CHAPTER

2

Software requirements analysis & specification

Requirements Analysis and Specification:

Requirements analysis and specification is considered to be a very important phase of software development and has to be undertaken with outmost care.

It starts once the feasibility study phase is complete and the project is found to be financially sound and technically feasible.

This phase consists of the following two activities:

(i) Requirement gathering and analysis (ii) Requirements specification

Requirements gathering:

This is a activity involves interviewing the end-users and customers and studying the existing document to collect all possible information regarding the system. If the project involves automating some existing procedures then the task of the system analyst becomes little easier as he can immediately obtain the input and output data forms and the details ofthe operational procedures. However, in the absence of a working system,much more imagination and creativity on the part of the system analyst is required.

Analysis of gathered requirements:

The main purpose of this activity is to clearly understand the exact requirements of the customer. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem :

• What is the problem?
• Why is it important to solve the problem?
• What are the possible solutions to the problem?
• What exactly are the data input to the system and what exactly are the data output required of the system?
• What are the likely complexities that might arise, while solving the problem?

Software Requirements Specification (SRS):

After the analyst has collected all the required information regarding the software to be developed and has removed all incompletences, inconsistencies, and amomalies from the specification, he starts to systematically organize the requirement in the form of an SRS document. The SRS documnet usually contains all the user requirements in an informal form.

• Among all the documents produced during a software development life cycle, writing the SRS document is expected to cater of audience.

Different people need the SRS document for very different purposes.

The important categories of users of the SRS document and their needs are as follows :

(i) Users, customer and marketing personnel

(ii) Software engineers
(iii) Test engineers
(iv) User documentation writer
(v) Project manager
(vi) Maintenance engineer

Contents of the SRS Document:

An SRS document should clearly document the following aspects of a system:

(i) Functional reuirement
(ii) Non functional requirements
(iii) Goals of implementation

• The functional requirements part should discuss the functionalities required for the system. To consider a system as performing a set of functions.

Each function of the system can be considered as a transformation of a set of input data (i_i) to the corresponding set of output data (o_i). The functional requirements of the system as documented in the SRS document should clearly describe each function which the system would support along with the corresponding input and output data set.

The nonfunctional requirements deal with the characteristics of the system that cannot be expressed as functions. Examples of non-functional requirements include aspects concerning maintainability, portability and usability. The non-functional requirement may also include reliablity issues, accuracy of the result, human-computer interface issues and constraint on the system implementation. The contraints on the system implementation describes, aspects such as the specific DBMS to be used as per customer request.

The goals of implementation part of the SRS document gives some general suggestions regarding developement. These suggestion guide trade-off among design decisions. The goal of implement section might document issues such as revision to the system functionalities that may be required to the future new devices to be supported in the future, reusuability issues etc. These are the items which the developers might keep in their mind during development system may meet some aspects that are not required immediately. It is useful to remember that anything that can be quantitavely stated is usually documented as a requirement and not as goal vice-versa.

Characteristics of a Good SRS document :

1. Concise: The SRS document should be concise and at the same time unambigous, consistent and complete.

2. Structured: The SRS document should be well structured. A well structured document is easy to understand and modify.

3. Black box view: It should only specify what the system should do and retain from stating how to do.

4. Conceptual Integrity: The SRS document should exhibit conceptual integrity. So, that the reader can easily understand the contents.

5. Reponse to undesired events: The document should characterize acceptable response to undesired events. These are called system response to exeptional conditions

6. Verifiable: All requirements of the system as documented in the SRS document should be verifiable.

Organization of SRS document:

1. Introduction
(a) Background (b) Overall description
(c) Environmental characteristics
(i) Hardware (ii) Peripherals (iii) People

2. Goals of implementation
3. Functional requirements
4. Non-functional requirements
5. Behavioural Description
(a) System states (b) Events and Actions

Techniques for Representing Complex logic :

There are two main techniques available to analyze and represent complex processing logic :

