A complete note on

Software Engineering and Project

CLASS 12

Table of Contents

Unit: One Introduction to Software Engineering

Program vs Software Product:

1. A program is a set of instructions that are given to a computer in order to achieve a specific task whereas software is when a program is made available for commercial business and is properly documented along with its licensing. Software = Program + documentation + licensing.
2. A program is one of the stages involved in the development of the software, whereas a software development usually follows a life cycle, which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

1.1 Software Engineering Definition

Software is a program or set of programs containing instructions that provide desired functionality. And Engineering is the process of designing and building something that serves a particular purpose and finds a cost-effective solution to problems.

Software Engineering *is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system.*

Dual Role of Software:

1. As a product

a) It delivers the computing potential across networks of Hardware.
b) It enables the Hardware to deliver the expected functionality.
c) It acts as an information transformer because it produces, manages, acquires, modifies, displays, or transmits information.

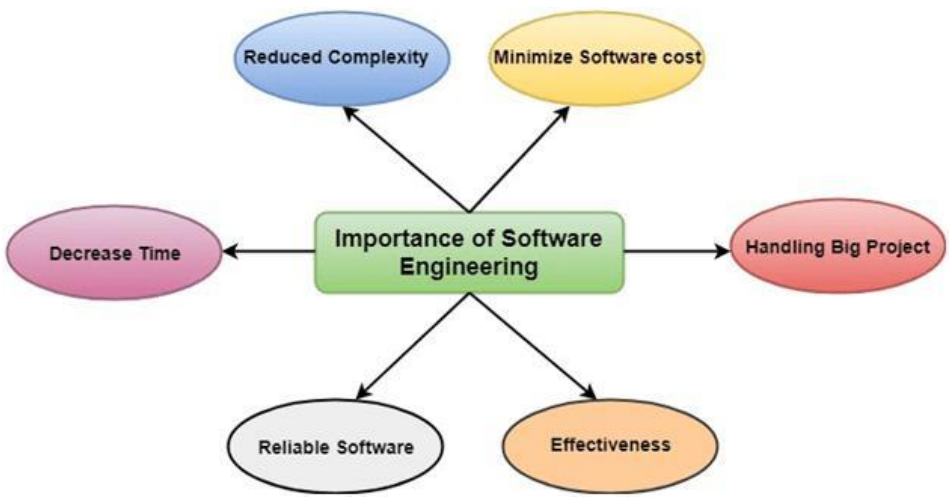
2. As a vehicle for delivering a product

a) It provides system functionality (e.g., payroll system)
b) It controls other software (e.g., an operating system)
c) It helps build other software (e.g., software tools)

Objectives of Software Engineering:

1. **Maintainability –** It should be feasible for the software to evolve to meet changing requirements.
2. **Efficiency –** The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. **Correctness –** A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.
4. **Reusability –** A software product has good reusability if the different modules of the product can easily be reused to develop new products.
5. **Testability –** Here software facilitates both the establishment of test criteria and the evaluation of the software with respect to those criteria.
6. **Reliability –** It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. **Portability –** In this case, the software can be transferred from one computer system or environment to another.
8. **Adaptability –** In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. **Interoperability** – Capability of 2 or more functional units to process data cooperatively.

1.2 Importance of Software Engineering



1 **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.

2 **To minimize software cost:** Software needs a lot of hard work and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.

3 **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So, if you are making your software according to the software engineering method, then it will decrease a lot of time.

4 **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So, to handle a big project without any problem, the company has to go for a software engineering method.

5 **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least its given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.

6 **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So, Software becomes more effective in the act with the help of software engineering.

1.3 Application of Software Engineering

Unit: Two Project Management Techniques

Software Project Management (SPM) is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored, and controlled. Software is a non-physical product. Most of the software products are made to fit clients' requirements. The most important is that the basic technology changes and advances so frequently and rapidly that experience of one product may not be applied to the other one. Such type of business and environmental constraints increase risk in software development hence it is essential to manage software projects efficiently. It is necessary for an organization to deliver quality products, keep the cost within the client's budget constrain and deliver the project as per schedule. Hence in order, software project management is necessary to incorporate user requirements along with budget and time constraints.

Software Project Management consists of different types of management:

1. **Conflict Management:** Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict. The goal of conflict management is to improve learning and group results including efficacy or performance in an organizational setting. Properly managed conflict can enhance group results.
2. **Risk Management:** Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resources to minimize, operate and control the possibility or effect of unfortunate events or to maximize the realization of opportunities.
3. **Requirement Management:** It is the process of analyzing, prioritizing, tracking, and documenting requirements and then supervising change and communicating to pertinent stakeholders. It is a continuous process during a project.
4. **Change Management:** Change management is a systematic approach for dealing with the transition or transformation of an organization's goals, processes, or technologies. The purpose of change management is to execute strategies for effecting change, controlling change, and helping people to adapt to change.
5. **Software Configuration Management:** Software configuration management is the process of controlling and tracking changes in the software, part of the larger cross-disciplinary field of configuration management. Software configuration management includes revision control and the inauguration of baselines.
6. **Release Management:** Release Management is the task of planning, controlling, and scheduling the build-in deploying releases. Release management ensures that the organization delivers new and enhanced services required by the customer while protecting the integrity of existing services.

Aspects of Software Project Management:

The list of focus areas it can tackle and the broad upsides of the Software Project. Management are:

1. **Planning:** The software project manager lays out the complete project's blueprint. The project plan will outline the scope, resources, timelines, techniques, strategy, communication, testing, and maintenance steps. SPM can aid greatly here.
2. **Leading:** A software project manager brings together and leads a team of engineers, strategists, programmers, designers, and data scientists. Leading a team necessitates exceptional communication, interpersonal, and leadership abilities. One can only hope to do this effectively if one sticks with the core SPM principles.
3. **Execution:** SPM comes to the rescue here also as the person in charge of software projects (if well versed with SPM/Agile methodologies) will ensure that each stage of the project is completed successfully. measuring progress, monitoring to check how teams' function, and generating status reports are all part of this process.
4. **Time management:** Abiding by a timeline is crucial to completing deliverables successfully. This is especially difficult when managing software projects because changes to the original project charter are unavoidable over time. To assure progress in the face of blockages or changes, software project managers ought to be specialists in managing risk and emergency preparedness. This Risk Mitigation and management is one of the core tents of the philosophy of SPM.

5. **Budget:** Software project managers, like conventional project managers, are responsible for generating a project budget and adhering to it as closely as feasible, regulating spending and reassigning funds as needed. SPM teaches us how to effectively manage the monetary aspect of projects to avoid running into a financial crunch later on in the project.
6. **Maintenance:** Software project management emphasizes continuous product testing to find and repair defects early, tailor the end product to the needs of the client, and keep the project on track. The software project manager makes ensuring that the product is thoroughly tested, analyzed, and adjusted as needed. Another point in favor of SPM.

2.1. Introduction to Project Development Techniques

Software development techniques and management are the concepts and processes that engineers use to structure software development projects. In projects involving dozens or hundreds of developers, expansive codebases, and rapid changes to code, it's critical to have a systematic software development technique and management methodology in place to ensure efficient operations across the project.

Software development techniques define specific strategies and processes that developers are supposed to follow in order to work efficiently alongside other developers. The exact rules vary from one technique to another. But in general, they include guidelines that govern how code written by individual developers is integrated into a larger codebase, how and when code is compiled, when to deploy new application updates, and how to plan a new set of feature updates. Most techniques also include conceptual guidelines designed to help developers decide what is important, and what is not, when writing code or planning features.

In most cases, software development technique and management strategies center around high-level recommendations, not rigid practices. They usually don't require developers to use particular tools, and they don't define exactly how developers spend each hour of their day. Instead, the various techniques leave a lot of room for interpretation and adaptation to unique circumstances.

Benefits of Software Development Techniques

The main reason to adopt a particular software development technique or management style is to provide coordination and structure for your project. By embracing a specific development technique, teams establish operational standards and procedures that help individual developers work together efficiently. Establishing such standards and procedures is particularly important in the context of large, complex software development projects. If you have hundreds of developers, each of whom is generating hundreds of lines of code per month, it can be quite challenging to ensure that the code written by one developer doesn't conflict or overlap with the code written by another. It's also hard to make sure that the work performed by individual developers adds up to meaningful feature updates that can be released according to a preset schedule.

1. Provides coordination and structure to software projects.
2. Enables teams to establish operational standards and procedures.
3. Helps individual developers work together efficiently and ensures they are all on the same page.
4. Prevents developers from wasting time writing code that is not useful.
5. Provides a better user experience.

The drawback is that it reduces the flexibility of a software project.

Some Common Software Development Techniques

Here's a look at the five most common software development technique and management strategies used by developers today.

Waterfall

The waterfall development technique organizes development projects into linear phases. One phase must be completed before the next phase can begin. Typically, waterfall results in infrequent releases — one or two application updates per year — and it can lead to delays because unexpected problems with one phase of development bring the entire project to a halt.

That said, waterfall is one of the simplest development techniques to implement and manage, which is the main reason why it remains in use by some projects today.

Agile

The agile development technique emerged in the early 2000s as part of an effort to address the shortcomings of the waterfall methodology. The agile technique prioritizes more rapid application updates, which it seeks to enable by breaking development efforts into small, incremental sets of changes.

