

UNIT-3

Software 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 focuses on:

- **Software's portability-** The ease with which the software can be transposed from one environment to another, Ex: adaptability, install ability.
- **Software's usability-** The degree to which the software is easy to use. Ex: understand ability, learnability and operability.
- **Software's reusability-** The extent to which a program can be reused in other application.
- **Software's correctness-** To extent to which software meets its specifications.
- **Software's maintainability-** The ease with which repair may be made to the software: analysability, changeability, stability, testability.
- **software's error control-** Error handling refers to the response and recovery procedures from error conditions present in a software application

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

Major Software Quality Assurance Activities:

1) SQA Management Plan

Make a plan for how you will carry out the SQA throughout the project. Think about which set of software engineering activities are the best for project. Check level of SQA team skills.

2) Multi testing Strategy:

SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

3) Multi testing Strategy:

Do not depend on a single testing approach. When you have a lot of testing approaches available use them.

4) Measure Change Impact:

The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.

5) Manage Good Relations:

In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of SQA team with programmer's team will impact directly and badly on project. Don't play politics.

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:

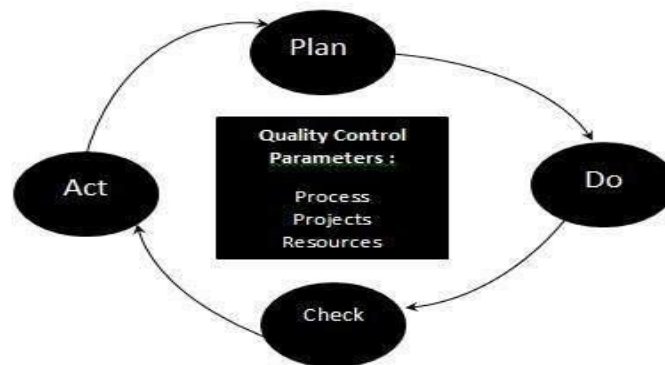
There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

QA Quality Assurance	QC Quality Control	QE Quality Engineer
Define the Testing Process	Actual Testing	QC = QE
They won't do Actual Testing	Executing Test Cases	These Days People prefer calling QC as QE
What Kind Of Templates	Ex: Reporting Defects	Ex: Automation
Smoke Testing	Perform Testing etc.	Coding Skill
Regression Testing		

What is 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.

What is Software Quality Management

Software Quality Management ensures that the required level of quality is achieved by submitting improvements to the product development process. SQA aims to develop a culture within the team and it is seen as everyone's responsibility.

Software Quality management should be independent of project management to ensure independence of cost and schedule adherences. It directly affects the process quality and indirectly affects the product quality.

Activities of Software Quality Management:

- **Quality Assurance** - QA aims at developing Organizational procedures and standards for quality at Organizational level.
- **Quality Planning** - Select applicable procedures and standards for a particular project and modify as required to develop a quality plan.
- **Quality Control** - Ensure that best practices and standards are followed by the software development team to produce quality products.

Objective Questions:

Q.1 The First step in Software Development Life Cycle is:

- a) **Preliminary Investigation and Analysis**
- b) System Design
- c) System Testing
- d) Coding

Q.2 The Study of existing system referred to as:

- a) System Planning
- b) **System Analysis**
- c) Feasibility Study
- d) Detailed DFD

Q.3 In Software Engineering Prototyping Means

- a) **End User Understanding and Approval**
- b) Program Logic
- c) Planning of Dataflow Organisation
- d) None of these

Q.4) what is Prototype?

- A) Mini Model of Existing System
- b) **Mini Model of Proposed System**
- c) Working Model of existing System
- d) None of these above

Q.5) Risk Analysis of Project is done in?

- a) System Analysis Phase
- b) **Feasibility Study**
- c) Implementation Phase
- d) Maintenance Phase

Q.6) In which Steps of SDLC Project Termination Could be done?

- a) Design Phase
- b) System Maintenance Phase
- c) **Feasibility Study Phase**
- d) Coding Phase

Q.7) The Fundamental Objective of System Analysis is to

- a) Understand Computer Hardware
- b) Train Manager in Mathematical Analysis
- c) **Study and understand the complex system and modify it in some way**
- d) Run Simulation Program

Q.8) Which of the following is not a stage of SDLC?

- a) System Analysis
- b) **Problem Identification**
- c) System design
- d) Feasibility Study

Q.9) An Iterative Process of Software Development in which requirement are converted into a working system that is continually revise through close work between analyst and end user?

- a) Waterfall Modelling
- b) Iterative Modelling
- c) Spiral Modelling**
- d) None of these

Q.10) The Phase of software development associated with the creation of test data?

- a) System analysis
- b) Physical Design
- c) System Acceptance**
- d) Logical Design

Quality Model

McCall Software Quality Model:

McCall software quality model was introduced in 1977. This model is incorporated with many attributes, termed as software factors, which influence software. The model distinguishes between two levels of quality attributes:

1. Quality Factors –

The higher level quality attributes which can be accessed directly are called quality factors. These attributes are external attributes. The attributes in this level are given more importance by the users and managers.

2. Quality Criteria –

The lower or second level quality attributes which can be accessed either subjectively or objectively are called Quality Criteria. These attributes are internal attributes. Each quality factor has many second levels of quality attributes or quality criteria.

Example –

Usability quality factor is divided into operability, training, communicativeness, input/output volume, input /output rate. This model classifies all software requirements into 11 software quality factors. The 11 factors are organised into three product quality factors – product operation, product revision, and product transition factors.

The following are the product quality factors –

1. Product Operation:

It includes five software quality factors, which are related with the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage and its correctness. These factors help in providing a better user experience.

- **Correctness** –
The extent to which software meets its requirements specification.
- **Efficiency** –
The amount of hardware resources and code the software needs to perform a function.
- **Integrity** –
The extent to which the software can control an unauthorized person from the accessing the data or software.
- **Reliability** –
The extent to which software performs its intended functions without failure.
- **Usability** –
The extent of effort required to learn, operate and understand the functions of the software.

2. Product Revision:

It includes three software quality factors, which are required for **testing** and **maintenance** of the software. They provide ease of maintenance, flexibility and testing effort to support the software to be functional according to the needs and requirements of the user in the future.

- **Maintainability** –
The effort required to detect and correct an error during maintenance phase.
- **Flexibility** –
The effort needed to improve an operational software program.
- **Testability** –
The effort required to verify software to ensure that it meets the specified requirements.