1. Decision trees   2. Decision tables

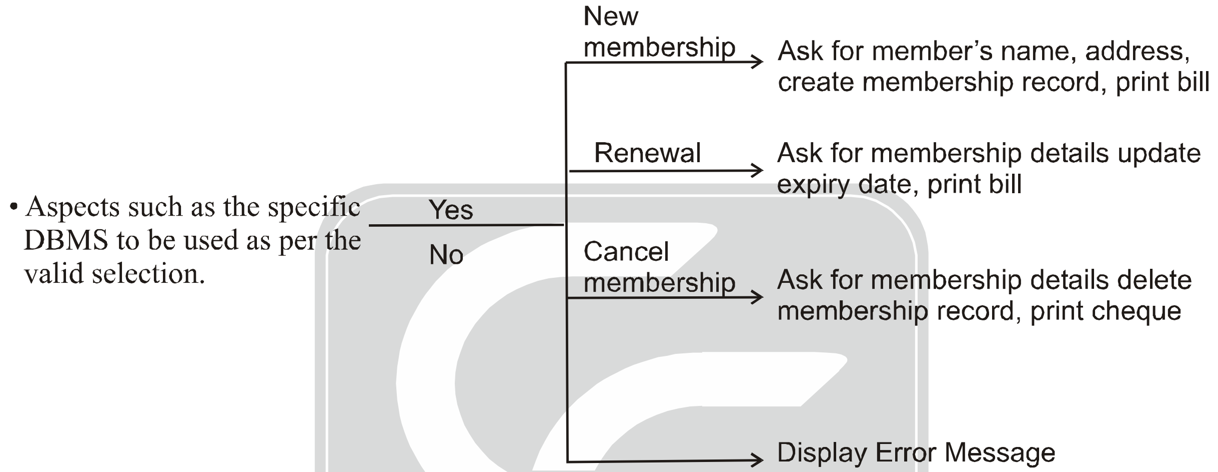
1. **Decision tree:** A decision tree gives a graphic view of the procesing logic involved in decision making and the corresponding action taken. Decision tables specify which variables are to be tested, and based on this what action need to be taken depending upon the outcome of the decision making logic and the order in which decision making is performed.

   The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

Example : A Library Membership Software (LMS) should support the following three options:

(a) New member   (b) Renewal   (c) Cancel Membership

Decision of LMS



1. Decision table: A decision table shows the decision making logic and the corresponding actions taken in tablular or matrix form. The upper rows of the table specify the variables or condition to be evaluated and the lower rows specify the actions to be taken. When an evaluation test is satisfied. A column in the table is called rule. A rule imples that if a condition is true, then the corresponding actions is to executed.

Decision table for the LMS

Condition				
Valid Selection	No	Yes	Yes	Yes
New Member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancelation	-	No	No	Yes
Actions	×			
Display error message	×			
Ask for members name etc.	×			
Build customer record		×		
Generate bill		×	×	
Ask for membership bill			×	×
Update expiry date			×	
Print cheque				×
Delete record				×

Formal System Development Techniques: Formal methods provide us with tools to describe a system and show that a system is correctly implemented. We say a system is correctly implemented when it satisfies in given specification. The specification of a system can be either as a list of desirable properties. (Property oriented approach) or an abstract model of the system model oriented approach

In a property oriented approach, we would probably start by listing of the properties of the system like, the consumer can start consuming only after the consumer has consumed the last item etc.
Example of property oriented specifiction styles are those of axiomatic specification and algebraic specification.
In model-oriented approach, we start by defining the basic opertions p (produce) and c (consume). Then we can start $S1 + p \Rightarrow S, S + c = S1$. Thus the model-oriented approaches essential specify a program by writing another. Examples of popular model-oriented specifiction technique or Z, CSP, CCS, etc.

Formal Technique: A formal technique is a mathematical method used to specify a hardware and/or a software system, verify whether a specificaion is reliable, verify whether an implementation satisfies in specification prove properties of a system without necessarily running the system and so on. More precisely a formal specification language consists of two sets syn and sem and relation at between them. The set syn is called the syntactic domain, the set sem is called sematic domain and the relation set is called the satisfaction relation.

Operational Semanties : The operational semanties of a formal method constitute the way computations are represented. There are different type of operational semanties :

1. **Linear Semantics :** In this approach, run of a system is described by a sequence of events or states.
2. **Branching Semantics :** In this approach the behaviour of a system is represented by a directed graph. The nodes of the graph represent the possible states in the evolution of a system
3. **Maximum Parallel Semantics :** In this approach, all the concurrent actions enabled at any state or assumed to be taken.
4. **Partial Order Semantics :** Under this view, the semantics described to a system constitute a structure of states satisfying a partial order relation among the states.