Agile's major downsides include the risk that constant small changes will fail to add up to meaningful large change over time. Agile also requires closer coordination between developers. Nonetheless, most developers today see agile as a more effective technique than waterfall.

DevOps and CI/CD

DevOps uses a development technique called continuous integration/continuous delivery, or CI/CD, to enable "continuous" release of application updates. The changes are not literally continuous, but DevOps usually involves the release of new application updates on a weekly — or, in many cases, daily — basis.

There's some debate about whether DevOps and CI/CD are a form of agile, or if they are a distinct methodology. But in general, most developers treat DevOps as an enhanced form of the agile technique, partly because DevOps addresses the entire application lifecycle instead of just the development phase.

Lean development

The lean development technique is similar in many ways to agile, with the difference that lean development focuses especially on efficiency. Lean developers carefully evaluate the value of a new feature before they begin to build it, and they strive to minimize wasted effort and wasted code at all stages of the development process.

In these ways, lean helps to avoid the risk (which, as noted above, can be a problem under the agile technique) of making rapid changes that lack real value.

Extreme programming

The extreme programming technique is another methodology derived from agile development, with the major difference being extreme programming emphasizes simplicity and quality. It encourages developers to avoid writing code unless the code is strictly necessary, and it prioritizes thorough testing of applications prior to release.

These strategies also help avoid the risks of pointless code releases or unnecessary complex codebases that can arise in an agile project.

2.2. PERT Introduction and Implementation

Project Evaluation and Review Technique (PERT) is a procedure through which activities of a project are represented in its appropriate sequence and timing. It is a scheduling technique used to schedule, organize and integrate tasks within a project. PERT is basically a mechanism for management planning and control which provides blueprint for a particular project. All of the primary elements or events of a project have been finally identified by the PERT.

In this technique, a PERT Chart is made which represent a schedule for all the specified tasks in the project. The reporting levels of the tasks or events in the PERT Charts is somewhat same as defined in the work breakdown structure (WBS).

Characteristics of PERT:

The main characteristics of PERT are as following:

1. It serves as a base for obtaining the important facts for implementing the decision-making.
2. It forms the basis for all the planning activities.
3. PERT helps management in deciding the best possible resource utilization method.
4. PERT take advantage by using time network analysis technique.
5. PERT presents the structure for reporting information.
6. It helps the management in identifying the essential elements for the completion of the project within time.

Advantages of PERT:

It has the following advantages:

1. Estimation of completion time of project is given by the PERT.
2. It supports the identification of the activities with slack time.
3. The start and dates of the activities of a specific project is determined.
4. It helps project manager in identifying the critical path activities.
5. PERT makes well organized diagram for the representation of large amount of data.

Disadvantages of PERT:

It has the following disadvantages:

1. The complexity of PERT is more which leads to the problem in implementation.
2. The estimation of activity time is subjective in PERT which is a major disadvantage.
3. Maintenance of PERT is also expensive and complex.
4. The actual distribution of may be different from the PERT beta distribution which causes wrong assumptions.
5. It under estimates the expected project completion time as there is chances that other paths can become the critical path if their related activities are deferred.

The PERT Method Implementation Steps

Track the following steps while creating a PERT Chart;

1. List the activities and milestones: The first step is to determine the tasks required to complete the project.

2. Determine the sequence of activities: The second step is to determine the order of the activities. Which activity is the predecessor which one is the successor? It is easy to determine the sequence of some activities however the sequence of some tasks may require deep analysis.

3. Build a network diagram: The third step is to create the network diagram with the help of software or by hand and place the activities on the diagram.

4. Estimate the activity durations: The PERT Method uses three duration estimates for activities which are;

- **Optimistic Estimate:** The shortest time required to complete the task.
- **Pessimistic Estimate:** The longest time required to complete the task.
- **Most Likely Estimate:** The most possible time (probable duration) required to complete the task.

Expected time is calculated with the help of the PERT Analysis formula below Expected time = (Optimistic + 4 x Most likely + Pessimistic) / 6

5. Determine the critical path: The critical path is the longest path of the network diagram. Forward and backward pass calculations is used to determine the critical path.

The PERT Method Example

We discussed the PERT Definition and analyze the formula above. Now we will provide a simple PERT Method example with the solution. In the following PERT Method Example, you are a project manager of a power plant project. You tracked the steps mentioned above and listed the following inputs;

- All the Activities
- Predecessors
- Optimistic, Pessimistic, and Most Likely Activity Durations, Expected duration for each activity.

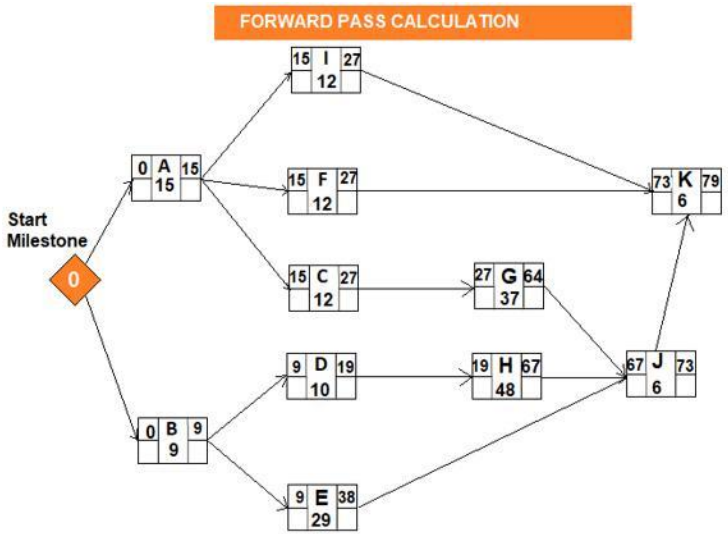
All the inputs are listed in the table below.

Activity	Description	Predecessors	Optimistic Duration (To)	Pessimistic Duration (Tp)	Most likely Duration (Tm)	Expected Duration (To + 4Tm + Tp)/6
0	Start Milestone	-	0	0	0	0
A	Select Technical Staff	0	12	18	15	15
B	Site Survey	0	6	12	9	9
C	Select Equipments	A	9	15	12	12
D	Prepare Designs	B	6	18	9	10
E	Bring Utilities to the Site.	B	18	36	30	29
F	Interview Applicants and Fill Positions	A	9	15	12	12
G	Purchase the Equipment.	C	36	42	36	37
H	Construct the Power Plant	D	42	54	48	48
I	Develop an Information System.	A	6	18	12	12
J	Install the Equipment.	H,G,E	3	9	6	6
K	Train the Staff to Run the System	F,J,I	3	9	6	6

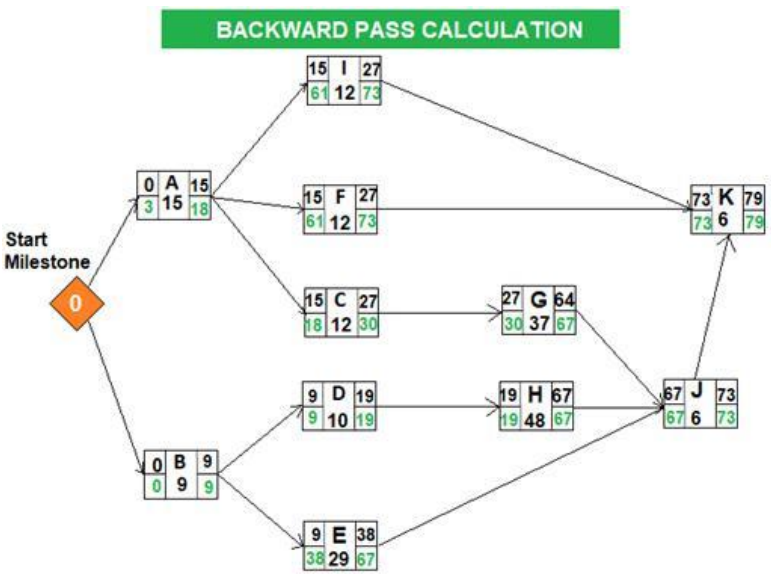
After building a network diagram and estimating the activity durations, you will determine the critical path by making forward and backward pass calculations.

Earliest Start	Duration	Earliest Finish
Activity Label		
Latest Start	Float	Latest Finish

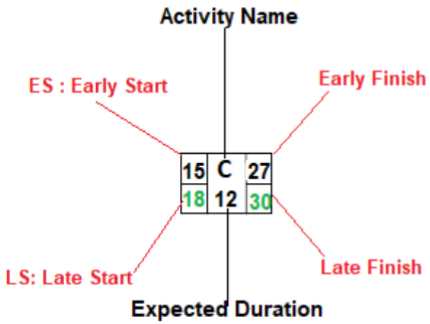
Forward Pass Calculations specify the minimum dates at which each activity can be performed and, ultimately, the minimum duration of a project.



Backward Pass Calculations of Program Evaluation and Review Technique determine the latest dates by which each activity can be performed without increasing the project's minimum duration.

