Unit 2 Notes

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.

Need of SRS

An SRS forms the basis of an organization's entire project. It sets out the framework that all the development teams will follow. It provides critical information to all the teams, including development, operations, quality assurance (QA) and maintenance, ensuring the teams are in agreement.

Characteristics of good SRS

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.

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

(3). Two or more requirements may define the same real-world object but use different terms for that object.

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.

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.

9. 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.

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. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?

Non-Functional Requirements

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

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

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Analysis Model

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

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 acts as a centralized repository and also helps in modeling data objects defined during software requirements.
   Data Dictionary is the major component in the structured analysis model of the system. It lists all the data items appearing in DFD. A data dictionary in Software Engineering means a file or a set of files that includes a database's metadata (hold records about other objects in the database), like data ownership, relationships of the data to another object, and some other data.

Components of Data Dictionary:

In Software Engineering, the data dictionary contains the following information:

- **Name of the item:** It can be your choice.
- **Aliases:** It represents another name.
- **Description:** Description of what the actual text is all about.
- **Related data items:** with other data items.
- **Range of values:** It will represent all possible answers.

2. **Entity Relationship Diagram (ERD):**

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

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

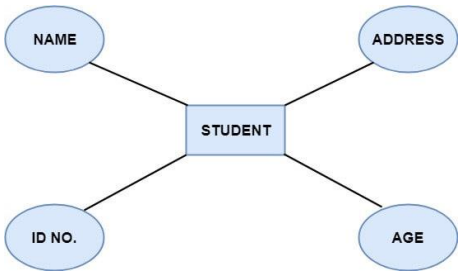
database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

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

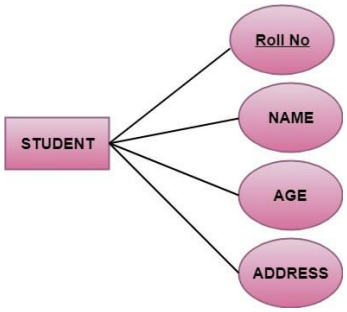
Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.



There are four types of Attributes:

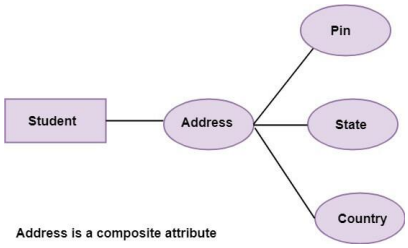
1. Key attribute
2. Composite attribute
3. Single-valued attribute
4. Multi-valued attribute
5. Derived attribute

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.



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

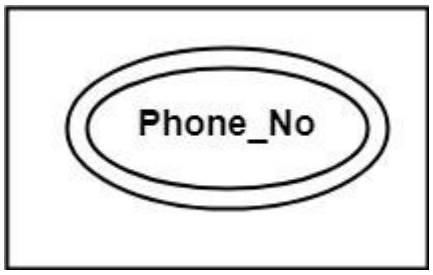
attribute as an address is composed of other characteristics such as pin code, state, country.



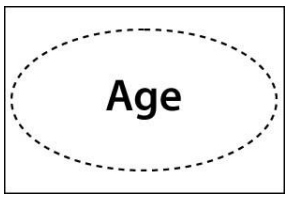
Address is a composite attribute

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.



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

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.

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.

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.

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.

Data Flow Diagram (DFD):

DFD is the abbreviation for **Data Flow Diagram**. The flow of data of a system or a process is represented by DFD. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present.

Rules for creating DFD

- The name of the entity should be easy and understandable without any extra assistance(like comments).
- The processes should be numbered or put in ordered list to be referred easily.
- The DFD should maintain consistency across all the DFD levels.
- A single DFD can have a maximum of nine processes and a minimum of three processes.

Symbols Used in DFD

- **Square Box:** A square box defines source or destination of the system. It is also called entity. It is represented by rectangle.
- **Arrow or Line:** An arrow identifies the data flow i.e. it gives information to the data that is in motion.
- **Circle or bubble chart:** It represents as a process that gives us information. It is also called processing box.
- **Open Rectangle:** An open rectangle is a data store. In this data is store either temporary or permanently.

