Chapter 3

ANALYSIS

CHAPTER OUTLINE

After studying this chapter, students will be able to understand the:

- System Requirements
- System Process Requirements
- System Data Requirements

INTRODUCTION

Analysis is the first systems development life cycle (SDLC) phase where we begin to understand, in depth, the need for system changes. Systems analysis involves a substantial amount of effort and cost, and is therefore undertaken only after management has decided that the systems development project under consideration has merit and should be pursued through this phase. The analysis team should not take the analysis process for granted or attempt to speed through it. Most observers would agree that many of the errors in developed systems are directly traceable to inadequate efforts in the analysis and design phases of the life cycle. Because analysis is a large and involved process, we divide it into two main activities to make the overall process easier to understand:

- **Requirements determination.** This is primarily a fact finding activity.
- **Requirements structuring.** This activity creates a thorough and clear description of current business operations and new information processing services.

DETERMINING SYSTEM REQUIREMENTS

Collection of information is at the core of systems analysis. Information requirement determination (IRD) is frequently and convincingly presented as the most critical phase of information system (IS) development, and many IS failures have been attributed to incomplete and inaccurate information requirements. System analysts must collect the information about the current system and how users would like to improve their performance with new information system. Accurately understanding the users' requirements will help the system developing team deliver a proper system to the end users in limited time and limited budget.

Information on what the new system should do is gathered from as many sources as possible: users of current system, from observing users, from the reports, forms and procedures. All this is documented and made ready for structuring. Main characteristics of a good systems analyst are listed below:

- **impertinence** (ask questions about everything that exists and also about what may exist in the future),
- **impartiality** (find the best solution to a business problem or opportunity – consider issues raised by all parties),
- **relax constraints** (eliminate unfeasibility, assume that anything is possible, traditions may not always be reasonable),
- **attention to details** (everything must fit together so that the system works properly),
- **Reframing** (every system is different and needs a new creative approach).

The primary deliverables from requirements determination are various forms of information gathered during the determination process: transcripts of interviews, notes from observations and analysis of documents, analyzed responses from questionnaires, sets of forms, reports, job descriptions, or computer-generated output such as system prototypes. We could group all this information in three groups:

- Information collected from conversations with or observations of users (interview transcripts, notes form observations etc.)
- Existing written information (business mission or strategy, forms, reports, training manuals, flow charts and documentation of existing system etc.)
- Computer-based information (results from JRP sessions, CASE repository contents)

TRADITIONAL METHODS FOR DETERMINING REQUIREMENTS

At the core of systems analysis is the collection of information. At the outset, we must collect information about the information systems that are currently being used and how users would like to improve the current systems and organizational operations with new or replacement information systems. One of the best ways to get this information is to talk to the people who are directly or indirectly involved in the different parts of the organizations affected by the possible system changes: users, managers, funders, and so on. Another way to find out about the current system is to gather copies of documentation relevant to current systems and business processes.

INTERVIEWING AND QUESTIONNAIRES

Interviewing is one of the primary ways analysts gather information about an information systems project. Early in a project, an analyst may spend a large amount of time interviewing people about their work, the information they use to do it, and the types of information processing that might supplement their work. Others are interviewed to understand organizational direction, policies, and expectations that managers have of the units they supervise. During interviewing, you gather facts, opinions, and speculation and observe body language, emotions, and other signs of what people want and how they assess current systems. Interviewing someone effectively can be done in many ways, and no one method is necessarily better than another. There are many ways to arrange an effectively interview and no one is superior to others. However, experience analysts commonly accept some following best practices for an effective interview:

- Prepare the interview carefully, including appointment, priming question, checklist, agenda, and questions.
- Listen carefully and take note during the interview (tape record if possible)
- Review notes within 48 hours after interview
- Be neutral
- Seek diverse views

The general nature of the interview should be explained to the interviewee in advance. You may ask the interviewee to think about specific questions or issues, or to review certain documentation to prepare for the interview. Spend some time thinking about what you need to find out, and write down your questions. Do not assume that you can anticipate all possible questions. You want the interview to be natural and, to some degree, you want to direct the

interview spontaneously as you discover what expertise the interviewee brings to the session. Prepare an interview guide or checklist so that you know in which sequence to ask your questions and how much time to spend in each area of the interview. The checklist might include some probing questions to ask as follow-up if you receive certain anticipated responses.

