

Module 3

Software Requirements

Requirements Engineering

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. It mainly focus on 'What a System must do' and not 'how'. Requirements engineering consists one large document contains a description of what a system without describing how it will do

Requirements engineering consists of 4 steps

1. Requirement elicitation
2. Requirement analysis
3. Requirement documentation
4. Requirement review

Requirement elicitation

Requirements elicitation is the process of researching and discovering the requirements of a system from users, customers, and other stakeholders. Before requirements can be analyzed, they must be gathered through an elicitation process. It is also called gathering information

Requirement analysis

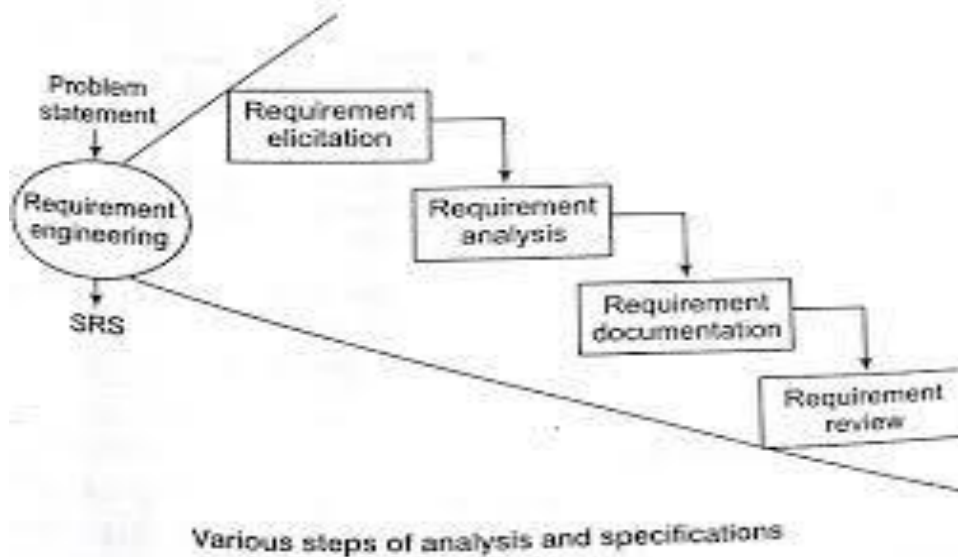
Analysis of requirements starts with requirement elicitation. Requirements are analysed in order to identify inconsistency, defects and omissions.

Requirement documentation

It is the end product of the elicitation and analysis. It is the foundation of the design phase. The document prepared in this step is called SRS (software requirement specification)

Requirement review

The review process is carried out to improve the quality of the SRS. It is also called requirement verification



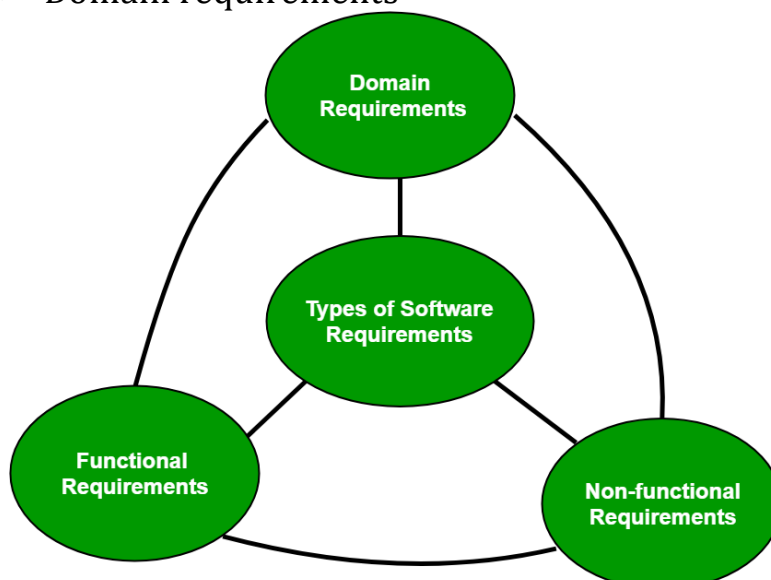
Types of requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

The known and unknown requirements may be functional or non-functional

A software requirement can be of 3 types:

- Functional requirements
- Non-functional requirements
- Domain requirements



Functional Requirements:

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.

Non-functional requirements:

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other.

They are also called non-behavioral requirements. They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

NFR's are classified into following types:

- Interface constraints
- Performance constraints: response time, security, storage space, etc.
- Operating constraints
- Life cycle constraints: maintainability, portability, etc.
- Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate.

Domain requirements:

Domain requirements are the requirements which are characteristic of a particular category or domain of projects. The basic functions that a system of a specific domain must necessarily exhibit come under this category.