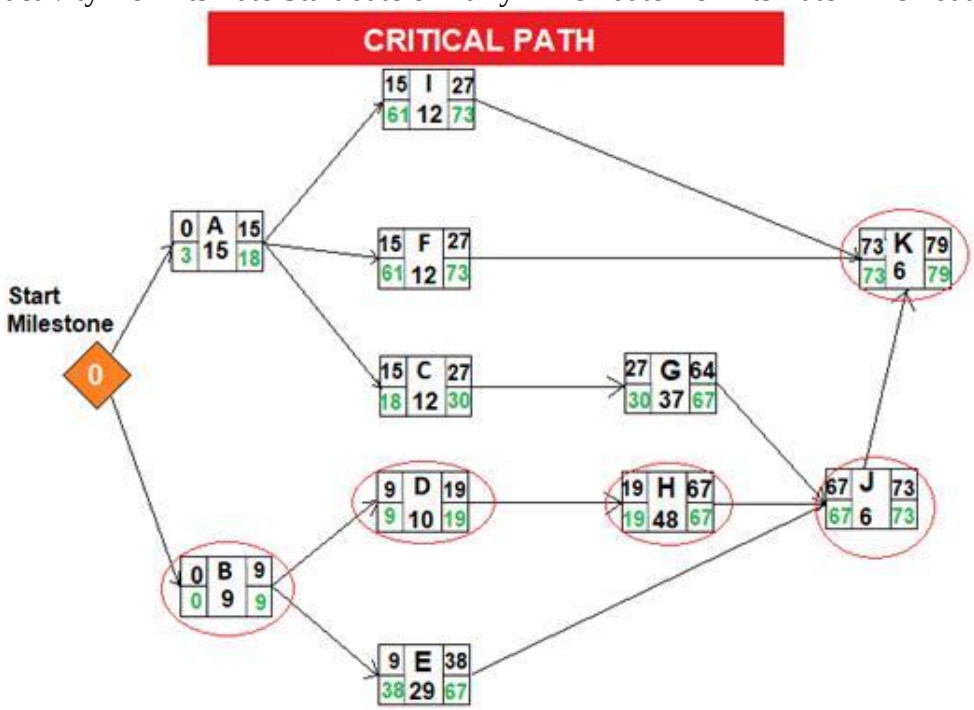


After completing the backward pass calculation, you can easily determine the critical path. In project management, *"float" or "slack" is the amount of time that a task can be delayed without affecting the deadlines of other subsequent tasks, or the project's final delivery date. Total float/slack is 0 on the critical path.*



PERT Method -Activity
Total Float: LS – ES = 18-15 = 3
Total Float: LF – EF = 30-27 = 3

The total float can be calculated by subtracting the Early Start date of an activity from its Late Start date or Early Finish date from its Late Finish date.



Critical Path

When we analyze the network diagram we will see that there are some paths and every path have duration. *The critical path is the longest path in the network diagram and the total float of the critical path is zero.*

2.3. CPM Introduction and Implementation

The Critical Path Method is used to determine the shortest possible time to complete the project. The CPM is a sequence of activities of a project's starting activity & time and the project's finish time and activity. This tool is totally based on mathematical calculations and is used for scheduling project activities. The CPM was developed by James E. Kelley and Morgan R. Walker in the 1950s. Initially, the CPM Method was used for managing plant maintenance projects. CPM method is used for planning and scheduling activity. This method also shows the interrelationship between the time and cost of the project. Critical Path Method helps us to assess the different possibilities of project planning.

CPM Method is used in the below fields

- Software Development
- Construction
- Aerospace & Defense
- Research Projects
- Product Development and many more fields

Advantages of CPM Method:

→ This tool is used for planning, scheduling, monitoring, and controlling various types of projects.
→ It helps us for proper communications between departments and functions.
→ We can estimate the expected project completion date.
→ This tool is very easy to use.
→ We can identify the critical path with the help of this tool.
→ It is very useful in monitoring costs
→ This tool is visually very effective so all people can easily understand.
→ We can also reduce the project completion timelines by using this method.
→ CPM Method improves the decision-making ability of the team.
→ By using this tool, we can able to determine which activities can be delayed without delaying the project

Disadvantages of CPM Method:

→ The reliability of this tool is based on accurate estimates and assumptions.
→ This tool does not give surety of the success of the project.
→ If we have many activities then our network diagram will also be complicated.
→ The estimate of activity timing is subjective and depends on the judgment.

Limitations of the Critical Path Method:

→ This method is not feasible for large and complex problems. We need to use software for this.
→ We cannot able to allocate resources by this method.

There are five steps in the CPM Method as mentioned below.

Step_1: List Out the Activity
Step_2: Forward Pass Calculation
Step_3: Backward Pass Calculation
Step_4: Float Calculation for Each Activity
Step_5: Identifying the Critical Path

Critical Path Method Example

Refer to the below example for understanding how to identify the critical path in project management. We will follow the below five steps for understanding the CPM Method. For calculation, we will refer to the activity from A to K, completion time duration, and the immediate predecessors as mentioned in 1st step.

Step 1: List Out the Activity

→ First of all, we will list out all activity, their immediate predecessors, and completion time. For this refer to the below table.
→ Now with the help of the above table, we will simplify and arrange all activity and time with the critical path method.
→ In this example, A, B, C, D, E, F, G, H, I, J, and K are representing the activities.

Data for Construction of CPM Network

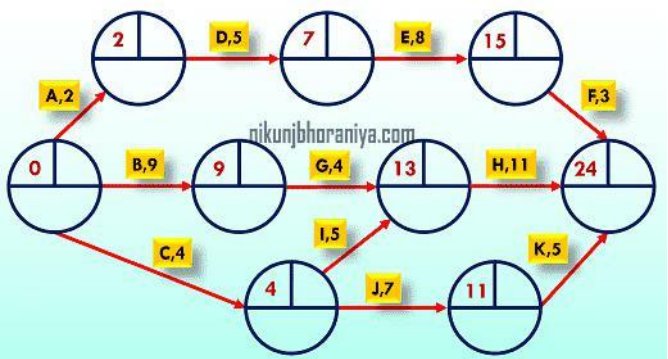
Activity	Immediate Predecessors	Completion Time
A		2
B		9
C		4
D	A	5
E	D	8
F	E	3
G	B	4
H	G & I	11
I	C	5
J	C	7
K	J	5

nikunjbhoraniya.com

Step 2: Forward Pass Calculation

→ In the forward pass, we calculate the Starting durations of all the activities.
→ So, we can easily show the calculations from the below picture.
→ It starts from the first activity and after that, we will add the time period as per the activity.
→ By repeating this process, we will get the final time of the end of all activity with the different paths.

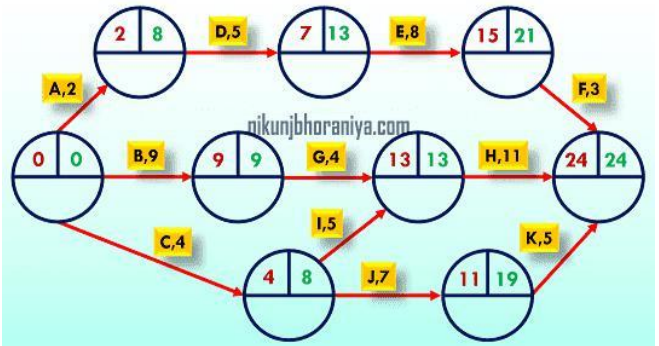
Forward Pass Calculation



Step 3: Backward Pass Calculation

→ Now, in the third step, we will calculate the backward pass.
→ In the backward pass calculation, we will take the end activities time as start time as a finish time. In this example, it is 24.
→ The backward pass is processed by subtracting the duration of the activities leading to the end activity node.
→ By repeating this process, we will arrive at the finish time which means at the beginning.

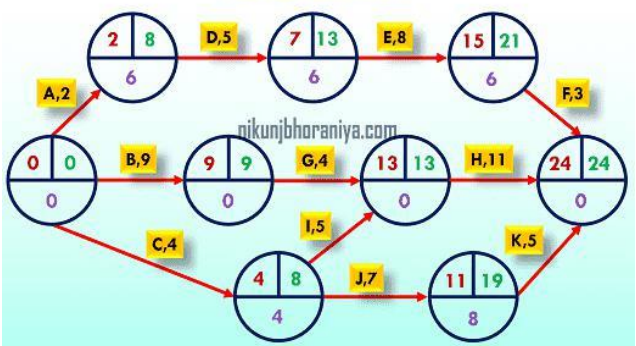
Backward Pass Calculation



Step 4: Float Calculation for Each Activity

→ Now the fourth step is to calculate the float of each activity.
→ The float is the simple difference between the backward pass count and the forward pass count.
→ We can easily see the float value for all activities from the below picture.

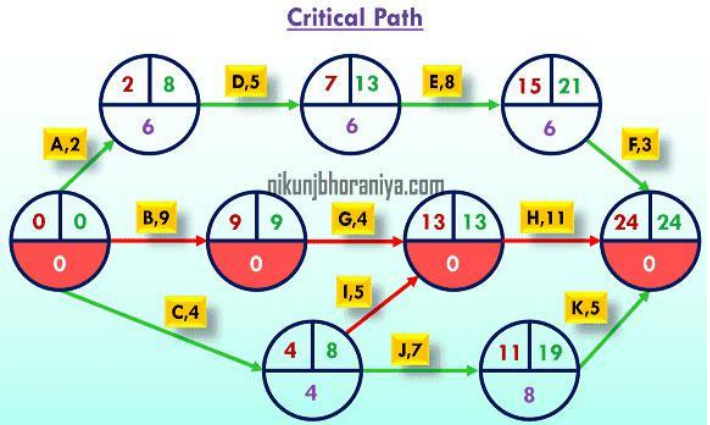
Float Calculation



Step 5: Identifying the Critical Path

→ Now the final step is to identify the critical path out of all different possible paths.
→ The critical path is the longest path in the network diagram.
→ The total float of the critical path is zero.

