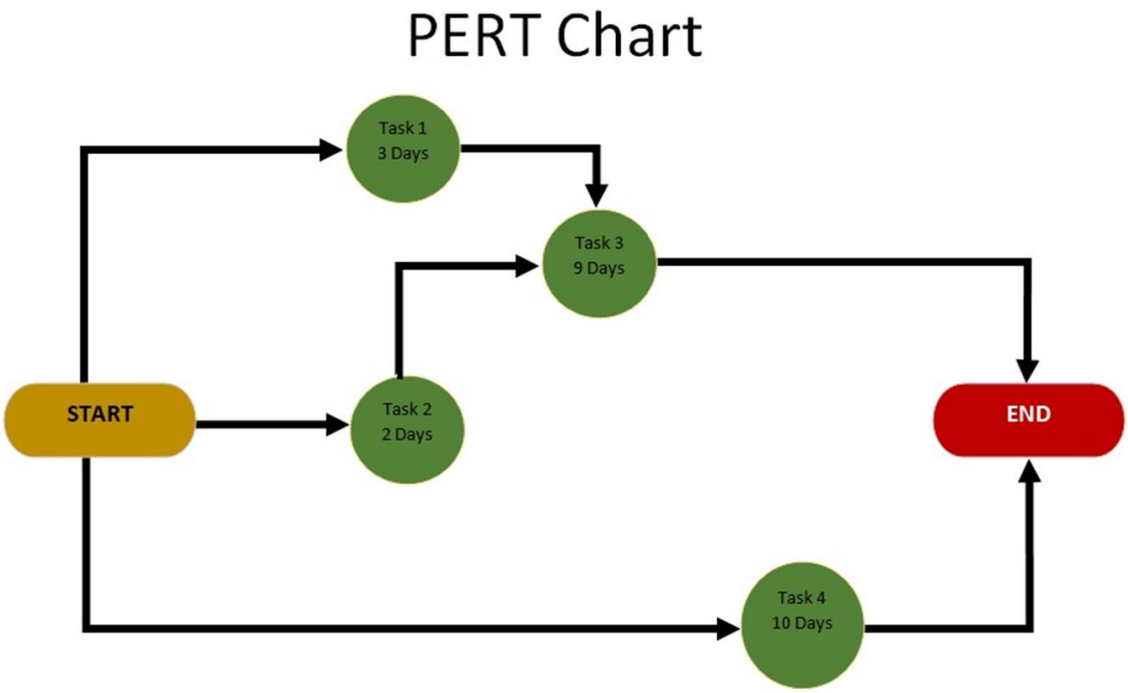
How do PERT Charts works?:

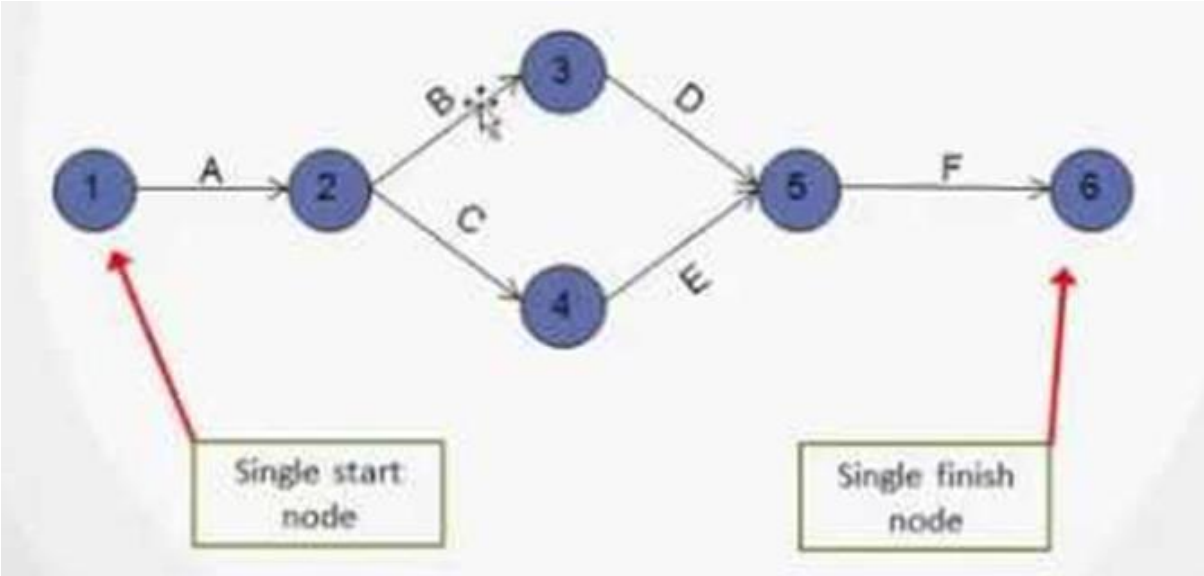
- In the diagram, for example, the tasks between nodes 1, 2, 4, 8 and 10 must be completed in sequence.
- These are called ***dependent*** or ***serial*** tasks.
- The tasks between nodes 1 and 2, and nodes 1 and 3 are not dependent on the completion of one to start the other and can be undertaken simultaneously.
- These tasks are called ***parallel*** or ***concurrent*** tasks.
- Tasks that must be completed in sequence but that don't require resources or completion time are considered to have ***event dependency***.
- These are represented by dotted lines with arrows and are called ***dummy activities***.
- For example, the dashed arrow linking nodes 6 and 9 indicates that the system files must be converted before the user test can take place, but that the resources

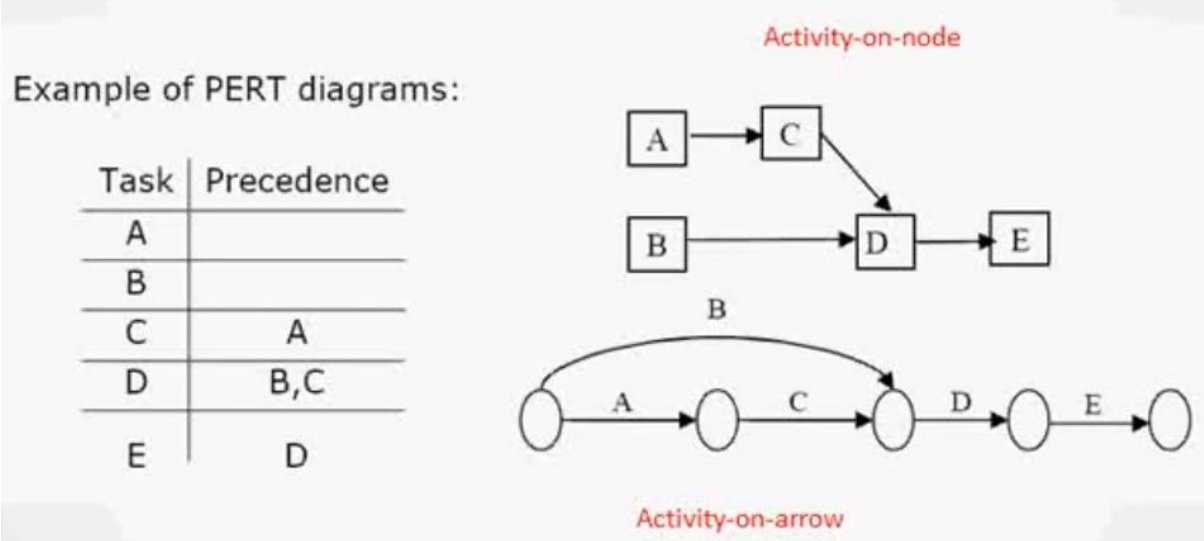
and time required to prepare for the user test (writing the user manual and user training) are on another path.

