Software Engineering
EG 3213 CT

Total: 6 hour /week
Year: III **Lecture: 3 hours/week**
Semester: VI **Tutorial: hour/week**
Practical: 3 hours/week

Course Description:

This course aims to guide the students in both the theoretical and practical aspects of developing computer solutions for real-world problems. One will study the tools and techniques used in analysis and design of software systems, and apply those tools within a recognized software development methodology and within the context of a case study.

Course Objectives:

After completing this course the students will be able to:

1. Introduce the theory and foundations of software engineering
2. Explain Software Project Management
3. Describe some key aspects of a software engineering process
4. Apply fact-finding and problem-solving skills
5. Determine the requirements for a software system
6. Enlist/Explain key aspects of models and processes for design of a software system
7. Apply current trends in the area of software engineering

Course Contents:

Unit	Topics	Contents	Hours	Methods/ Media	Marks
1	**Introduction to software engineering**	1.1 Introduction to software 1.2 Program Vs software 1.3 Software components 1.4 . Characteristics of software 1.5 Types of software 1.6 Generic view of software engineering 1.7 Software process and software process model.	**[4]**		
2	**Software Development Life Cycles Models:**	2.1 Build and fix model 2.2 The waterfall model 2.3 Prototyping model 2.4 Iterative enhancement model	**[4]**		

Unit	Topics	Contents	Hours	Methods/ Media	Marks
		2.5 Spiral model 2.6 Rapid application development model (RAD) 2.7 Selection criteria of a lifecycle model			
3	**Software Project Management:**	3.1 Activities in project management 3.2 Software project planning 3.3 Software project management plan 3.4 Software project scheduling and techniques 3.5 Software project team management and organization 3.6 Project estimation techniques 3.7 COCOMO model 3.8 Risk analysis and management 3.9 Risk management process 3.10 Software configuration management 3.11Software change management 3.12Version and release management	**[7]**		
4	**Software Requirement Analysis & Specification:**	4.1 Requirement engineering 4.2 Requirement elicitation 4.2.1 Interviews 4.2.2 Brainstorming series 4.2.3 Use case approach 4.3 Requirement analysis 4.3.1. Data flow diagram 4.3.2 Data dictionary 4.3.3 Entity-Relationship diagram 4.3.4 Software prototyping 4.4 Requirement documentation 4.4.1 Nature of SRS 4.4.2 Characteristics of a good SRS 4.4.3 Organization of SRS	**[6]**		

Unit	Topics	Contents	Hours	Methods/ Media	Marks
5	**Software Design:**	5.1 Objectives of design 5.2 Design framework 5.3 Software design models 5.4 Design process 5.5 Architecture design 5.6 Low level design 5.7 Coupling and cohesion 5.8 Software design strategies 5.9 Function oriented design 5.10 Object oriented design 5.11 Function oriented design Vs Object oriented design	**[5]**		
6	**Software Metrics:**	6.1 Software metrics: what & why? 6.2 Token count 6.3 Data structure metrics 6.4 Information flow metrics 6.5 Metrics analysis	**[4]**		
7	**Software Reliability:**	7.1 Basic Concepts 7.2 Software quality 7.3 Software reliability model 7.4 Capability maturity model (CMM)	**[4]**		
8	**Software Testing:**	8.1 Testing process 8.2 Some important terminologies 8.3 Unit testing 8.4 Integration testing 8.5 System testing 8.6 Regression Testing 8.7 Performance testing 8.8 White Box testing and black box testing 8.9 Acceptance testing 8.10 Alpha and Beta testing 8.11 Debugging techniques, tools and approaches	**[6]**		
9	**Software Maintenance:**	9.1 Need for software maintenance 9.2 Types of software maintenance	**[3]**		

Unit	Topics	Contents	Hours	Methods/ Media	Marks
		9.3 Software maintenance process model. 9.4 Software maintenance cost			
10	**Quality assurance**	10.1 Software quality attributes 10.2 Quality factors 10.3 Quality control 10.4 Quality assurance 10.5 Software quality assurance 10.6 Software safety 10.7 The ISO 9000 model 10.8 SEI capability maturity model 10.9 Verification and validation	**(2)**		
	Practical:	The practical should contain all features mentioned above.	**[45]**		

Recommended books:

1. Software engineering, Udit Agarwal, publication
2. Fundamentals of Software Engineering by Ghezzi, Jayazeri and Mandrioli, Prentice-Hall.
3. Fundamentals of Software Engineering by Rajib Mall
4. Software Engineering by Ian Sommerville, Addison-Wesley, ISBN 0-201-17568-1
5. Software Engineering by Roger Jones
6. Modern System analysis and design, Jeffrey A. Hoffer, Joey F. George, Joseph S. Valacich

Introduction of Software

Software, commonly known as programs or apps, consists **of all the instructions that tell the hardware how to perform a task**. ...
System software: Helps run the computer hardware and computer system itself. System software includes operating systems, device drivers, diagnostic tools and more.

Program v/S software

On the basis of	Program	Software
Definition	A computer program is a set of instructions that is used as a process of creating a software program by using programming language.	Software is a set of programs that enables the hardware to perform a specific task.
Types	Programs do not have further categorization.	The software can be of three types: system software, application software, and programming software.
User Interface	A program does not have a user interface.	Every software has a user interface that may be in graphical format or in the form of a command prompt.

Size	Programs are smaller in size, and their size exists between Kilobyte (Kb) to a megabyte (Mb).	Software's are larger in size, and their size exists between megabytes (Mb) to gigabytes (Gb).
Time taken	A program takes less time to be developed.	Whereas software requires more time to be developed.
Features and functionality	A program includes fewer features and limited functionalities.	It has more features and functionalities.
Development approach	The development approach of a program is unorganized, unplanned, and unprocedural.	The development approach of software is well planned, organized, and systematic.
Documentation	There is a lack of documentation in the program.	Softwares are properly documented.
Examples	Examples of the program are - video games, malware, and many more.	Examples of software are - Adobe Photoshop, Adobe Reader, Google Chrome, and many more.

SUBASH SUBEDI

Software components

Your system has three basic types of software: **application programs, device drivers, and operating systems**. Each type of software performs a completely different job, but all three work closely together to perform useful work.

Types of software components

- **Application Programs**
- *Application programs* are the top software layer. You can perform specific tasks with these programs, such as using a word processor for writing, a spreadsheet for accounting, or a computer-aided design program for drawing. The other two layers, device drivers and the operating system, play important support roles. Your system might run one application program at a time, or it might run many simultaneously.
- **Device Drivers**
- *Device drivers* are a set of highly specialized programs. Device drivers help application programs and the operating system do their tasks. Device drivers (in particular, adapters), do not interact with you. They interact directly with computer hardware elements and shield the application programs from the hardware specifics of computers.

Operating System

An *operating system* is a collection of programs that controls the running of programs and organizes the resources of a computer system. These resources are the hardware components of the system, such as keyboards, printers, monitors, and disk drives. Your AIX operating system comes with programs, called *commands* or *utilities*, that maintain your files, send and receive messages, provide miscellaneous information about your system, and so on.
An application program relies on the operating system to perform many detailed tasks associated with the internal workings of the computer. The operating system also accepts commands directly from you to manage files and security. There are many extensions to the AIX operating system that allow you to customize your environment.

Characteristics of software

- **Functionality:**
  It refers to the degree of performance of the software against its intended purpose. Required functions are:

- **Reliability:**
  A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time. Required functions are:

Efficiency:
It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements. Required functions are:

Usability:
It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software. Required functions are:

Maintainability:
It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors. Required functions are:

- **Portability:**
  A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes. Required functions are:

Types of software

- Application Software.
- System Software.
- Firmware.
- Programming Software.
- Driver Software.
- Freeware.
- Shareware.
- Open Source Software.

Application software: -

Application software, is **software that performs specific tasks for an end-user**.

System Software: -

System software is **software designed to provide a platform for other software**. Examples of system software include operating systems (OS) like macOS, Linux, Android.

Firmware: -

In computing, firmware is **a specific class of computer software that provides the low-level control for a device's specific hardware.**

Driver Software: -

A software driver is **a type of software program that controls a hardware device**.

Generic view of software

- **01. Definition Phase:**
- The definition phase focuses on “what”. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. During this, three major tasks will occur in some form: system or information engineering, software project planning and requirements analysis.

- **02. Development Phase:**
  - The development phase focuses on “how”. That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how interfaces are to be characterized, how the design will be translated into a programming language, and how testing will be performed. During this, three specific technical tasks should always occur; software design, code generation, and software testing.

- **03. Support Phase:**

  - The support phase focuses on “change” associated with error correction, adaptations required as the software’s environment evolves, and changes due to enhancements brought about by changing customer requirements. Four types of change are encountered during the support phase:

Software Process

- A software process is **the set of activities and associated outcome that produce a software product**. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. ... Software evolution: The software must evolve to meet changing client needs.

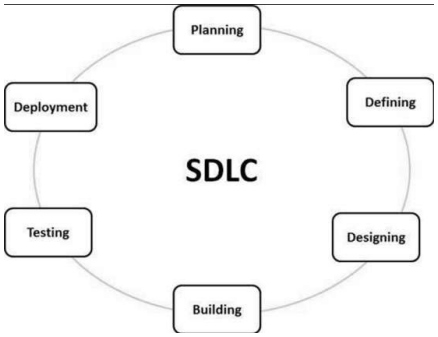
Software process model

- A software process model is **an abstraction of the actual process**, which is being described. It can also be defined as a simplified representation of a software process. Each model represents a process from a specific perspective.

Unit 2: Software Development Life Cycles Models:

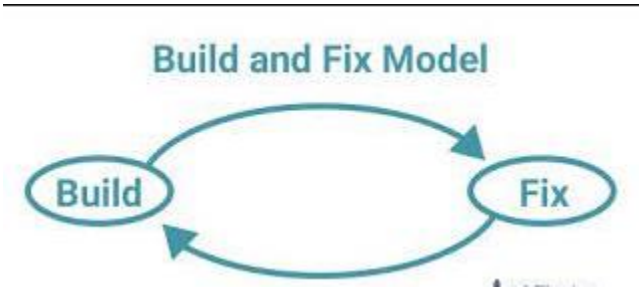
Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

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.



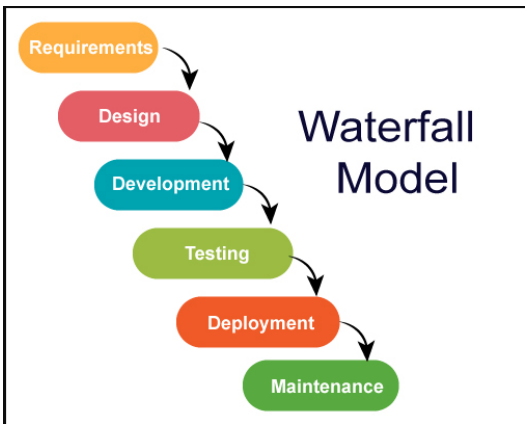
Build & Fix Model

- In this most simple model **of software development**, the product is constructed with minimal requirements, and generally no specifications nor any attempt at design, and testing is most often neglected. This is a representation of what is happening in many software development projects.



The Waterfall Model

- Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.



1.1. Requirement's analysis and specification phase:

The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how. "In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.

1.2. Design Phase:

This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

1.3. Implementation and unit testing:

During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

1.4. Integration and System Testing:

This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs,

and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

1.5. Operation and maintenance phase:

Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

When to use SDLC Waterfall Model?

- When the requirements are constant and not changed regularly.
- A project is short
- The situation is calm
- Where the tools and technology used is consistent and is not changing
- When resources are well prepared and are available to use.

Advantages of Waterfall model

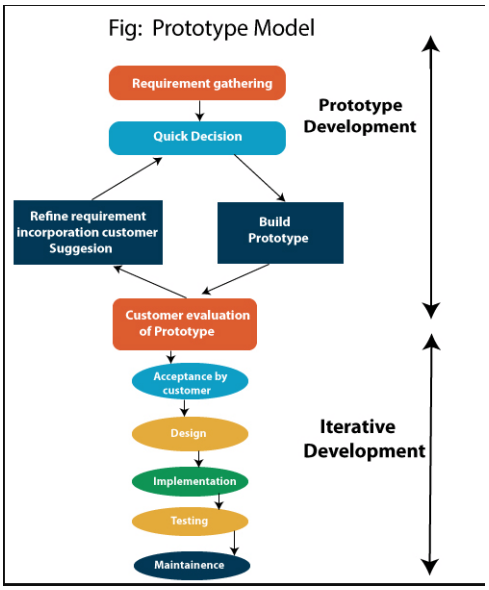
- This model is simple to implement also the number of resources that are required for it is minimal.
- The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
- The start and end points for each phase is fixed, which makes it easy to cover progress.
- The release date for the complete product, as well as its final cost, can be determined before development.
- It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

- In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

2. Prototyping Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.



Steps of Prototype Model

1) Requirement Gathering and Analyst
2) Quick Decision
3) Build a Prototype
4) Assessment or User Evaluation
5) Prototype Refinement
6) Engineer Product

Advantage of Prototype Model

1) Reduce the risk of incorrect user requirement
2) Good where requirement is changing/uncommitted
3) Regular visible process aids management
4) Support early product marketing
5) Reduce Maintenance cost.
6) Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

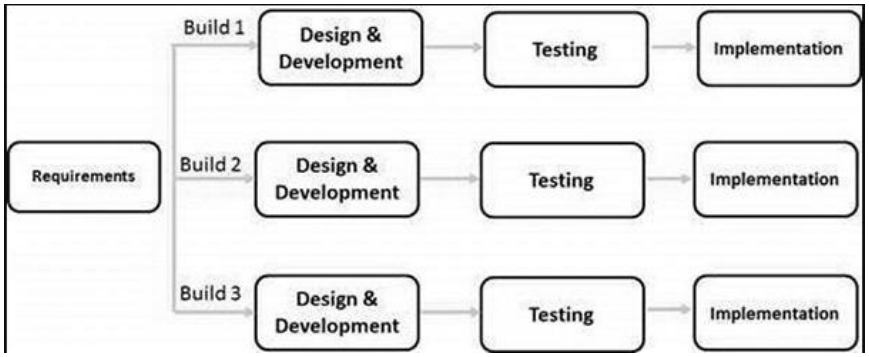
1) An unstable/badly implemented prototype often becomes the final product.
2) Require extensive customer collaboration
* Costs customer money
* Needs committed customer
* Difficult to finish if customer withdraw
* May be too customer specific, no broad market
3) Difficult to know how long the project will last.
4) Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5) Prototyping tools are expensive.

6) Special tools & techniques are required to build a prototype.
7) It is a time-consuming process.

3. Iterative Model:

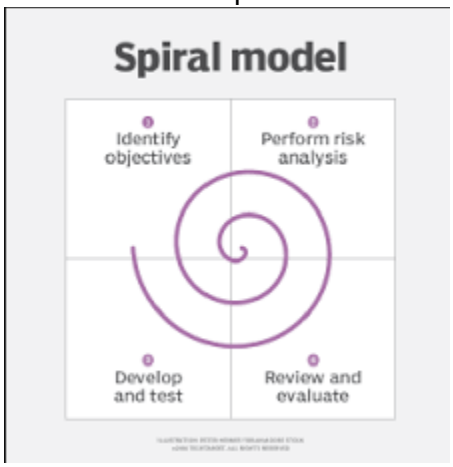
In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.



4. Spiral Model:

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.



4.1. Design:

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

4.2. Construct & Build:

The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

4.3. Evaluation & Risk Analysis:

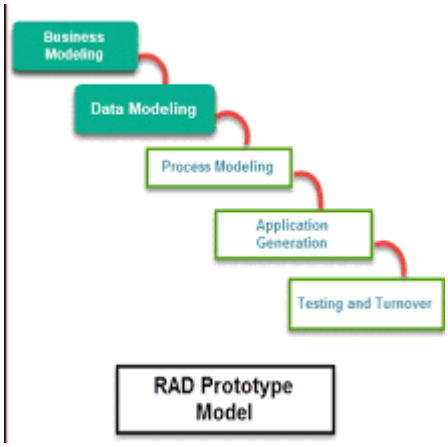
Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.

5. RAD Model:

The **RAD (Rapid Application Development)** model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.