3. Product Transition:

It includes three software quality factors that allow the software to adapt to the change of environments in the new platform or technology from the previous.

- **Portability** –
The effort required to transfer a program from one platform to another.
- **Re-usability** –
The extent to which the program's code can be reused in other applications.
- **Interoperability** –
The effort required to integrate two systems with one another.

Bohem Software Quality Model

In 1978, B.W. Boehm introduced his software quality model. The model represents a hierarchical quality model similar to McCall Quality Model to define software quality using a predefined set of attributes and metrics, each of which contributes to overall quality of software.

The difference between Boehm's and McCall's model is that McCall's model primarily focuses on precise measurement of high-level characteristics, whereas Boehm's quality model is based on a wider range of characteristics.

Example –

Characteristics of **hardware performance** that are missing in McCall's model.

The Boehm's model has three levels for quality attributes. These levels are divided based on their characteristics. These levels are primary uses (high level characteristics), intermediate constructs (mid-level characteristics) and primitive constructs (primitive characteristics).

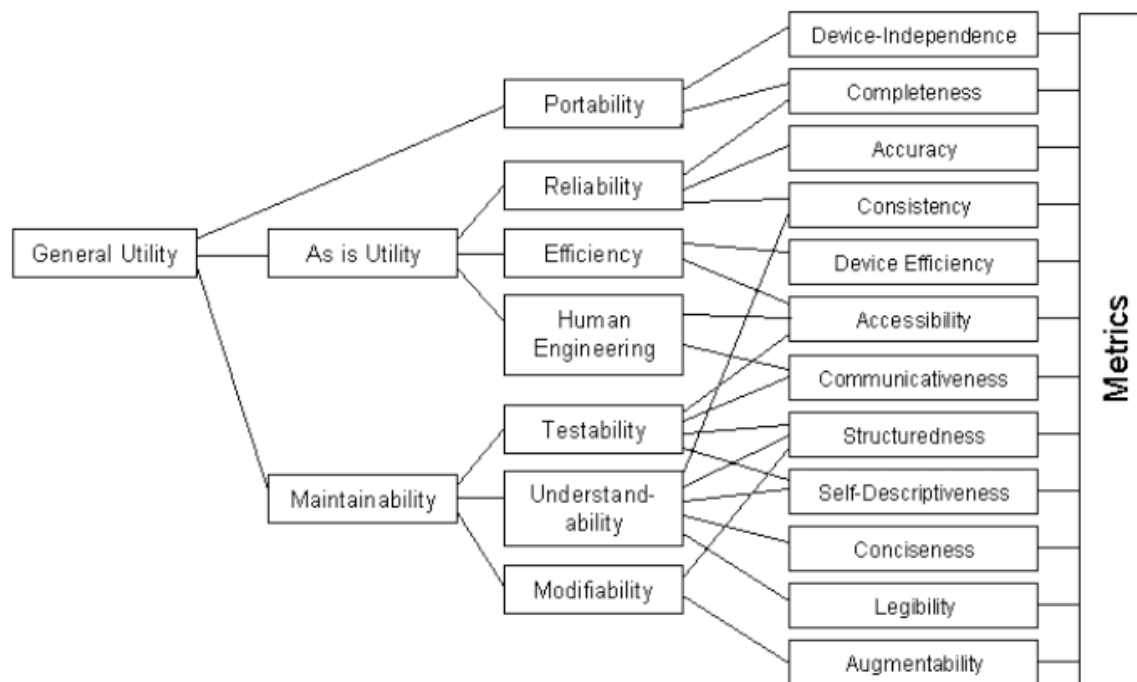
The highest level of Boehm's model has following three primary uses stated as below

1. **As is utility** –
Extent to which, we can use software as-is.
2. **Maintainability** –
Effort required detecting and fixing an error during maintenance.
3. **Portability** –
Effort required changing software to fit in a new environment.

The next level of Boehm's hierarchical model consists of seven quality factors associated with three primary uses, stated as below –

1. **Portability** –
Effort required to change software to fit in a new environment.
2. **Reliability** –
Extent to which software performs according to requirements.
3. **Efficiency** –
Amount of hardware resources and code required to execute a function.
4. **Usability (Human Engineering)** –
Extent of effort required to learn, operate and understand functions of the software.
5. **Testability** –
Effort needed for validating the modified software.
6. **Understandability** –
Effort required for a user to recognize logical concept and its applicability.
7. **Modifiability** –
Effort required modifying software during maintenance phase.

Boehm further classified characteristics into Primitive constructs as follows- device independence, accuracy, completeness, consistency, device efficiency, accessibility, communicativeness, self-descriptiveness, legibility, structuredness, conciseness, augment-ability. For example- Testability is broken down into: - accessibility, communicativeness, structuredness and self-descriptiveness.



Advantages:

- It focuses and tries to satisfy the needs of the user.
- It focuses on software maintenance cost effectiveness.

Disadvantages:

- It doesn't suggest how to measure the quality characteristics.
- It is difficult to evaluate the quality of software using the top-down approach.

ISO 9126 Quality Model

- 1. ISO stands for “International Organisation for Standardization”**
- 2. It is an organisation to verify the quality of a product during the development of the product.**
- 3. ISO 9001 specify the quality management system means if you want to check the quality of the software product then you should use ISO 9001.**
- 4. ISO implicitly use the TQM Technology to provide certificate for our organisation based on quality of our product.**