Feasibility Studies

Feasibility Study in Software Engineering is a study to evaluate feasibility of proposed project or system. Feasibility study is one of stage among important four stages of Software Project Management Process. As name suggests feasibility study is the feasibility analysis or it is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view. Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc.

Types of Feasibility Study :

1. Technical Feasibility –

In Technical Feasibility current resources both hardware software along with required technology are analyzed/assessed to develop project. This technical feasibility study gives report whether there exists correct required resources and technologies which will be used for project development. Along with this, feasibility study also analyzes technical skills and capabilities of technical team, existing technology can be used or not, maintenance and up-gradation is easy or not for chosen technology etc.

2. Operational Feasibility –

In Operational Feasibility degree of providing service to requirements is analyzed along with how much easy product will be to operate and maintenance after deployment.

3. Economic Feasibility –

In Economic Feasibility study cost and benefit of the project is analyzed. Under this feasibility study a detail analysis is carried out what will be cost of the project for development which includes all required cost for final development like hardware and software resource required, design and development cost and operational cost and so on. ot.

4. Legal Feasibility –

In Legal Feasibility study project is analyzed in legality point of view. This includes analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc..

5. Schedule Feasibility –

In Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project

Requirements Engineering

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. It mainly focus on 'What a System must do' and not 'how'. Requirements engineering consists one large document contains a description of what a system without describing how it will do

Requirements engineering consists of 4 steps

1. Requirement elicitation
2. Requirement analysis
3. Requirement documentation
4. Requirement review

I. Requirement elicitation

Requirements elicitation is perhaps the most difficult, most error-prone and most communication intensive software development. It can be successful only through an effective customer-developer partnership. It is needed to know what the users really need.

Requirements elicitation techniques

1. Interview

Objective of conducting an interview is to understand the customer's expectations from the software. It is impossible to interview every stakeholder(all the responsible person) hence representatives from groups are selected based on their expertise and credibility.

Interviews maybe be open-ended or structured.

1. In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
2. In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

2. Brainstorming Sessions:

- It is a group technique
- It is intended to generate lots of new ideas hence providing a platform to share views

- A highly trained facilitator is required to handle group bias and group conflicts.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3. **Facilitated Application Specification Technique: (FAST)**

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4. **Quality Function Deployment: (QFD)**

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified –

- **Normal requirements –**
In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc
- **Expected requirements –**
These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- **Exciting requirements –**
It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

5. Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements. The use cases describe the 'what', of a system and not 'how'. Hence, they only give a functional view of the system. The component of the use case design includes three major things – Actor, Use cases, and use case diagram.

1. Actor

It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.

- Primary actors – It requires assistance from the system to achieve a goal.
- Secondary actor – It is an actor from which the system needs assistance.

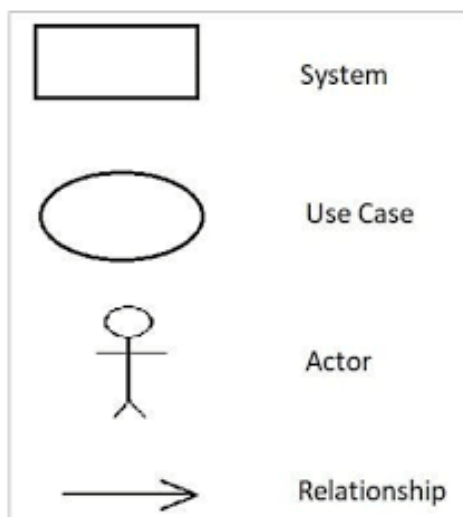
2. Use cases –

They describe the sequence of interactions between actors and the system.