Levels of DFD

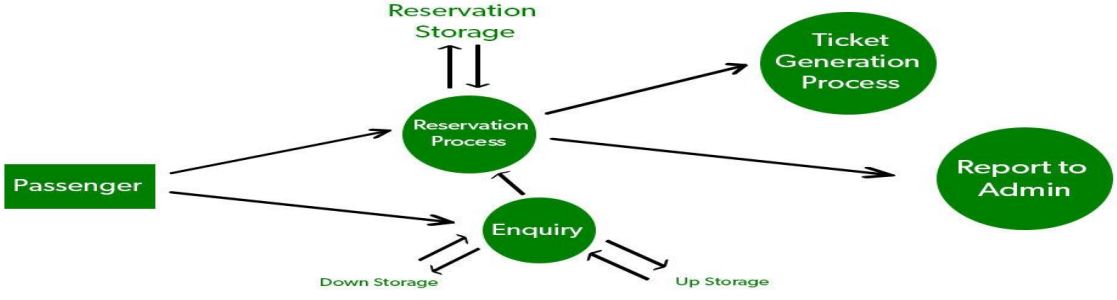
DFD uses hierarchy to maintain transparency thus multilevel DFD's can be created. Levels of DFD are as follows:

- 0-level DFD: It represents the entire system as a single bubble and provides an overall picture of the system.
- 1-level DFD: It represents the main functions of the system and how they interact with each other.
- 2-level DFD: It represents the processes within each function of the system and how they interact with each other.
- 3-level DFD: It represents the data flow within each process and how the data is transformed and stored.

- **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.

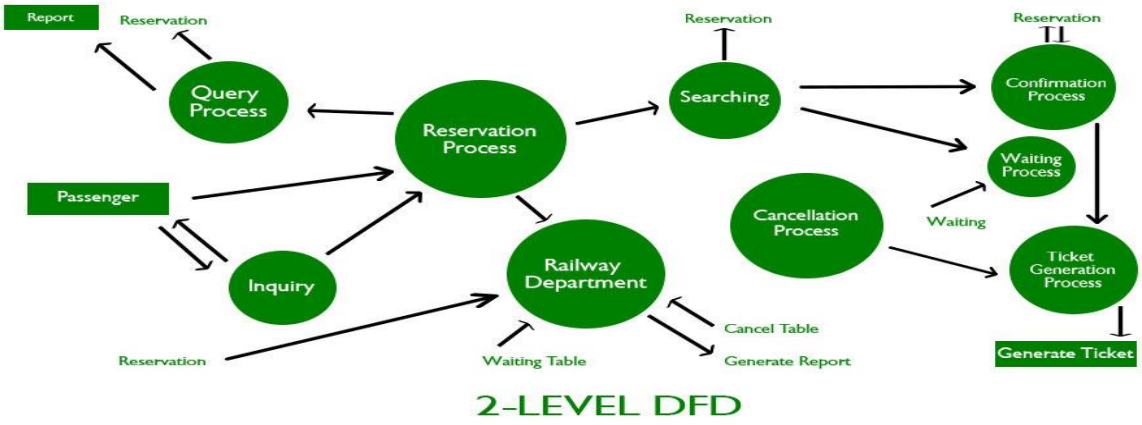
0-LEVEL DFD

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



1-LEVEL DFD

- **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.



Differences Between DFD and ERD

DFD	ERD
It stands for Data Flow Diagram.	It stands for Entity Relationship Diagram or Model.
Main objective is to represent the processes and data flow between them.	Main objective is to represent the data object or entity and relationship between them.
It explains the flow and process of data input, data output, and storing data.	It explains and represent the relationship between entities stored in a database.
Symbols used in DFD are: rectangles (represent the data entity), circles (represent the process), arrows (represent the flow of data), ovals or parallel lines (represent data storing).	Symbols used in ERD are: rectangles (represent the entity), diamond boxes (represent relationship), lines and standard notations (represent cardinality).
Rule followed by DFD is that at least one data flow should be there entering into and leaving the process or store.	Rule followed by ERD is that all entities must represent the set of similar things.
It models the flow of data through a system.	It model entities like people, objects, places and events for which data is stored in a system.

4. **State Transition Diagram:**
   It shows various modes of behavior (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 behavior 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.

References

- Software Engineering: A Practitioner's Approach | 9th Edition By Roger S. Pressman, Bruce R. Maxim
- Software Engineering 9th Edition by Ian Sommerville.
- https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/
- https://www.coursera.org/learn/introduction-to-software-engineering
- https://en.wikipedia.org/wiki/Software_engineering