5.1. Business Modelling

The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

5.2. Data Modelling

The information gathered in the Business Modelling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

5.3. Process Modelling

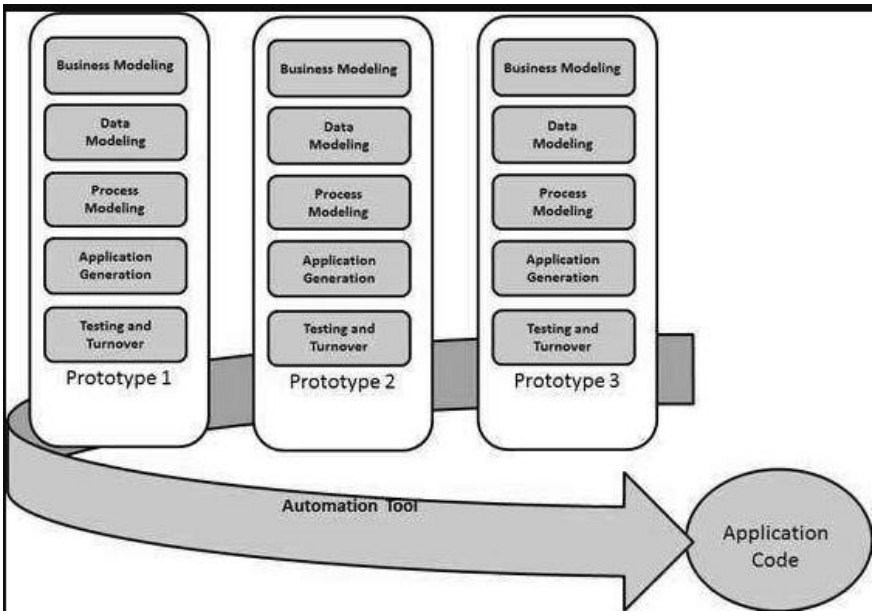
The data object sets defined in the Data Modelling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given

5.4. Application Generation

The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

5.5. Testing & Turnover

The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.



6. Selection Criteria Of a Lifecycle model:

Selection Process parameters plays an important role in software development as it helps to choose the best suitable software life cycle model. Following are the parameters which should be used to select a SDLC.

6.1. Requirements Characteristics:

- Reliability of Requirements
- How often the requirements can change
- Types of requirements
- Number of requirements
- Can the requirements be defined at an early stage
- Requirements indicate the complexity of the system

6.2. Development Team:

- Team size
- Experience of developers on similar type of projects
- Level of understanding of user requirements by the developers
- Environment
- Domain knowledge of developers
- Experience on technologies to be used
- Availability of training

6.3. User Involvement In the Project:

- Expertise of user in project
- Involvement of user in all phases of the project

- Experience of user in similar project in the past

6.4. Project type and associated risk :

- Stability of funds
- Tightness of project schedule
- Availability of resources
- Type of project
- Size of the project
- Expected duration for the completion of project
- Complexity of the project
- Level and the type of associated risk

3 Software Project Management

3.1 Activities in Project Management

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

1. **Project Planning:** It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

2. **Scope Management:** It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

3. **Estimation management:** This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.
   If we talk about the size, then Line of code depends upon user or software requirement.

And if we talk about cost, it includes all the elements such as:

- Size of software
- Quality
- Hardware
- Communication
- Training
- Additional Software and tools
- Skilled manpower

4. **Scheduling Management:** Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.
   For scheduling, it is compulsory

- Find out multiple tasks and correlate them.
- Divide time into units.
- Assign the respective number of work-units for every job.
- Calculate the total time from start to finish.
- Break down the project into modules.

5. **Project Resource Management:** In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

- Create a project team and assign responsibilities to every team member
- Developing a resource plan is derived from the project plan.
- Adjustment of resources.

6. **Project Risk Management:** Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

- The Experienced team leaves the project, and the new team joins it.
- Changes in requirement.
- Change in technologies and the environment.
- Market competition.

7. **Project Communication Management:** Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

8. **Project Configuration Management:** Configuration management is about to control the changes in software like requirements, design, and development of the product.

Some reasons show the need for configuration management:

- Several people work on software that is continually update.
- Help to build coordination among suppliers.

- Changes in requirement, budget, schedule need to accommodate.
- Software should run on multiple systems.

3.2 Software Project Planning

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

Need of Software Project Management

Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities. The most significant is that the underlying technology changes and advances so generally and rapidly that experience of one element may not be connected to the other one.

Software Project Manager

Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & cost estimation scheduling, project monitoring, personnel selection evaluation & risk management. To plan a successful software project, we must understand:

- Scope of work to be completed
- Risk analysis
- The resources mandatory
- The project to be accomplished
- Record of being followed

3.3 Software project Management plan

Once project designing is complete, project managers document their plans during a software package Project Management set up (SPMP) document. The SPMP document ought to discuss an inventory of various things that are mentioned below.

1 Introduction:

Objectives

- Major Functions
- Performance Issues
- Management and Technical Constraints

2 Project Estimates:

- Historical Data Used
- Estimation Techniques Used
- Effort, Resource, Cost, and Project Duration Estimates

3 Schedule:

- Work Breakdown Structure
- Task Network Representation
- Gantt Chart Representation
- PERT Chart Representation

4 Project Resources:

- People
- Hardware and Software
- Special Resources

5 Staff Organization:

- Team Structure
- Management Reporting

6 Risk Management Plan:

- Risk Analysis
- Risk Identification
- Risk Estimation
- Risk Abatement Procedures

7 Project Tracking and Control Plan

8 Miscellaneous Plans:

- Process Tailoring
- Quality Assurance Plan
- Configuration Management Plan
- Validation and Verification
- System Testing Plan
- Delivery, Installation, and Maintenance Plan

3.4 Software project Scheduling and Techniques

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following.

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

Project scheduling techniques

Below are four popular scheduling techniques used by project managers: Critical Path Method, Program Evaluation and Review Technique, Fast-tracking and crashing, and Gantt charts.

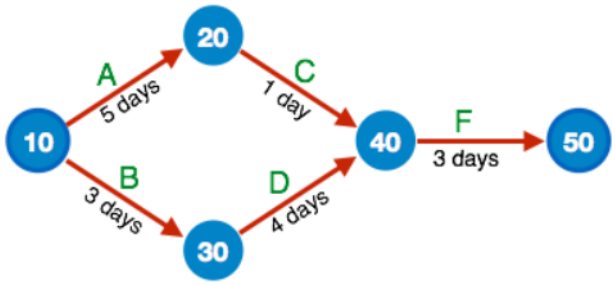
1. Critical Path Method (CPM)

The CPM method is a commonly used construction scheduling method that helps managers predict the project schedule based on its tasks. To do this, you need to:

- List all of the assignments required to complete your project in a Work Breakdown Structure
- Estimate the duration of each task
- Identify your task dependencies and deliverables for your project

2. Program Evaluation and Review Technique (PERT)

PERT is another construction scheduling method that project managers use to estimate the duration of a project. PERT charts offer a visual representation of the major activities (and dependencies) in a project. The activities are displayed sequentially (like a project roadmap) and the tasks are connected via an activity line on the chart. This technique requires creating a visual timeline known as a PERT chart, which you can do by using a Work Breakdown Structure and then mapping out your project and dependencies in a Gantt chart.



3. Fast-tracking and crashing

Fast-tracking and crashing are known as duration compression tactics. They're used to shorten the schedule of your project, but should be used cautiously as they can lead to bigger problems.

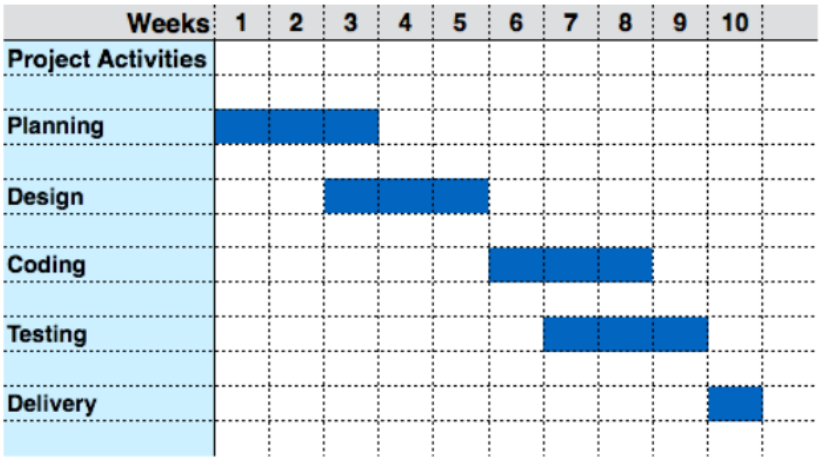
Fast-tracking is essentially multitasking. It requires you to find your project's critical path, and work on those activities while simultaneously working on your float tasks.

Crashing is dumping extra resources into a project to finish it quicker. This is usually done when a project is in danger of running past a deadline. Crashing could be adding extra members to a team or having your members work overtime until they meet their goals.

4. Gantt charts

Gantt charts offer a graphical representation of your project timeline from start to finish. Using a Gantt chart can increase workflow transparency by allowing you to observe who's working on what so you know where everyone is at in relation to the project schedule.

That way, you can measure progress and planning timelines, identify bottlenecks, and manage resources easier and more effectively.



3.5 Software project team management and organization

There are many ways to organize the project team. Some important ways are as follows:

1. Hierarchical team organization
2. Chief-programmer team organization
3. Matrix team, organization
4. Egoless team organization
5. Democratic team organization

Hierarchical team organization :

In this, the people of organization at different levels following a tree structure. People at bottom level generally possess most detailed knowledge about the system. People at higher levels have broader appreciation of the whole project.

Chief-programmer team organization :

This team organization is composed of a small team consisting the following team members

- The Chief programmer : It is the person who is actively involved in the planning, specification and design process and ideally in the implementation process as well.
- The project assistant : It is the closest technical co-worker of the chief programmer.
- The project secretary : It relieves the chief programmer and all other programmers of administration tools.
- Specialists : These people select the implementation language, implement individual system components and employ software tools and carry out tasks.

Matrix Team Organization :

In matrix team organization, people are divided into specialist groups. Each group has a manager. Example of Metric team organization is as follows :

Egoless Team Organization :

Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. In this team organization goals are set and decisions are made by group consensus. Here group, 'leadership' rotates based on tasks to be performed and differing abilities of members.

Democratic Team Organization :

It is quite similar to the egoless team organization, but one member is the team leader with some responsibilities :

- Coordination
- Final decisions, when consensus cannot be reached.

3.6 Project Estimation techniques

An estimate is a rough calculation of something. For example, a project cost estimate is a general idea of the price model for a project.

Estimation techniques are ways to create project estimates. When your client or another project stakeholder asks you to estimate an aspect of the project, these techniques help you come up with a realistic number to give them.
Here are six common estimating methods in project management:

1. Top-down estimate

A top-down estimating technique assigns an overall time for the project and then breaks it down into discrete phases, work, and tasks — usually based on your project's work breakdown structure (WBS).

If a client tells you the project has to be done within six months, a top-down approach allows you to take that overall timeline and estimate how much time you can take for each activity within the project and still complete it on time.

2. Bottom-up estimate

A bottom-up estimate is the reverse of top-down. Using this estimation technique, you start by estimating each individual task or aspect of the project. Then you combine all those separate estimates to build up the overall project estimate.

Since each activity is being assessed individually, this type of estimate tends to be more accurate than the top-down approach. But it also takes more time.

3. Expert judgment

Expert judgment is one of the most popular estimation techniques, as it tends to be quick and easy. This technique involves relying on the experience and gut feel of experts to estimate projects.

It's most useful when you're planning a standard project that is similar to ones your team has completed before. Expert judgment can be used for creating top-down or bottom-up estimates.

4. Comparative or analogous estimation

Comparative estimation uses past project data combined with a top-down approach to estimate project duration. If the average completion time of similar projects was eight months, you'd assume the current one will take eight months. Then you can break those eight months down across tasks and activities to get your lower-level work estimates.

5. Parametric model estimating

Parametric modeling also uses past project data, but it attempts to adjust the data to reflect each project's differences. This technique takes the detail of past projects and pro-rates it to estimate the current project.
Imagine your company builds houses. Parametric modeling could take the cost of all past construction projects divided by each project's square footage to come up with an average

project cost per square foot of the home. Then, you'd multiply that number by the planned square footage of the current home to create your overall project budget.

6. Three-point estimating
Three-point estimating is a technique sometimes used for creating bottom-up estimates. Rather than assuming one duration for a task, you may assign three: optimistic, pessimistic, and most likely. These three numbers are averaged to create your actual estimate.
The PERT (Program Evaluation and Review Technique) method uses three-point estimating, but it takes a weighted average of the three points.

3.7 COCOMO Model

Before the start of any software project, various factors are taken into consideration. These factors decide how the project development phase will proceed, how the product will be deployed, and what are the risks involved.

One of these factors includes the time and cost factor. It is important to know how much time will it take before the product is ready for deployment. Cost does also plays a very important factor during the development of the product.

Keeping this in mind in 1981, Boehm proposed a model – COCOMO model (Constructive Cost Estimation model) to estimate the cost and effort required to complete any product based on its size. This model was not very accurate but still gives a good rough idea and is still in use.

COCOMO or Constructive Cost Estimation Model is a model that estimates the effort and time taken to complete the model based on the size of the source code.
The Cocomo model divides software projects into 3 types-

- Organic
- Semi-detached
- Embedded

ORGANIC – A software development project comes under organic type if the development team is small, the problem is not complex, and has been solved before. Also, a nominal experience of team members is considered. Some of the projects that can be organic are- small data processing systems, basic inventory management systems, business systems
SEMIDETACHED – A software project that is in-between the organic and embedded system is a semi-detached system. Its characteristics include – a middle-sized development team, with a mix of both- experienced and inexperienced members, and the project complexity is neither too high nor too low.

The problems faced in this project are a few known and a few unknown, which have never been faced before. Few such projects are- Database Management System(DBMS), new unknown operating system, difficult inventory management system.
EMBEDDED – A project requiring a large team with experienced members, and having the highest level of complexity is an embedded system. The members need to be creative enough

to be able to develop complex models. Such projects include- air traffic models, ATMs, complex banking systems, etc.

Types of COCOMO Model

The different types of COCOMO models define the depth of cost estimation is required for the project. It depends on the software manager, what type of model do they choose.

According to Boehm, the estimation should be divided into 3 stages-

- Basic Model
- Intermediate Model
- Detailed Model

Boehm had set different expressions to calculate effort and development time through KLOC (Kilo line of code). All these calculations are roughly correct and not exact because of the absence of sufficient factors.

Basic Model – This model is based on rough calculations thus, there is very limited accuracy. The whole model is based on only lines of source code to estimate the calculation and other factors are neglected.

Intermediate Model – The intermediate model dives deeper and includes more factors such as cost drivers into the calculation. This enhances the accuracy of estimation. The cost driver includes attributes like reliability, capability, and experience.

These cost drivers can be classified under-

Product attributes

- size of the application's database
- the complexity of the product
- software's reliability extent

Hardware attributes

- Total turnabout time
- Memory constraints
- Run-time performance constraints

Project attributes

- software tools used
- required development schedule

Personnel attributes

- Applications experience
- Coding experience

- Software developer's capability
- Analyst experience

Detailed Model – The detailed model or the complete model includes all the factors of the both-the basic model and the intermediate model. In the detailed model for each cost driver property, various effort multipliers are used.

The whole software is divided into smaller phases and then the COCOMO calculations or model is applied to them. The effort is then estimated and then the sum of the efforts is calculated.

The software is divided into 6 different phases namely-

- Planning and requirements
- System structure
- Integration and test
- Cost Constructive model
- Complete structure
- Module code and test

3.8 Risk Analysis and Management

Risk Analysis in project management is a sequence of processes to identify the factors that may affect a project's success. These processes include risk identification, analysis of risks, risk management and control, etc. Proper risk analysis helps to control possible future events that may harm the overall project.
A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

- Project risks
- Technical risks
- Business risks

1. **Project risks:** Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

2. **Technical risks:** Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty,

and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

3. **Business risks:** This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

Other risk categories

1. Known risks: Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

3.9 Risk Management Process

Risk Management is an important part of project planning activities. It involves identifying and estimating the probability of risks with their order of impact on the project.