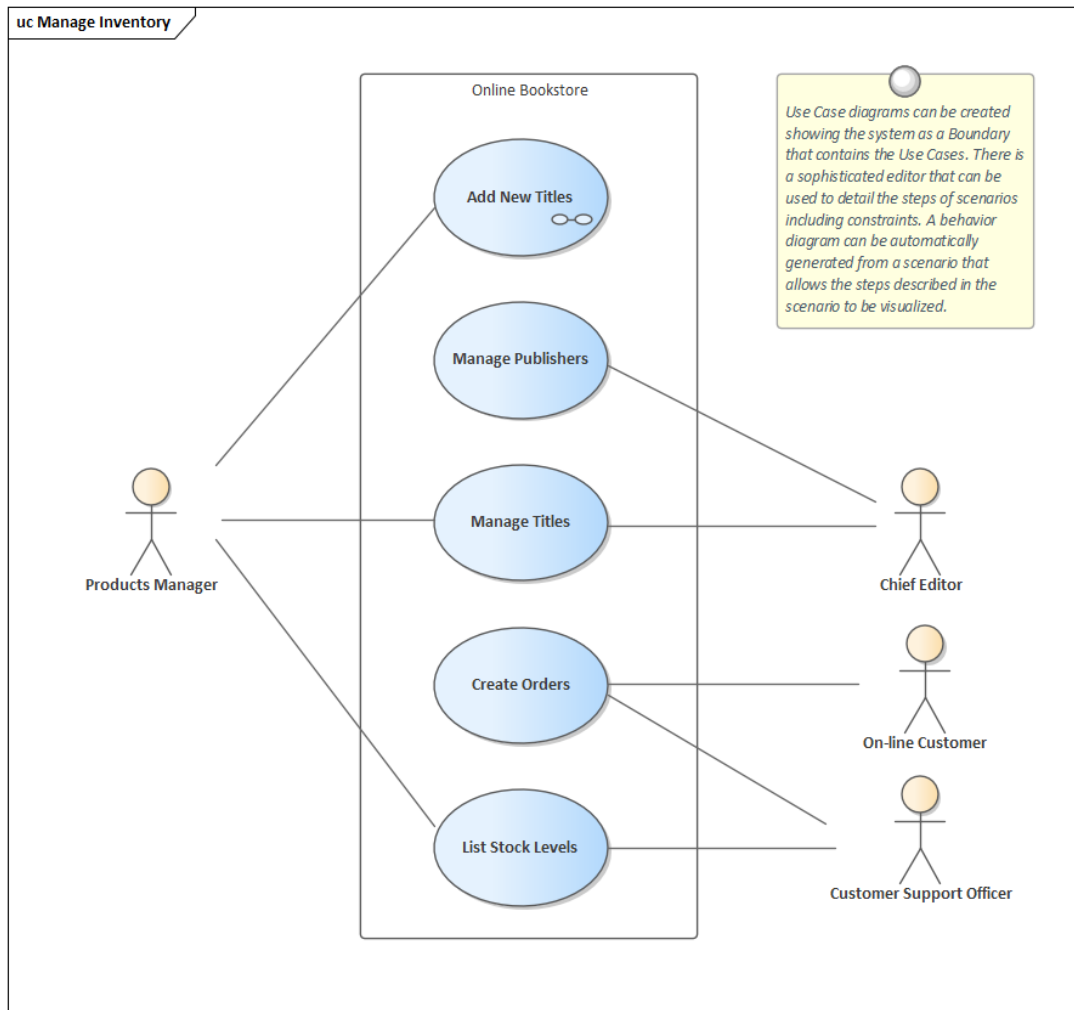
3. Use case diagram –

A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.