→ It is shown as light red color in the below picture.
→ If activity durations are determined wrong then the critical path of the project will be wrong.
→ If there are many other similar duration paths available in the project at that time it is very critical to identify the Critical Path of the project.



2.4. Implementation of Project Management Techniques in Real World

There are various tools and techniques available for Software Project Management. Depending upon the availability of resources and feasibility of project any method or tool can be adopted.

Traditional Methodologies	**Change Management Methodologies**
• The Waterfall Model • The Critical Path Method • The Critical Chain Method	• Event Chain Methodology • Extreme Project Management
Agile Methodologies	**Other Methodologies**
• Agile Project Management • The Scrum Framework • Kanban • Extreme Programming • Adaptive Project Framework	• Rational Unified Process • Program Evaluation & Review Technique • The PMBOK Method • Gantt Chart • WBS

Unit: Three Software Development Phases

Software Development is the development of software for distinct purposes. For software development, there is a specific programming language like Java, Python, C/C++, etc. The entire process of software development isn't as simple as its definition, it's a complicated process. Accordingly, it requires an efficient approach from the developer in the form of the Software Development Life Cycle (SDLC).

Proper planning and execution are the key components of a successful software development process.

Hence, it's vital for a software developer to have prior knowledge of this software development process.

These 6 stages are discussed below.

1. Planning and Requirements Analysis
2. Defining Requirements [System Requirements Specification (SRS)]
3. System Design
4. System Development
5. System Integration and Testing
6. System Implementation [Deployment], Maintenance and reviews [Support]

- **Stage-1: Planning and Requirement Analysis:**
  Planning is the crucial step in everything and so as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from the inputs from the customers, sales department/market surveys.
  The information from this analysis forms the building block of a basic project. The quality proof of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.

- **Stage-2: Defining Requirements:**
  In this stage, all the requirements for the target software are specified. These requirements get approval from the customers, market analysts, and stakeholders.
  This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

- **Stage-3: Designing Architecture:**
  SRS is a reference for software designers to come out with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS). This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for the development.

- **Stage-4: Developing Product:**
  At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

- **Stage-5: Product Testing and Integration:**
  After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC.
  Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

- **Documentation, Training and Support:**
  Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions and maintenance. Documentation also provides information about how to use the product. Thoroughly-written documentation should involve the required documentation. Software architecture documentation, technical documentation and user documentation. Training in an attempt to improve

the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.

- **Stage 6: Deployment and Maintenance of Product:**
  After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. Because it is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.

3.1 Importance and need of SDLC

Software Development Life Cycle is the sequential list of activities that a software undergoes before its completed. Hence, it defines all the steps to be followed while developing a software. Therefore, SDLC allows developers to identify the problems or risks that might arise during development and appropriate solutions to them.

- It acts as a guide to the project and meet client's objectives.
- It helps in evaluating, scheduling and estimating deliverables.
- It provides a framework for a standard set of activities.
- It ensures correct and timely delivery to the client.

3.2 System Study

System Study refers to the detailed study of the software before starting the development. It might be a comparative study in reference to the existing similar software or a new research from scratch or reverse engineering of a software. It allows the developers to forsee potential risks, propose appropriate solutions and make a systematic work plan.

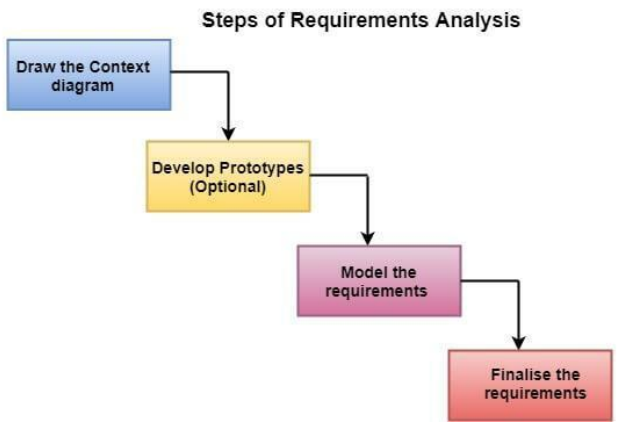
3.3 Feasibility study and its types

This phase involves analyzing the feasibility; of the plans and requirements made in the first stage. It studies weather the project can be accomplished or not. During feasibility study not only, the time constraints but account of resources like manpower and technology are also considered. The following are some of the questions that need to be answered in this phase:

- **Economic feasibility:** Are there enough funds to invest in the development of the software?
- **Legal feasibility:** Is the company able to comply with related regulations?
- **Operational feasibility:** Is it possible to meet the workflow and operational requirements set in the requirement stage?
- **Technical feasibility:** Does the organization have the necessary technology and human resources for the SDLC process?
- **Schedule feasibility:** Can the development process be completed on time?

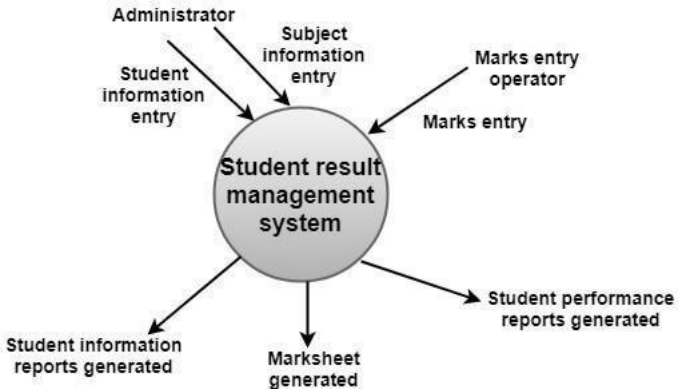
3.4 System Requirements and Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others. The various steps of requirement analysis are shown in figure below.



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the

system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want. We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalize the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

3.5 System Requirements Specification (SRS)

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

1. Introduction
   - (i) Purpose of this document
   - (ii) Scope of this document
   - (iii) Overview
2. General description
3. Functional Requirements
4. Interface Requirements
5. Performance Requirements
6. Design Constraints
7. Non-Functional Attributes
8. Preliminary Schedule and Budget
9. Appendices

Software Requirement Specification (SRS) Format as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-functional depending upon type of requirement. The interaction between different customers and contractor is done because it's necessary to fully understand needs of customers.

Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

1. ***Introduction***:

   (i) Purpose of this Document: At first, main aim of why this document is necessary and what's purpose of document is explained and described.

   (ii) Scope of this document: In this, overall working and main objective of document and what value it will provide to customer is described and

explained. It also includes a description of development cost and time required.

(iii) Overview: In this, description of product is explained. It's simply summary or overall review of product.

2. ***General description:*** In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.
3. ***Functional Requirements:*** In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
4. ***Interface Requirements:*** In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.
5. ***Performance Requirements:*** In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.
6. ***Design Constraints:*** In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc.
7. ***Non-Functional Attributes:*** In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.
8. ***Preliminary Schedule and Budget:*** In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
9. ***Appendices:***
   In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

3.6 System Design

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language. It is the phase where the SRS document is converted into a format that can be implemented and decides how the system will operate.

In this phase, the complex activity of system development is divided into several smaller sub-activities, which coordinate with each other to achieve the main objective of system development.

Following are the activities performed in System Design Phase

1. Identify Design Goals
2. System Decomposition
3. Identification of Concurrency
4. Hardware Allocation
5. Database Management
6. Global Resources Hand Leap
7. Software Control Implementation
8. Boundary Conditions

Hence, the activities of design phase can be summarized as:

- Includes the design of application, network, databases, user interfaces, and system interfaces.
- Transform the SRS document into logical structure, which contains detailed and complete set of specifications that can be implemented in a programming language.
- Create a contingency, training, maintenance, and operation plan.
- Review the proposed design. Ensure that the final design must meet the requirements stated in SRS document.
- Finally, prepare a design document which will be used during next phases

Following are the purposes of Software design:

1. **Correctness:** Software design should be correct as per requirement.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:** Resources should be used efficiently by the program.
4. **Flexibility:** Able to modify on changing needs.
5. **Consistency:** There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

Inputs to System Design: System design takes the following inputs.

- Statement of work
- Requirement determination plan
- Current situation analysis
- Proposed system requirements including a conceptual data model, modified DFDs, and Metadata (data about data).

Outputs for System Design: System design gives the following outputs

- Infrastructure and organizational changes for the proposed system.
- A data schema, often a relational schema.
- Metadata to define the tables/files and columns/data-items.
- A function hierarchy diagram or web page map that graphically describes the program structure.
- Actual or pseudocode for each module in the program.
- A prototype for the proposed system.

3.7 System Development

- Implement the design into source code through coding.
- Combine all the modules together into training environment that detects errors and defects.
- A test report which contains errors is prepared through test plan that includes test related tasks such as test case generation, testing criteria, and resource allocation for testing.
- Integrate the information system into its environment and install the new system.

3.8 System Testing

3.8.1 Software Testing

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases. The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:

1. **Verification:** it refers to the set of tasks that ensure that the software correctly implements a specific function.
2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

Verification: "Are we building the product right?"
Validation: "Are we building the right product?"