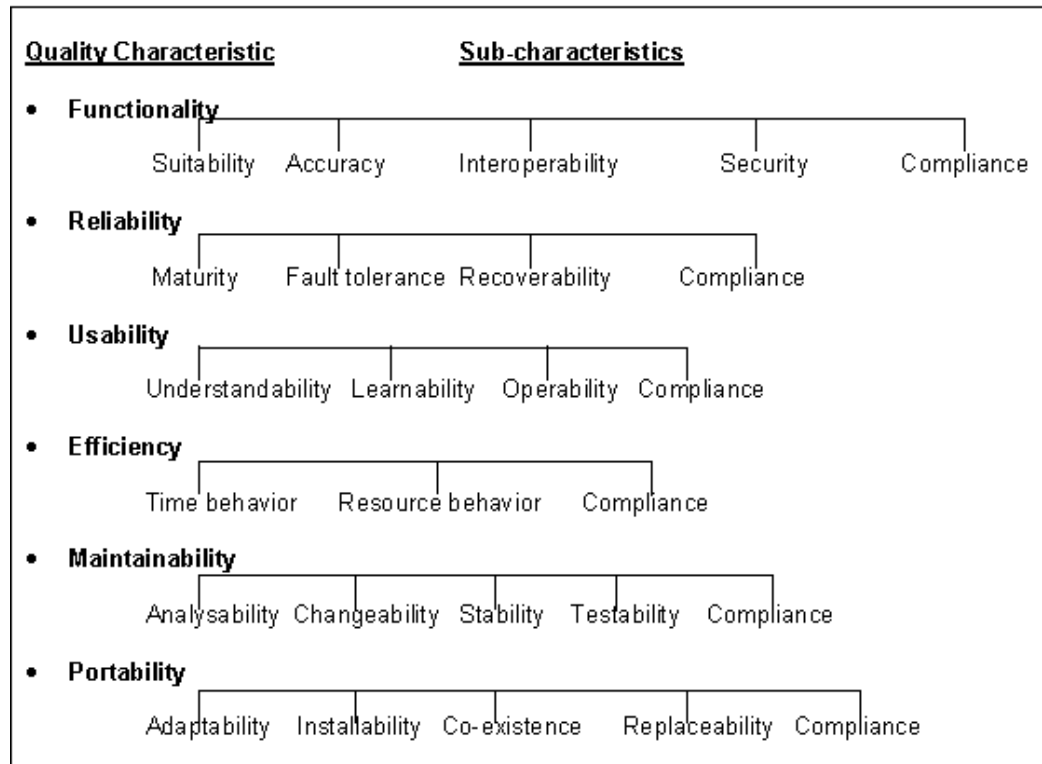
Why ISO Certificate?

- 1. If a company is ISO certified then the company has big name in the international market.**
- 2. The Company acquires the ISO certificate will change the internal structure mechanism just to implement the quality of services in the process but not finalize product.**
- 3. We can increase revenue and business because once we ISO certified then the customer starts believing on the organisation.**
- 4. ISO provides some benchmark and guidelines for every organisation those certified by the ISO.**
- 5. The progress of product can be easily measure as well as we can easily provide the report to top level management.**

ISO 9126: Hierarchical Model with Six Major Attributes contributing to Quality.

These attributes are:

- **Functionality**
- **Reliability**
- **Usability**
- **Efficiency**
- **Maintainability**
- **Portability**

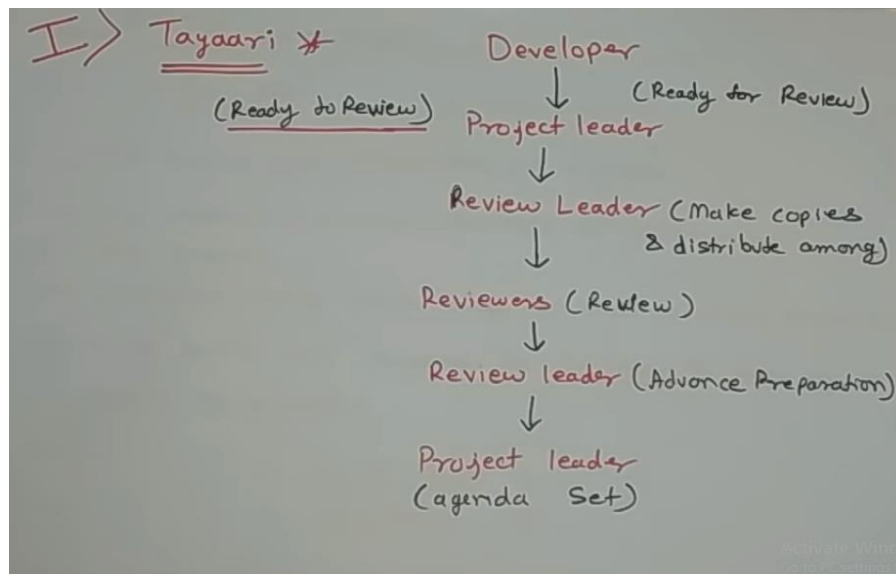


Formal Technical Review (FTR) in Software Engineering

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.

Objectives of formal technical review (FTR): Some of these are:

- Useful to uncover error in logic, function and implementation for any representation of the software.
- The purpose of FTR is to verify that the software meets specified requirements.
- To ensure that software is represented according to predefined standards.
- It helps to review the uniformity in software that is development in a uniform manner.
- To makes the project more manageable.



2. FTR Meetings:

- FTR begins with introduction of agenda.
- The Producer (Developer) proceed to walkthrough the product.
- Reviewer raised issue based on their advance preparation.
- Valid points are record by recorder and summary report is generated.
- At the end all attendees of FTR must decide
 - Accept the product without further modification.
 - Reject the product due to severs errors.
 - Accept the product provisionally.(with some modification)

Summary Report

- It Answer three question
- What is reviewed?
- Who is reviewed?
- What were the findings and conclusions?

Purpose of Summery Report

- Makes Errors identification easy
- Serves as a checklist for developer

The review meeting: Each review meeting should be held considering the following constraints- *Involvement of people:*

1. Between 3, 4 and 5 people should be involved in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Configuration Management

1. Configuration management is the set of policies, processes and tools which we use to manage the changes in software.
2. Why we need a special system to manage the changes in software?
3. Because when software is changing with time it can slip from its track.

Configuration management activities

1. Configuration management is a whole system which comprise on 4 activities i.e.
 1. Change management
 2. Version management
 3. System building
 4. Release management

Change management

1. In change management we keep track of changes requests coming from the customers or developers.
2. Change management also check the cost and impact of the changes.
3. It also decide if the change is really required? If so, when it is required.

Version Management

- It keep track of the version numbers of the software system it name is already defining it.

System building

1. This is the process of assembling and linking all the program components and libraries to make the executable system.
2. It is a complex process, this phase also communicate with the version control system to check the previous build and to decide what should be the next build.
3. Some time developers build a system just for there testing purpose before giving it to the client, it is important to keep track of that system is well.
4. It also keep check and balance of the **"timestamp"**.

Activate V
Go to PC sett



**Timestamp
in
system building**

Activate V
Go to PC sett

Release management

1. Release management keep track of the releases of the software to the customers.
2. It also keep track of the all other kind of releases.
3. There are two kind of release major release and minor release.
4. Like If the release number or let say version number is 8.4.1 it means minor release number is 1 and major release is 4 and the main software is 8.