Risk Management Process:
There are some steps that need to be followed in order to reduce risk. These steps are as follows:

1. Risk Identification:
Risk identification involves brainstorming activities. it also involves the preparation of a risk list. Brainstorming is a group discussion technique where all the stakeholders meet together. this technique produces new ideas and promotes creative thinking.
Preparation of risk list involves identification of risks that are occurring continuously in previous software projects.

2. Risk Analysis and Prioritization:
It is a process that consists of the following steps:

- Identifying the problems causing risk in projects
- Identifying the probability of occurrence of problem
- Identifying the impact of problem
- Assigning values to step 2 and step 3 in the range of 1 to 10
- Calculate the risk exposure factor which is the product of values of step 2 and step 3
- Prepare a table consisting of all the values and order risk on the basis of risk exposure factor

Risk No	Problem	Probability of occurrence of problem	Impact of problem	Risk exposure	Priority
R1	Issue of incorrect password	2	2	4	10
R2	Testing reveals a lot of defects	1	9	9	7
R3	Design is not robust	2	7	14	5

3. Risk Avoidance and Mitigation:
The purpose of this technique is to altogether eliminate the occurrence of risks. so the method to avoid risks is to reduce the scope of projects by removing non-essential requirements.

4. Risk Monitoring:
In this technique, the risk is monitored continuously by reevaluating the risks, the impact of risk, and the probability of occurrence of the risk.
This ensures that:

- Risk has been reduced
- New risks are discovered
- Impact and magnitude of risk are measured

3.10 Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.
Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.
The elements that comprise all information produced as a part of the software process are collectively called a software configuration.
As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.
These are handled and controlled by SCM. This is where we require software configuration management.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.
Therefore, SCM is the discipline which

- Identify change
- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.
The objective is to maximize productivity by minimizing mistakes (errors).
CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

3.11 Software Change Management

Change Management in software development refers to the transition from an existing state of the software product to another improved state of the product. It controls, supports, and manages changes to artifacts, such as code changes, process changes, or documentation changes. Where CCP (Change Control Process) mainly identifies, documents, and authorizes changes to a software application.

Each software development process follows Software Development Life Cycle (SDLC) where each phase is accordingly followed to finally deliver a good quality software product. Change Management does not come under any phases of SDLC still it has great importance in the entire software development process. There are various types of change management tools that are used for various purposes like to adopt, control, represent and effect the change required. For example Change management tools for Flow Charting, Project Planning, Data collection, etc.

Process of Change Management:
When any software application/product goes for any changes in an IT environment, it undergoes a series of sequential processes as follows:

- Creating a request for change
- Reviewing and assessing a request for change
- Planning the change
- Testing the change
- Creating a change proposal
- Implementing changes
- Reviewing change performance
- Closing the process

Importance of Change Management:

- For improving performance
- For increasing engagement
- For enhancing innovation

- For including new technologies
- For implementing new requirements
- For reducing cost

3.12 Version and release Management

The process involved in version and release management are concerned with identifying and keeping track of the versions of a system. Versions managers devise procedures to ensure that versions of a system may be retrieved when required and are not accidentally changed by the development team. For products, version managers work with marketing staff and for custom systems with customers, to plan when new releases of a system should be created not distributed for deployment.

A system instance is an instance of a system which can be different from other instances in some way. There is a chance in which versions of the system may have different functionality, enhanced performance or repaired software faults. Some versions may be functionally equivalent but designed for different hardware or software configuration. Versions with only small differences are sometimes called variants.

A system release may be a version that's distributed to customers. Each system release should either include new functionality or should be intended for a special hardware platform. There are normally many more versions of a system than release. Versions are created with an organization for internal development or testing and are not intended for release to customers.

Software Requirement Analysis & Specification

4.1 Requirements engineering (RE)

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

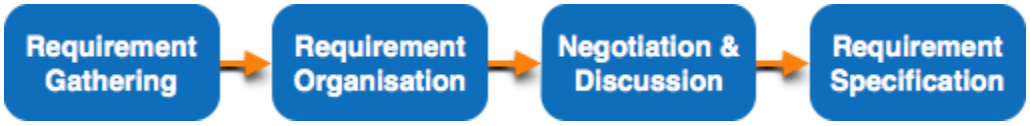
Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management

4.2 Requirement Elicitation

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirements gathering -** The developers discuss with the client and end users and know their expectations from the software.

- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency and convenience.

- **Negotiation & discussion -** If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.
  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

4.2 Requirement Elicitation Techniques

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Questionnaires
A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.
A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis
Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis
Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming
An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Use case approach
A use case is a software and system engineering term that **describes how a user uses a system to accomplish a particular goal**. A use case acts as a software modeling technique that defines the features to be implemented and the resolution of any errors that may be encountered.
The steps in designing use cases are:

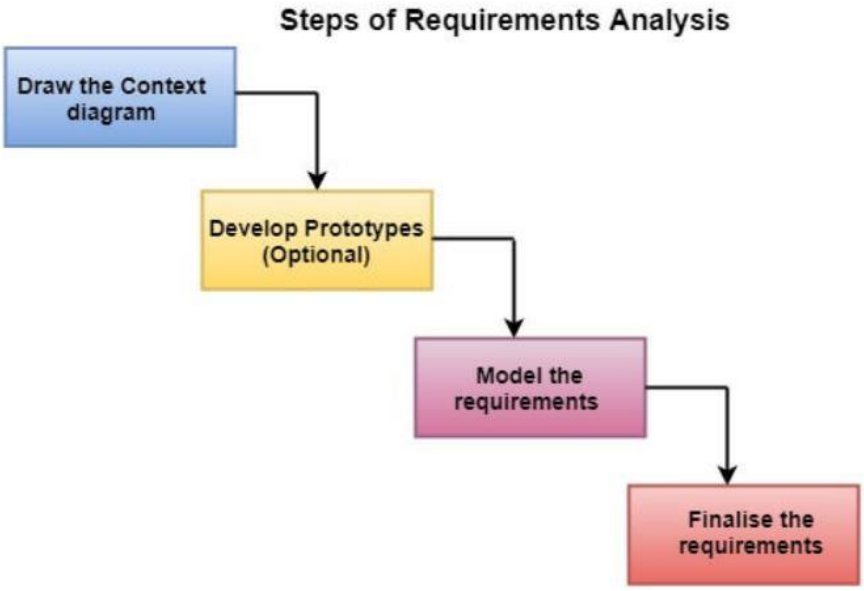
- Identify the users of the system
- For each category of users, create a user profile. This includes all roles played by the users relevant to the system.
- Identify significant goals associated with each role to support the system. The system's value proposition identifies the significant role.
- Structure the use cases
- Review and validate the users

4.3 Requirement 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 fig:

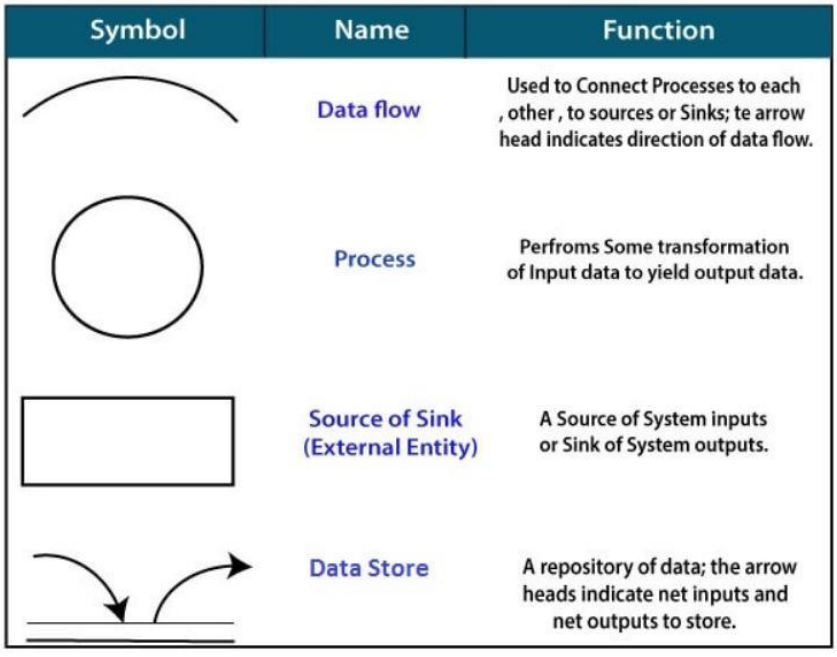


4.3.1 Data Flow Diagram

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole.



Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)
The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD:
It is also known as a context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as a single bubble with input and output data indicated by incoming/outgoing arrows.

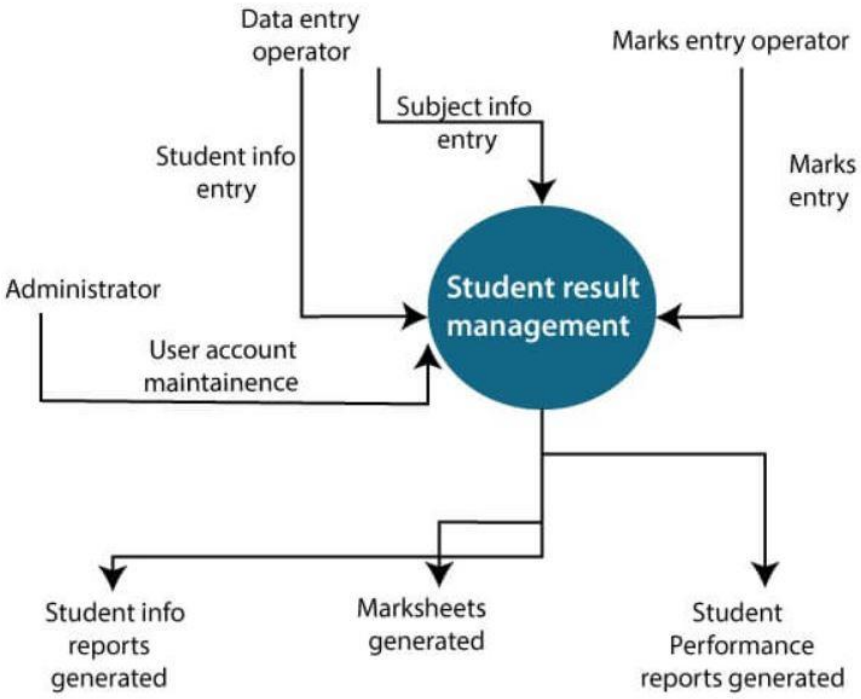


Fig: Level-0 DFD of result management system

1-level DFD:
In 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

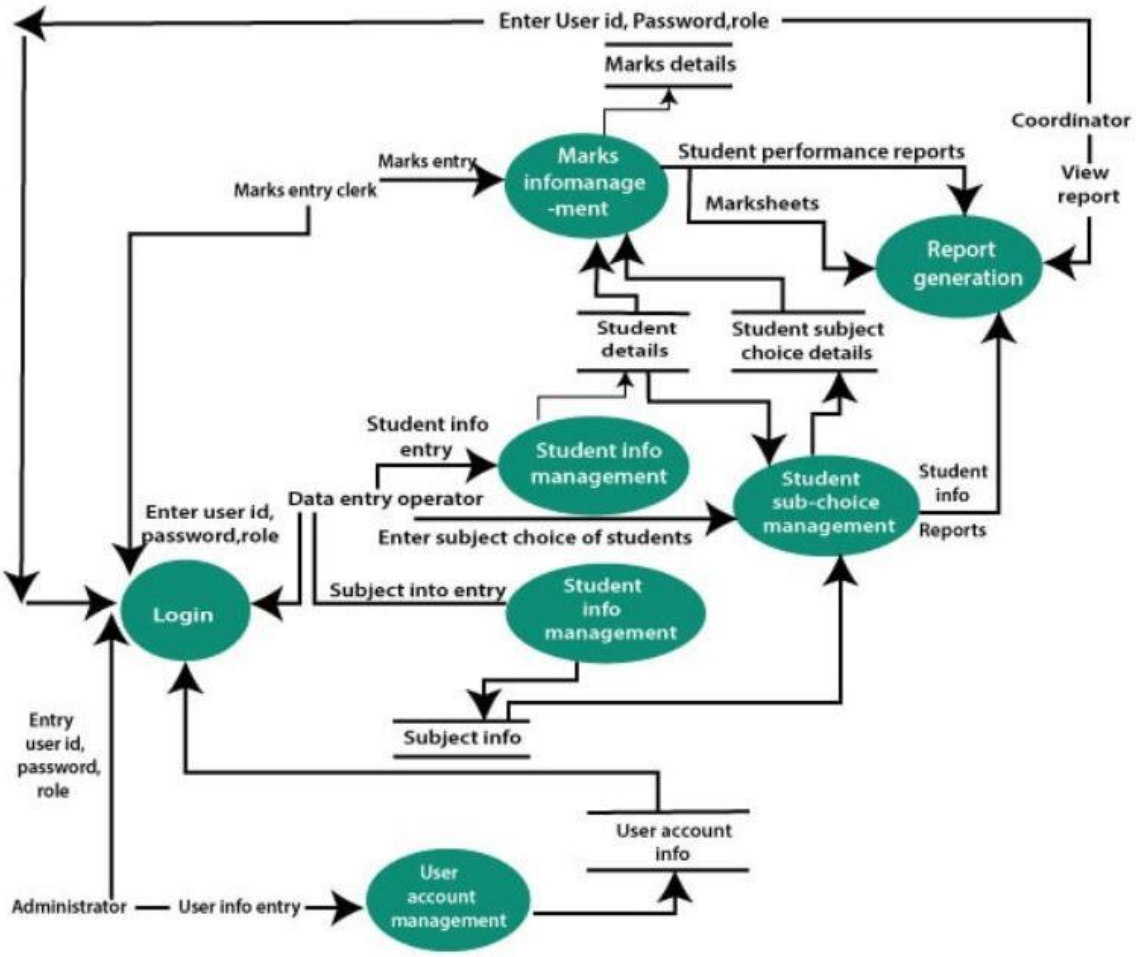
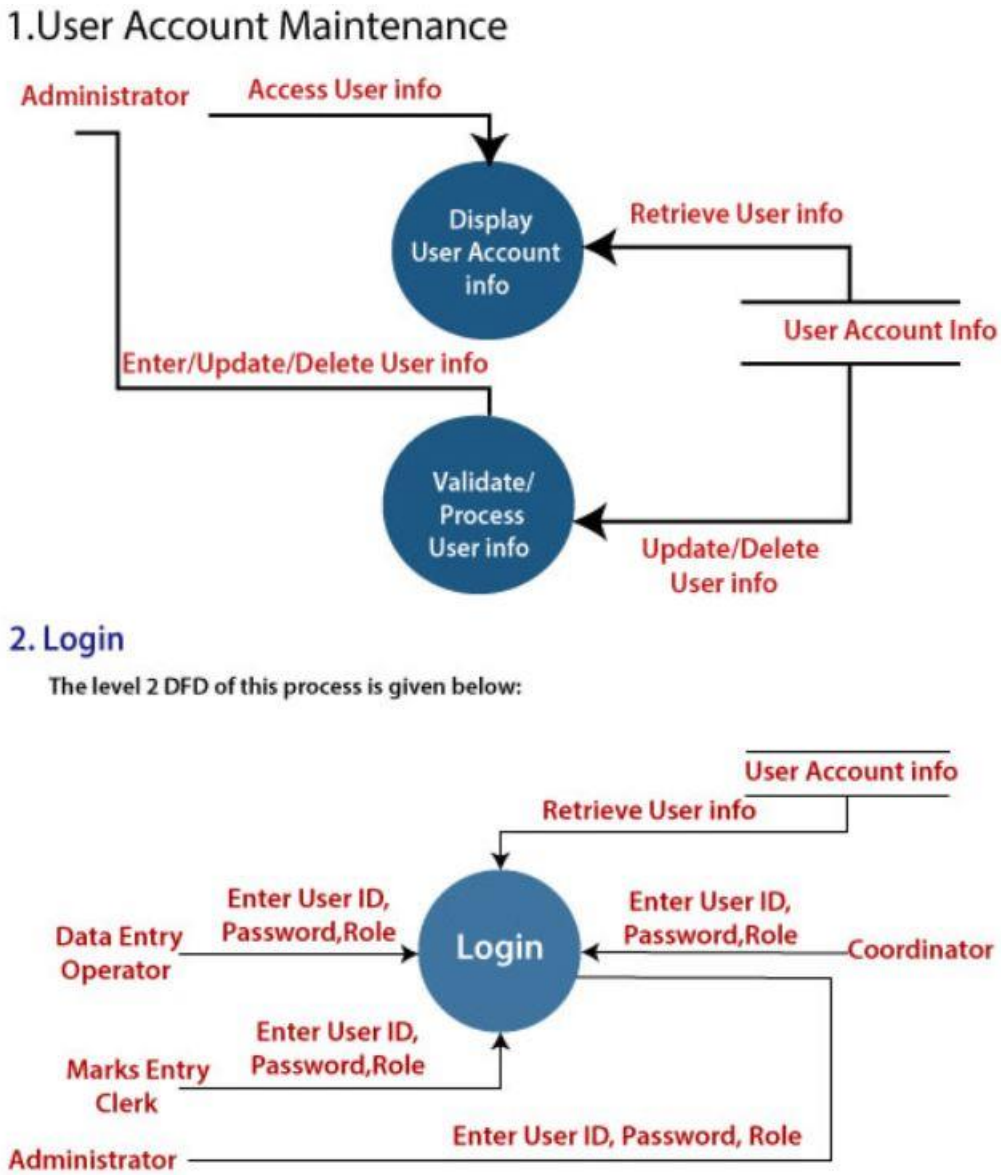


Fig: Level-1 DFD of result management system

2-level DFD:
2-level DFD goes one step deeper into parts of 1-level DFD. It can be used to plan or record the specific/necessary detail about the system's functioning.