Symbols of use case diagram



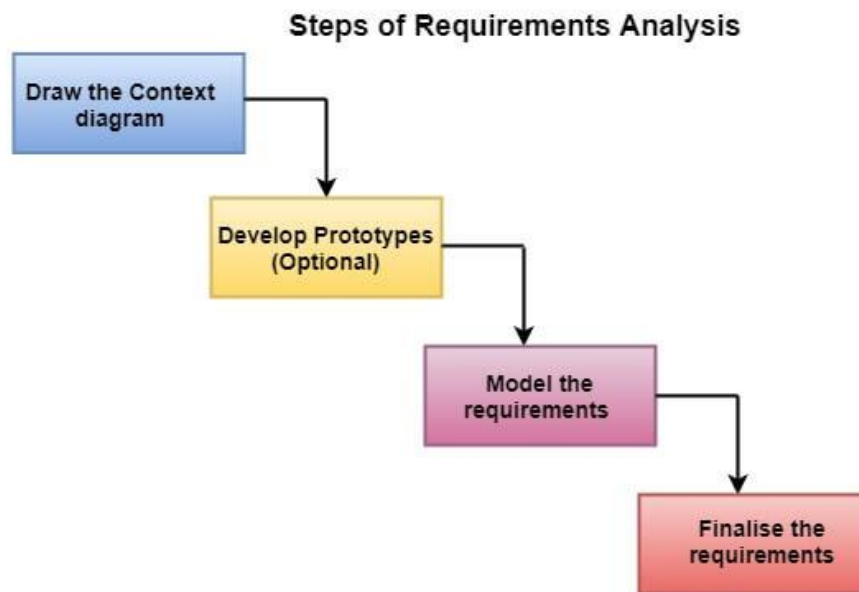
Example



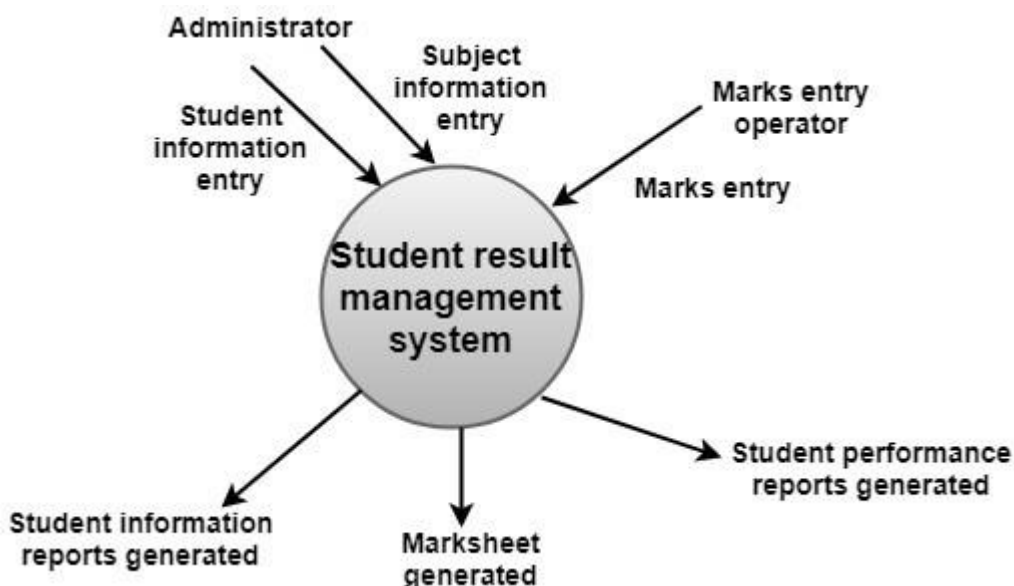
II. Requirements Analysis

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.

The various steps of requirement analysis are shown in fig:



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



(ii) **Development of a Prototype (optional):** One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.


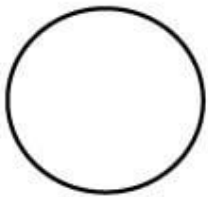
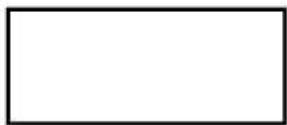
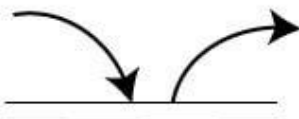
We can use their feedback to modify the prototype until the customer is satisfied continuously.

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

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 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. The DFD is also called as a data flow graph or bubble chart.

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the 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

Draw the example from your note book

2. Entity-Relationship diagram

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes.

Entity

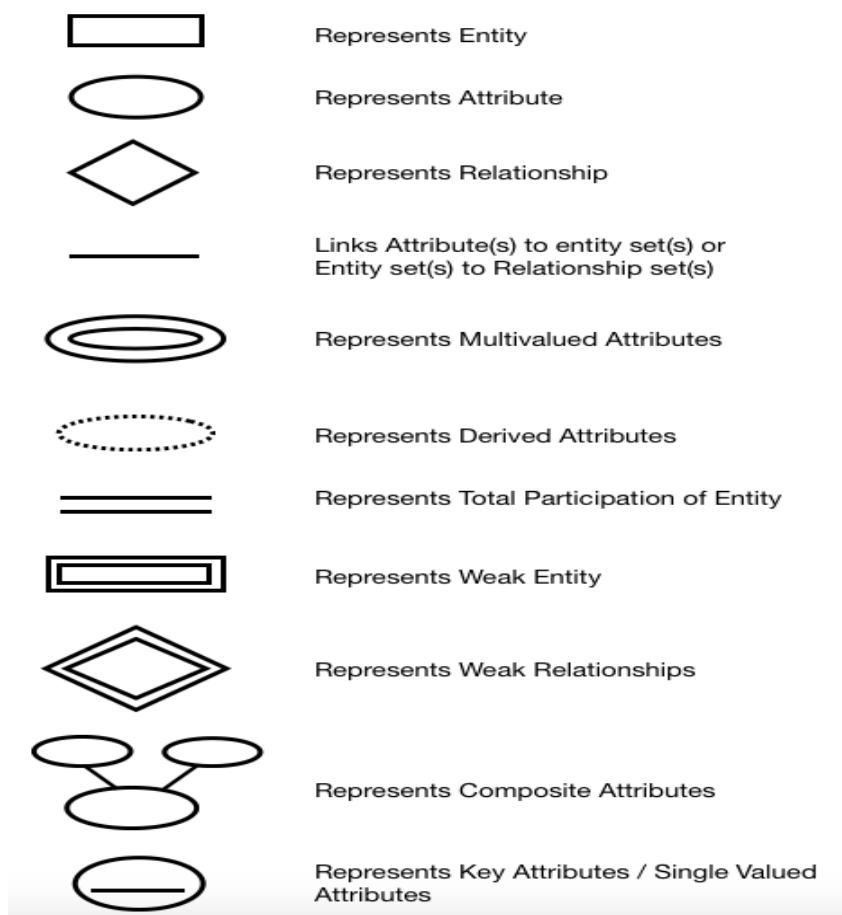
An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