Activate V
Go to PC sett

Analysis Concepts and Principles:

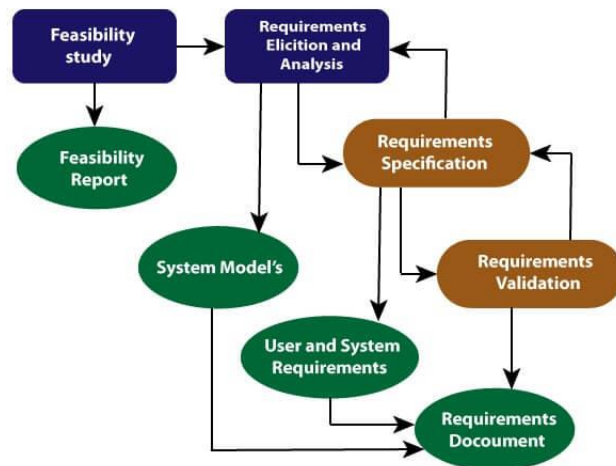
Requirement Engineering-

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, analysing 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 behaviour 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



Requirement Engineering Process

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

2. Requirement Elicitation and Analysis:

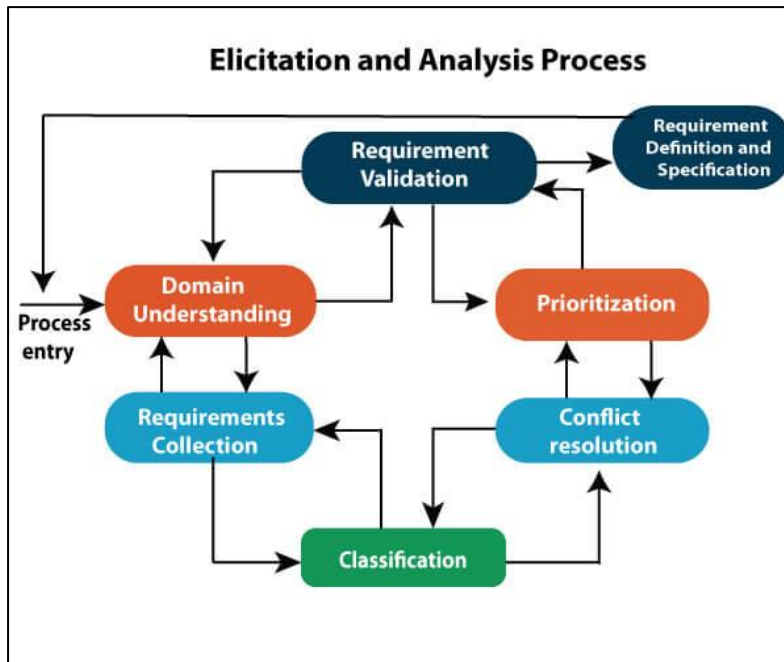
This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analysed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want

- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.



3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modelling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.

- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "*E-R diagram*." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be checked against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Software Requirement Management:

- Requirement management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.
- The priority of requirements from different viewpoints changes during development process.
- The business and technical environment of the system changes during the development.

Prerequisite/Condition of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

- Clear - The extent to which documents are clearly and accurate written.
- Correct - The extent to which a software meets its specifications.
- Consistent - The extent to which a software is consistent and give result with precision.
- Coherent - Coherence is defined as the quality of being logical, consistent and forming a unified whole.
- Comprehensible - The effort required for a user to recognize the logical concept and its applicability.
- Modifiable - The effort required to modify a software during maintenance phase.
- Verifiable - Software is verifiable if satisfaction of desired properties can be easily determined.
- Prioritized - Priority determines where a task ranks in order relative to all the other tasks that need to be completed
- Unambiguous - A Requirements Document is unambiguous if and only if every requirement stated therein has only one interpretation.
- Traceable - The extent to which an error is traceable in order to fix it.
- Credible source - credible sources that have been written by an expert/academic in a field relevant to your project and reviewed by other experts in the same field

Software Requirements: Largely software requirements must be categorized into two categories:

1. **Functional Requirements:** Requirements, which are related to functional/working aspects of software fall into this category. Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements are describing the behaviour of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** Requirements are expected characteristics of target software(Security,Storage,Configuration,Performance,Cost,Interoperability, Flexibility, Disaster Recovery, Accessibility). This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviours of the system.

Non-functional requirements are divided into two main categories:

- Execution qualities like security and usability, which are observable at run time.
- Evolution qualities like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

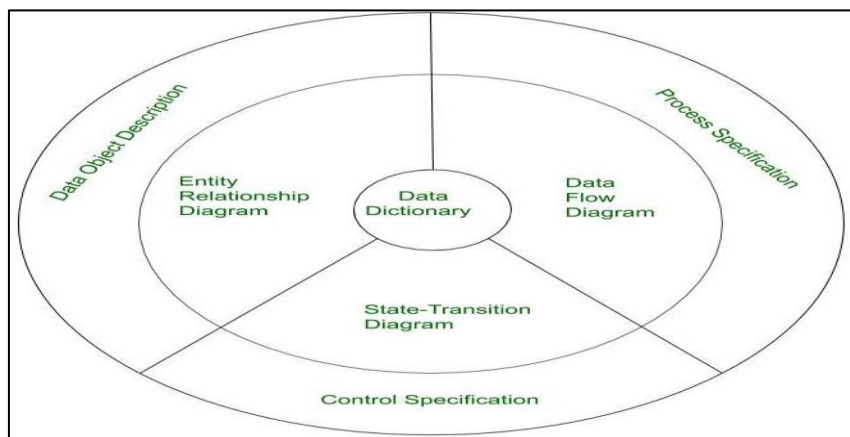
Analysis principles

Analysis Model is a technical representation of the system. It acts as a link between system description and design model. In Analysis Modelling, information, behaviour, and functions of the system are defined and translated into the architecture, component, and interface level design in the design modelling.

Objectives of Analysis Modelling:

- It must establish a way of creating software design.
- It must describe the requirements of the customer.
- It must define a set of requirements that can be validated, once the software is built.

Elements of Analysis Model:



1. Data Dictionary:

It is a repository that consists of a description of all data objects used or produced by the software. It stores the collection of data present in the software. It is a very crucial element of the analysis model. It acts as a centralized repository and also helps in modelling data objects defined during software requirements.

- A Structure place to keep details of the contents of data flow process and data store.
- It is a structured repository of data about data.
- It is a set of definition of all DFD elements.

Advantages

- A. Documentation- It is a valuable reference in any organisation.
- B. Improve Analyst/ User Communication.
- C. It is an important step in building a Database.