- Numbers on the opposite sides of the vectors indicate the time allotted for the task.

Some examples of PERT Chart:







When to use PERT Charts?:

- PERT charts should be used when a project manager needs to:
  - Determine the project's critical path in order to guarantee all deadlines are met.
  - Display the various interdependencies of tasks.
  - Estimate the amount of time needed to complete the project.
  - Prepare for more complex and larger projects

Advantages of PERT Charts:

- A PERT chart allows managers to evaluate the time and resources necessary to manage a project.
- PERT analysis incorporates data and information from multiple departments.
- It also improves communication during the project.
- PERT charts are useful for what-if analyses.

Disadvantages of PERT Charts:

- The use of a PERT chart is highly subjective
- Success of Pert Chart depends on the management's experience.
- It can include unreliable data for cost or time.
- PERT charts are deadline-focused.
- PERT chart is labor intensive.

Critical Paths Method:

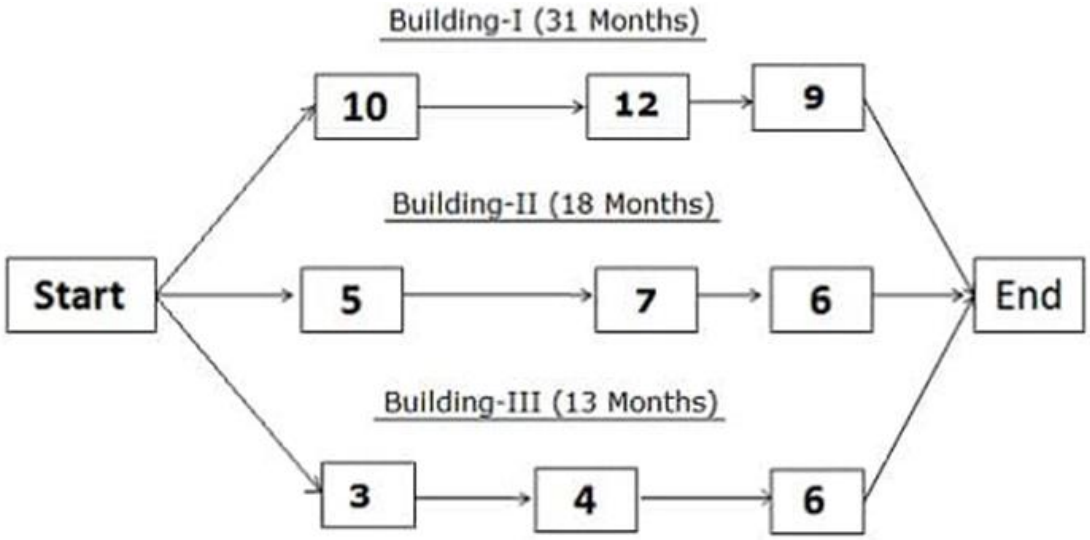
- Critical path method (CPM); a project modelling technique developed by Morgan R. Walker and James E. Kelly in late 1950
- The ***critical path method (CPM)*** is used extensively by project planners worldwide for developing the project schedule in all types of projects including IT, research, and construction.
- This method is the basis of the project schedule.
- In any network diagram, there are many paths originating from one point and ending at another point.
- Every path will have some duration.
- The path with the longest duration is known as the ***critical path***.
- The critical path can be defined in many ways including:
    - The longest path in the network diagram, or
    - The shortest duration in which the project can be completed.
- Now, are these two definitions similar, or opposite to each other?
- Both definitions are trying to convey the same message.
- They might seem opposite because the first definition is talking about the longest path and the second definition is talking about the shortest duration.
- However, they both are the same.

Example:

- For example, let's say suppose we have received a project to build three buildings in one location.
- The first building is the largest building, the second building is a medium-sized building, and the third building is the smallest building.
- Develop the network diagram which consists of three paths; each path resembles each building.
- Then calculate the duration for each path.
    - For the first building, the duration is 31 months,

- The second building will take 18 months, and
- The third building will require 13 months.

- The first path represents
  - the largest building;
  - the second path represents the medium-sized building, and
  - the third path, the smallest building.

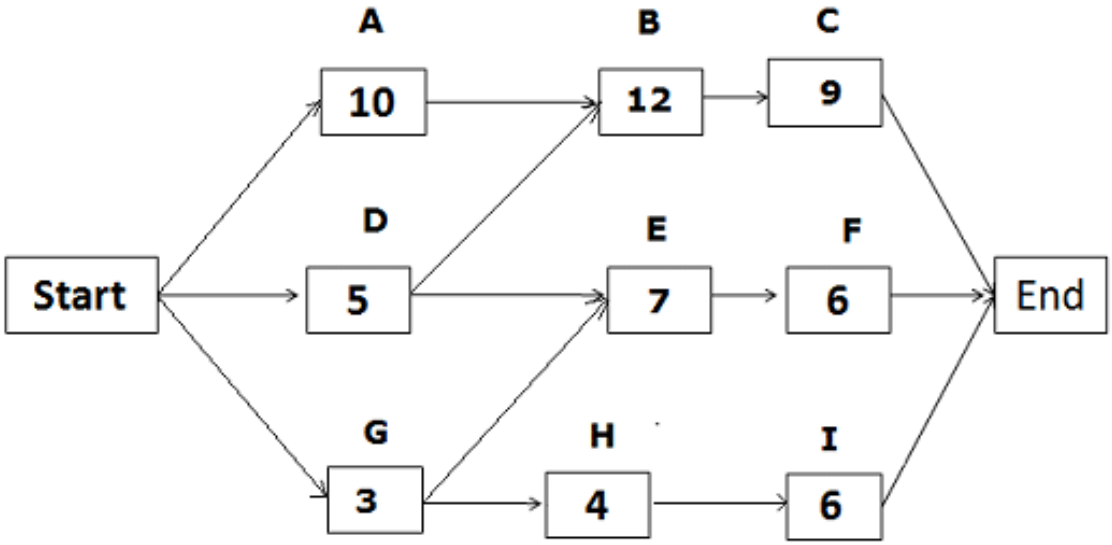


- Now, let us review the above diagram.
  - Notice that the path for the first building is the longest duration of all three.
  - It is 13 months longer than the second path, and 18 months longer than the third path.
  - This means that we can wait 13 months before working on the second building
    - If we start working on the first building because we can complete the second building in 18 months.
  - Likewise, we could wait 18 months to start working on the third building because it will take only 13 months to complete.
    - This means that even if we start working on the third building after 18 months from the project start date, we can finish it on time.
  - This waiting period is known as the **float** or **slack**.
  - **So, which is the critical path in this network diagram of three paths?**

- Of course, it is the longest path on the network diagram, because we cannot complete the project before constructing the first building.
- Although we can complete the other two buildings quickly, but the project is not considered complete until we complete the first building.
- **Hence, the critical path is the longest path on the network diagram.**
- **Now, what is the shortest duration in which we can complete the project?**
- Sure enough, it is 31 months, because we cannot complete the project before 31 months, and this is the duration of the critical path.
- **Hence, the critical path is the shortest duration in which we can complete the project.**
- Hence, we can see that both definitions are the same.
- We can conclude that the critical path is the sequence of activities from start to end, and it has the longest duration among all paths in a network diagram.
- **Procedure for Finding the Critical Path in a Network Diagram**
    - The following are the procedures to find the critical path on a network diagram:
        - Draw the network diagram.
        - Identify all paths in the network diagram.
        - Find the duration of each path.
        - The path with the largest duration is the critical path.
    - Let's look at the above procedure in action.