4.3.2 Data Dictionary

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is an essential component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

The data dictionary, in general, includes information about the following:

- Name of the data item
- Aliases
- Description/purpose
- Related data items
- Range of values

- Data structure definition/Forms

The **name of the data item** is self-explanatory.
Aliases include other names by which this data item is called DEO for Data Entry Operator and DR for Deputy Registrar.

Description/purpose is a textual description of what the data item is used for or why it exists.

Related data items capture relationships between data items e.g., total_marks must always equal to internal_marks plus external_marks.

Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.

Data structure Forms: Data flows capture the name of processes that generate or receive the data items. If the data item is primitive, then data structure form captures the physical structures of the data item. If the data is itself a data aggregate, then data structure form capture the composition of the data items in terms of other data items.

The mathematical operators used within the data dictionary are defined in the table:

Notations	Meaning
x=a+b	x includes of data elements a and b.
x=[a/b]	x includes of either data elements a or b.
x=a x	includes of optimal data elements a.
x=y[a]	x includes of y or more occurrences of data element a
x=[a]z	x includes of z or fewer occurrences of data element a
x=y[a]z	x includes of some occurrences of data element a which are between y and z.

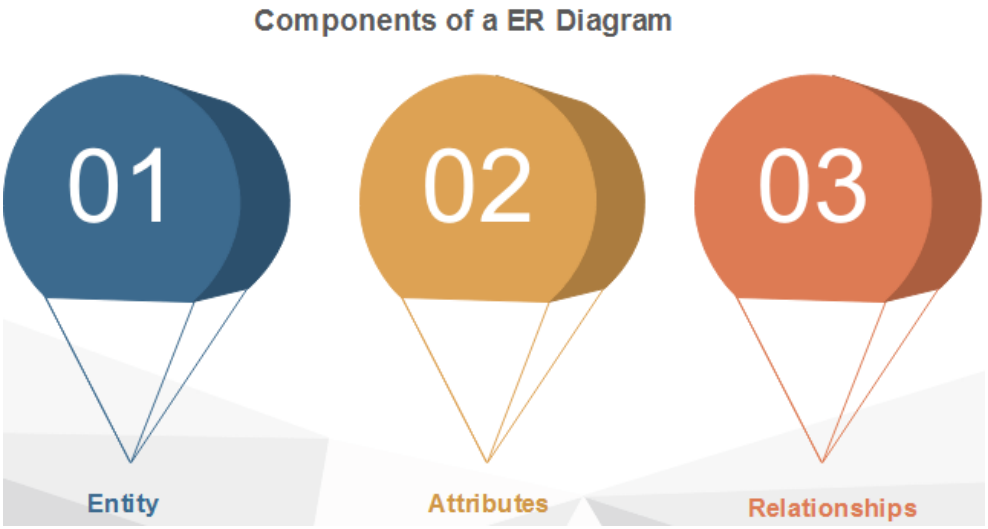
4.3.3 Entity-Relation Diagram

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD

- The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
- The ERD serves as a documentation tool.
- Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of an ER Diagrams



1. Entity

An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

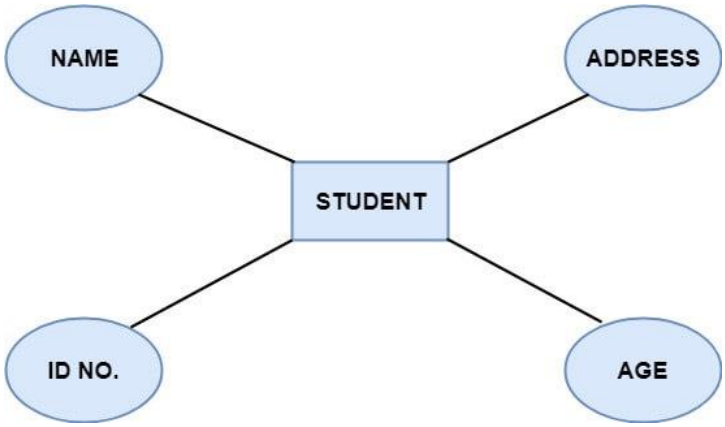
Entity Set

An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.

2. Attributes

Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

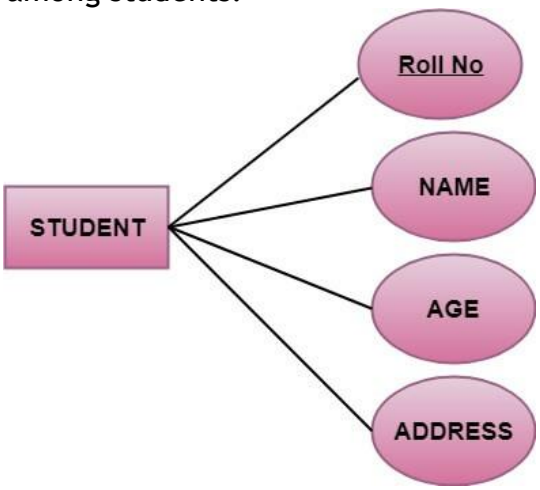
There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

1. Key attribute: Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll_number of a student makes him identifiable among students.



There are mainly three types of keys:

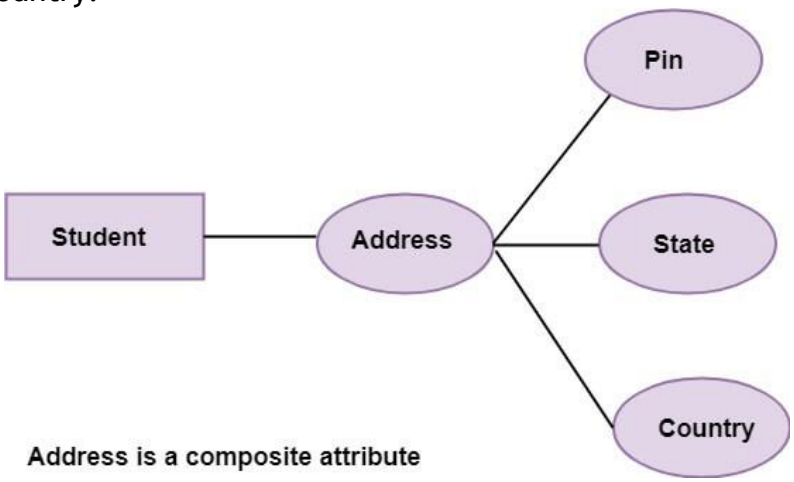
1. **Super key:** A set of attributes that collectively identifies an entity in the entity set.

2. **Candidate key:** A minimal super key is known as a candidate key. An entity set may have more than one candidate key.

3. **Primary key:** A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

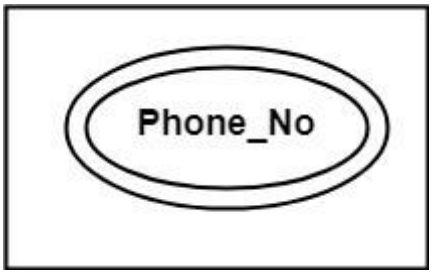
2. Composite attribute: An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is

a composite attribute as an address is composed of other characteristics such as pin code, state, country.

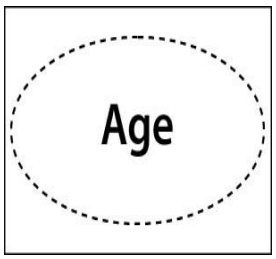


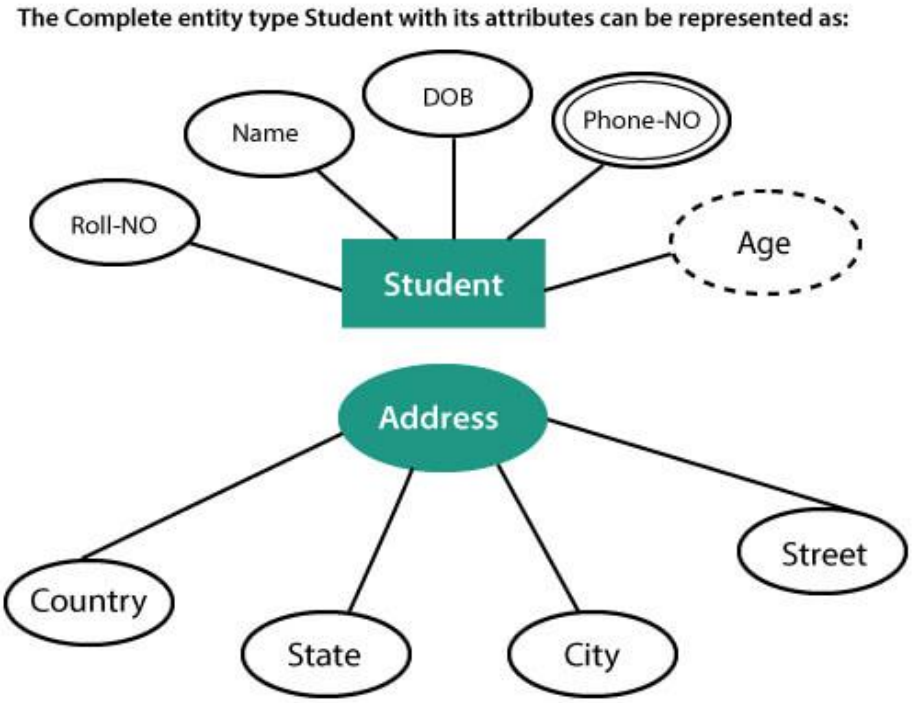
3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



5. Derived attribute: Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date_of_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.





6. Relationships

The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works_at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

Fig: Relationships in ERD

Relationship set

A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of a relationship set

The number of participating entities in a relationship describes the degree of the relationship. The three most common relationships in E-R models are:

1. Unary (degree1)
2. Binary (degree2)
3. Ternary (degree3)

1. Unary relationship: This is also called recursive relationships. It is a relationship between the instances of one entity type. For example, one person is married to only one person.

Fig: Unary Relationship

2. Binary relationship: It is a relationship between the instances of two entity types. For example, the Teacher teaches the subject.

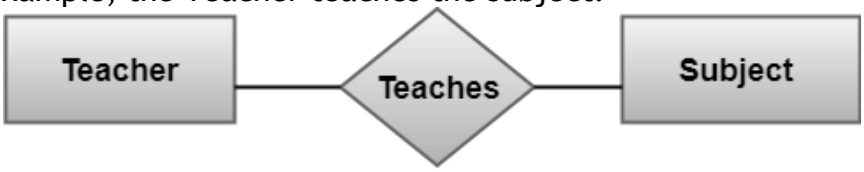


Fig: Binary Relationship

3. Ternary relationship: It is a relationship amongst instances of three entity types. In fig, the relationships "**may have**" provide the association of three entities, i.e., TEACHER, STUDENT, and SUBJECT. All three entities are many-to-many participants. There may be one or many participants in a ternary relationship.
In general, "**n**" entities can be related by the same relationship and is known as **n-ary** relationship.

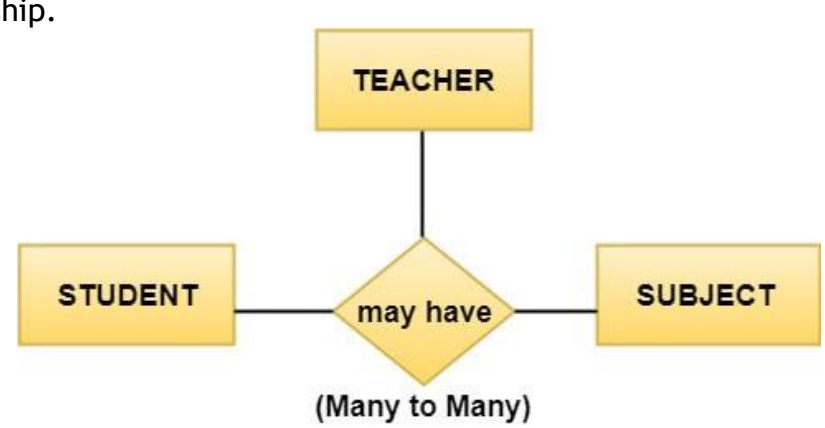
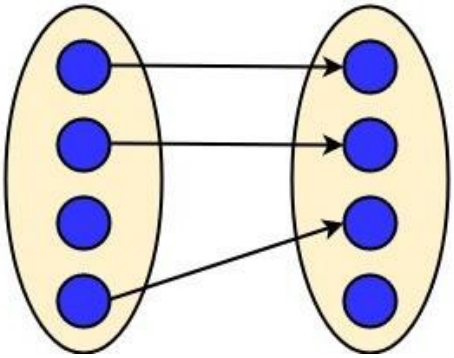


Fig: Ternary Relationship

Cardinality
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

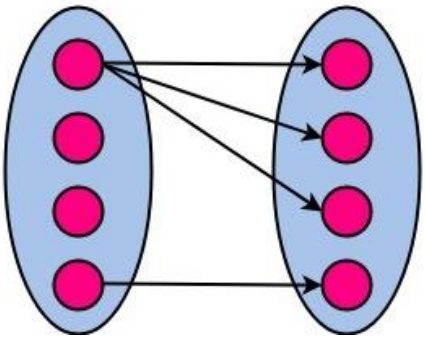
Types of Cardinalities
1. One to One: One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.

Using Sets, it can be represented as:



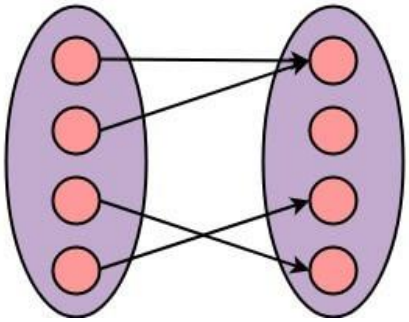
2. One to many: When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.

Using Sets, it can be represented as:



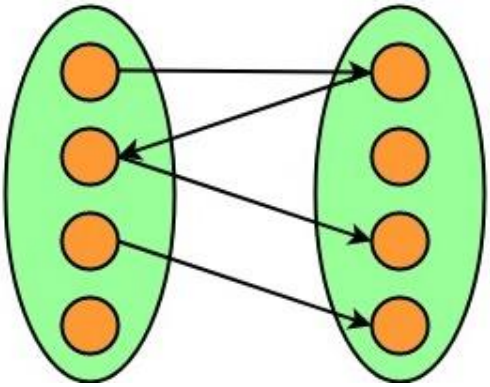
3. Many to One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. For example - many students can study in a single college, but a student cannot study in many colleges at the same time.

Using Sets, it can be represented as:



4. Many to Many: One entity from A can be associated with more than one entity from B and vice-versa. For example, the student can be assigned to many projects, and a project can be assigned to many students.

Using Sets, it can be represented as:



4.3.4 Software Prototyping

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

Following is a stepwise approach explained to design a software prototype.

Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and Enhance the Prototype

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like – time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

4.4 Requirement Documentation

4.4.1 Nature of SRS

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user

requirements for a system as well as detailed specifications of the system requirements.