Axiomatic Specificaton: In axiomatic specification of a system, the first order logic is used to write the pre and post condition in order to specify the opertions of the system in the form of axioms. The pre-conditions basically capture the conditions that must be satisfied before an operation can be successfully involved. In essence the pre-conditions capture the requirements on the input parameters of a function. The post-conditions are the condition that must be satisfied when a function to be considered to have executed successfully. Thus, the post condition essentially check the constraints on the result produced for the function execution to be consider successful.

Steps to develop an axiomatic specification :

1. Establish the range of input values over which the function should behave correctly. Establish the constraints on the input parameters as a predicate.
2. Specify a predicate defining the condition which must hold on the output of the function if it behaved properly.
3. Establish the changes made to the functions input parameters after execution of the function. Pure mathematical functions do not change their input and therefore this type of assertion is not necessary for pure functions.
4. Combines all of the above into pre and post condition of the functions.

Algebraic Specification: In the algebraic specification technique an object class or type is specified in terms of relationship existing between the operations defined on that type.
Essentially, algebraic specification define a system as a heterogenous algebra. A heterogenous algebra is a collection of different sets on which several operationals are defined. Traditional algebras are homogeneous. A homogeneous algebra consists of a single set and several operations (+, –, *, /).
Each set of symbols in the algebra is called a sort of the algebra.
An algeraic specification is divided into four sections :

1. **Type Section:** In this section the sorts being used are specified.

2. **Exception section:** This section gives the names of the exceptional conditions that might occur when different operations are carried out.
3. **Equations section:** This section given a set of rewrited rules (or equation) defining the meaning of the interface procedures in terms of each other.

Pros and Cons. of Algebraic Specification: Algebraic specification have a strong mathematical base and can be viewed as heterogeneous algebra. therefore, they are unambigous and precise. Using an algebrais specification, the effect of any arbitary sequences of operations involving the interface procedures can be automatically.

A major short coming of algebraic specification is that they cannot deal with side effects. Therefore, algebraic specification are difficult to integrates with typical programming language.
Also, algebraic specification are hard to understand.

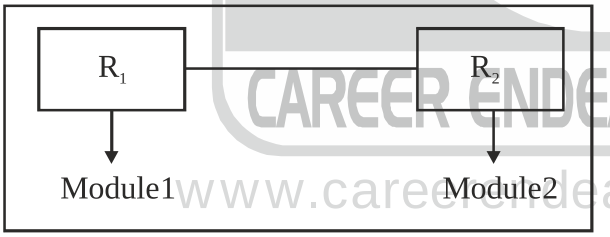
SRS Document (IEEE–830):
(i) Introduction : (a) Overview (b) Motivation (c) Brief history
Introduction work like header.

(ii) Goals of Implementation: Miles stone are written
(iii) Functional requirement: Functionalities are define. It take input and give output
Login

$$\left\{\begin{matrix}\text{username}\\ \text{password}\end{matrix}\right\} \rightarrow \text{sucessful login ouptut}$$

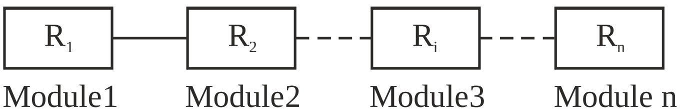
(iv) Non-functional requirements: Tell the qualities of functional requirements
[login-non-functional requirement is response time]

(v) Environmental constraints: Constraints of environments ae eleborated.
Reliability: Probability that system works correctly in given period time [system works without failure]
e.g. Reliability of ceiling fan = 0.99 for one year.
The arrangements of modules either serial or parallel.
Reliability of serial system:



Reliability of serial system = series system works if module1 and module2 both works

$$\boxed{= R_1 \cdot R_2}$$



Reliability (R) = $R_1 \cdot R_2 ....R_i.............R_n$

$$\boxed{R = \prod_{i=1}^{n} R_i}$$

SOLVED EXAMPLES

1. In serieal system there are two component having reliabilty 0.9 each. What is reliability of serial system

Soln. $R = R_1 \cdot R_2$

$R = 0.9 \times 0.9$

$R = 0.81$

2. In serial system there are three component having reliability 0.9 each. What is reliability of serial system.

Soln. $R = R_1 \cdot R_2 \cdot R_3$

$R = 0.9 \times 0.9 \times 0.9$

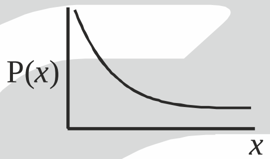
$R = 0.729$

Note: "If we increase the component in serial system the reliability decreases"

$$\boxed{0 \le R_i \le 1}$$

- Reliability follows : 'Exponential Distribution'
  Exponential Distribution

3. $P(x) = e^{-\lambda x}$



Mean $= E(x) = \int_{-\infty}^{\infty} x \cdot e^{-\lambda x} dx = \lambda$

Variance = λ^2

x is Random variable $\rightarrow$[vary value]

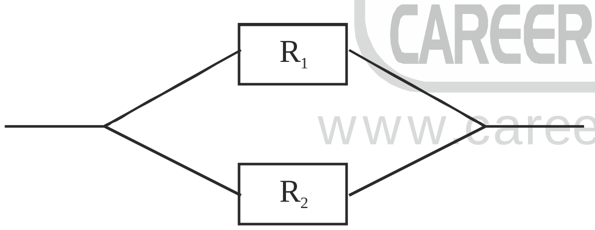
Bionomial Distribution:

n = number of trials

Mean = np p = success problem

Variance = npq q = 1–p

Reliability in parallel system: If we have two component in parallel system



System works when either component R_1 or component R_2 or both.

$$\boxed{R = R_1(1-R_2) + (1-R_2)R_1 + R_1R_2}$$

Or

$$\boxed{R = 1-(1-R_1)(1-R_2)}$$

If we have 'n' components then reliability

$$R = 1-(1-R_1)(1-R_2)..........(1-R_i).........(1-R_n)$$

$$\boxed{R = 1-\prod_{i=1}^{n}(1-R_i)}$$

Example: If we have two component having reliability 0.9 each. What is reliability in parallel system?

Soln. $R = 1-(1-R_1)(1-R_2) = 1-(1-0.9)(1-0.9) = 1-0.1\times 0.1 = 1-0.01 = 0.99$

Example: If we have three component having reliability 0.9 each. What is reliability in parallel system?

Soln. $R = 1-(1-R_1)(1-R_2)(1-R_3) = 1-(1-0.9)(1-0.9)(1-0.9) = 1-0.1\times 0.1\times 0.1 = 1-0.001 = 0.999$

Note "If we increase number of component in parallel system then reliability increase"

Availability: Probability that system is available in given period of time

Availability of serial system:

For two sytsem, $A = A_1.A_2$

More than two, $A = A_1, A_2, \dots\dots\dots A_n = \prod_{i=1}^{n} A_i$

Availability of Parallel system:

Two component $A = 1-(1-A_1)(1-A_2)$

More than two $A = 1-\prod_{i=1}^{n}(1-A_i)$

Predicate: Generalized version of preposition.

e.g. x is even number, $\forall\ x / \exists x$

s.t. $x \in \{2, 4, 6\}$

Predicate logic for more then one.

SRS Document representation:

(i) English Language (logic)
(ii) Algebric Expression (Regular Expression)
(iii) Transition Diagrams (State Diagram)
(iv) UML (unified modelling language (Analysis Modelling)

Decision table and Decision tree are used to ensure the **correctness of SRS**

Decision table: It is table which contains conditions, C_1, C_2, C_3

Rules/Actions – A_1, A_2 ....A_n

Condition either true or false i.e. if there are n conditions then = 2×2×2 .... n times

$\boxed{= 2^n \text{ are maximum actions}}$

If all actions are present than that table i.e. known as 'complete table'

	Rule1	Rule2
C_1	T	F
C_1	X	T
C_1	F	T
A_1	✓	
A_2		✓

$X \rightarrow$ not clear

There are 3 conditions and 2 action. So, it is not complete table

3 condition = $2^3 = 8$ action

Rule 1: if (C_1 = = T) and (C_3 = = F) then A1

Rule 2 : if (C_1 = = F) and (C_2 = = T) and (C_3 = = T) then A_2.