Questionnaires

Questionnaires have the advantage of gathering information from many people in a relatively short time and of being less biased in the interpretation of their results. Choosing right questionnaires respondents and designing effective questionnaires are the critical issues in this information collection method. People usually are only use a part of functions of a system, so they are always just familiar with a part of the system functions or processes. In most situations, one copy of questionnaires obviously cannot fit to all the users. To conduct an effective survey, the analyst should group the users properly and design different questionnaires for different group. Moreover, the ability to build good questionnaires is a skill that improves with practice and experience. When designing questionnaires, the analyst should concern the following issues at least:

- The ambiguity of questions.
- Consistence of respondents' answers.
- What kind of question should be applied, open-ended or close-ended?
- What is the proper length of the questionnaires?

Interview Outline	
Interviewee: Name of person being interviewed	**Interviewer**: Name of person leading interview
Location/Medium: Office, conference room, or phone number	**Appointment Date:** **Start Time:** **End Time:**
Agenda: Introduction Background Project Overview of Interview Topics to be covered Permission to Tape Record Topic 1 Questions Topic 2 Questions Summary of Major Points Questions from Interviewee Closing	**Approximate Time:**
General Observations: Interviewee seemed busy - probably need to call in a few days for follow-up questions since he gave only short answers. PC was turned off- probably not a regular PC user.	

Unresolved Issues, Topics not covered: He needs to look up figures of previous years. He raised the issue of how to handle the problem, but he did not have to discuss.	
When to ask question, if conditional Question number 1 Have you used current system? If so, how often? If yes, go to Question number 2.	**Answer** Yes, I ask for report weekly **Observation** Seemed anxious - may be over-estimating usage frequency
Question 2 What do you like least about this system?	**Answer** Using wrong units Observations: Options determine the right units, but user chose the wrong one.

Figure 3.1: A typical interview guide

Interview Guidelines

- First, with either open- or closed-ended questions, do not phrase a question in a way that implies a right or wrong answer. Respondents must feel free to state their true opinions and perspectives and trust that their ideas will be considered. Avoid questions such as "Should the system continue to provide the ability to override the default value, even though most users now do not like the feature?" because such wording predefines a socially acceptable answer.
- Second, listen carefully to what is being said. Take careful notes or, if possible, record the interview on a tape recorder (be sure to ask permission first!). The answers may contain extremely important information for the project. Also, this may be your only chance to get information from this particular person. If you run out of time and still need more information from the person you are talking to, ask to schedule a follow-up interview.
- Third, once the interview is over, go back to your office and key in your notes within forty-eight hours with a word processing program such as Microsoft Word. For numerical data, you can use a spreadsheet program such as Microsoft Excel. If you recorded the interview, use the recording to verify your notes. After forty-eight hours, your memory of the interview will fade quickly. As you type and organize your notes, write down any additional questions that might arise from lapses in your notes or ambiguous information. Separate facts from your opinions and interpretations. Make a list of unclear points that need clarification. Call the person you interviewed and get answers to these new questions. Use the phone call as an opportunity to verify the accuracy of your notes. You may also want to send a written copy of your notes to the person you interviewed to check your notes for accuracy. Finally, make sure to thank the person for his or her time. You may need to talk to your respondent again. If the interviewee will be a user of your system or is involved in some other way in the system's success, you want to leave a good impression.