Attributes

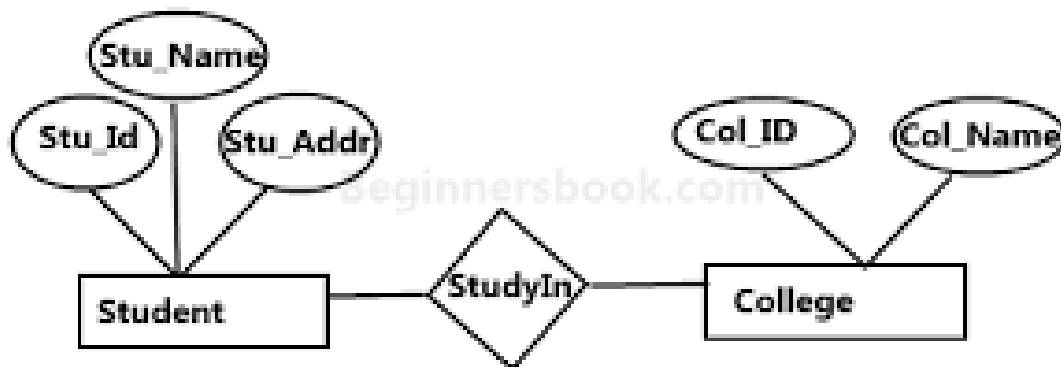
Each entity has attributes. An attribute defines set of properties that are used to describe an entity. For example, an employee entity has attributes like employeeid, employee_name, designation, salary etc.,

Relationship

A relationship is an association between several entities..



Examples of ER Diagrams



Sample E-R Diagram

3. Data dictionaries

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. 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.

III. Requirement documentation

It is the activity after requirement elicitation and analysis. This is the way to represent requirements in a consistent format. The requirement documentation is called SRS (Software requirement specification). SRS is a document created by system analyst after the requirements are collected

Nature of Software Requirement Specification (SRS):

The basic issues that SRS writer shall address are the following:

1. Functionality

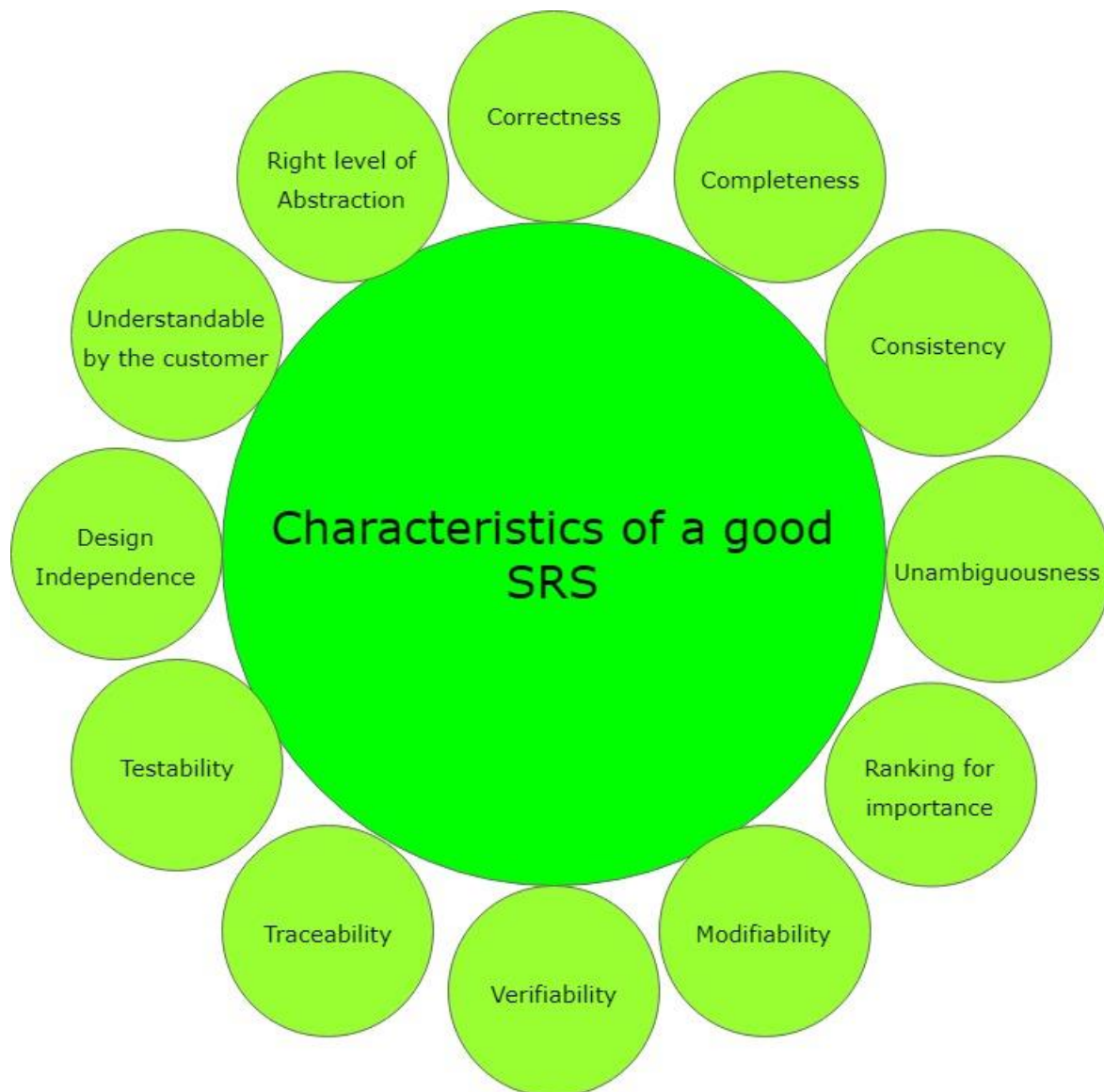
2. External Interfaces:

3. Performance: What is the speed, availability, response time, recovery time etc.

4. Attributes: What are the considerations for portability, correctness, maintainability, security, reliability etc.

5. Design Constraints Imposed on an Implementation: Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment etc.

Characteristics of a good SRS



Following are the characteristics of a good SRS document:

1. **Correctness:**

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 etc.

4. Unambiguousness:

A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation.

5. Ranking for importance and stability:

There should be 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

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.

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.

Requirement validation

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation.

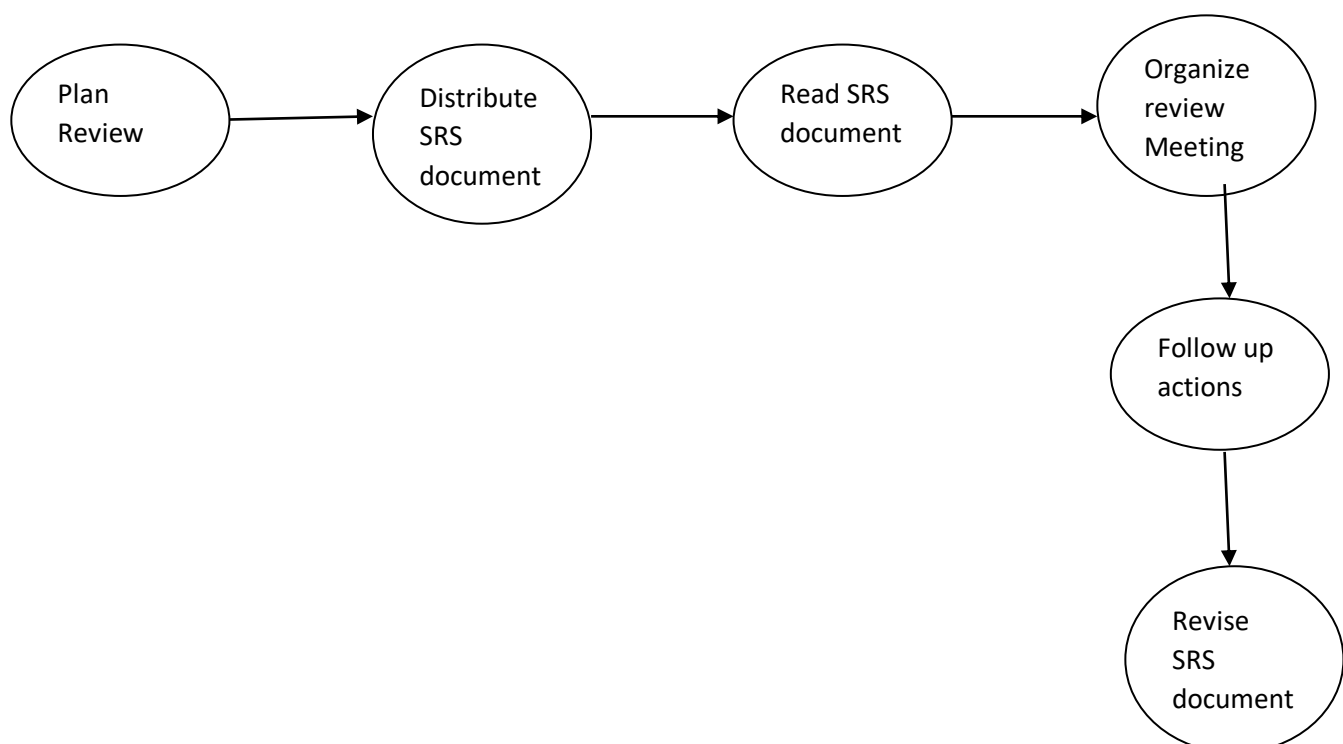


Requirements Validation

- The requirements document should be formulated and organized according to the standards of the organization.
- The **organizational knowledge** is used to estimate the realism of the requirements of the system.
- The **organizational standards** are specified standards followed by the organization according to which the system is to be developed.
- The **lists of problems** indicate the problems encountered in the requirements document of the requirements validation process.
- The **agreed actions** is a list that displays the actions to be performed to resolve the problems depicted in the problem list.