Items to be defined in Data Dictionary

- A. Data Elements- Smallest unit of data that provide for no further decomposition. Eg. Data Consist of Day, Month or year.
- B. Data Structure- Group of Data Element Handled as an unit. Ex- Phone is a Data Structure Consisting of four data element. Area- Code- Exchange no- Extensions.
- C. Data Flows and Data Stores- whenever Data Structure is in an motion it is called Data Flow and whenever it is an rest mode it is called Data Store.

2. Entity Relationship Diagram (ERD):

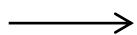
It depicts the relationship between data objects and is used in conducting data modelling activities. The attributes of each object in the Entity-Relationship Diagram can be described using Data object description. It provides the basis for activity related to data design.

3. Data Flow Diagram (DFD):

It depicts the functions that transform data flow and it also shows how data is transformed when moving from input to output. It provides the additional information which is used during the analysis of the information domain and serves as a basis for the modelling of function. It also enables the engineer to develop models of functional and information domains at the same time.

- a. A graphical tool, useful for communicating with users, managers and other personnel.
- b. Useful for analysing existing as well as proposed system.
- c. Focus on the movement of the data between external entities and processes, and between processes and data store.
- d. A relatively Simple technique to learn and use.

DFD Elements

- a. **Source/Sinks (External entities)** – Entity that supplies data to the System.
 - Sink- Entity that receives data from the system.
- b. **Data Flows** – Marks movement of data through the system – a pipeline to carry data. 
 - Connects the processes, external entities, and data stores.
 - Generally unidirectional, if same data flows in both directions, double-headed arrow can be used.


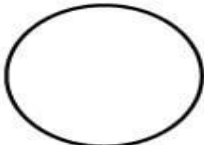
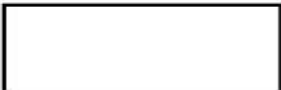
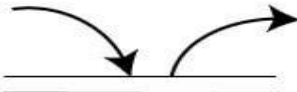
c. **Processes-** A circle (bubble) shows a process that transforms data inputs into data outputs.

- Straight line with incoming arrows is input data flow.
- Straight line with outgoing arrows is output data flow.
- Labels are assigned to data flow.

d. **Data Stores/ Database** - 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.

- A data store is a repository of a data.
- Data can be written into the data store. This is depicted by incoming arrows.
- Data can be read from a data stores. This is depicted by outgoing arrows.
- External entities cannot read or write to the data store.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Rules of data flow

1. Data can flow from

- External entity to process.
- Process to external entity.
- Process to store and back.

- **Process to process.**

2. Data cannot flow from

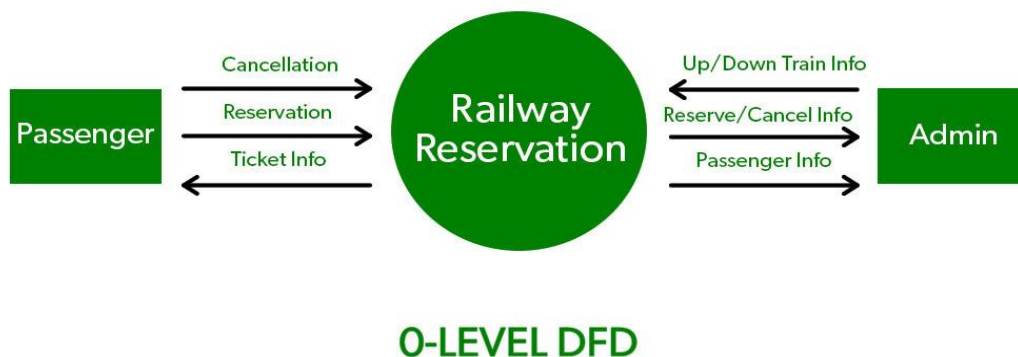
- External Entity to External Entity.
- External Entity to store.
- Store to External Entity.
- Store to Store.

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.

1. 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.



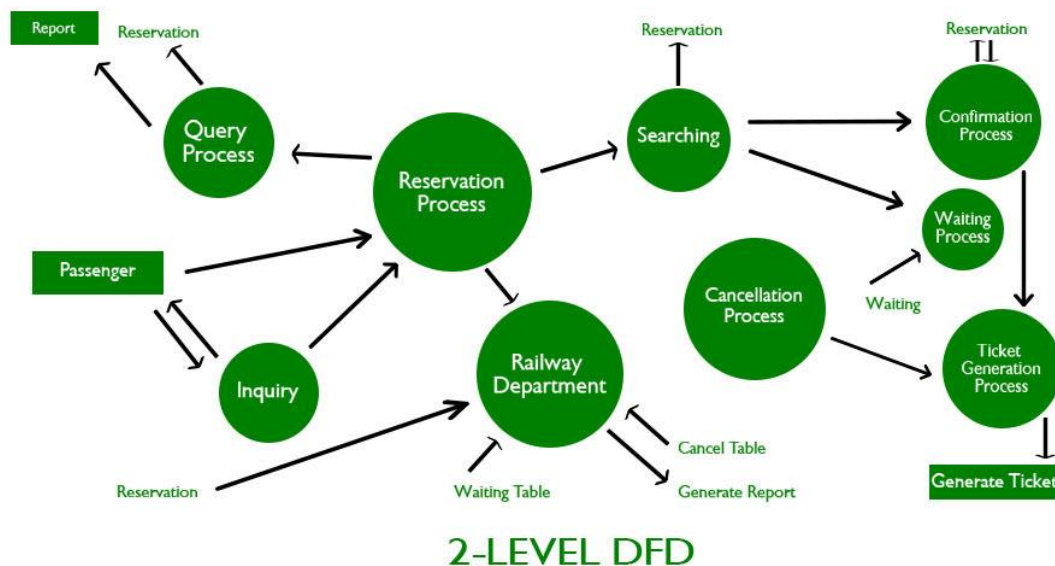
2. 1-level DFD:

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 sub processes.



3. 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. State Transition Diagram:

It shows various modes of behaviour (states) of the system and also shows the transitions from one state to another state in the system. It also provides the details of how the system behaves due to the consequences of external events. It represents the behaviour of a system by presenting its states and the events that cause the system to change state. It also describes what actions are taken due to the occurrence of a particular event.

5. Process Specification:

It stores the description of each function present in the data flow diagram. It describes the input to a function, the algorithm that is applied for the transformation of input, and the output that is produced. It also shows regulations and barriers imposed on the performance characteristics that are applicable to the process and layout constraints that could influence the way in which the process will be implemented.