Example:

- Based on the below network diagram, identify the total paths, critical path, and float for each path.



Solution:

The above network diagram has five paths; the paths and their duration are as follows:

1. Start -> A -> B -> C-> End, duration: ***31* days**.
2. Start ->D -> E ->F -> End, duration: **18 days**.
3. Start -> D -> B -> C -> End, duration: **26 days.**
4. Start -> G ->H ->I -> End, duration: **13 days.**
5. Start -> G -> E ->F -> End, duration: **16** days.

Since the duration of the first path is the longest, it is the ***critical path.***

The ***float*** on the critical path is ***zero.***

The float for the second path

"Start ->D -> E ->F -> End" = duration of the critical path – duration of the path "Start ->D -> E ->F -> End"

= 31 – 18 = 13

Hence, the float for the second path is 13 days.

Using the same process, we can calculate the float for other paths as well.

Float for the third path = 31 – 26 = 5 days.

Float for the fourth path = 31 – 13 = 18 days.

Float for the fifth path = 31 – 16 = 15 days.

Advantages of Critical Path Methods:

The following are a few benefits of the critical path method:

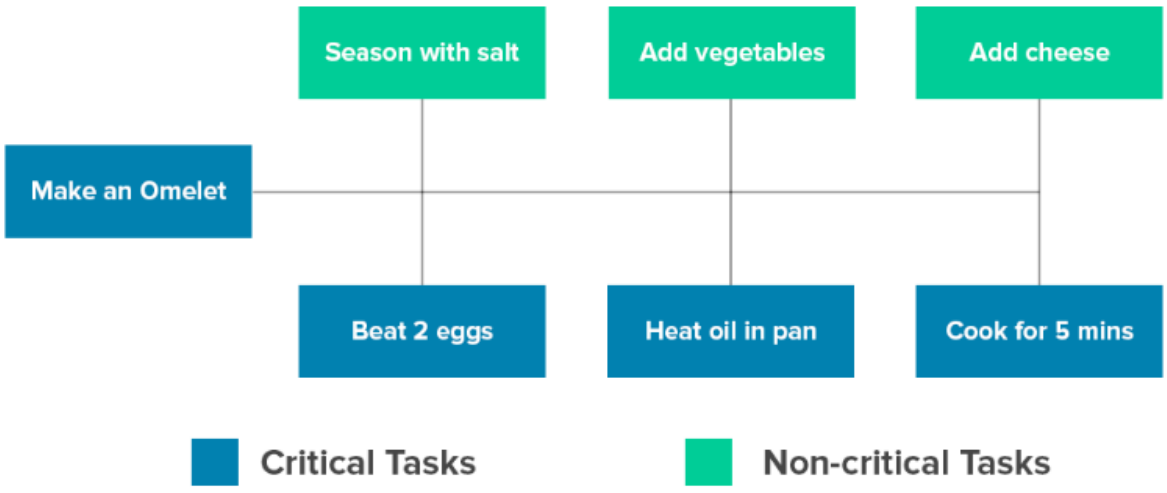
- It shows the graphical view of the project.
- It discovers and makes dependencies visible.
- It helps in project planning, scheduling, and controlling.
- It helps in emergency planning.
- It shows the critical path, and identifies critical activities requiring special attention.
- It shows you where you need to take action to bring project back on track.

Disadvantages of Critical Path Methods:

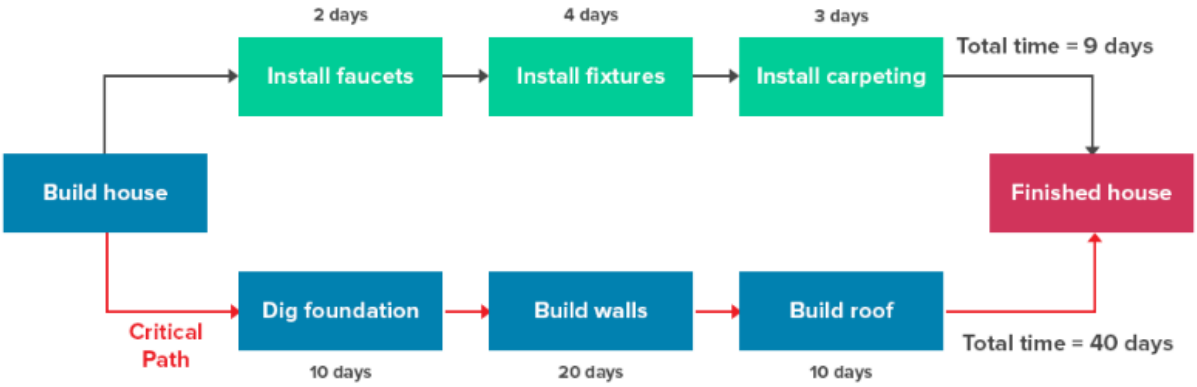
- It does not consider resource dependencies.
- There are chances of misusing float or slack.
- Less attention on non-critical activities.
- Projects based on the critical path often fail to be completed within the approved time duration.

Examples:

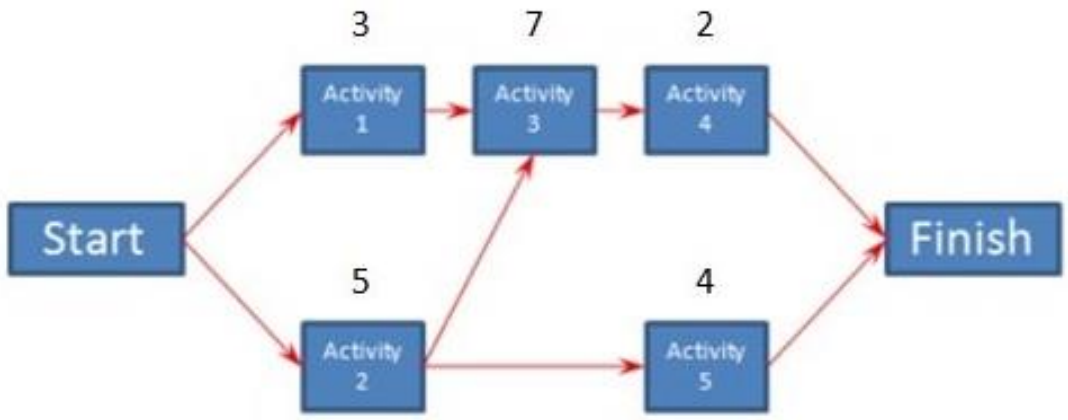
- 1. The three steps in the recipe describe the *critical* tasks necessary to make the omelet making project a success.



- 2. For example, if you're building a house, you would have several task sequences as follows:



- 3. Consider following example:



Using the Critical Path Method (CPM)

There are three paths through this project...

Critical Path

Start → Activity 1 → Activity 3 → Activity 4 → Finish	3 + 7 + 2 = 12
Start → Activity 2 → Activity 3 → Activity 4 → Finish	5 + 7 + 2 = 14
Start → Activity 2 → Activity 5 → Finish	5 + 4 = 9

Use Critical Path Analysis to find your Critical Path

Gantt Chart:

A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist.

Frequently used in project management,

A *Gantt chart* provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.