Problem: Which one of the following is NOT desired in a good software Requirement Specifications (SRS) document ?

(a) Functional Requirements
(b) Non-Functional Requirements
(c) Goals of Implementation
(d) Algorithm for Software Implementation

[GATE-2011]

Ans. **(d)**

Soln. A software requirements specification (SRS), a requirements specification for a software system, is a complete description of the behaviour of a system to be developed. In addition it also contains non-functional requirements. Algorithms are developed during design phase.

Problem: A Software Requirements Specification (SRS) document should avoid discussing which one of the following ?

(a) User interface issues
(b) Non-functional requirements
(c) Design specification
(d) Interfaces with third party software

[GATE-2015]

Ans. **(c)**

Soln. SRS document should avoid discussing design specification.

Data Flow Diagrams (DFD):

DFD shows the flow of data through system. The system may be compnay an organization, a set of procedures, a computer hardware system, a software system, or any combination of the precedding. DFD is also known as a data flow graph or a bubble chart in DFD:

(i) All name should be unique

(ii) DFD is not a flowchart, because Arrow $(\rightarrow)$ in a flow chart shows the flow of events, where as arrow in DFD shows the flow of data.

(iii) No logical decision in DFD (unlike flow chart)

(iv) Do not become bogged down with details.

• The following symbols used in DFD:

• 

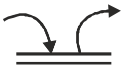
Data Flow: Used to connect process to each other. The arrow head indicated directions of data flow.

• 

Process: Perform some transformation of I/P data to yield O/P data.

- 

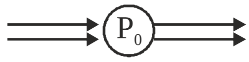
  Source/Sink (External Entity): A source of system inputs, or sink of system outputs.

- 

  Data Store: A repository of data; he arroead indicates net inputs and not outputs to store.

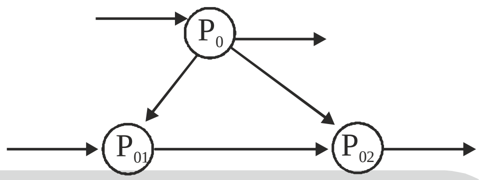
• The DFD is mainly used to represent a system a software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functionality details.

• A O-level DFD, also called a fundamental DFD represents the entire S/W element as a single with I/P and O/P data.

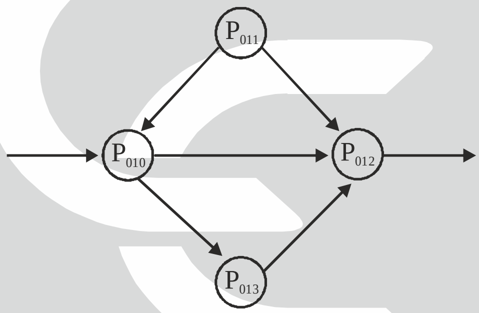


(O-level DFD)

Now the system is decomposed and represented as a DFD with multiple bubbles called 1-level DFD.



If again process of P_{01} does not give the more details then further it can be divided into multiple bubbles, and this type of DFD is called 2-levels DFD.



- Entity Relationship (E-R) Diagram

  It is a detailed logical representation of the data and uses the main constructs namely Entity, Relationship and Attributes.
- **Entity:**

  An entity is a thing or object in a real world that is distinghuishable from other objects. Entity is described by a set of attributes.
- **Entityt set:**

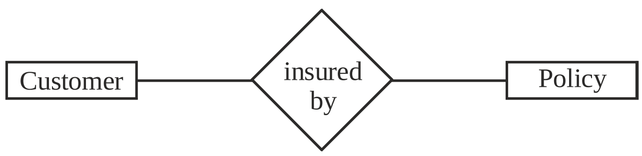
  It is a entities of same type that share same prosperties or attributes. It is denoted by

  $\boxed{E}$

  e.g. Customer, Employee etc.