What are different types of software testing?

Software Testing can be broadly classified into two types:

1. **Manual Testing:** Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

   Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.
2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.

Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

What are the different types of Software Testing Techniques?

Software testing techniques can be majorly classified into two categories:

1. **Black Box Testing:** The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software is known as black-box testing.
2. **White-Box Testing:** The technique of testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.

Black Box Testing	White Box Testing
Internal workings of an application are not required.	Knowledge of the internal workings is a must.
Also known as closed box/data-driven testing.	Also known as clear box/structural testing.
End users, testers, and developers.	Normally done by testers and developers.
This can only be done by a trial and error method.	Data domains and internal boundaries can be better tested.

What are different levels of software testing?

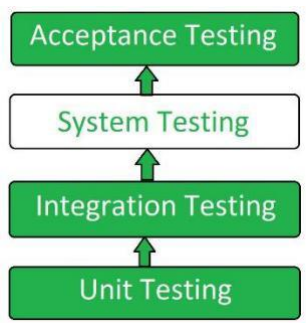
Software level testing can be majorly classified into 4 levels:

1. **Unit Testing:** A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
2. **Integration Testing:** A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
3. **System Testing:** A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.
4. **Acceptance Testing:** A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

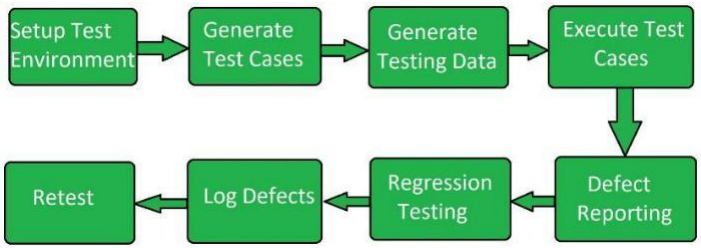
3.8.2 System Testing

System testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.

System Testing Process: System Testing is performed in the following steps:



- **Test Environment Setup:** Create testing environment for the better-quality testing.
- **Create Test Case:** Generate test case for the testing process.
- **Create Test Data:** Generate the data that is to be tested.
- **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:** Defects in the system are detected.
- **Regression Testing:** It is carried out to test the side effects of the testing process.
- **Log Defects:** Defects are fixed in this step.
- **Retest:** If the test is not successful then again test is performed.



Types of System Testing:

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Tools used for System Testing:

Some tools used for system testing are: JMeter, Gallen Framework, Selenium

Advantages of System Testing:

- The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing.

Disadvantages of System Testing:

- This testing is time consuming process than another testing technique since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.

3.9 System implementation

Implementation is a process of ensuring that the information system is operational. It involves;

- Constructing a new system from scratch
- Constructing a new system from the existing one.

Implementation allows the users to take over its operation for use and evaluation. It involves training the users to handle the system and plan for a smooth conversion.

Training

The personnel in the system must know in detail what their roles will be, how they can use the system, and what the system will or will not do. The success or failure of well-designed and technically elegant systems can depend on the way they are operated and used.

Training Systems Operators

Systems operators must be trained properly such that they can handle all possible operations, both routine and extraordinary. The operators should be trained in what common malfunctions may occur, how to recognize them, and what steps to take when they come.

User Training

End-user training is an important part of the computer-based information system development, which must be provided to employees to enable them to do their own problem solving. User training involves how to operate the equipment, troubleshooting the system problem, determining whether a problem that arose

is caused by the equipment or software. Most user training deals with the operation of the system itself. The training courses must be designed to help the user with fast mobilization for the organization.

Conversion

It is a process of migrating from the old system to the new one. It provides understandable and structured approach to improve the communication between management and project team.

3.10 System Maintenance and reviews

Maintenance means restoring something to its original conditions. Enhancement means adding, modifying the code to support the changes in the user specification. System maintenance conforms the system to its original requirements and enhancement adds to system capability by incorporating new requirements. Thus, maintenance changes the existing system, enhancement adds features to the existing system, and development replaces the existing system. It is an important part of system development that includes the activities which corrects errors in system design and implementation, updates the documents, and tests the data.

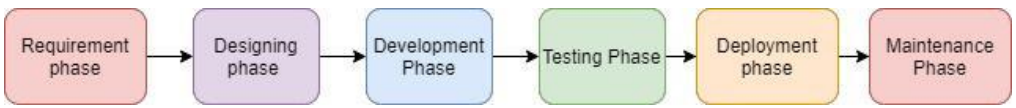
Maintenance Types

System maintenance can be classified into three types

- **Corrective Maintenance:** Enables user to carry out the repairing and correcting leftover problems.
- **Adaptive Maintenance**: Enables user to replace the functions of the programs.
- **Perfective Maintenance**: Enables user to modify or enhance the programs according to the users' requirements and changing needs.

Unit: Four Software Development Life Cycle Models

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.
SDLC specifies the task(s) to be performed at various stages by a software engineer/developer. It ensures that the end product is able to meet the customer's expectations and fits in the overall budget. Different phases of the software development cycle

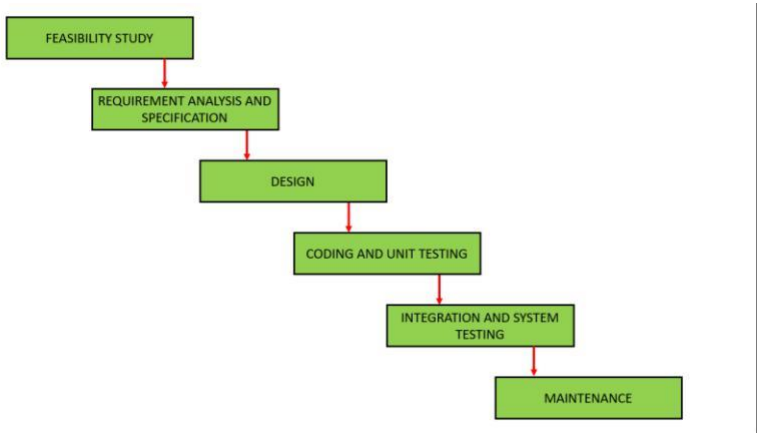


There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "**Software Development Process Models**." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

4.1 Waterfall Model

The classical waterfall model is the basic **software development life cycle** model. It is very simple but idealistic. Earlier this model was very popular but nowadays it is not used. But it is very important because all the other software development life cycle models are based on the classical waterfall model.

The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus, the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other. The different sequential phases of the classical waterfall model are shown in the below figure:



Let us now learn about each of these phases in brief detail:

1. **Feasibility Study**: The main goal of this phase is to determine whether it would be financially and technically feasible to develop the software. The feasibility study involves understanding the problem and then determining the various possible strategies to solve the problem. These different identified solutions are analyzed based on their benefits and drawbacks, the best solution is chosen and all the other phases are carried out as per this solution strategy.
2. **Requirements analysis and specification**: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.
   - **Requirement gathering and analysis:** Firstly, all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (an inconsistent requirement is one in which some part of the requirement contradicts some other part).
   - **Requirement specification:** These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between the development team and

customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

3. **Design**: The goal of this phase is to convert the requirements acquired in the SRS into a format that can be coded in a programming language. It includes high-level and detailed design as well as the overall software architecture. A Software Design Document is used to document all of this effort (SDD)

4. **Coding and Unit testing**: In the coding phase software design is translated into source code using any suitable programming language. Thus, each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

5. **Integration and System testing**: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.
System testing consists of three different kinds of testing activities as described below:

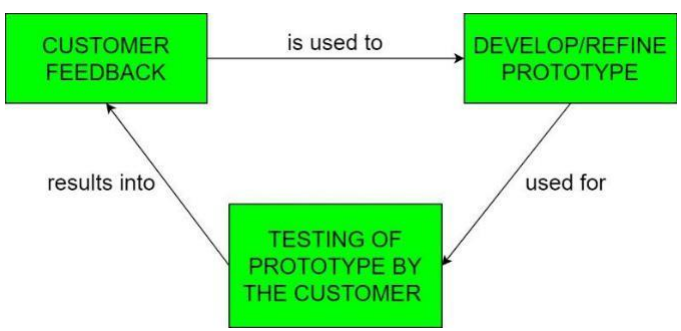
   - **Alpha testing:** Alpha testing is the system testing performed by the development team.
   - **Beta testing:** Beta testing is the system testing performed by a friendly set of customers.
   - **Acceptance testing:** After the software has been delivered, the customer performed acceptance testing to determine whether to accept the delivered software or reject it.

6. **Maintenance:** Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is 60% of the total effort spent to develop a full software. There are basically three types of maintenance:
   - **Corrective Maintenance:** This type of maintenance is carried out to correct errors that were not discovered during the product development phase.
   - **Perfective Maintenance:** This type of maintenance is carried out to enhance the functionalities of the system based on the customer's request.
   - **Adaptive Maintenance:** Adaptive maintenance is usually required for porting the software to work in a new environment such as working on a new computer platform or with a new operating system.

Advantages	• Simple and easy to use. • Clearly defined stages. • Good for small projects. • Detailed requirement analysis is done before starting the project.
Disadvantages	• Cannot accommodate changes. • No working software is achieved until all phases are completed • Not good for complex projects • Limited interactions with the customer
Applications	• When project is small. • Requirement is clear. • Resources are available and trained. • Requirements are not changing

4.2 Prototyping Model

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale facsimile of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

In this process model, the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle. The process starts by interviewing the customers and developing the incomplete high-level paper model. This document is used to build the initial prototype supporting only the basic functionality as desired by the customer. Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues until the user approves the prototype and finds the working model to be satisfactory.