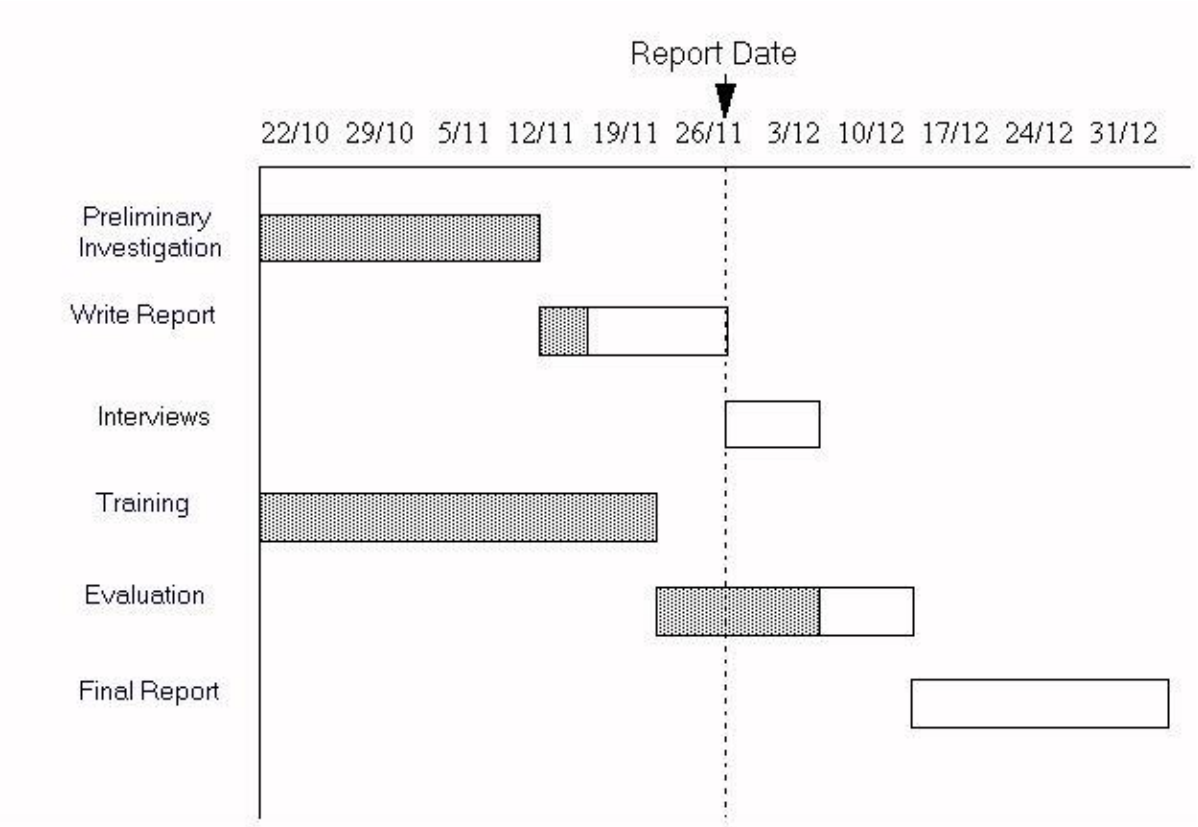


Figure 1: Gantt Chart

- Gantt charts may be simple versions created on graph paper or more complex automated versions created using project management applications such as Microsoft Project or Excel.
- A Gantt chart is constructed with a horizontal axis representing the total time span of the project, broken down into increments (for example, days, weeks, or months) and a vertical axis representing the tasks that make up the project
- (for example, if the project is outfitting your computer with new software, the major tasks involved might be: conduct research, choose software, install software).
- Horizontal bars of varying lengths represent the sequences, timing, and time span for each task.

What is a Gantt chart used for?

- Gantt Charts are mostly used for planning the Project Activities.
- Here are the uses of Gantt Type Chart:
    - Gantt charts used for visually representing the Project Schedules.
    - It can be used to make the Project Plans to understand clearly
    - It helps visually representing the Tasks
    - It can be used for understanding the Due date and Deadline of activities

Advantages of Gantt chart:

- It is to represent the Project schedules and Activities
- Easy to represent Tasks, Sub-tasks, Milestones and Projects Visually on a Graph
- Clear visibility of Dates and Time Frames
- Helps to effectively manage the Team
- Easy to Check the Project Status
- Gantt chart is good tool for presenting in Team Meetings

Disadvantages of Gantt chart:

- Require more efforts for Creating and Managing the Chart
- Updating a Chart is Very Time Consuming
- All Tasks are not visible in a single view of a Gantt
- Stacks represents only the time and not the hours of the work

Risk Management

- Risk is unavoidable in a business organization when undertaking projects.
- However, the project manager needs to ensure that risks are kept to a minimal.
- Risks can be mainly divided between two types,
    - **negative impact** risk and
    - **positive impact** risk.
- Not all the time would project managers be facing **negative impact** risks as there are **positive impact** risks too.
- Once the risk has been identified, project managers need to come up with a mitigation plan or any other solution to counter attack the risk.

What is Risk Management?:

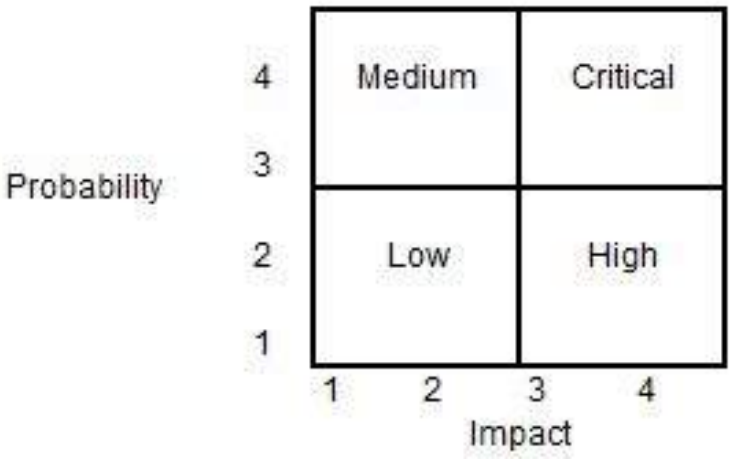
- ***Project risk management*** is the process of identifying, analysing and then responding to any risk that arises over the life cycle of a project.
- It will help the project remain on track and meet its goal.

Steps for Project Risk Management?:

- Managers can plan their strategy based on four steps of risk management which prevails in an organization.
- Following are the steps to manage risks effectively in an organization:
    - Risk Identification
    - Risk Quantification
    - Risk Response
    - Risk Monitoring and Control

- **Risk Identification:**
    - Managers face many difficulties when it comes to identifying and naming the risks that occur when undertaking projects.
    - These risks could be resolved through structured or unstructured brainstorming or strategies.
    - It's important to understand that risks pertaining to the project can only be handled by the project manager and other stakeholders of the project.
    - Risks, such as operational or business risks will be handled by the relevant teams.
    - The risks that often impact a project are,
        - supplier risk,
        - resource risk and
        - budget risk.
    - ***Supplier risk*** would refer to risks that can occur in case the supplier is not meeting the timeline to supply the resources required.
    - ***Resource risk*** occurs when the human resource used in the project is not enough or not skilled enough.
    - ***Budget risk*** would refer to risks that can occur if the costs are more than what was budgeted.

- **Risk Quantification:**
    - Risks can be evaluated based on quantity.
    - Project managers need to analyse the likely chances of a risk occurring with the help of a matrix.



- **Risk Response:**
    - When it comes to risk management, it depends on the project manager to choose strategies that will reduce the risk to minimal.
    - Project managers can choose between the four risk response strategies, which are outlined below.
        - Risks can be avoided
        - Pass on the risk
        - Take corrective measures to reduce the impact of risks
        - Acknowledge the risk
- **Risk Monitoring and Control:**
    - Risks can be monitored on a continuous basis to check if any change is made.
    - New risks can be identified through the constant monitoring and assessing mechanisms.