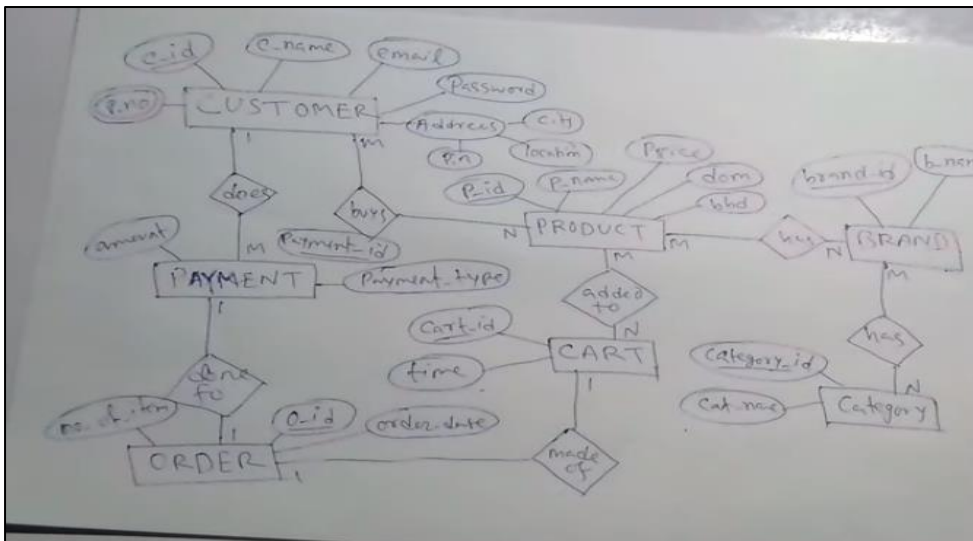
6. Control Specification:

It stores additional information about the control aspects of the software. It is used to indicate how the software behaves when an event occurs and which processes are invoked due to the occurrence of the event. It also provides the details of the processes which are executed to manage events.

7. Data Object Description:

It stores and provides complete knowledge about a data object present and used in the software. It also gives us the details of attributes of the data object present in the Entity Relationship Diagram. Hence, it incorporates all the data objects and their attributes.

ER Diagram



Entity-Relationship Diagrams

ER-modelling is a data modelling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modelling 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.

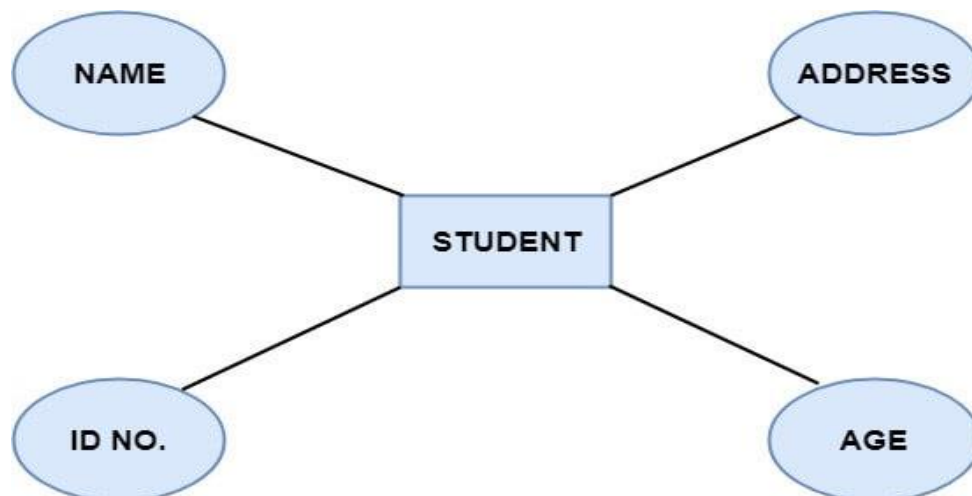
Entity Set

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

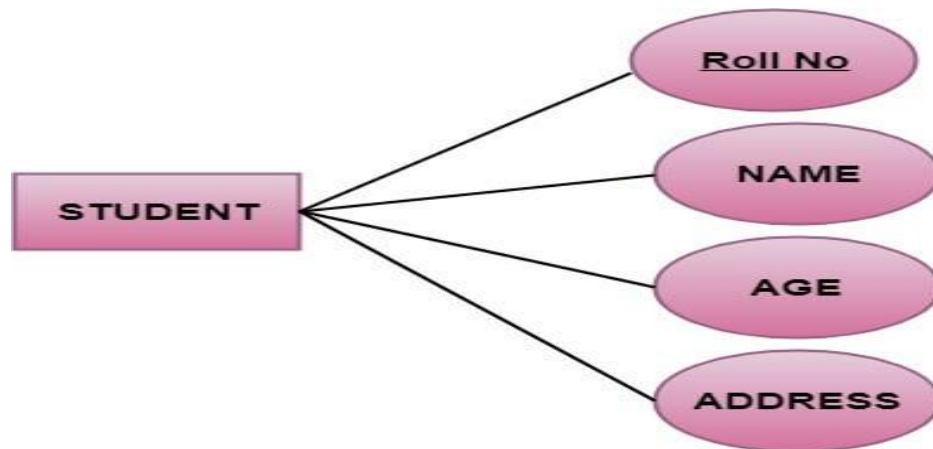
- Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.
- There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.



There are four types of Attributes:

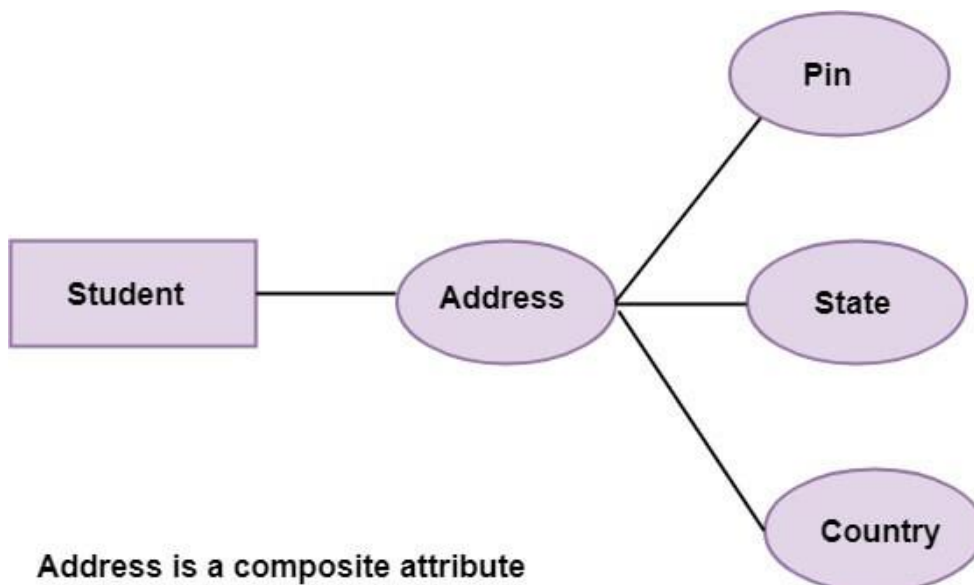
1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

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.