4.4.2 Characteristics of a good SRS

1. **Correctness:**
   User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

2. **Completeness:**
   Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

3. **Consistency:**
   Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements. Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

4. **Unambiguousness:**
   A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation. Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

5. **Ranking for importance and stability:**
   There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential. An identifier mark can be used with every requirement to indicate its rank or stability.

6. **Modifiability:**
   SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent. Modifications should be properly indexed and cross-referenced.

7. **Verifiability:**
   A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met by the system. For example, a requirement starting that the system must be user-friendly is not verifiable and listing such requirements should be avoided.

8. **Traceability:**
   One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

9. **Design Independence:**
   There should be an option to choose from multiple design alternatives for the final system. More specifically, the SRS should not include any implementation details.

10. **Testability:**
    A SRS should be written in such a way that it is easy to generate test cases and test plans from the document.

11. **Understandable by the customer:**
    An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

12. **Right level of abstraction:**
    If the SRS is written for the requirements phase, the details should be explained explicitly. Whereas, for a feasibility study, fewer details can be used. Hence, the level of abstraction varies according to the purpose of the SRS.

4.4.3 Organization Of SRS

In order to form a good SRS, here you will see some points which can be used and should be considered to form a structure of good SRS. These are as follows :

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

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.

5 Software Design

5.1 Objective of 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.

The software design phase is the first step in **SDLC (Software Design Life Cycle)**, which moves the concentration from the problem domain to the solution domain. In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.

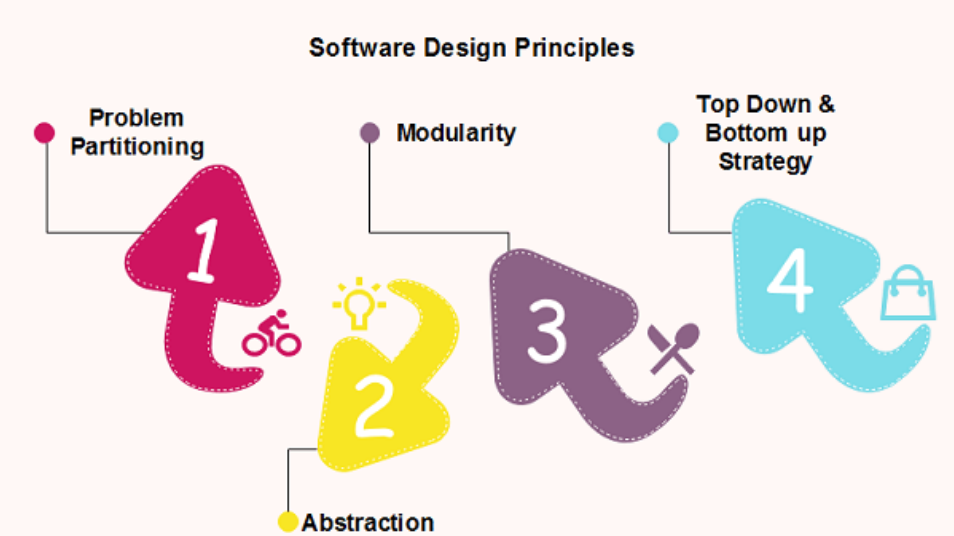
Following are the purposes of Software design:

Objectives 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.

5.2 Design framework

Software design principles are concerned with providing means to handle the complexity of the design process effectively.



Problem Partitioning

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

i. A module is specified by the method it performs.
ii. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for Function oriented design approaches.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for Object Oriented design approaches.

Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software.

Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

Advantages of Modularity

There are several advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

There are several disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

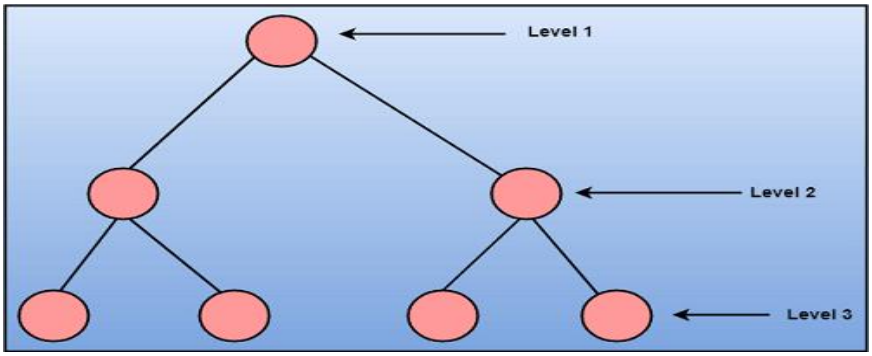
Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs.

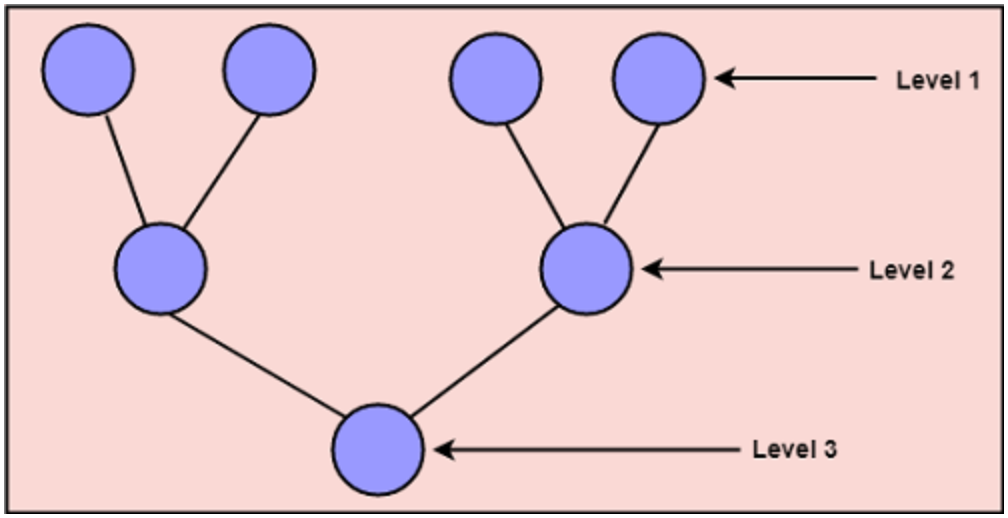
To design a system, there are two possible approaches:

1. **Top-down Approach**
2. **Bottom-up Approach**

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



3. **Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



5.3 Software Design Models

Designing a model is an important phase and is a multi-process that represent the data structure, program structure, interface characteristic, and procedural details. It is mainly classified into four categories – Data design, architectural design, interface design, and component-level design.

1. Data design elements

- The data design element produced a model of data that represent a high level of abstraction.
- This model is then more refined into more implementation specific representation which is processed by the computer based system.
- The structure of data is the most important part of the software design.

2. Architectural design elements

- The architecture design elements provides us overall view of the system.
- The architectural design element is generally represented as a set of interconnected subsystem that are derived from analysis packages in the requirement model.

The architecture model is derived from following sources:

- The information about the application domain to built the software.
- Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
- The architectural style and pattern as per availability.

3. Interface design elements

- The interface design elements for software represents the information flow within it and out of the system.
- They communicate between the components defined as part of architecture.

Following are the important elements of the interface design:

1. The user interface
2. The external interface to the other systems, networks etc.
3. The internal interface between various components.

4. Component level diagram elements

- The component level design for software is similar to the set of detailed specification of each room in a house.
- The component level design for the software completely describes the internal details of the each software component.
- The processing of data structure occurs in a component and an interface which allows all the component operations.

5. Deployment level design elements

- The deployment level design element shows the software functionality and subsystem that allocated in the physical computing environment which support the software.
- Following figure shows three computing environment as shown. These are the personal computer, the CPI server and the Control panel.

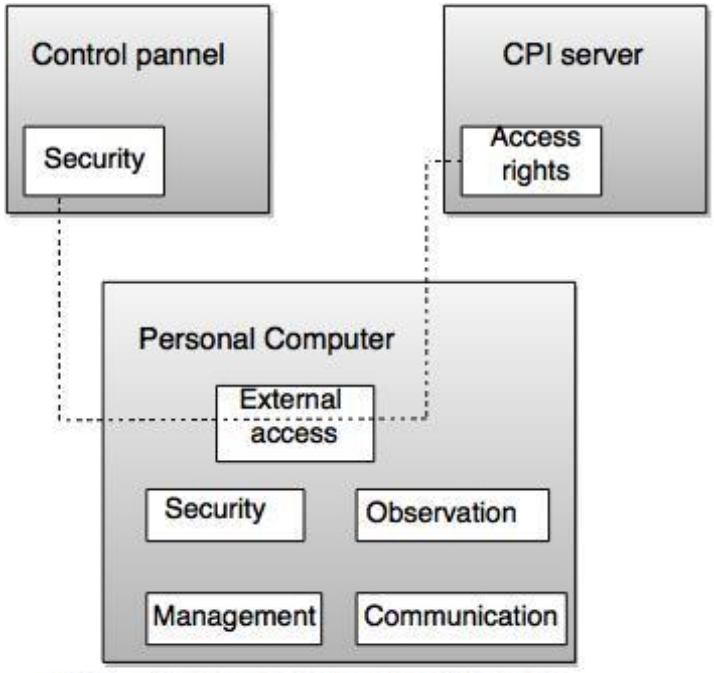


Fig. - Deployment level diagram

5.4 Design Process

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.
Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.
- **High-level Design-** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules

5.5 Architecture Design

The software needs the architectural design to represents the design of software. IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system." The software that is built for computer-based systems can exhibit one of these many architectural styles.
Each style will describe a system category that consists of:

- A set of components (eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

The use of architectural styles is to establish a structure for all the components of the system.

1 **Data centered architectures:**

- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
- The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
- Data can be passed among clients using blackboard mechanism.

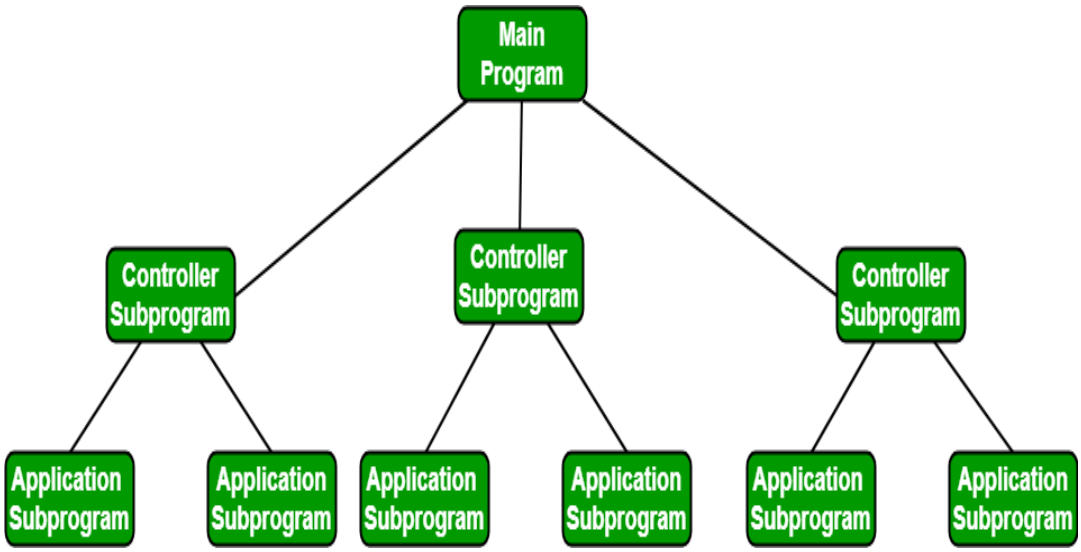
2 Data flow architectures:

- This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.
- Pipes are used to transmit data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.

3 Call and Return architectures:

It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



4 Object Oriented architecture:

The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

5 Layered architecture:

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
- Intermediate layers to utility services and application software functions.

5.6 Low Level Design

1. High Level Design :

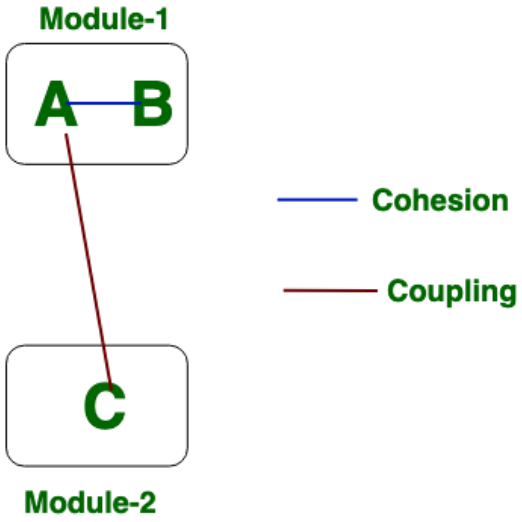
High Level Design in short HLD is the general system design means it refers to the overall system design. It describes the overall description/architecture of the application. It includes the description of system architecture, data base design, brief description on systems, services, platforms and relationship among modules. It is also

known as macro level/system design. It is created by solution architect. It converts the Business/client requirement into High Level Solution. It is created first means before Low Level Design.

2. Low Level Design :

Low Level Design in short LLD is like detailing HLD means it refers to component-level design process. It describes detailed description of each and every module means it includes actual logic for every system component and it goes deep into each modules specification. It is also known as micro level/detailed design. It is created by designers and developers. It converts the High Level Solution into Detailed solution. It is created second means after High Level Design.

5.7 Coupling and cohension



Cohesion:
Cohesion is the indication of the relationship within module. It is concept of intra-module. Cohesion has many types but usually highly cohesion is good for software.

Types of Modules Cohesion

1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.

7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

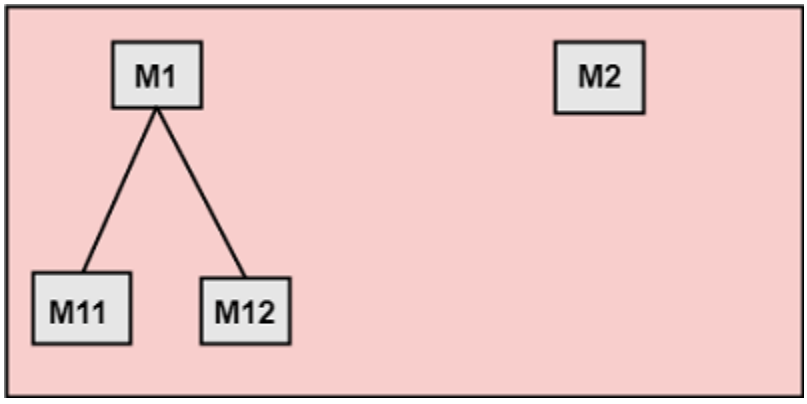
Coupling:

Coupling is also the indication of the relationships between modules. It is concept of Inter-module. Coupling has also many types but usually low coupling is good for software.

A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large.

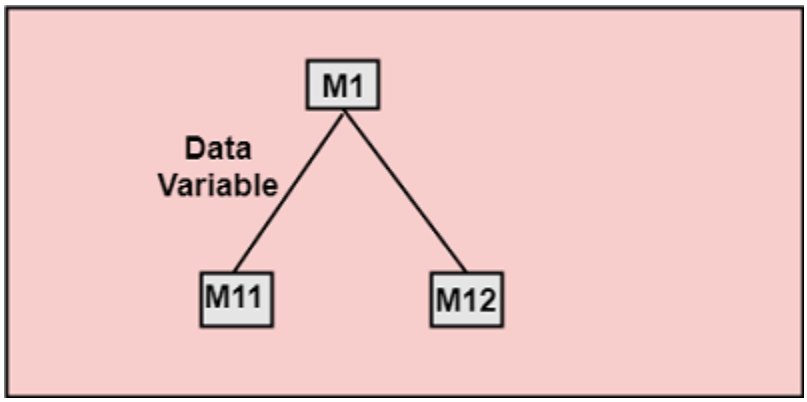
Types of Module Coupling

1. **No Direct Coupling:** There is no direct coupling between M1 and M2.



In this case, modules are subordinates to different modules. Therefore, no direct coupling.

2. **Data Coupling:** When data of one module is passed to another module, this is called data coupling.



3. **Stamp Coupling:** Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc.

4. **Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. **External Coupling:** External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

6. **Common Coupling:** Two modules are common coupled if they share information through some global data items.