Risk Management Process:

- Following are the considerations when it comes to risk management process:
    - Each person involved in the process of planning needs to identify and understand the risks pertaining to the project.
    - Once the team members have given their list of risks, the risks should be consolidated to a single list in order to remove the duplications.

- Assessing the probability and impact of the risks involved with the help of a matrix.
- Split the team into subgroups where each group will identify the triggers that lead to project risks.
- The teams need to come up with a emergency plan whereby to strategically eliminate the risks involved or identified.
- Plan the risk management process. Each person involved in the project is assigned a risk in which he/she looks out for any triggers and then finds a suitable solution for it.

Risk Register:

- Risk register will often consists of diagrams to aid the reader as to the types of risks that are dealt by the organization and the course of action taken.
- The risk register should be freely accessible for all the members of the project team.

Project Risk: an Opportunity or a Threat?

- As mentioned above, risks contain two sides.
  - It can be either viewed as a negative element or
  - a positive element.
- Negative risks can be harmful factors that can careless situations for a project.
- Therefore, these should be control once identified.
- On the other hand, positive risks can bring about acknowledgements from both the customer and the management.
- All the risks need to be addressed by the project manager.

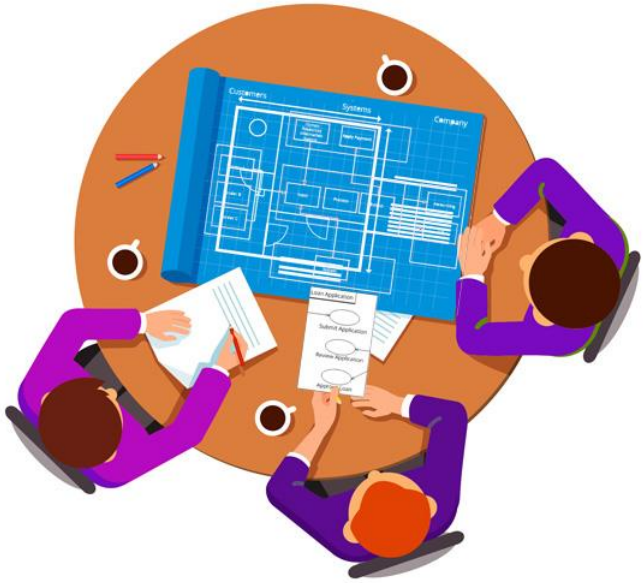
Chapter 5: Requirement Gathering and Analysis:

Requirement Gathering:

- Requirement gathering is the most important part of a software project.
- If we get the requirements wrong, the resulting application won't solve the users' problems.
- *Requirements* are the features that the application must provide.

What is Requirement Gathering?;

Requirements Gathering is the process of generating a list of requirements (functional, system, technical, etc.) from all the stakeholders (customers, users, vendors, IT staff) that will be used as the basis for the formal definition of what the project is.



The Importance of Requirement Gathering:

- Requirements need to be formally captured in one document.
- Requirement can be used as a reference during software development.
- Good gathering, processing and management of requirements is important as it sets clear targets for everyone to aim for.

Characteristics of Good Requirement:

- **Clear:**
    - Good requirements are clear, concise, and easy to understand.
- **Unambiguous:**
    - Unambiguous means a single interpretation.

- If a requirement is defined in such a manner that it can only be interpreted in one way, it means that the requirement is unambiguous.

- **Consistent:**
  - Consistency is an important aspect of requirements.
  - It means that all inputs to any process, must be processed similarly.
  - They cannot contradict each other

- **Prioritized:**
  - It's very important to prioritized the requirement.
  - What to include and what not.

- **Verifiable:**
  - Requirements must be verifiable.
  - Requirement can be verified and validated using following methods
    - Inspection
    - Demonstration
    - Testing

Requirement Categories:

- There are FOUR ways to categorise the requirement:
  1. Audience-Oriented Requirements
  2. FURPS
  3. FURPS+
  4. Common Requirements

1. Audience-Oriented Requirement:

- These categories focus on different audiences and the different points of view that each audience has.
- Audience-oriented requirements are divided into following categories:
- ***Business Requirements***
  - *Business requirements* lay out the project's high-level goals.
  - They explain what the customer hopes to achieve with the project.
- ***User Requirements***
  - *User requirements* (also called *stakeholder requirements)*, describe how the project will be used by the eventual end users.
- ***Functional Requirements***
  - *Functional Requirements* tells how the system should react to particular inputs, and how the system should behave in particular situations.
  - They're similar to the user requirements but they may also include things that the users won't see directly.

- ***Non-Functional Requirements***
  - *Non-functional requirements* are about the quality of the application's behaviour or constraints on how it produces a desired result.
  - They specify things such as the application's performance, reliability, and security characteristics.
- ***Implementation Requirements***
  - *Implementation requirements* include hiring new staff, buying new hardware, preparing training materials, and actually training the users to use the new system.

2. FURPS:

- FURPS is an acronym for this system's requirement categories: functionality, usability, reliability, performance, and scalability.
- It was developed by Hewlett-Packard (and later extended by adding a + at the end to get FURPS+).
- The following list summarizes the FURPS categories:
  - **Functionality** —What the application should do
  - **Usability** —What the program should look like.
  - **Reliability** —How reliable the system should be.
  - **Performance** —How efficient the system should be.
  - **Supportability** —How easy it is to support the application.

3. FURPS+:

- FURPS was extended into FURPS+ to add a few requirements categories that software engineers thought were missing.
- The following list summarizes the new categories:
  - **Design constraints** —Constraints on the design by other factors such as the hardware platform, software platform, network characteristics, or database.
  - **Implementation requirements** —These are constraints on the way the software is built.
  - **Interface requirements** —These are constraints on the system's interfaces with other systems.
  - **Physical requirements** —These are constraints on the hardware and physical devices that the system will use.

4. Common Requirements:

The following list summarizes some specific requirements that arise in many applications.

- **Screens** —What screens are needed?

- **Menus** —What menus will the screens have?
- **Navigation** —How will the users navigate through different parts of the system? Will they click buttons, use menus, or click forward and backward arrows? Or some combination of those methods?
- **Work flow** —How does data (work orders, purchase requests, invoices, and other data) move through the system?
- **Login** —How is login information stored and validated? What are the password formats (such as, must require at least one letter and number) and rules (as in, passwords must be changed monthly)?
- **User types** —Are there different kinds of users such as order entry clerk, shipping clerk, supervisor, and admin? Do they need different privileges?
- **Audit tracking and history** —Does the system need to keep track of who made changes to the data? (For example, so you can see who changed a customer to premier status.)
- **Archiving** —Does the system need to archive older data to free up space in the database? Does it need to copy data into a data warehouse for analysis?
- **Configuration** —Should the application provide configuration screens that let the system administrators change the way the program works?

Gathering Requirements:

- Following are the several techniques which can use to gather and refine requirements.
- **Brainstorming**
  - It is used to get as many ideas as possible from group of people.
  - It is used to identify possible solutions to problems, and clarify details of opportunities.
- **Document Analysis**
  - It is an important gathering technique.
  - It helps in evaluating the documentation of a present system.
  - It also useful in driving the gap analysis for scoping of the migration projects
- **Focus Group**
  - It is a gathering of people who are representative of the users or customers of a product to get feedback.
- **Interface Analysis**
  - Interfaces for a software product can be human or machine.
  - Integration with external systems and devices is just another interface.
- **Interview**
  - Interviews of users and stakeholders are important in creating wonderful software.