There are four types of models available:

A) Rapid Throwaway Prototyping

This technique offers a useful method of exploring ideas and getting customer feedback for each of them. In this method, a developed prototype need not necessarily be a part of the ultimately accepted prototype. Customer feedback helps in preventing unnecessary design faults and hence, the final prototype developed is of better quality.

B) Evolutionary Prototyping

In this method, the prototype developed initially is incrementally refined on the basis of customer feedback till it finally gets accepted. In comparison to Rapid Throwaway Prototyping, it offers a better approach which saves time as well as effort. This is because developing a prototype from scratch for every iteration of the process can sometimes be very frustrating for the developers.

C) Incremental Prototyping

In this type of incremental Prototyping, the final expected product is broken into different small pieces of prototypes and being developed individually. In the end, when all individual pieces are properly developed, then the different prototypes are collectively merged into a single final product in their predefined order. It's a very efficient approach that reduces the complexity of the development process, where the goal is divided into sub-parts and each sub-part is developed individually. The time interval between the project's beginning and final delivery is substantially reduced because all parts of the system are prototyped and tested simultaneously. Of course, there might be the possibility that the pieces just do not fit together due to some lack of ness in the development phase – this can only be fixed by careful and complete plotting of the entire system before prototyping starts.

D) Extreme Prototyping

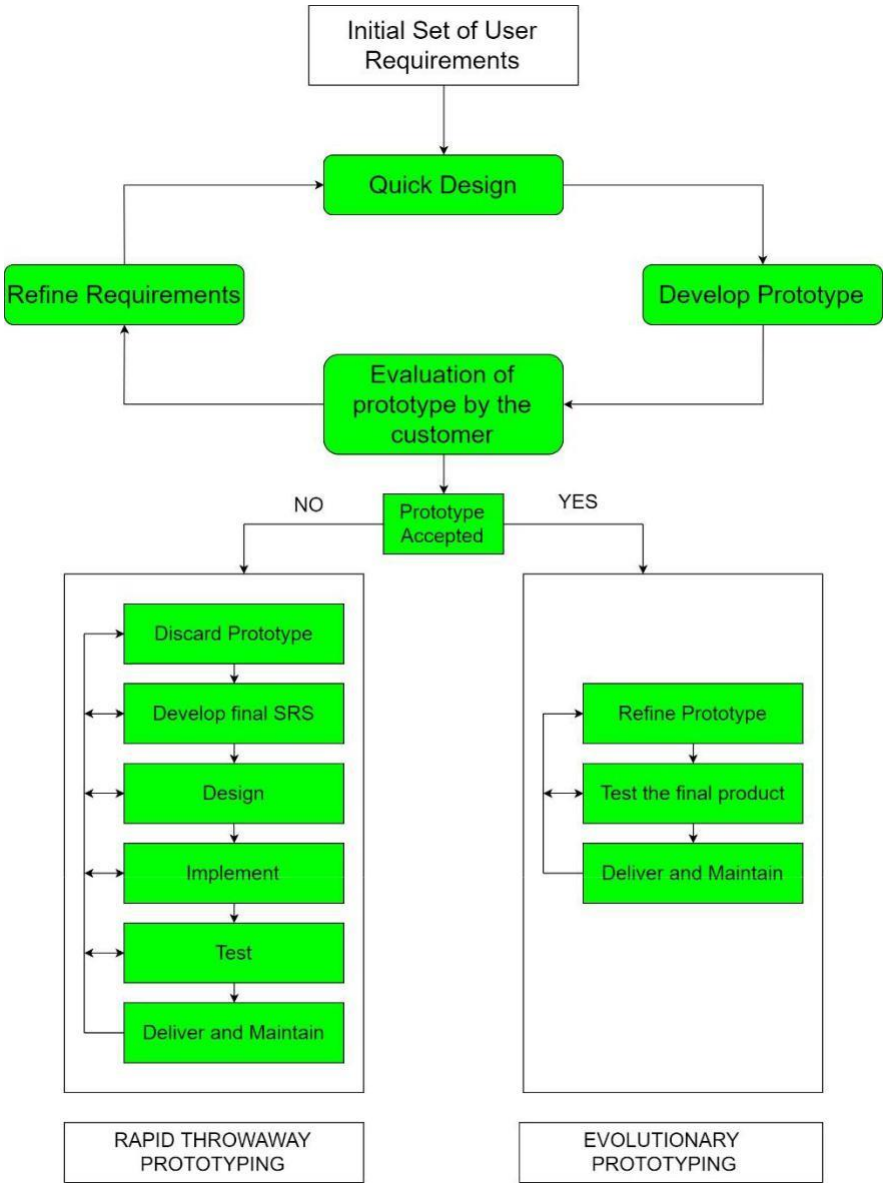
This method is mainly used for web development. It consists of three sequential independent phases:

D.1) In this phase a basic prototype with all the existing static pages are presented in the HTML format.

D.2) In the 2nd phase, Functional screens are made with a simulated data process using a prototype services layer.

D.3) This is the final step where all the services are implemented and associated with the final prototype.

This Extreme Prototyping method makes the project cycling and delivery robust and fast, and keeps the entire developer team focus centralized on products deliveries rather than discovering all possible needs and specifications and adding unnecessitated features.



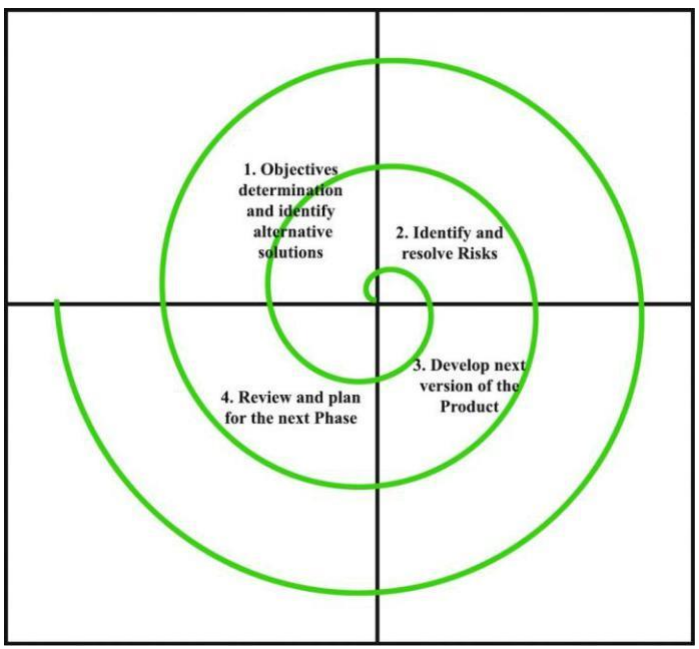
Advantages	• The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort. • New requirements can be easily accommodated as there is scope for refinement. • Missing functionalities can be easily figured out. • Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software. • The developed prototype can be reused by the developer for more complicated projects in the future. • Flexibility in design.
Disadvantages	• Costly w.r.t time as well as money. • There may be too much variation in requirements each time the prototype is evaluated by the customer. • Poor Documentation due to continuously changing customer requirements. • It is very difficult for developers to accommodate all the changes demanded by the customer. • There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer. • After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon. • Developers in a hurry to build prototypes may end up with sub-optimal solutions. • The customer might lose interest in the product if he/she is not satisfied with the initial prototype.
Applications	• When requirements of the product are not clearly understood or unstable. • Changing requirements. • System with complex algorithm and interface.

4.3 Spiral Model

Spiral model is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **Phase of the software development process.** The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

The below diagram shows the different phases of the Spiral Model:



Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below

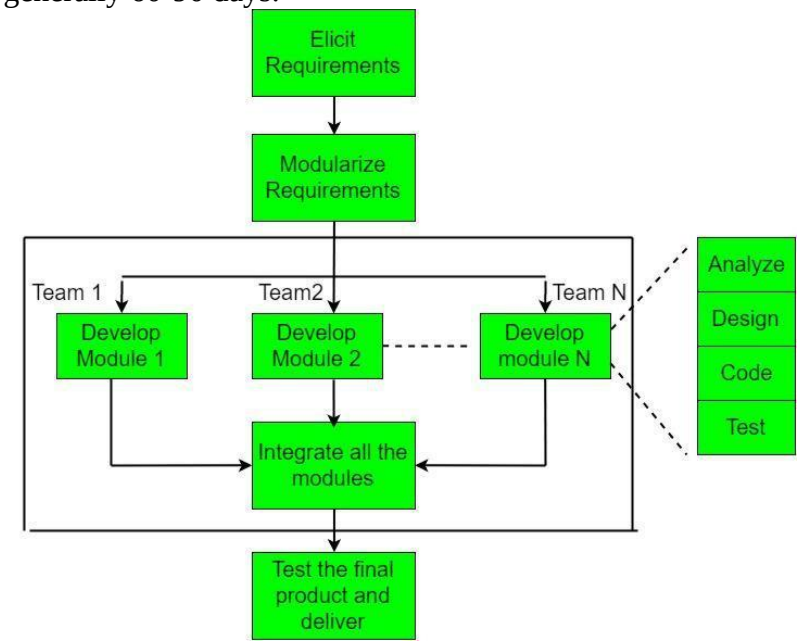
1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.
3. **Develop next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

Advantages	• Change can be incorporated at any time. • Cost estimation and risk handling is easy. • Development is fast • Features are added in systematic approach Customer feedbacks are addressed.
assessment. **Disadvantages**	• Risk of going off the schedule. • Needs risk management experts for risk • Demands strict to the protocols. • Too much documentation needed. • Not good for small projects.
Applications	• When project is large. • Releases are required frequently. • When risk and cost evaluation is important. When changes are required at any time.