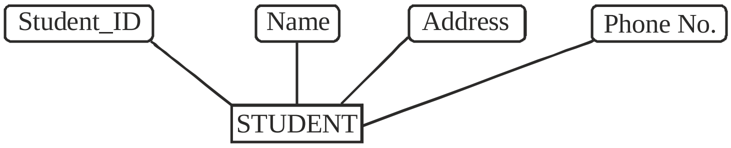
  Relationship: Relationship is an associate among entities. It is dentoed by  diamond

  e.g. A customer is insured by a policy is represented as below:

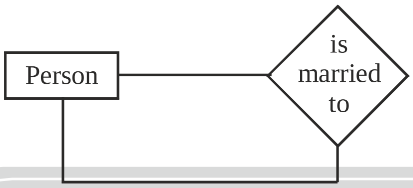


Attributes: An attribute is a property or characteristic of an entity and each entity type has a set of attributes assocaited with it. It is denoted by (A).

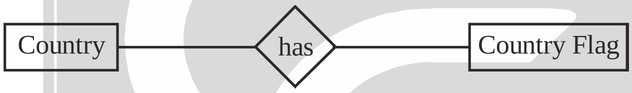
e.g. student entity has student_ID, Name, Address, and Phone No, attributes and is represented as follows:



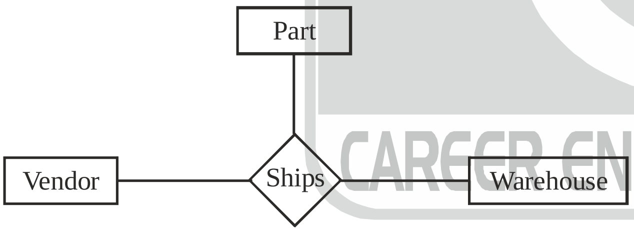
- Degree of relationship
  The 3 most common relationships in E-R models are-unary, binary, and ternary.
- **UNARY Relationship**
  This is also called recursive relationship. It is a relationship between the instances of one entity set.
  e.g. each person may be married to one another person.



- **Binary Relatinship:**
  It is a relationship between instances of two entity sets. A binary relationship is of degree 2. e.g. A country has a country flag.

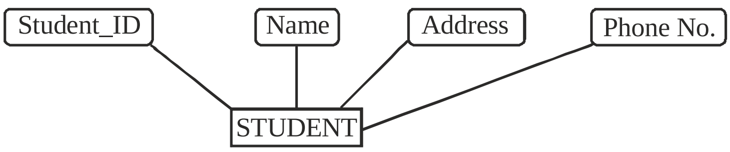


- **Ternary Relationship:**
  It is a simultaneous relationship among instances of three entity sets. A ternary relationship if of degree 3.
  e.g. The relationship ship tracks the part, that is shipped by a vendor to a selected warehouse.



Candidate keys and Identifier:

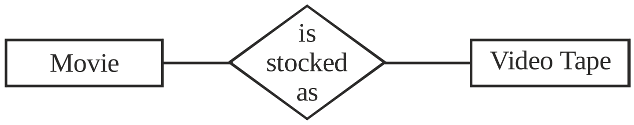
- A candidate key may be of one or more attribute or combination of attributed or combination of attributes that uniquely identifies each instancesof an entity set.
- An identifier is a candidate key that has been selected to be used for uniquely identifies each instance of an entity set.
  e.g. The candidate keys for a student entity might be student ID, or student with Address. Here we have shown student-ID as an key indentifier for student entity.



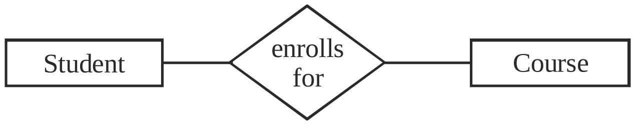
Candinalities and optionality:

- The candinality express the number of entities of an entity set to which another entity can be associated via a relationship, it can be one of the following

one-to-one

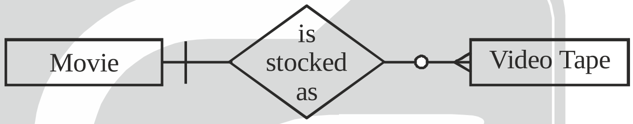


one-to-many

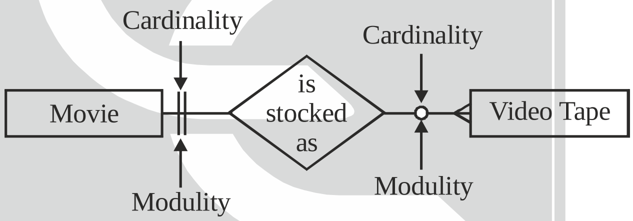


many-to-many

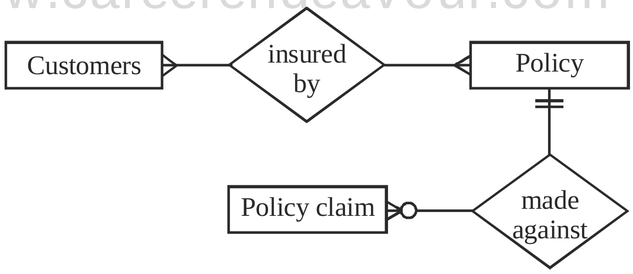
- An optionality shows an entity set may be/may not be associated with another entity set through the relationship. It is denoted by 0 in the cordinality
  e.g. A movie may have many video tapes in stock or may not have any video tape in stock. It is represent by as follows.



- If there is one video tape exist in stock, then if contains one move. So, it is represented as follows:



Example: Consider example of insurance that offfers both home and automobile insurance policies. These policies are offered to individuals and business. A customer can be insured by many policies. Similarly, a policy can cover many customers. One policy can be impacted by many policy claims. However, a policy may not have any policy claims field against it. Draw the E-R diagrams.

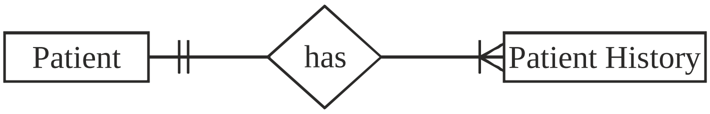


- A customer can be insured by many 'policy' and a 'policy' can cover many 'customer'. This is a many-to-many relationship. A 'Policy Claim' is made against one 'Policy' and one 'Policy' can be imacted by many 'Policy Claims'. This is a one-to-one and one-to-many relationship.
- A 'Policy Claim' is made against one 'Policy' and one 'Policy' can be imacted by many 'Policy Claims'. This is a one-to-one and one-to-many relationship
  (a) 'Policy' may not have nay 'Policy Claim' field against it, it means the relationship is optional.

- **Example:** Draw the E-R diagrams of the following
  (a) Patient has patient history. Each patient has one or more patient histories. Assume that the initial patient visit is always recorded as an instance of patient history. Each instance of patient history belongs to exactly one patient.
  (b) 'Employee is asigned to project. Each project has altleast one assigned employee. Some projects have than one employee. Each employee may or may not be assigned to any existing project, or may be assigned to served projects.
  (c) Person is married to Person.

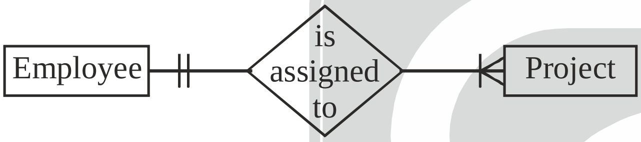
- **Explanation:**

(a)



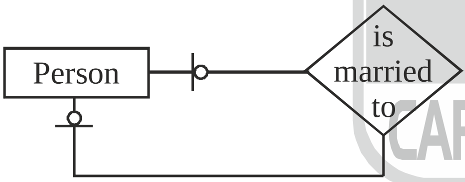
• There are two entities 'Patient' and Patient-History' Each 'patient' has one or more 'patient-History'. This is one-top-many relationship with mandatory cardinality.
• Initial 'patient visit is always recorded as an instance of 'Patient History'. This is mandatory cardinality near the manycardinality of 'patient History'.
• Each 'patient History' belongs ot exactly one 'Patient'. This is modulity near the mandotry cardinality relationship of 'Patient'.

(b)



• There are two entities 'Employee' and 'Project'. Each 'Project' has atleast one assigned 'Employee'. This is one-to-many relationship with mandatory cordinality.
• Each 'Employee' may or may not be assigned to any existing 'Project', or may not be assigned to any existing 'Project', or may be assigned to several Project. This is one-to-many relationship with optional cardinality.