IV. Requirement review

In this process a group of people will read the SRS document and look for possible errors



Plan review:- the review team is selected and the time and place for the meeting is fixed

Distribute SRS document:-The SRS document is distributed to all the members

Read the SRS document:-each member read the document carefully to find the omissions, inconsistencies, deviations etc

Organise the review meeting:-each member present his/her views and identified problems

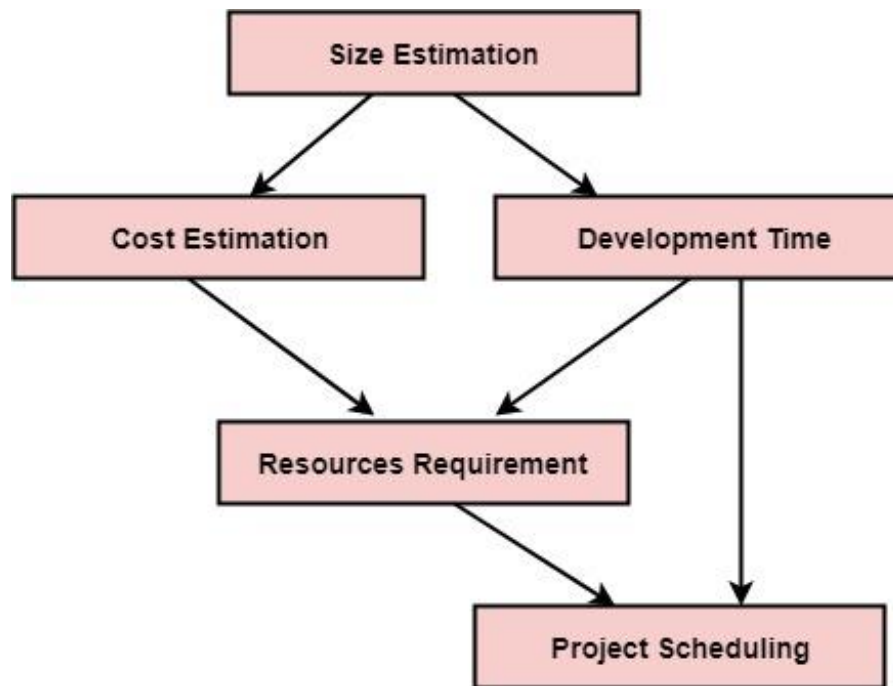
Follow-up-actions:-corresponding actions regarding the problems are identified

Revise the SRS document:- the SRS document is revised to reflect the changes

Project Planning

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.**

Software Project planning starts before technical work start. The various steps of planning activities are:



The size is the crucial parameter for the estimation of other activities. Resources requirement are required based on cost and development time.

Project size estimation

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

1. **Lines of Code (LOC)**: As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- **KLOC**- Thousand lines of code
- **NLOC**- Non comment lines of code
- **KDSI**- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer's perspective.
- Simple to use.

2. Number of entities in ER diagram: The number of entities in ER model can be used to measure the estimation of size of project. Number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages:

- Size estimation can be done during initial stages of planning.
- Number of entities is independent of programming technologies used.

3. Total number of processes in detailed data flow diagram: Data Flow Diagram (DFD) represents the functional view of a software. Utilization of number of functions in DFD is used to predict software size.

Advantages:

- It is independent of programming language.
- Each major processes can be decomposed into smaller processes. This will increase the accuracy of estimation

4. Function Point Analysis

Function Point Analysis was initially developed by Allan J. Albercht in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA provides standardized method to functionally size the software work product.

Types of FPA:**1. Transactional Functional Type –**

- **(i) External Input (EI):** EI processes data or control information that comes from outside the application's boundary. The EI is an elementary process.
- **(ii) External Output (EO):** EO is an elementary process that generates data or control information sent outside the application's boundary.
- **(iii) External Inquiries (EQ):** EQ is an elementary process made up of an input-output combination that results in data retrieval.

2. Data Functional Type –

- **(i) Internal Logical File (ILF):** A user identifiable group of logically related data or control information maintained within the boundary of the application.
- **(ii) External Interface File (EIF):** A group of user recognizable logically related data allusion to the software but maintained within the boundary of another software.

Counting Function Point (FP):

- **Step-1:**
 $F = 14 * \text{scale}$
 Scale varies from 0 to 5 according to character of Complexity Adjustment Factor (CAF). Below table shows scale:
 0 - No Influence
 1 - Incidental
 2 - Moderate
 3 - Average
 4 - Significant
 5 - Essential
- **Step-2:** Calculate Complexity Adjustment Factor (CAF).
 $CAF = 0.65 + (0.01 * F)$
- **Step-3:** Calculate Unadjusted Function Point (UFP).
 TABLE (Required)

Function Units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Multiply each individual function point to corresponding values in TABLE.