- **Observation**
  - Observation can either be passive or active.
  - Passive observation provides better feedback to refine requirements
  - Active observation works best for obtaining an understanding over an existing business process.
- **Prototyping**
  - It is a relatively modern technique for gathering requirements.
  - This approach is used to gather preliminary requirements which use to build an initial version of the solution - a prototype.
- **Requirements Workshops**
  - Workshops can be very effective for gathering requirements.
  - More structured than a brainstorming session, involved parties collaborate to document requirements.
- **Survey/ Questionnaire**
  - The survey insists the users to choose from the given options agree / disagree or rate something.
  - A well designed survey must give qualitative guidance for characterizing the market.

Recording Requirements:

Following are the various methods for recording requirements:

a) UML

b) User Stories

c) Use Cases

a) UML:

- **Unified Modeling Language (UML)** is a general purpose modelling language.
- Main aim of UML is to define a standard way to visualize the way a system has been designed.
- It is quite similar to blueprints used in other fields of engineering.
- UML is not a programming language.
- It is rather a visual language.
- UML diagrams are used to portray the behaviour and structure of a system.
- UML helps software engineers, businessmen and system architects with modelling, design and analysis.

b) User Stories:

- A user story is a tool used to capture a description of a software feature from an end user perspective.
- The user story describes the type of user, what they want and why.
- A user story helps to create a simplified description of a requirement.

- The purpose of a user story is to write down how a project will deliver value back to the user.
- User stories can be written by product developers to help prioritize how functionality is going to be added to a project over the project timeframe.

c) **Use Cases:**

- A use case is a methodology used in system analysis to identify, clarify, and organize system requirements.
- Use case is made up of a set of possible sequences of interactions between systems and users.
- It consists of a group of elements (for example, classes and interfaces).
- Use case should contain all system activities that have significance to the users.

Validation and Verification:

- ***Requirement validation*** is the process of making sure that the requirements say the right things.
- ***Requirement verification*** is the process of checking that the finished application actually satisfies the requirements.

Validation	Verification
• It includes testing and validating the actual product.	• It includes checking documents, design, codes and programs.
• Validation is the dynamic testing.	• Verification is the static testing.
• It includes the execution of the code.	• It does *not* include the execution of the code.
• It checks whether the software meets the requirements and expectations of a customer or not.	• It checks whether the software conforms to specifications or not.
• Validation is executed on software code with the help of testing team.	• Quality assurance team does verification.
• It comes after verification.	• It comes before validation.

Chapter 6: High-level Design

Design is not just what it looks like and feels like. Design is how it works.

—Steve Jobs

- High-level design is the first step in breaking an application into pieces that are small enough to implement.
- High-level design provides a view of the system at an abstract level.
- It shows how the major pieces of the finished application will fit together and interact with each other.
- The things which specify in a high-level design includes following:
    - Security
    - Hardware
    - User interface
    - Internal interfaces
    - External interfaces
    - Architecture
    - Reports
    - Other outputs
    - Database
    - Configuration data
    - Data flows and States
    - Training

Security:

- **Security** is an idea implemented to protect software against malicious attack and other hacker risks so that the software continues to function correctly under such potential risks.
- Security is necessary to provide integrity, authentication and availability.
- Following types of Security is required:
    - **Operating system security**
        - It includes the type of login procedures, password expiration policies, and password standards.
    - **Application security**
        - It means providing the right level of access to different users.

- **Data security**
  - It means keeping data secure from hackers.
- **Network security**
  - It means keeping network data secure
- **Physical security**
  - It means maintaining the security of office labs etc.

Hardware:

- Hardware is any physical device used in or with your machine.
- It includes mainframes, desktops, laptops, tablets, and phones.
- Mini-computers act sort of as a mini-mainframe that can serve a handful of users.
- Personal Digital Assistants (PDAs) are small computers that are basically miniature tablets.
- Additional hardware that need to specify include the following:
  - Printers
  - Network components (cables, modems, gateways, and routers)
  - Servers (database servers, web servers, and application servers)
  - Specialized instruments (scales, microscopes, programmable signs, and GPS units)
  - Audio and video hardware (webcams, headsets, and VOIP)

User Interface:

- **User interface** is the front-end application view to which user interacts in order to use the software.
- User can manipulate and control the software as well as hardware by means of user interface.
- User Interface can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination.
- The software becomes more popular if its user interface is:
  - Attractive
  - Simple to use
  - Responsive in short time
  - Clear to understand
  - Consistent on all interfacing screens

Internal Interfaces:

- With **internal interfaces** we control both endpoints of the connection.
- If something goes wrong we can look at all involved components and the infrastructure and decide how to fix it (and what to do to avoid it in the future).
- Those internal interface are typically between the webserver and caches, or between the application server and database.

External Interfaces:

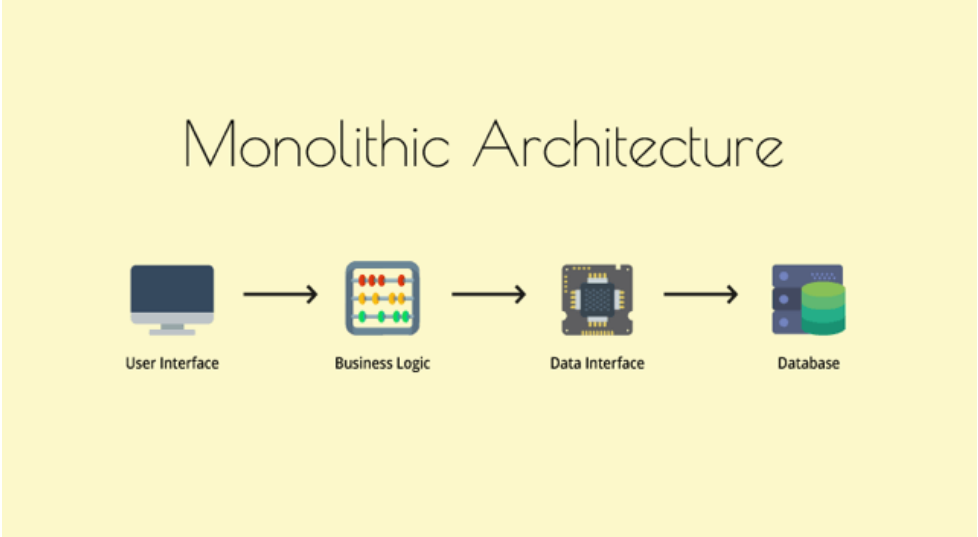
- **External interfaces** limit our influence to one part of the connection.
- Their second part is run by a third party, typically in a different data centre.
- Those interfaces are typically social media networks but can also be external mail servers, authentication, databases from partner companies, or anything else.

Architecture:

- An application's architecture describes how its pieces fit together at a high level.
- Developers use a lot of "standard" types of architectures.
- Many of these address particular characteristics of the problem being solved.
- Following are the most common architectures:
    1. Monolithic
    2. Clint/Server
    3. Component-Based
    4. Service Oriented
    5. Data-Centric
    6. Event-Driven
    7. Rule-Based
    8. Distributed
    9. Mix and Match

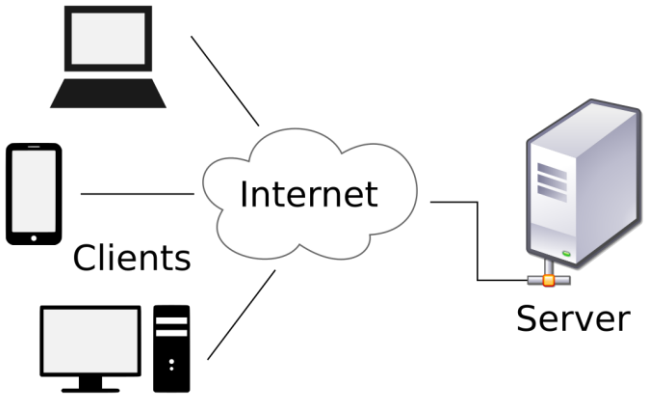
1. Monolithic:

- Monolith means composed all in one piece.
- In a ***monolithic architecture***, a single program does everything.
- It displays the user interface, accesses data, processes customer orders, prints invoices, launches missiles, and does whatever else the application needs to do.