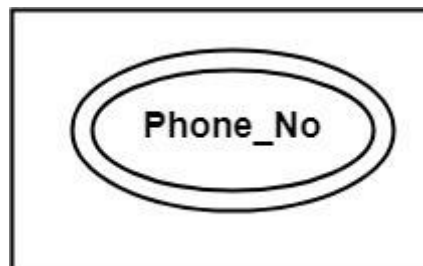
There are mainly three types of keys:

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.

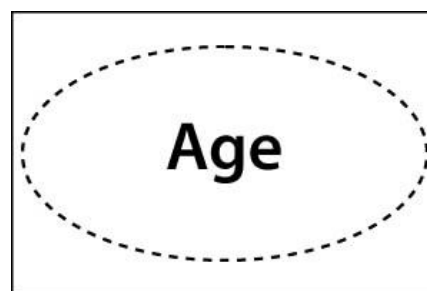


3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social_Security_Number.

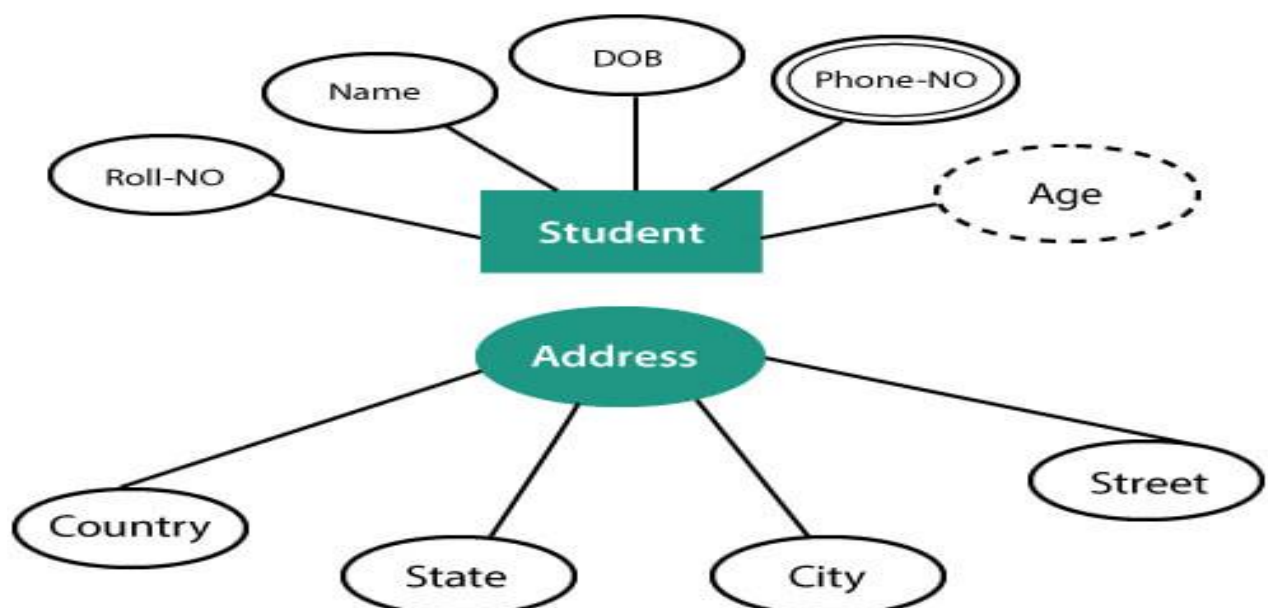
4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.



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.



The Complete entity type Student with its attributes can be represented as:



3. 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.

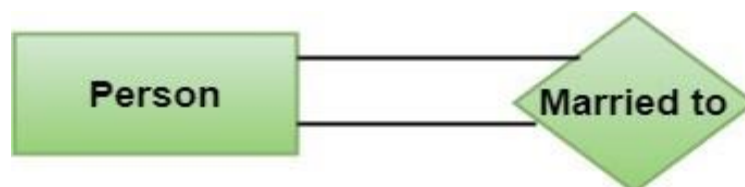


Fig: Unary Relationship

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.

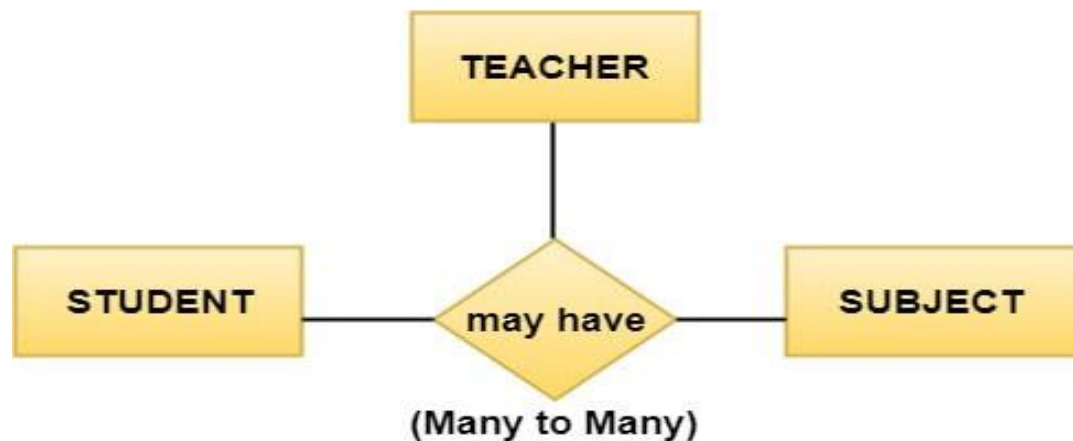


Fig: Ternary Relationship

Cardinality

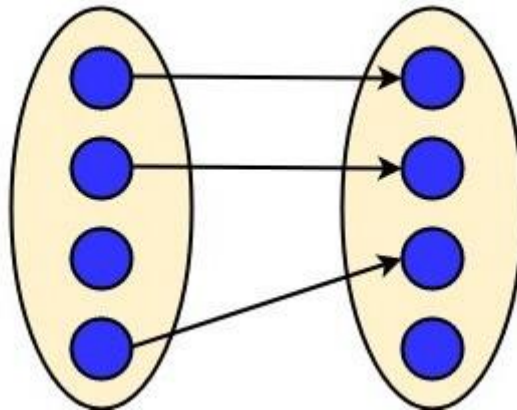
Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

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.