(c)



• There is only one entity 'Person'. A 'Person' may or may not be married and if married then one person with one person and not more than person. This is one-to-one relationship with optional cardinality.

DATA FLOW DIAGRAM (DFD):
We will explore one of the three major graphical modeling tools of structured analysis; the dataflow diagram. The dataflow diagram is a modeling tool that allows us to picture a system as a network of funtional processes, connected to one another by "pipelines" and "holding tanks" of data. In the computer literature, and in your conversions with other systems analysis and users, you may use any of the following terms as synonyms for dataflow diagram:

• Bubble chart
• DFD
• Bubble diagram.
• Process model
• Work flow diagram
• Function model
• "a picture of what's going on around here"

The dataflow diagram is one of the commonly used systems-modeling tools, particularly for operational systems in which the functions of the system are of paramout importance and more complex than the data that the system manipulates. DFSs were list used in the software engineering field as a notation for studying system design issues (e.g. in early structured design books and articles such as, in turn, the notation had been borrowed from earlier papers on graph theory, and it continues to be used as a convenient notation by software engineers concerned with direct implementation of models of user requirements.

This is interesting background, but is likely to be irrelevant to the users to whom you show DFD system models; indeed, probably the worst thing you can do say, “Mr. Users, I’d like to show you a top-down, partitioned, graph-theoretic model of your system”. Actually, many users will be familiar with the uderlying concept of DFD’s because the same kind of notation has been used by operations research scientists for nearly 70 years to build work-flow models information - processing systems, but also as a way of modeling whole organizations, that is, as a tool for business planning and strategic planning.

We will begin our study of dataflow diagrams by examining the components of typical dataflow diagram: the process, the flow, the store, and the terminator. We will use a fairly standard notation for DFDs. However, we will also include DFD notation for modeling real-time systems (i.e. control flows and control processes). This additional notation is generally not required for business-oriented systesms, but is crucial when modeling a variety of engineering and scientific systesm.

SOLVED PROBLEMS

1. The diagram that helps in understanding and representing user requirements for a software project using UML (Unified Modeling Language) is

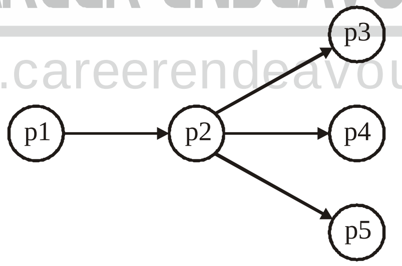
(a) Entity Relationship Diagram (b) Deployment Diagram
(c) Data Flow Diagram (d) Use Case Diagram

[GATE-2004]

Ans. (d)

Soln. Use case Diagram: It gives the external view of the system that helps in understanding and representing user requirements for a software project using UML.

2. In a data flow diagram, the segment shown below is identified as having transaction flow characteristics, with p2 identified as the transaction center



A first level architectural design of this segment will result in a set of process modules with an associated invocation sequence. The most appropriate architecture is

(a) p1 invokes p2, p2 invokes either p3, or p4, or p5.
(b) p2 invokes p1, and then invokes p3, or p4, or p5.
(c) A new module Tc is defined to control the transaction flow. This module Tc first invokes p1 and then invokes p2. p2 in turn invokes p3, or p4, or p5.
(d) A new module Tc is defined to control the transaction flow. This module Tc invokes p2. p2 invokes p1, then invokes p3, or p4 or p5.

[GATE-2005]

Ans. (c)

Soln. A new module Tc is defined to control the transaction flow. This module Tc first invokes p1 and then invokes p2. p2 in turn invokes p3, or p4 or p5.

3. 9. Which of the following statements are TRUE ?

I. The context diagram should depict the system as a single bubble.
II. External entities should be identified clearly at all levels of DFD.
III. Control information should not be represented in a DFD.
IV. A data store can be connected either to another data store to an external entity.

(a) II and III (b) I, II and IV (c) I and III (d) I, II and III

[GATE-2009]

Ans. **(c)**

Soln. The context diagram depict the system as a single bubble and control information should not represent in DFD. In DFD external entities are represented only at level 0 DFD or context diagram. Between every external agent and data store process is compulsory.