7. **Content Coupling:** Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

The differences between cohesion and coupling are given below:

Cohesion	Coupling
Cohesion is the concept of intra module.	Coupling is the concept of inter module.

Cohesion represents the relationship within module.	Coupling represents the relationships between modules.
Increasing in cohesion is good for software.	Increasing in coupling is avoided for software.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
Highly cohesive gives the best software.	Where as loosely coupling gives the best software.
In cohesion, module focuses on the single thing.	In coupling, modules are connected to the other modules.

5.8 Software Design Strategies

Software design is a process to conceptualize the software requirements into software implementation.

Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.
This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.
Let us see the important concepts of Object Oriented Design:

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.
  In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.
- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their on set of sub-system and components and creates hierarchical structure in the system. Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

5.9 Function Oriented Design

The design process for software systems often has two levels. At the first level the focus is on deciding which modules are needed for the system on the basis of SRS (Software Requirement Specification) and how the modules should be interconnected.

Function Oriented Design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function.

Function Oriented Design Strategies:

Function Oriented Design Strategies are as follows:

1. Data Flow Diagram (DFD):
   A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

2. **Data Dictionaries:**
   Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirement stage, data dictionaries contains data items. Data dictionaries include Name of the item, Aliases (Other names for items), Description / purpose, Related data items, Range of values, Data structure definition / form.

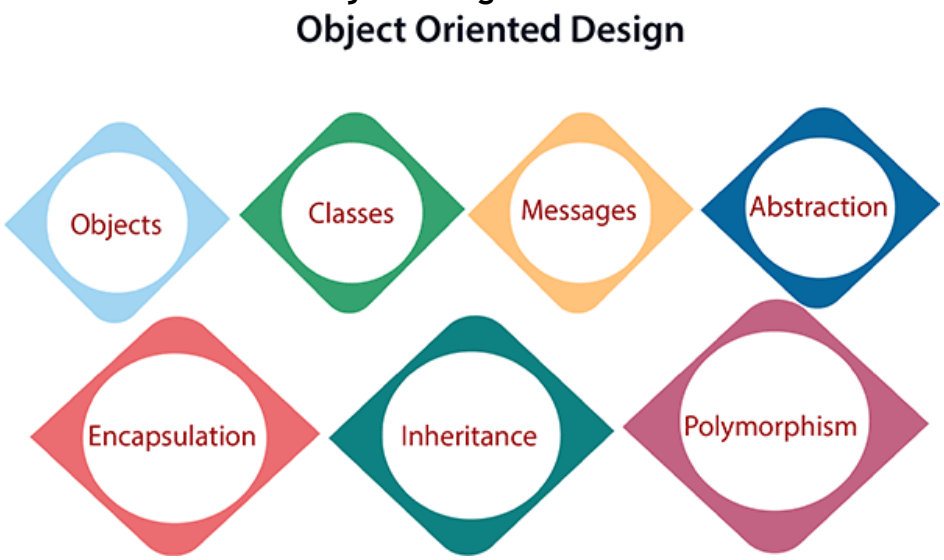
3. **Structure Charts:**
   It is the hierarchical representation of system which partitions the system into black boxes (functionality is known to users but inner details are unknown). Components are read from top to bottom and left to right. When a module calls another, it views the called module as black box, passing required parameters and receiving results.

4. **Pseudo Code:**
   Pseudo Code is system description in short English like phrases describing the function. It use keyword and indentation. Pseudo codes are used as replacement for flow charts. It decreases the amount of documentation required.

5.10 Object oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data.

The different terms related to object design are:



1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.

2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.

3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.

4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.

6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.

7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

5.11 Function oriented design vs Object Oriented Design

Sr.no.	Functional – oriented	Object - oriented
1	Basic abstractions are real world functions, such as sort, merge, track, display etc.	Basic abstractions are the data abstractions where the real world entities are customer, students, employee etc.
2	State information is available in a centralized shared data store.	State information exists in the form of data distributed among several objects of the system
3	Functions are grouped together by which a higher level function is obtained	Functions are grouped together on the basis of the data they operate on
4	Orients on functions	Orients on objects
5	Functions, get some data, process it and then return result, or do some actions	Objects is data with methods of processing it.

6 Software Metrics

6.1 Software Metrics what and why?

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

- For analysis, comparison, and critical study of different programming language concerning their characteristics.
- In comparing and evaluating the capabilities and productivity of people involved in software development.
- In the preparation of software quality specifications.
- In the verification of compliance of software systems requirements and specifications.
- In making inference about the effort to be put in the design and development of the software systems.
- In getting an idea about the complexity of the code.
- In taking decisions regarding further division of a complex module is to be done or not.
- In guiding resource manager for their proper utilization.

6.2 Token Count

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand.

In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

The basic measures are

n1 = count of unique operators.

n2 = count of unique operands.

N1 = count of total occurrences of operators.

N2 = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2

6.3 Data Structure metrics

Essentially the need for software development and other activities are to process data. Some data is input to a system, program or module; some data may be used internally, and some data is the output from a system, program, or module.

Example:

Program	Data Input	Internal Data	Data Output
Payroll	Name/Social Security No./Pay rate/Number of hours worked	Withholding rates Overtime Factors Insurance Premium Rates	Gross Pay withholding Net Pay Pay Ledgers
Spreadsheet	Item Names/Item Amounts/Relationships among Items	Cell computations Subtotal	Spreadsheet of items and totals
Software Planner	Program Size/No of Software developer on team	Model Parameter Constants Coefficients	Est. project effort Est. project duration

That's why an important set of metrics which capture in the amount of data input, processed in an output form software. A count of this data structure is called Data Structured Metrics.
There are some Data Structure metrics to compute the effort and time required to complete the project. There metrics are:

1. The Amount of Data.
2. The Usage of data within a Module.
3. Program weakness.
4. The sharing of Data among Modules.

1. The Amount of Data: To measure the amount of Data, there are further many different metrics, and these are:

- **Number of variable (VARS):** In this metric, the Number of variables used in the program is counted.
- **Number of Operands (η_2):** In this metric, the Number of operands used in the program is counted.
  η_2 = VARS + Constants + Labels
- **Total number of occurrence of the variable (N2):** In this metric, the total number of occurrence of the variables are computed

2. The Usage of data within a Module: The measure this metric, the average numbers of live variables are computed. A variable is live from its first to its last references within the procedure.

$$\text{Average no of Live variables (LV)} = \frac{\text{Sum of count live variables}}{\text{Sum of count of executable statements}}$$

3. Program weakness: Program weakness depends on its Modules weakness. If Modules are weak (less Cohesive), then it increases the effort and time metrics required to complete the project.

4.There Sharing of Data among Module: As the data sharing between the Modules increases (higher Coupling), no parameter passing between Modules also increased, As a result, more effort and time are required to complete the project. So Sharing Data among Module is an important metrics to calculate effort and time.

6.4 Information Flow metrics

The other set of metrics we would live to consider are known as Information Flow Metrics. The basis of information flow metrics is found upon the following concept the simplest system consists of the component, and it is the work that these components do and how they are fitted together that identify the complexity of the system. The following are the working definitions that are used in Information flow:

- **Component:** Any element identified by decomposing a (software) system into it's constituent's parts.
- **Cohesion:** The degree to which a component performs a single function.
- **Coupling:** The term used to describe the degree of linkage between one component to others in the same system.

Information Flow metrics deal with this type of complexity by observing the flow of information among system components or modules. This metrics is given by **Henry and Kafura**. So it is also known as Henry and Kafura's Metric.

This metrics is based on the measurement of the information flow among system modules. It is sensitive to the complexity due to interconnection among system component. This measure includes the complexity of a software module is defined to be the sum of complexities of the procedures included in the module. A process contributes complexity due to the following two factors.

1. The complexity of the procedure code itself.
2. The complexity due to the procedure's connections to its environment. The effect of the first factor has been included through LOC (Line Of Code) measure. For the quantification of the second factor, Henry and Kafura have defined two terms, namely FAN-IN and FAN-OUT.

FAN-IN: FAN-IN of a procedure is the number of local flows into that procedure plus the number of data structures from which this procedure retrieve information.

FAN -OUT: FAN-OUT is the number of local flows from that procedure plus the number of data structures which that procedure updates.

6.5 Metrics Analysis

Metrics are measurements. A software development project is an experiment that is based on a number of assumptions theories and estimates. To confirm these measurements are taken for all significant processes and then analyzed. The results of the analysis are used to make decisions.

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

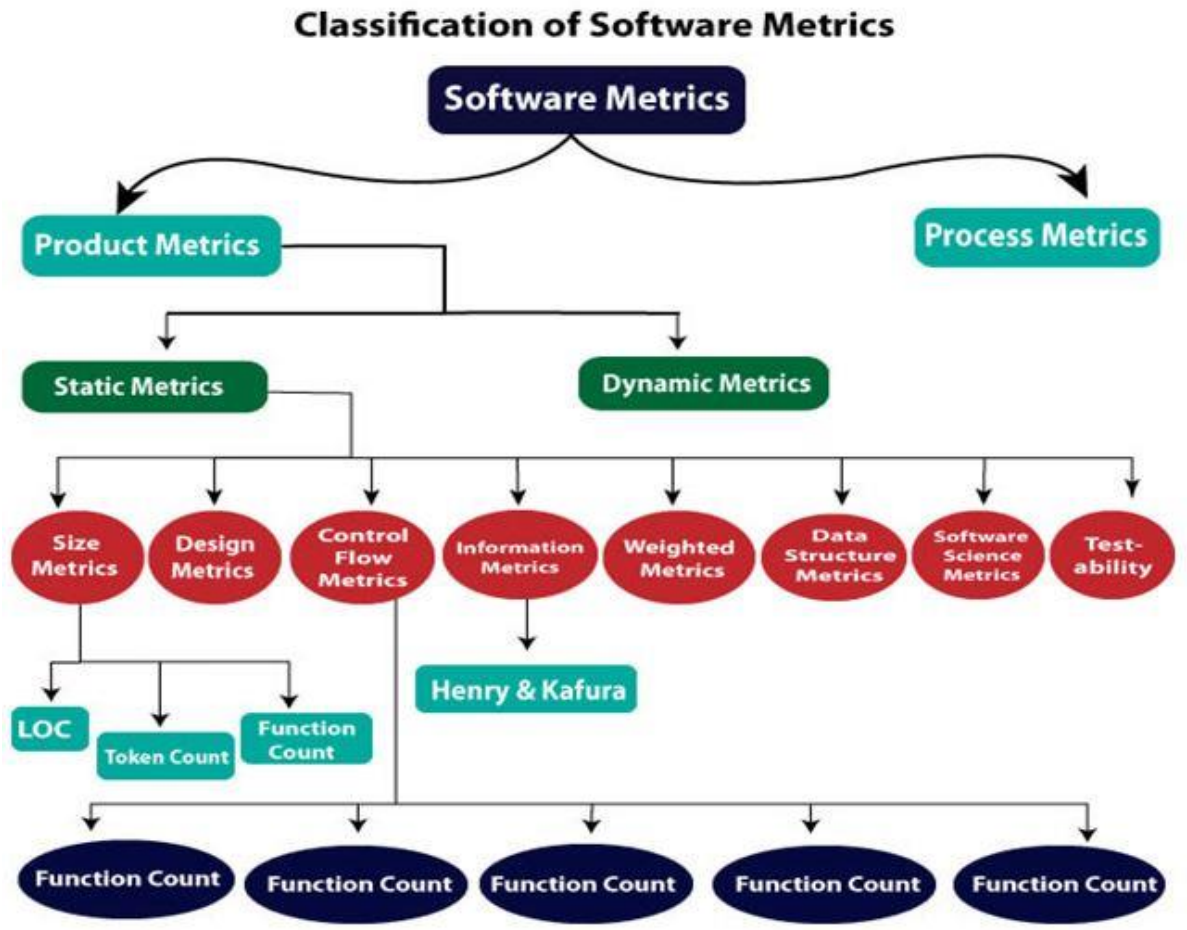
Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Types of Metrics

Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

Comparative study of various design methodology of software systems.

- For analysis, comparison, and critical study of different programming language concerning their characteristics.
- In comparing and evaluating the capabilities and productivity of people involved in software development.
- In the preparation of software quality specifications.
- In the verification of compliance of software systems requirements and specifications.
- In making inference about the effort to be put in the design and development of the software systems.
- In getting an idea about the complexity of the code.
- In taking decisions regarding further division of a complex module is to be done or not.
- In guiding resource manager for their proper utilization.
- In comparison and making design tradeoffs between software development and maintenance cost.

Disadvantage of Software Metrics

- The application of software metrics is not always easy, and in some cases, it is difficult and costly.
- The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.
- These are useful for managing software products but not for evaluating the performance of the technical staff.

7 Software Reliability

7.1 Basic Concepts

Software Reliability means **Operational reliability**. It is described as the ability of a system or component to perform its required functions under static conditions for a specific period.
Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.
Software Reliability is an essential connect of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turn to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

7.2 Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

The modern view of a quality associated with a software product several quality methods such as the following:
Portability: A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

Usability: A software product has better usability if various categories of users can easily invoke the functions of the product.

Reusability: A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

Correctness: A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

Maintainability: A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

7.3 Software Reliability Model

A software reliability model indicates the form of a random process that defines the behavior of software failures to time.

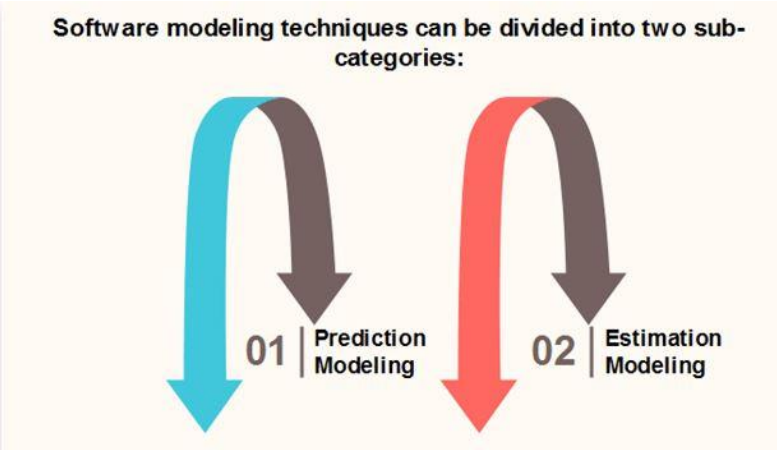
Software reliability models have appeared as people try to understand the features of how and why software fails, and attempt to quantify software reliability.

Over 200 models have been established since the early 1970s, but how to quantify software reliability remains mostly unsolved.

There is no individual model that can be used in all situations. No model is complete or even representative.

Most software models contain the following parts:

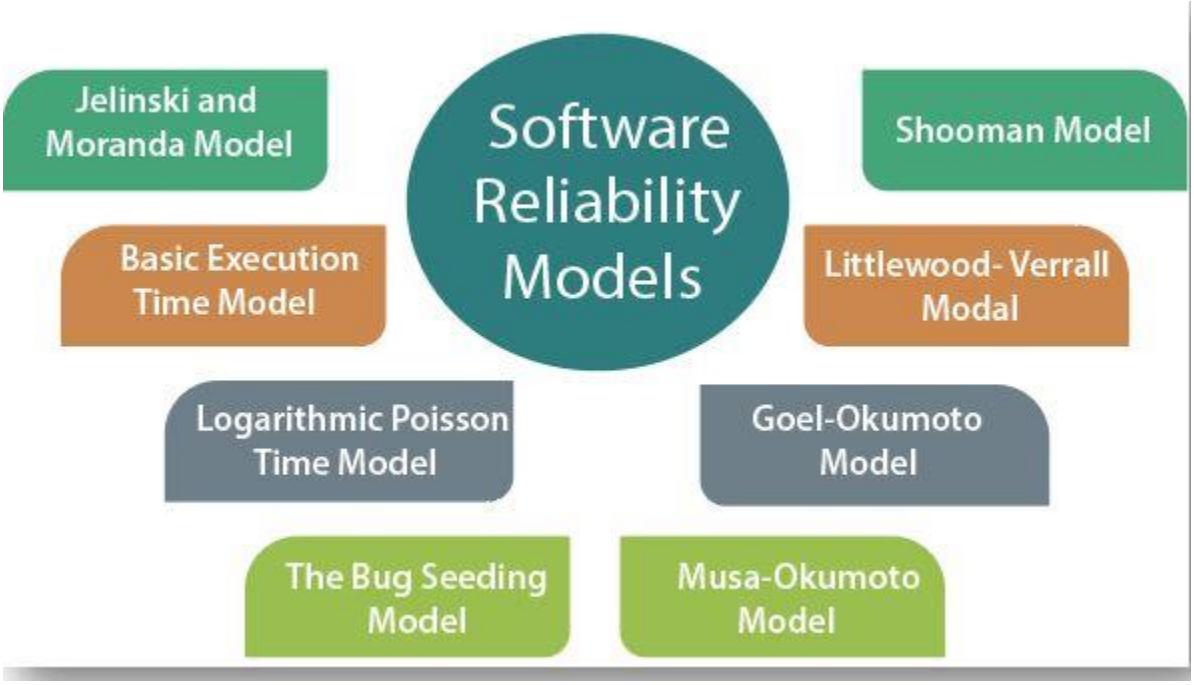
- Assumptions
- Factors



Basics	Prediction Models	Estimation Models
Data Reference	Uses historical information	Uses data from the current software development effort.
When used in development cycle	Usually made before development or test phases; can be used as early as concept phase.	Usually made later in the life cycle (after some data have been collected); not typically used in concept or development phases.