4.4 RAD Model

The Rapid Application Development Model was first proposed by IBM in the 1980s. The critical feature of this model is the use of powerful development

tools and techniques. A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product. Development of each module involves the various basic steps as in the waterfall model i.e analyzing, designing, coding, and then testing, etc. as shown in the figure. Another striking feature of this model is a short time span i.e the time frame for delivery(time-box) is generally 60-90 days.



The use of powerful developer tools such as JAVA, C++, Visual BASIC, XML, etc. is also an integral part of the projects. This model consists of 4 basic phases:

1. **Requirements Planning –** It involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.
2. **User Description –** This phase consists of taking user feedback and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.
3. **Construction –** In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform process and data models into the final working product. All the required modifications and enhancements are too done in this phase.
4. **Cutover –** All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

The process involves building a rapid prototype, delivering it to the customer, and taking feedback. After validation by the customer, the SRS document is developed and the design is finalized.

Advantages	• Changing requirements can be accommodated. • Progress can be measured. • Increase reusability of components. • Production with fewer people in short time. • Accommodates customer feedback.
Disadvantages .	• Only modular systems can be developed. • Requires highly skilled developers. Not good for small and low budget projects. • Increased management complexity.
Applications.	• System is modular. • When high skilled manpower is available. Budget is not a problem. • If customer interaction is important.

Unit: Five Software Analysis and Design Tools

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do. Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

5.1 Dataflow Diagram (DFD)

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

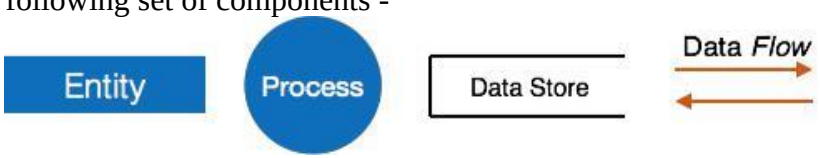
Types of DFD

Data Flow Diagrams are either Logical or Physical.

- **Logical DFD** - This type of DFD concentrates on the system process, and flow of data in the system. For example in a Banking software system, how data is moved between different entities.
- **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and closer to the implementation.

DFD Components

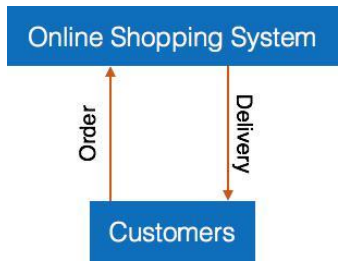
DFD can represent Source, destination, storage and flow of data using the following set of components -

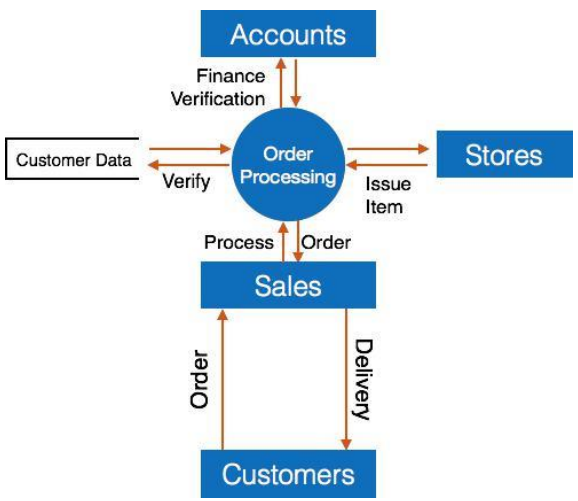


- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangle with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.





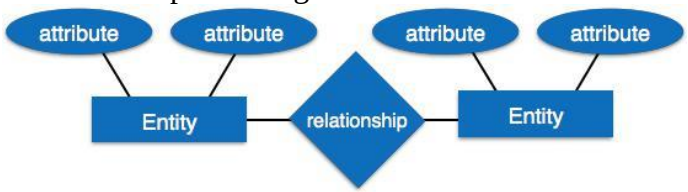
- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

- **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

5.2 Entity Relationship (ER) Diagram

Entity-Relationship model is a type of database model based on the notion of real-world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them. ER Model is best used for the conceptual design of database. ER Model can be represented as follows:

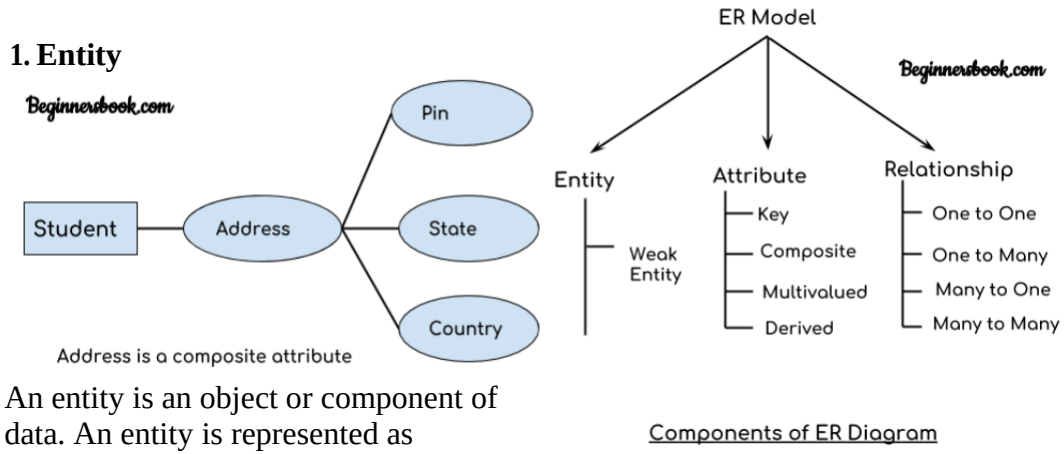


Components of a ER Diagram

As shown in the above diagram, an ER diagram has three main components:

1. Entity 2. Attribute 3. Relationship

1. Entity



Components of ER Diagram

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram. For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many student's study in a single college. We will read more about relationships later, for now focus on entities.

Weak Entity:

An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example; a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

2. Attribute

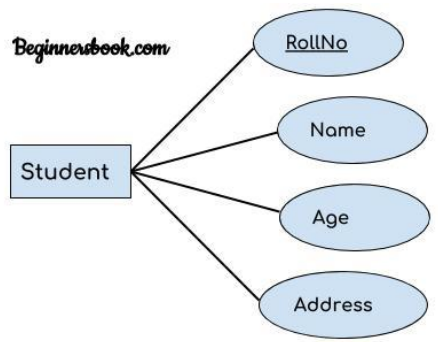
An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

1. Key attribute: A key attribute can uniquely identify an entity from an entity set. For example, student roll number can uniquely identify a student from a set of students. Key attribute is represented by oval same as other attributes however the **text of key attribute is underlined**.

2. Composite attribute:

An attribute that is a combination of other attributes is known as composite attribute. For example, in student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.



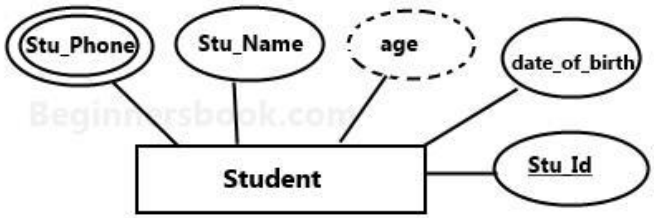
3. Multivalued attribute:

An attribute that can hold multiple values is known as multivalued attribute. It is represented with **double ovals** in an ER Diagram. For example – A person can have more than one phone numbers so the phone number attribute is multivalued.

4. Derived attribute:

A derived attribute is one whose value is dynamic and derived from another attribute. It is represented by **dashed oval** in an ER Diagram. For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).

E-R diagram with multivalued and derived attributes:



3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One Relationship

When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.

2. One to Many Relationship

When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.

3. Many to One Relationship

When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.

4. Many to Many Relationship

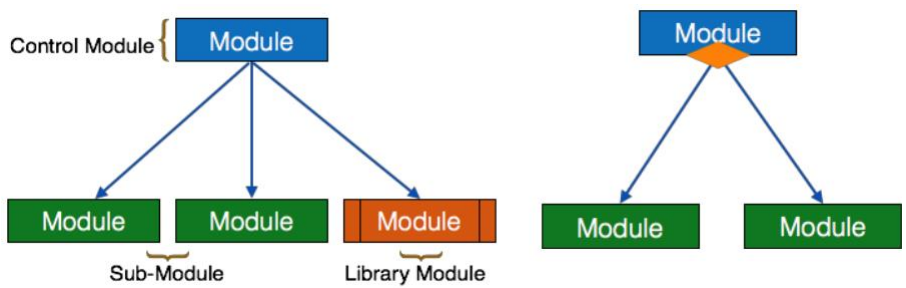
When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationships. For example, a can be assigned to many projects and a project can be assigned to many students.