- **Step-4:** Calculate Function Point.

$$FP = UFP * CAF$$

Example:

Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are average.

User Input = 50

User Output = 40

User Inquiries = 35

User Files = 6

External Interface = 4

Explanation:

- **Step-1:** As complexity adjustment factor is average (given in question), hence,
- scale = 3.

$$F = 14 * 3 = 42$$

- **Step-2:**

$$CAF = 0.65 + (0.01 * 42) = 1.07$$

- **Step-3:** As weighting factors are also average (given in question) hence we will multiply each individual function point to corresponding values in TABLE.

$$UFP = (50*4) + (40*5) + (35*4) + (6*10) + (4*7) = 628$$

- **Step-4:**

$$\text{Function Point} = 628 * 1.07 = 671.96$$

This is the required answer.

Cost estimation

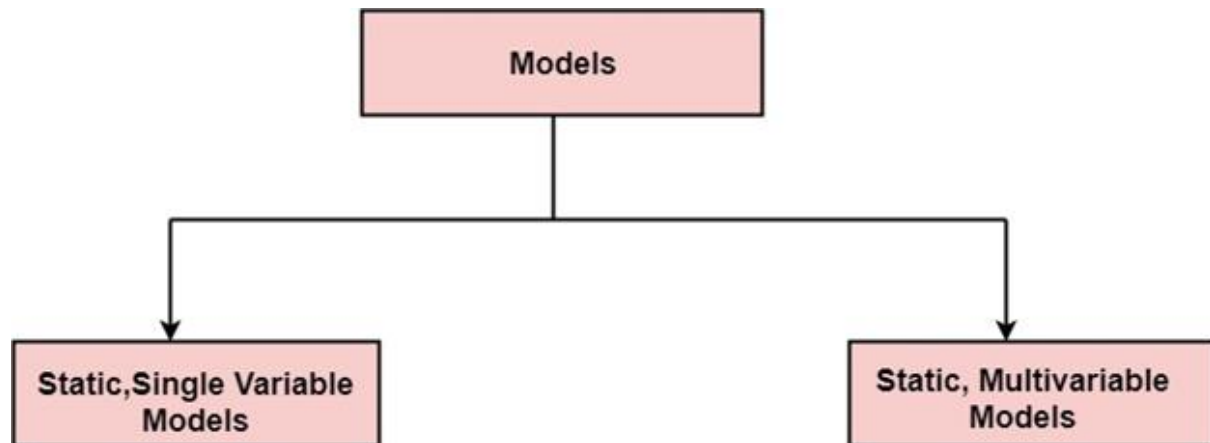
For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take

uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variable are interdependent, and there is no basic variable



Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

Where C = Costs

L= size

a and b are constants

Static, Multivariable Models: In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

$$E=5.2L^{0.91}$$

In the same manner duration of development is given by

$$D=4.1L^{0.36}$$

COCOMO Model

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981. COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

In COCOMO, projects are categorized into three types:

1. Organic
 2. Semidetached
 3. Embedded
1. **Organic** – A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
 2. **Semi-detached** – A software project is said to be a Semi-detached type if the vital characteristics such as team-size, experience, knowledge of the various programming environment lie in between that of organic and Embedded.
 3. **Embedded** – A software project with requiring the highest level of complexity, creativity, and experience requirement fall under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

All the above system types utilize different values of the constants used in Effort Calculations.

Types of Models: COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

Estimation of Effort: Calculations -

I. Basic Model -

$$E = a(KLOC)^b$$

$$time = c(Effort)^d$$

$$Personrequired = Effort/time$$

Productivity (P)=KLOC/E

E→ Effort applied in person month

T→ Duration or development time in months

a, b, c and d are co-efficient

Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example:

Explain the levels of cocomo model assume that the size of an organic software product has been estimated to be 3200 lines of code. Determine the effort required to develop the software product and nominal development time. Also find out the average staff size and productivity

$$E = a(KLOC)^b \quad T = c(E)^d$$

Organic

$$E = 2.4(3200)^{1.05}$$

$$= 11497.905 \text{ pm (person month)}$$

$$T = 2.5(11497.905)^{.38}$$

=87.292 m

Average staff size= $E/D = 11497.905/87.292$
 =131.77 persons

Productivity=KLOC/E = $3200/11497.905 = .278$ loc/pm

II. **Intermediate Model -**

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system's effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes:

(i) Product attributes -

- Required software reliability extent
- Size of the application database
- The complexity of the product

(ii) Hardware attributes -

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

(iii) Personnel attributes -

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

(iv) Project attributes -

- Use of software tools
- Application of software engineering methods
- Required development schedule

The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table.

These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor).

The Intermediate COCOMO formula now takes the form:

$$E = a(KLOC)^b * EAF$$

EAF is the effort adjustment factor

$$D = c(E)^d$$

Software Projects	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi Detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

III. Detailed Model -

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System design
3. Detailed design
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.