Time Frame	Predict reliability at some future time.	Estimate reliability at either present or some next time.

A reliability growth model is a numerical model of software reliability, which predicts how software reliability should improve over time as errors are discovered and repaired. These models help the manager in deciding how much efforts should be devoted to testing. The objective of the project manager is to test and debug the system until the required level of reliability is reached.



7.4 Capability maturity Model (CMM)

The Software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization. The higher the level, the better the software development process, hence reaching each level is an expensive and time-consuming process.

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.

Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

The 5 levels of CMM are as follows:

Level-1: Initial –
- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –
- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.
- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –
- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –
- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.

- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.
- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.

8 Software Testing

8.1 Testing Process

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.
According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?
It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called **Unit Testing**. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

When to Start Testing?
An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.
It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −

- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process −

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management dec

8.2 Some Important Terminologies

White-box testing

In white-box testing, the tester is aware of exactly what the 'module under the test' does, and how it does it. In other words, they know the inner workings of the module. One of the basic goals of white-box testing is to verify working flow for an application.

Black-box testing

In black-box testing, functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as 'Behavioral Testing'.

- The tester gets the requirements and specifications of the system.
- Choose valid inputs (positive testing) and invalid inputs (negative testing).
- The tester determines expected outputs for all those inputs.
- The tester constructs the test case and executes it.
- Compares the actual output with the expected output.
- Defects are fixed (if any) and retested.

Unit Testing

Unit testing consists of the testing of individual modules or components. Its objective is to test each unit of the software.

Functional testing

The testing of the system's functionality and behavior; Functional tests verify that our application does what it's designed to do. More specifically, we are aiming to test each functional element of our software to verify that the output is correct. Functional testing covers Unit testing, Component testing, and UI testing among others.

Assertion

An assertion is used in automated testing to assert the expected behavior of the test. Assertions are commonly used in Unit testing, but the same concept applies to other forms of automated tests.

Test case
A test case is the complete set of pre-requisites, required data, and expected outcomes for a given instance of a Test. A test case is designed to pass or to fail. Often this depends on the data passed to the Test.

Test Scenario
A sequence of activities performed in a system, such as logging in, signing up a customer, ordering products, and printing an invoice. You can combine test cases to form a scenario especially at higher test levels.

Test Suite
A Test Suite is a collection of test cases. In automated testing, it can mean a collection of test scripts.

Mocking
Mocking means creating a fake version of an external or internal service that can stand-in for the real one, helping your tests run more quickly and more reliably.

8.3 Unit Testing

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team.
The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing
Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.
There is a limit to the number of scenarios and test data that a developer can use to verify a source code.

8.4 Integration Testing
Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

Bottom-up integration
This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.

Top-down integration

In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.
In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing.

8.5 System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.
System testing is important because of the following reasons −

- System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- The application is tested thoroughly to verify that it meets the functional and technical specifications.
- The application is tested in an environment that is very close to the production environment where the application will be deployed.
- System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

8.5 Regression Testing

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.
Regression testing is important because of the following reasons −

- Minimize the gaps in testing when an application with changes made has to be tested.
- Testing the new changes to verify that the changes made did not affect any other area of the application.
- Mitigates risks when regression testing is performed on the application.
- Increase speed to market the product.

8.7 Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. There are different causes that contribute in lowering the performance of a software −

- Network delay
- Client-side processing
- Database transaction processing
- Load balancing between servers
- Data rendering

Performance testing is considered as one of the important and mandatory testing type in terms of the following aspects −

- Speed (i.e. Response Time, data rendering and accessing)

- Capacity
- Stability
- Scalability

Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing

It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time.

Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Stress Testing

Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.

This testing can be performed by testing different scenarios such as −

- Shutdown or restart of network ports randomly
- Turning the database on or off
- Running different processes that consume resources such as CPU, memory, server, etc.

8.8 White Box Testing and Black Box Testing

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

The following table lists the advantages and disadvantages of white-box testing.

Advantages	**Disadvantages**
As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.	Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
It helps in optimizing the code.	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
Extra lines of code can be removed which can bring in hidden defects.	It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.	

Black-Box Testing

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

The following table lists the advantages and disadvantages of black-box testing.

Advantages	**Disadvantages**
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.

8.9 Acceptance Testing

he most important type of testing, as it is conducted by the Quality Assurance Team who will measure whether the application meets the intended specifications and satisfies the client’s requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated.

Acceptance tests are not only intended to point out simple spelling mistakes, errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.

8.10 Alpha and Beta Testing

- Alpha testing requires a testing environment whereas Beta testing doesn’t require any such environment.
- Alpha testing uses both black and white box testing while Beta testing uses only blackbox testing.
- Alpha testing is done by testers and quality analysts inside the organization whereas Beta testing is done by real users who will be actually using the software.

- Alpha testing takes longer duration to complete execution while Beta testing gets completed within a few weeks.
- Alpha testing is done at the developer's site while Beta testing is done at the client's site.
- Alpha testing does not check security and reliability of the product while Beta testing checks for security and reliability of the product in depth by the end users.
- Multiple test cycles are organized in alpha testing while in beta testing only one or two test cycles are there.

8.11 Debugging Techniques, tools and approaches

The important technique to find and remove the number of errors or bugs or defects in a program is called Debugging. It is a multistep process in software development. It involves identifying the bug, finding the source of the bug and correcting the problem to make the program error-free.

To perform the debugging process easily and efficiently, it is necessary to follow some techniques. The most commonly used debugging strategies are,

- Debugging by brute force
- Induction strategy
- Deduction strategy
- Backtracking strategy and
- Debugging by testing.

Debugging by brute force is the most commonly used technique. This is done by taking memory dumps of the program which contains a large amount of information with intermediate values and analyzing them, but analyzing the information and finding the bugs leads to a waste of time and effort.

Induction strategy includes the Location of relevant data, the Organization of data, the Devising hypothesis (provides possible causes of errors), and the Proving hypothesis.

Deduction strategy includes Identification of possible causes of bugs or hypothesis Elimination of possible causes using the information refining of the hypothesis (analyzing one-by-one)

The backtracking strategy is used to locate errors in small programs. When an error occurs, the program is traced one step backward during the evaluation of values to find the cause of bug or error.

Debugging by testing is the conjunction with debugging by induction and debugging by deduction technique. The test cases used in debugging are different from the test cases used in the testing process.

Debugging Tools

A software tool or program used to test and debug the other programs is called a debugger or a debugging tool. It helps to identify the errors of the code at the various stages of the software development process. These tools analyze the test run and find the lines of codes that are not executed. Simulators in other debugging tools allow the user to know about the display and behavior of the operating system or any other computing device.

Mostly used **Debugging Tools** are GDB, DDD, and Eclipse.

- **GDB Tool:** This type of tool is used in UNIX programming. GDB is pre-installed in all Linux systems if not, it is necessary to download the GCC compiler package.
- **DDD Tool:** DDD means Data Display Debugger, which is used to run a Graphic User Interface (GUI) in UNIX systems.
- **Eclipse:** An IDE tool is the integration of an editor, build tool, debugger and other development tools. IDE is the most popular Eclipse tool. It works more efficiently when compared to the DDD, GDB and other tools.

The list of debugging tools is listed below.

- AppPuncher Debugger is used for debugging Rich Internet Applications
- AQtime debugger
- CA/EZ TEST is a CICS interactive test/debug software package
- CharmDebug is a Debugger for Charm++
- CodeView debugger
- DBG is a PHP Debugger and Profiler
- FusionDebug
- Debugger OpenGL, OpenGL ES, and OpenCL Debugger and Profiler. For Windows, Linux, Mac OS X, and iPhone
- GNU Debugger (GDB), GNU Binutils
- Intel Debugger (IDB)

9 Software Maintenance

9.1 Need For Software Maintenance

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Software Maintenance must be performed in order to:

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

9.2 Types of software Maintenance

1. **Corrective Maintenance:** Corrective Maintenance is a reactive process that is focused on fixing failures in the system. It refers to the modification and enhancement done to the coding and design of a software to fix errors or defects detected by the user or concluded by error user report. This type of maintenance is initiated in the system to resolve any new or missed defects in the software. Corrective Maintenance is further divided into two types:
    - Emergency Repairs.
    - Scheduled Repairs.

2. **Adaptive Maintenance:** Adaptive Maintenance is initiated as a consequence of internal needs, like moving the software to a different hardware or software platform compiler, operating system or new processor and to match the external completion and requirements. The main goal of Adaptive Maintenance is to keep the software program up-to-dated and to meet the needs and demands of the user and the business.

3. **Perfective Maintenances:** Here enhancements, modifications and updates are done in order to keep the software usable for a long period of time. It aims at achieving reduced costs in using the system and increasing its maintainability. The process of perfective maintenance includes making the product faster, cleaner structured, improving its reliability and performance, adding new features, and more.

4. **Preventive Maintenance:** Most commonly known as Software Re-engineering, the purpose of this type of maintenance is to prevent future problems in the software by making it more understandable, enhancing its features and improving its existing qualities, which will facilitate future maintenance work.

The objective of Preventive Maintenance is to attend problems, which may seem insignificant but can cause serious issues in the future.

9.3 Software Maintenance Process Model

Software Maintenance is an important phase of Software Development Life Cycle (SDLC), and it is implemented in the system through a proper software maintenance process, known as **Software Maintenance Life Cycle (SMLC)**. This life cycle consists of seven different phases, each of which can be used in iterative manner and can be extended so that customized items and processes can be included. These seven phases of Software Maintenance process are:

- **Identification Phase:** In this phase, the requests for modifications in the software are identified and analyzed. Each of the requested modification is then assessed to determine and classify the type of maintenance activity it requires. This is either generated by the system itself, via logs or error messages, or by the user.

- **Analysis Phase:** The feasibility and scope of each validated modification request are determined and a plan is prepared to incorporate the changes in the software.

- **Design Phase:** The new modules that need to be replaced or modified are designed as per the require.

- **Implementation Phase:** In the implementation phase, the actual modification in the software code are made, new features that support the specifications of the present software are added, and the modified software is installed.

- **System Testing Phase: Regression testing** is performed on the modified system to ensure that no defect, error or bug is left undetected. Furthermore, it validates that no new faults are introduced in the software as a result of maintenance activity.

- **Acceptance Testing Phase: Acceptance testing** is performed on the fully integrated system by the user or by the third party specified by the end user.

- **Delivery Phase:** Once the acceptance testing is successfully accomplished, the modified system is delivered to the users. In addition to this, the user is provided proper consisting of manuals and help files that describe the operation of the software along with its hardware specifications. The final testing of the system is done by the client after the system is delivered.

Software Maintenance Models:

To overcome internal as well as external problems of the software, Software maintenance models are proposed. These models use different approaches and techniques to simplify the process of maintenance as well as to make is cost effective. Software maintenance models that are of most importance are:

Quick-Fix Model:

This is an ad hoc approach used for maintaining the software system. The objective of this model is to identify the problem and then fix it as quickly as possible. The advantage is that it performs its work quickly and at a low cost. This model is an approach to modify the software code with little consideration for its impact on the overall structure of the software system.

Iterative Enhancement Model:

Iterative enhancement model considers the changes made to the system are iterative in nature. This model incorporates changes in the software based on the analysis of the existing system. It assumes complete documentation of the software is available in the beginning. Moreover, it attempts to control complexity and tries to maintain good design.

Iterative Enhancement Model is divided into three stages:

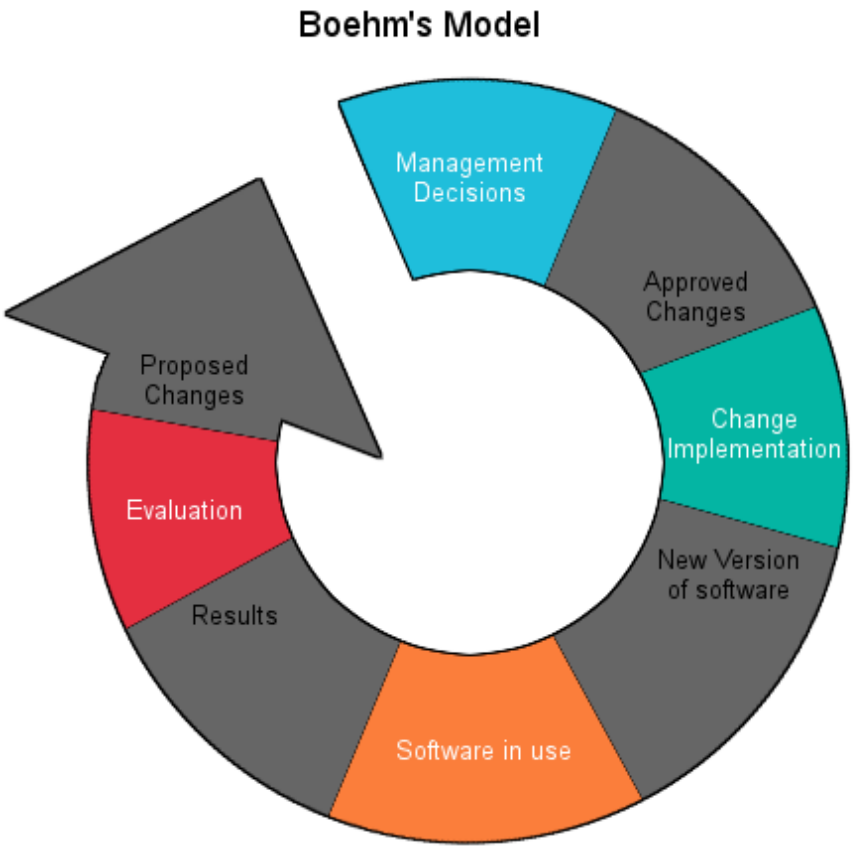
1. Analysis of software system.
2. Classification of requested modifications.
3. Implementation of requested modifications.

The Re-use Oriented Model:

The parts of the old/existing system that are appropriate for reuse are identified and understood, in Reuse Oriented Model. These parts are then go through modification and enhancement, which are done on the basis of the specified new requirements. The final step of this model is the integration of modified parts into the new system.