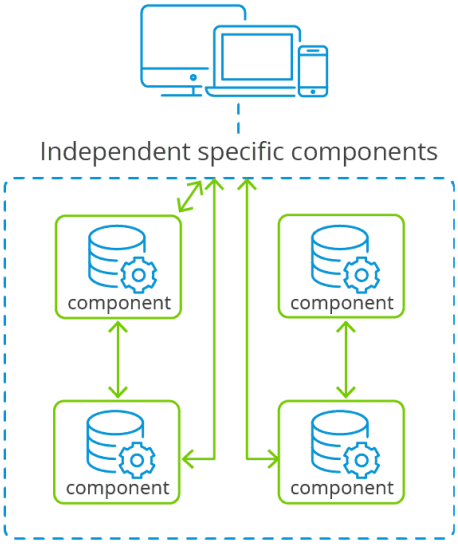
2. Client/ Server:

- ***Client/server architecture*** is a computing model in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.
- This type of architecture has one or more client computers connected to a central server over a network or internet connection.
- This system shares computing resources.



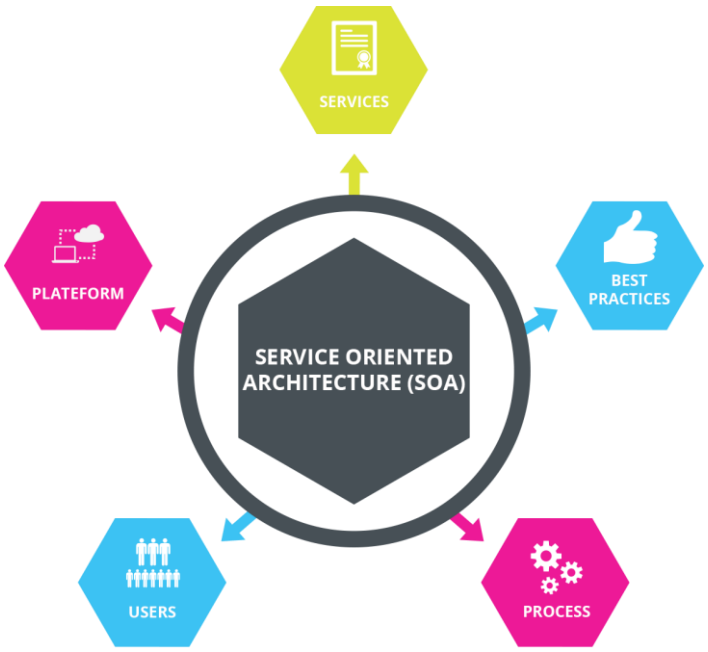
3. Component-Based:

- **Component-based architecture** is a way of building software with independent, modular, and reusable pieces.
- The primary objective of component-based architecture is to ensure **component reusability**.
- There are many standard component frameworks such as COM/DCOM, JavaBean, EJB, CORBA, .NET, web services, and grid services.



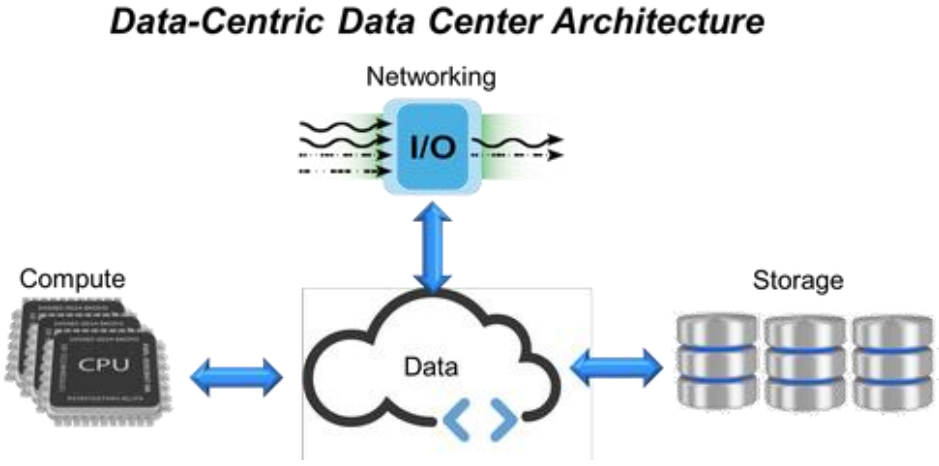
4. Service Oriented:

- **Service-Oriented Architecture (SOA)** is a style of software design where services are provided to the other components by application components, through a communication protocol over a network.
- In **service oriented architecture**, a number of services communicate with each other, in one of two ways: through passing data or through two or more services coordinating an activity.



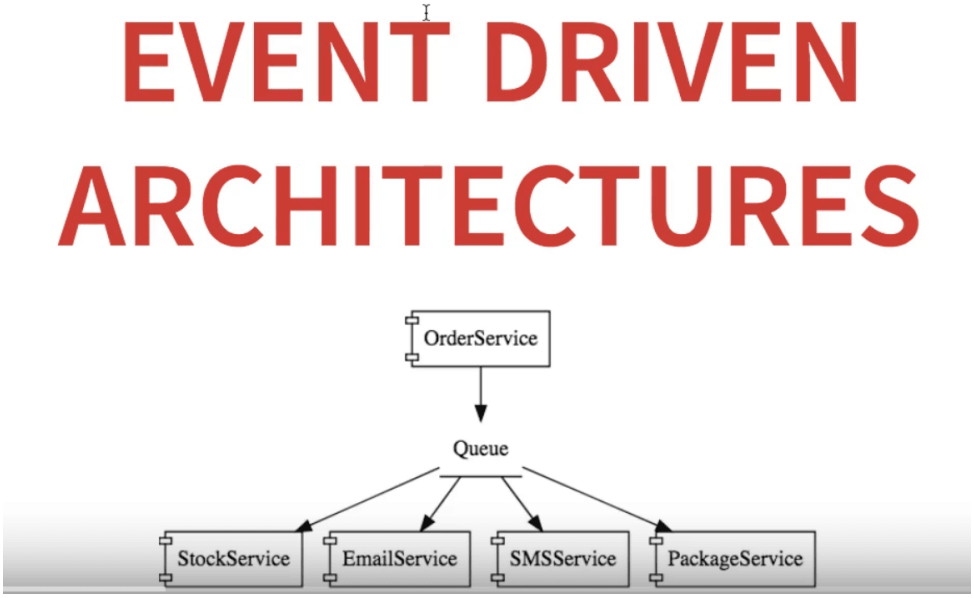
5. Data-Centric:

- ***Data-centric*** or ***database-centric architectures*** come in a variety of flavors that all use data in some central way.
- The following list summarizes some typical data-centric designs:
  - Storing data in a relational database system.
  - Using tables instead of hard-wired code to control the application.
  - Using stored procedures inside the database to perform calculations and implement business logic.