5.3 Structure Chart

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD. Structure chart represents hierarchical structure of modules. At each layer a specific task is performed. Here are the symbols used in construction of structure charts -

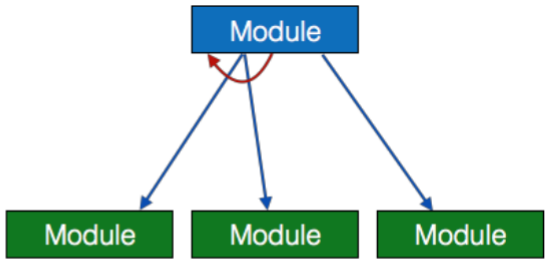
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.

- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



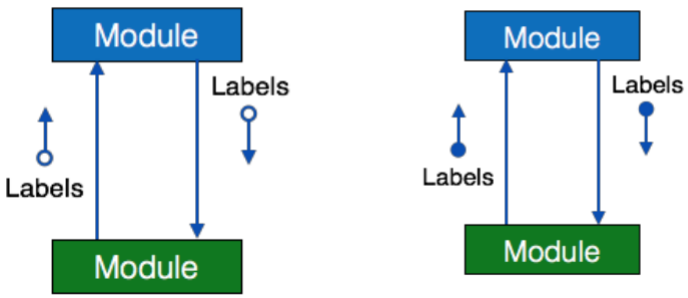
- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.

- **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



- **Data flow** - A directed arrow with empty circle at the end represents data flow.

- **Control flow** - A directed arrow with filled circle at the end represents control flow.



5.4 Decision Table

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Creating Decision Table

To create the decision table, the developer must follow basic four steps:

- Identify all possible conditions to be addressed
- Determine actions for all identified conditions
- Create Maximum possible rules
- Define action for each rule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Example

Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions.

We list all possible problems under column conditions and the prospective actions under column Actions.

	Conditions/Actions	**Rules**							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

Table: Decision Table – In-house Internet Troubleshooting

5.5 Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions *to a problem/decision based on given conditions.*
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

- In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Decision Tree Steps

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

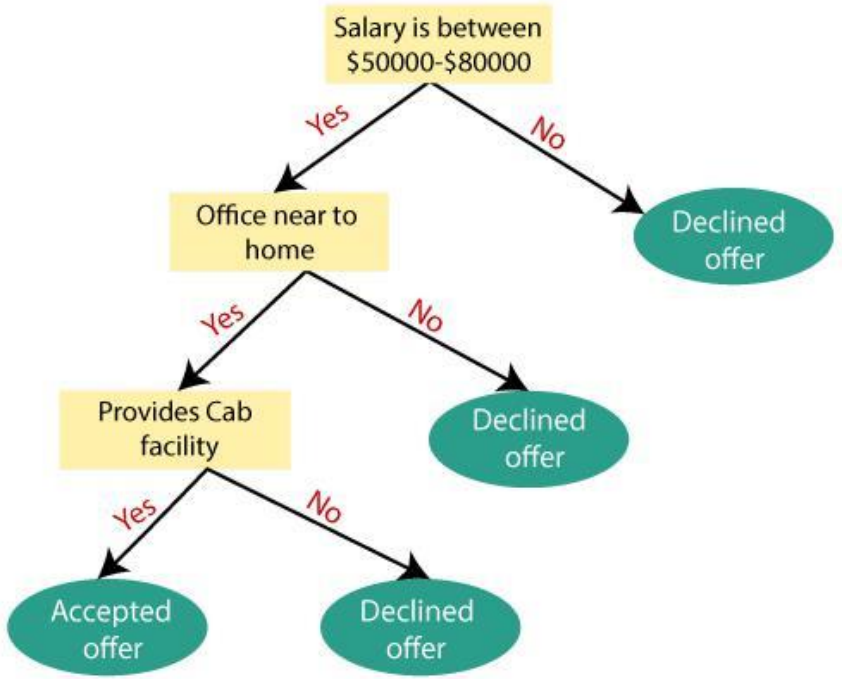
Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

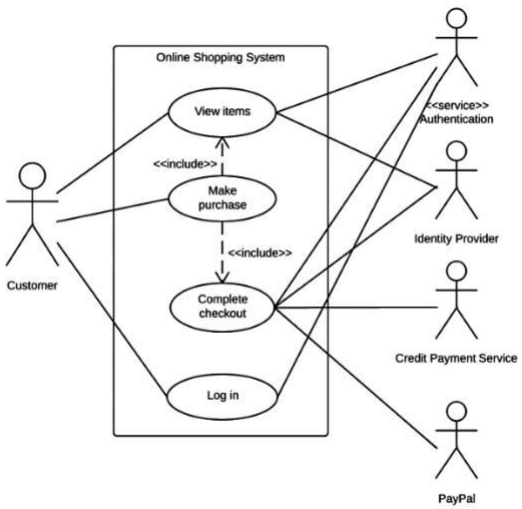
- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

5.6 Use Case Diagram

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.

Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.



ADVANTAGES

- The use cases are mainly composed of narrative text. Hence, unlike many other modeling techniques, the non-technical stake holders (e.g. customers, end users, salesperson etc) are also able understand the model for the software system. This means that feedback can be obtained at a very early stage of the development from the customers and the end users.
- Another major advantage of use case modeling is that it requires the identification of exceptional scenarios for the use cases. This helps in discovering subtle alternate requirements in the system.
- The use case model can be utilized in several other aspect of software development as well, e.g. Cost Estimation, Project Planning, Test Case Preparation and User Documentation.
- The use case diagram provides a comprehensive summary of the whole software system in a single illustration.

DISADVANAGES

- They do not capture the non-functional requirements easily.
- There might be a learning curve for the developer and/or specially, the client in using these use cases.

Use case diagram components

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

- Actors: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- System: A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- Goals: The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Use case diagram symbols and notation

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams. You can use this guide to learn how to draw a use case diagram if you need a refresher. Here are all the shapes you will be able to find in Lucidchart:

- Use cases: Horizontally shaped ovals that represent the different uses that a user might have.
- Actors: Stick figures that represent the people actually employing the use cases.
- Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
- System boundary boxes: A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system.

For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.

- Packages: A UML shape that allows you to put different elements into groups. Just as with component diagrams, these groupings are represented as file folders.

5.7 Sequence Diagram

The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

Benefits of a Sequence Diagram

1. It explores the real-time application.
2. It depicts the message flow between the different objects.
3. It has easy maintenance.
4. It is easy to generate.
5. Implement both forward and reverse engineering.
6. It can easily update as per the new change in the system.

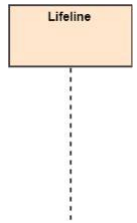
The drawback of a Sequence Diagram

1. In the case of too many lifelines, the sequence diagram can get more complex.
2. The incorrect result may be produced, if the order of the flow of messages changes.
3. Since each sequence needs distinct notations for its representation, it may make the diagram more complex.
4. The type of sequence is decided by the type of message.

Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It models generic interactions or some certain instances of interaction.

Notations of a Sequence Diagram

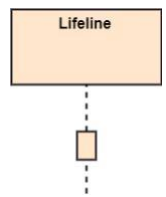


Lifeline

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.

Actor

A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



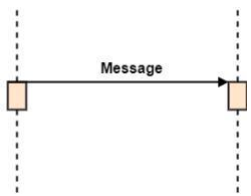
Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

Messages

The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

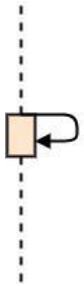
Following are types of messages enlisted below:



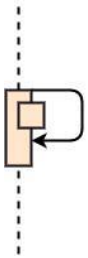
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.

- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.

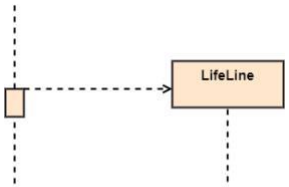
- **Self-Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



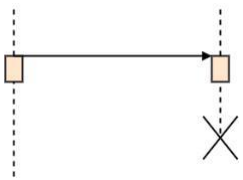
- **Recursive Message:** A self-message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self-message as it represents the recursive calls.



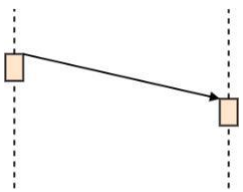
- **Create Message:** It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



- **Destroy Message:** It describes a communication,particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



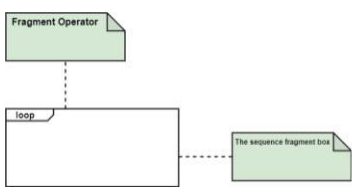
- **Duration Message:** It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.



Note

A note is the capability of attaching several remarks to the element. It basically carries useful information for the modelers.

Sequence Fragments



1. Sequence fragments have been introduced by UML 2.0, which makes it quite easy for the creation and maintenance of an accurate sequence diagram.
2. It is represented by a box called a combined fragment, encloses a part of interaction inside a sequence diagram.
3. The type of fragment is shown by a fragment operator.

Unit: Six Project Work

6.1 Web page development

6.2 Game development

6.3 Mobile application development

6.4 Software Piracy Protection System

6.5 e-Learning Platform