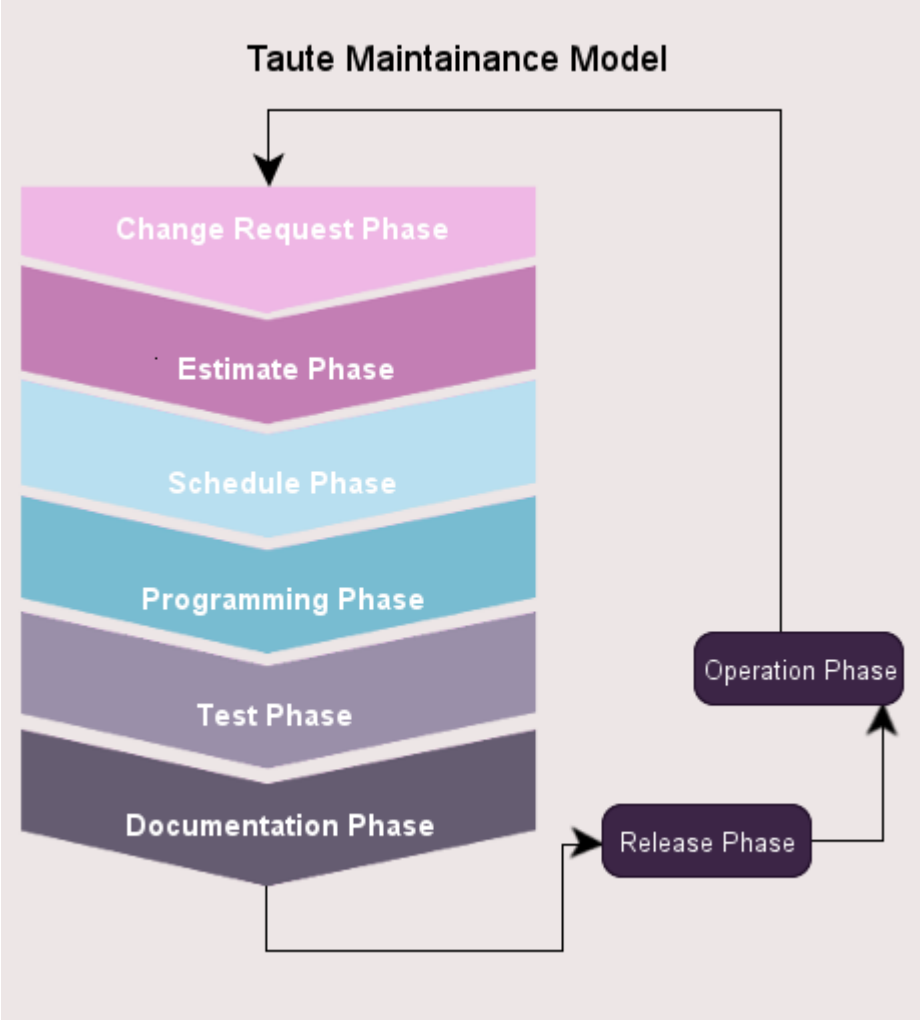
Boehm's Model:

Boehm's Model performs maintenance process based on the economic models and principles. It represents the maintenance process in a closed loop cycle, wherein changes are suggested and approved first and then are executed.



Taute Maintenance Model:

Named after the person who proposed the model, Taute's model is a typical maintenance model that consists of eight phases in cycle fashion. The process of maintenance begins by requesting the change and ends with its operation. The phases of **Taute's Maintenance Model** are:



1. **Change request Phase.**
2. **Estimate Phase.**
3. **Schedule Phase.**
4. **Programming Phase.**
5. **Test Phase.**
6. **Documentation Phase.**
7. **Release Phase.**
8. **Operation Phase.**

9.4 software maintenance cost

Software Maintenance is a very broad activity that takes place once the operation is done. It optimizes the software performance by reducing errors, eliminating useless lines of codes and applying advanced development. It can take up to 1-2 years to build

a software system while its maintenance and modification can be an ongoing activity for 15-20 years.

Categories of Software Maintenance:
1. Corrective Maintenance
2. Adaptive Maintenance
3. Perfective Maintenance
4. Preventive Maintenance

The cost of system maintenance represents a large proportion of the budget of most organizations that use software system. More than 65% of software lifecycle cost is expanded in the maintenance activities.
Cost of software maintenance can be controlled by postponing the. development opportunity of software maintenance but this will cause the following intangible cost:
- Customer dissatisfaction when requests for repair or modification cannot be addressed in a timely manner.
- Reduction in overall software quality as a result of changes that introduce hidden errors in maintained software.

Software maintenance cost factors
The key factors that distinguish development and maintenance and which lead to higher maintenance cost are divided into two subcategories:
1. Non-Technical factors
2. Technical factors

Non-Technical factors:
The Non-Technical factors include:
1. Application Domain
2. Staff stability
3. Program lifetime
4. Dependence on External Environment
5. Hardware stability

Technical factors:
Technical factors include the following:
1. module independence
2. Programming language
3. Programming style
4. Program validation and testing
5. Documentation
6. Configuration management techniques

10 Quality Assurance

10.1 Software quality attributes

1) Usability
It is described as how the user is utilizing a system effectively and the ease of which users can learn to operate or control the system. The well-known principle of usability is KISS (Keep It Simple Stupid). Software applications should be user-friendly.

2) Reliability
It is the ability of a system to continue to keep operating over time

3) Availability
It is the ratio of the available system time to the total working time it is required or expected to function.

4) Portability
It is the ability of a software application to run on numerous platforms such as data portability, hosting, viewing, etc.,

5) Testability
It shows how well the system or component facilitates to perform tests to determine whether the predefined test criteria have been met.

6) Scalability
It is the ability of a system to handle the demand for stress caused by increased usage without decreasing performance.

7) Flexibility
It is the ability of a system to adapt to future changes.

8) Reusability
It is the use of existing software I more than one software with small or no change. It is a cost-efficient and time-saving quality attribute.

9) Maintainability
It is the ability of a software application to maintain easily and support changes cost-effectively.

10) Supportability
It is the ability of a system that satisfies necessary requirements and needs to identifying and solving problems.

11) Interoperability

It is the ability of two or more systems to communicate or exchange data easily and to use the data that has been exchanged.

12) Performance
It is the ability of a system in the form of responsiveness to various actions within a certain period of time.

13) Security
It is the ability of a system to resist or block malicious or unauthorized attempts that destroy the system and at the same time provide access to legitimate users.

10.2 Quality Factors

1- Product operation:

Correctness:
Correctness is the extent to which a program satisfies its specifications.

Reliability:
Reliability is the property that defines how well the software meets its requirements.

Efficiency:
Efficiency is a factor relating to all issues in the execution of software; it includes considerations such as response time, memory requirement, and throughput.

Usability:
Usability, or the effort required locating and fixing errors in operating programs.
2-Producttransition:

Portability:
Portability is the effort required to transfer the software from one configuration to another.

Reusability:
Reusability is the extent to which parts of the software can be reused in other related applications.

2- Product revision:

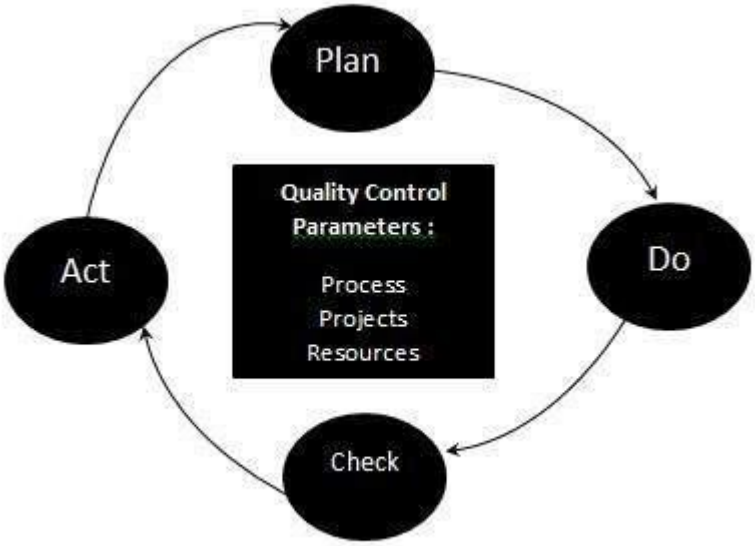
Maintainability:
Maintainability is the effort required to maintain the system in order to check the quality.

Testability:
Testability is the effort required to test to ensure that the system or a module performs its intended function.

Flexibility:
Flexibility is the effort required to modify an operational program.

10.3 Quality Control
Quality control is a set of methods used by organizations to achieve quality parameters or quality goals and continually improve the organization's ability to ensure that a software product will meet quality goals.

Quality Control Process:



The three class parameters that control software quality are:
- Products
- Processes
- Resources

The total quality control process consists of:
- Plan - It is the stage where the Quality control processes are planned
- Do - Use a defined parameter to develop the quality
- Check - Stage to verify if the quality of the parameters are met
- Act - Take corrective action if needed and repeat the work

Quality Control characteristics:
- Process adopted to deliver a quality product to the clients at best cost.
- Goal is to learn from other organizations so that quality would be better each time.
- To avoid making errors by proper planning and execution with correct review process.

10.4 Quality Assurance

Software Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

Software Quality Assurance has:

1. A quality management approach
2. Formal technical reviews
3. Multi testing strategy
4. Effective software engineering technology
5. Measurement and reporting mechanism

Benefits of Software Quality Assurance (SQA):

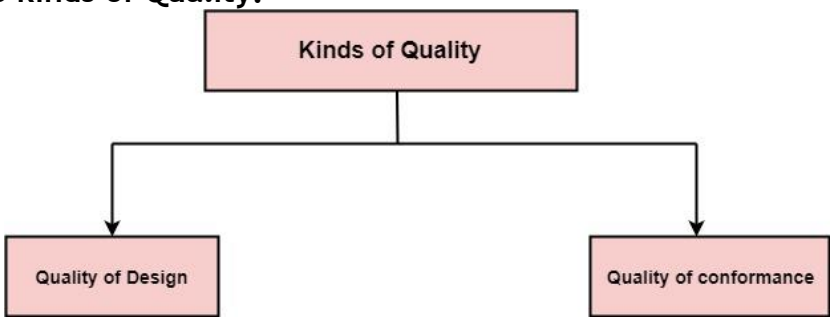
1. SQA produces high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.
4. SQA is beneficial in the condition of no maintenance for a long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

Disadvantage of SQA:

There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

10.4 Software quality Assurance Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

There are two kinds of Quality:



Quality of Design: Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

Quality of conformance: Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

Software Quality: Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

Quality Assurance focuses on how the engineering and management activity will be done?
As anyone is interested in the quality of the final product, it should be assured that we are building the right product.
It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

10.6 Software Safety

Safety software refers to software that as its primary purpose improves the safety of an organization through the more efficient management of its safety protocols. Safety software allows management to familiarize themselves with corporate safety activities, immediately identify and minimize risk, as well as improve company culture that creates trust between front line employees and management.

Safety software allows organizations to standardize their safety procedures and track, analyze, and optimize safety related activities more efficiently. This enables organizations to focus on measurable outcomes and make safety related decisions that are based on empirical data.

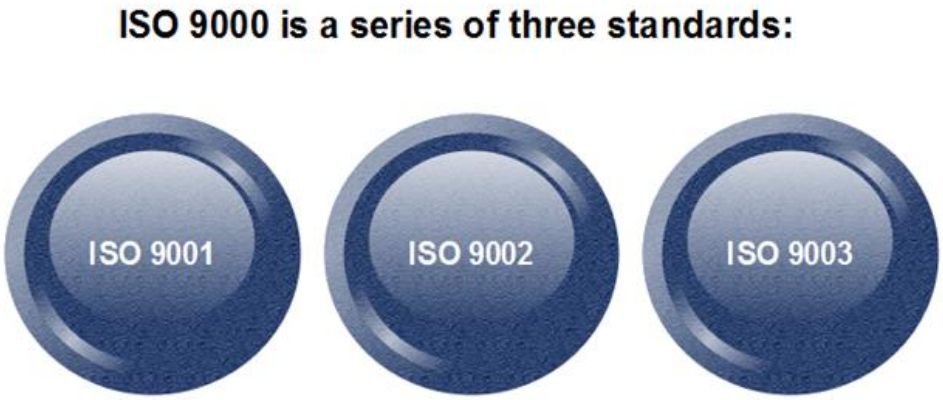
To achieve an acceptable level of safety for software used in critical applications, software system safety engineering must be given primary emphasis early in the requirements definition and system conceptual design process. Safety-critical software must then receive continuous management emphasis and engineering analysis throughout the development and operational lifecycles of the system.

- Functional safety is achieved through engineering development to ensure correct execution and behavior of software functions as intended
- Safety consistent with mission requirements, is designed into the software in a timely, cost effective manner.
- On complex systems involving many interactions safety-critical functionality should be identified and thoroughly analyzed before deriving hazards and design safeguards for mitigations.
- Failure modes, including hardware, software, human and system are addressed in the design of the software
- Safety issues and safety attributes are addressed as part of the software testing effort at all levels.

10.7 The ISO 9000 Model

ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc.

Types of ISO 9000 Quality Standards



The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically.

The types of industries to which the various ISO standards apply are as follows.

1. **ISO 9001:** This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.

2. **ISO 9002:** This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category

industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.

3. **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:

1. **Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.

2. **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.

3. **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.

4. **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.

5. **Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.

6. **Continued Inspection:** The registrar continued to monitor the organization time by time.

10.8 SEI Capability maturity Model

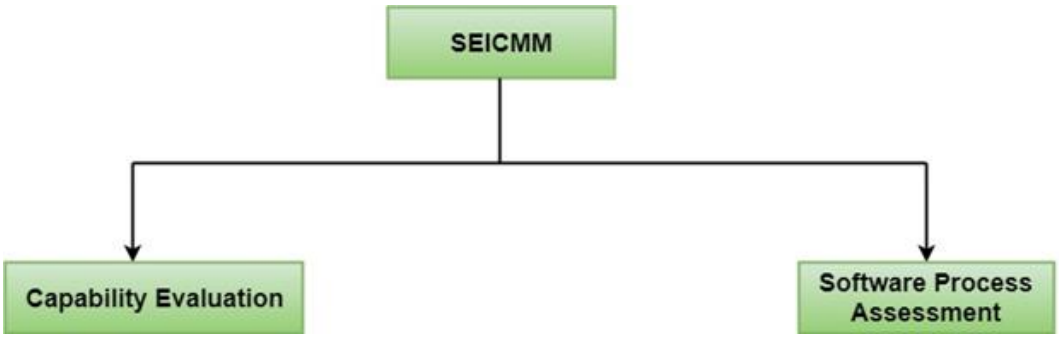
The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.

The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD). Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

Methods of SEICMM

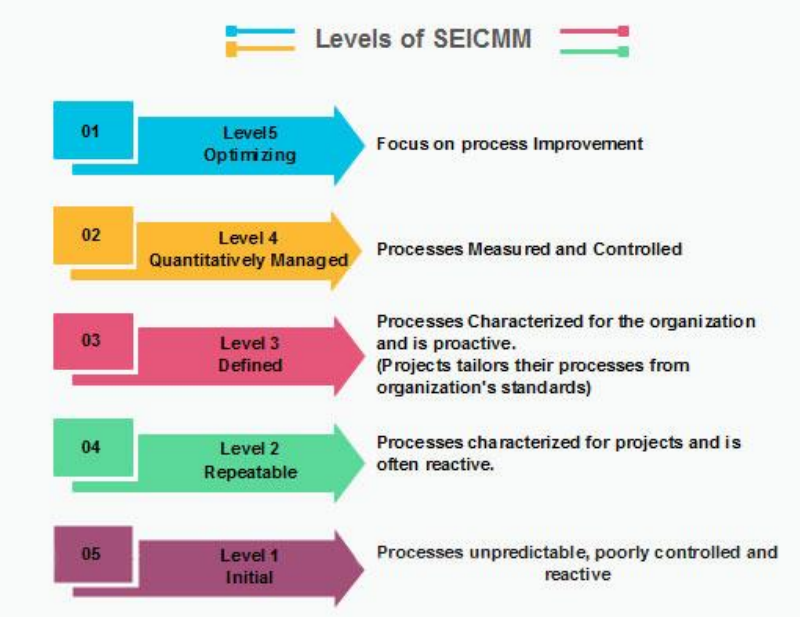
There are two methods of SEICMM:



Capability Evaluation: Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work. Therefore, the results of the software process capability assessment can be used to select a contractor.

Software Process Assessment: Software process assessment is used by an organization to improve its process capability. Thus, this type of evaluation is for purely internal use.

SEI CMM categorized software development industries into the following five maturity levels. The various levels of SEI CMM have been designed so that it is easy for an organization to build its quality system starting from scratch slowly.



10.9 Verification and Validation

Verification	Validation
1. Verification is a static practice of verifying documents, design, code and program.	1. Validation is a dynamic mechanism of validating and testing the actual product.
2. It does not involve executing the code.	2. It always involves executing the code.
3. It is human based checking of documents and files.	3. It is computer based execution of program.
4. Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.	4. Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Verification** is to check whether the software conforms to specifications.	5. **Validation** is to check whether software meets the customer expectations and requirements.
6. It can catch errors that validation cannot catch. It is low level exercise.	6. It can catch errors that verification cannot catch. It is High Level Exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	8. Validation is carried out with the involvement of testing team.
9. It generally comes first-done before validation.	9. It generally follows after **verification**.