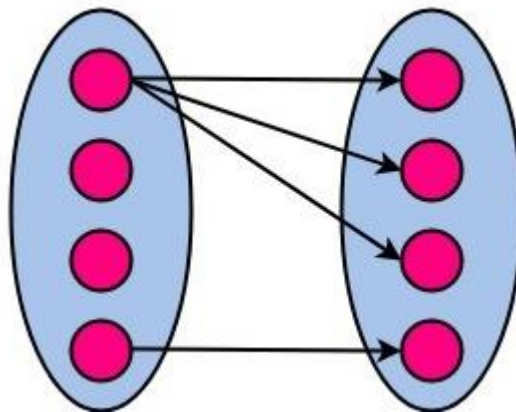
Using Sets, it can be represented as:



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.



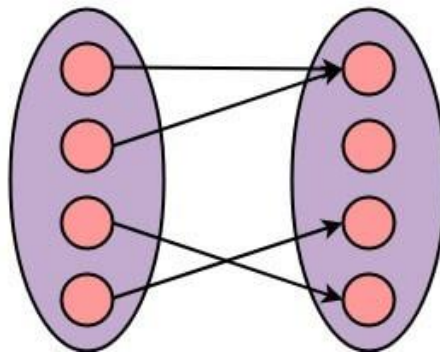
Using Sets, it can be represented as:



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.



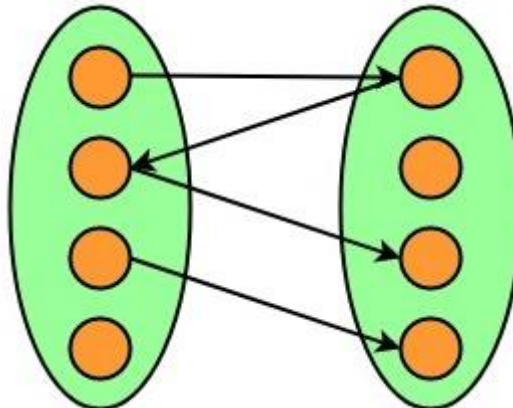
Using Sets, it can be represented as:



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:



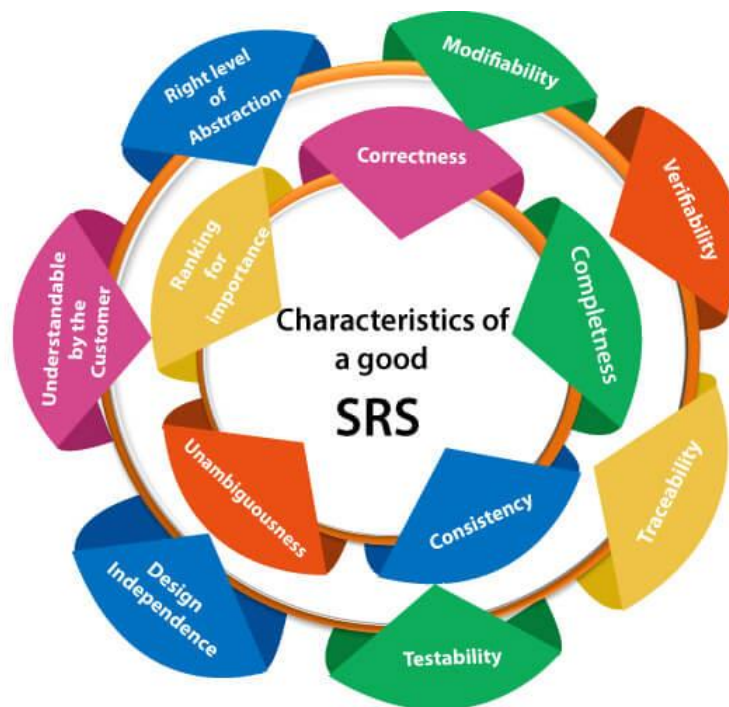
Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructed when entire requirements are elicited and analysed. **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.

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.

Characteristics of good SRS



Following are the features of a good SRS document:

1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

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

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

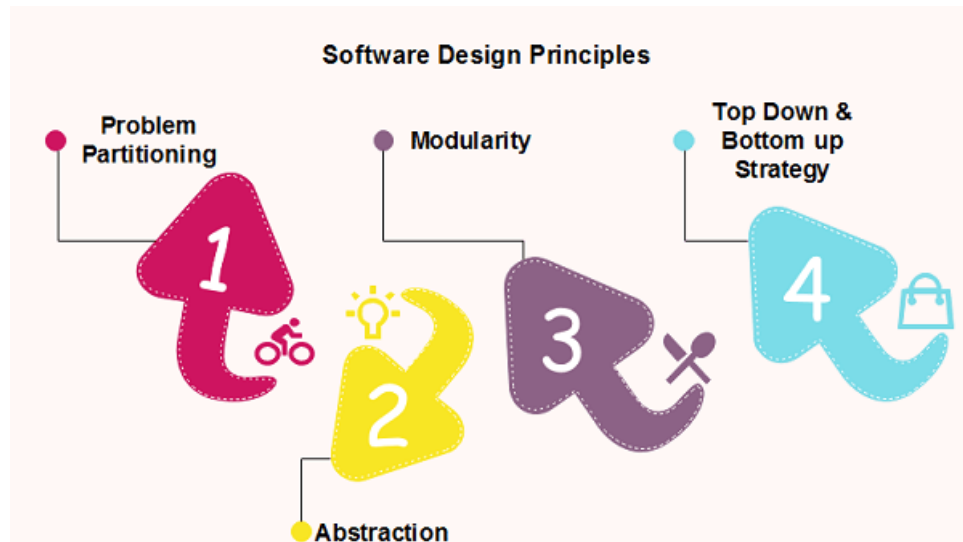
Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Software Design Principles

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design



Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

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. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.
- Modules are simpler from outside than inside.

Advantages and Disadvantages 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 simplifies the overlay procedure of loading a large program into main storage.
- 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.