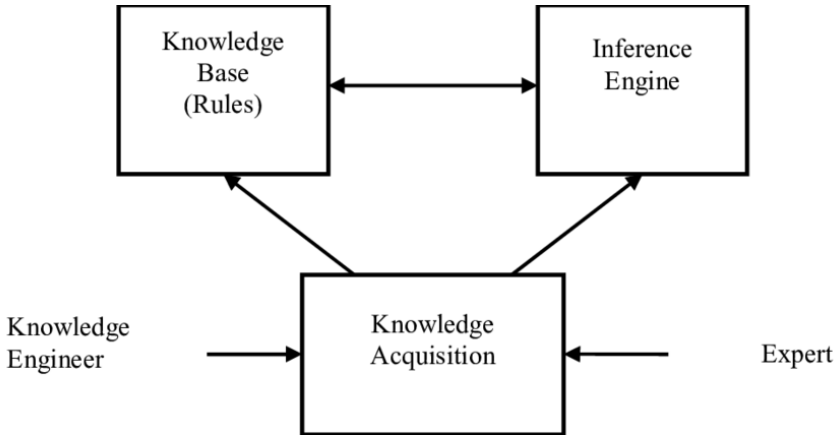
6. Event-Driven:

- **Event-driven architecture** (**EDA**) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.
- An *event* can be defined as "a significant change in state".
- For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold".



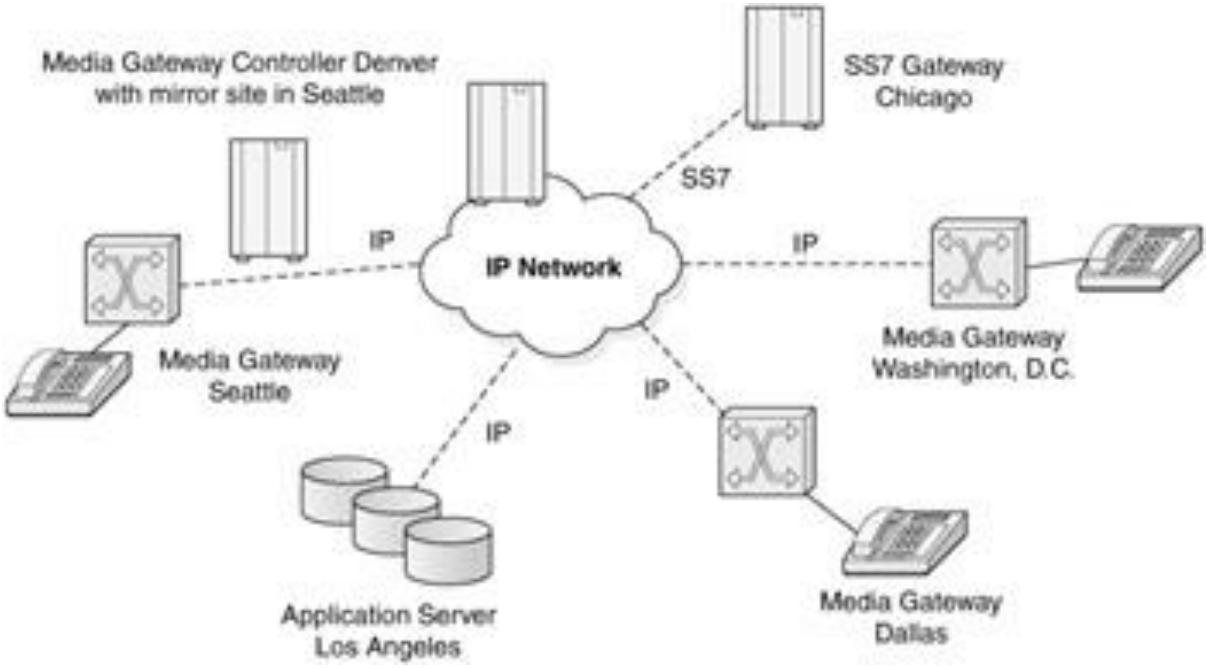
7. Rule-Based:

- **Rule-based architecture** are used as a way to store and manipulate knowledge to interpret information in a useful way.
- In software development, rule-based systems can be used to create software that will provide an answer to a problem in place of a human expert.
- These type of system may also be called an *expert system.*
- Rule-based systems are also used in AI (***a****rtificial* ***i****ntelligence*) programming and systems.



8. Distributed:

- In **distributed architecture**, components are presented on different platforms and several components can cooperate with one another over a communication network in order to achieve a specific objective or goal.
- In this architecture, information processing is not limited to a single machine rather it is distributed over several independent computers.



9. Mix and Match:

- An application doesn't need to stick with a single architecture.
- Different pieces of the application might use different design approaches.
- Mix and Match Architecture is the combination of various architecture.
- Combining different architectures can be impressive, like we can decide to go with an event-driven multitier approach using rule-based distributed components.

Reports:

- A **report** is a document that presents information in an organized format for a specific audience and purpose.
- Although summaries of **reports** may be delivered orally, complete **reports** are almost always in the form of written documents.
- A large application might have dozens or even hundreds of reports.

- Business applications include reports that deal with
    - customers (who's buying, who has unpaid bills, where customers live),
    - products (inventory, pricing, what's selling well), and
    - users (which employees are selling a lot, employee work schedules).

Other Outputs:

- In addition to normal reports, we can consider **other** kinds of **outputs** that the application might create.
- The other kinds of outputs are
    - printouts, web pages, data files, image files,
    - audio (to speakers or to audio files), video,
    - output to special devices (such as electronic signs),
    - e-mail, or text messages.

Database:

- Database design is an important part of most applications.
- The first part of database design is to decide what kind of database the program will need.
- Need to specify whether the application will store data in text files, XML files, a full-fledged relational database.
- Many applications store their data in relational databases such as Access, SQL Server, Oracle, or MySQL.
- In relational database, is important to sketch out the tables it contains and their relationships during high-level design.
- Need to use good database design practices to ensure that the database is properly normalized.

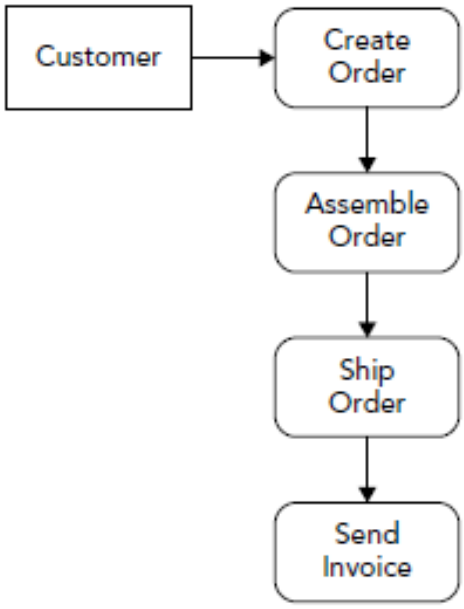
Configuration Data:

- **Configuration Data** means the Facility-specific **data** that is used in conjunction with the Software, including
    - without limitation,

- tuning and set point constants,
- Graphical, pictorial and text files, that translate the Software for the specific Facility environment.

Data Flows and States:

- **Data Flow** means the path of data from source document to data entry to processing to final reports.
- Data changes format and sequence (within a file) as it moves from program to program.
- Many applications use data that flows among different processes.
- Consider this diagram:
  - Here the data flow diagram shows how data such as a customer order flows through various processes.



Training:

- **Training** constitutes a basic concept in human resource development. It is concerned with developing a particular skill to a desired standard by instruction and practice.
- **Training** is the act of increasing the knowledge and skill of an employee for doing a particular job.
- In Software Engineering project manager train the users who are going work on the developed software.
- They are responsible to prepare training schedules, training materials so that user can able to work on the system.