- Fourth, be careful during the interview not to set expectations about the new or replacement system unless you are sure these features will be part of the delivered system. Let the interviewee know that there are many steps to the project. Many people will have to be interviewed. Choices will have to be made from among many technically possible alternatives. Let respondents know that their ideas will be carefully considered. Because of the repetitive nature of the systems development process, however, it is premature to say now exactly what the ultimate system will or will not do.
- Fifth, seek a variety of perspectives from the interviews. Talk to several different people: potential users of the system, users of other systems that might be affected by this new system, managers and superiors, information systems staff, and others. Encourage people to think about current problems and opportunities and what new information services might better serve the organization. You want to understand all possible perspectives so that later you will have information on which to base a recommendation or design decision that everyone can accept.

Comparison between Interviews and Questionnaires		
Characteristics	**Interviews**	**Questionnaires**
Information Richness	High (many channel)	Medium to low (only responses)
Time Required	Can be extensive	Low or moderate
Expense	Can be high	Moderate
Confidentiality	Interviewee is known to interviewer	Respondent can be unknown
Chance of Follow-up & Probing	Good: probing and clarification questions can be asked by either interviewer or interviewee	Limited: probing and follow-up done after original data collection
Involvement of Subject	Interviewee is involved and committed	Respondent is passive, no clear commitment
Potential Audience	Limited numbers, but complete responses from those interviewed	Can be quite large, but lack of response from some can bias results

Directly Observing Users

People are not always very reliable informants, even when they try to be reliable and tell what they think is the truth. People often do not have a completely accurate appreciation of what they do or how they do it. This we especially true concerning infrequent events, issues from the past, or issues for which people have considerable passion. Since people cannot always be trusted to reliably interpret and report their own actions, analyst can supplement and corroborate what people say by watching what they do or by obtaining relatively objective measures of how people behave in work situation. However, observation can cause people to change their normal operation behavior. It will make the gathered information biased.

Analyzing Procedures and Other Documents

By examining existing system and organizational documentation, system analyst can find out details about current system and the organization these systems support. In documents analyst can find information, such as problem with existing systems, opportunities to meet new needs if only certain information or information processing were available, organizational direction that can influence information system requirements, and the reason why current systems are designed as they are, etc.

However, when analyzing those official documentations, analysts should pay attention to the difference between the systems described on the official documentations and the practical systems in real world. For the reason of inadequacies of formal procedures, individual work habits and preferences, resistance to control, and other factors, the difference between so called formal system and informal system universally exists.

In documents you can find information about:

- Problems with existing systems (e.g., missing information or redundant steps)
- Opportunities to meet new needs if only certain information or information processing were available (e.g., analysis of sales based on customer type)
- Organizational direction that can influence information system requirements (e.g., trying to link customers and suppliers more closely to the organization)
- Titles and names of key individuals who have an interest in relevant existing systems (e.g., the name of a sales manager who has led a study of buying behavior of key customers)
- Values of the organization or individuals who can help determine priorities for different capabilities desired by different users (e.g., maintaining market share even if it means lower short-term profits)
- Special information-processing circumstances that occur irregularly that may not be identified by any other requirements determination technique (e.g., special handling needed for a few large-volume customers who require use of customized customer ordering procedures)
- The reason why current systems are designed as they are, which can suggest features left out of current software that may now be feasible and desirable (e.g., data about a customer's purchase of competitors' products not available when the current system was designed; these data now available from several sources)
- Data, rules for processing data, and principles by which the organization operates that must be enforced by the information system (e.g., each customer assigned exactly one sales department staff member as primary contact if customer has any questions)

CONTEMPORARY METHODS FOR DETERMINING SYSTEM REQUIREMENTS

Even though we called interviews, questionnaires, observation, and document analysis traditional methods for determining a system's requirements, all of these methods are still used by analysts to collect important information. Today, however, additional techniques are available to collect information about the current system, the organizational area requesting the new system, and what the new system should be like. In this section, you learn about two modern information-gathering techniques for analysis: joint application design (JAD) and prototyping. These techniques can support effective information collection and structuring while reducing the amount of time required for analysis.

JOINT APPLICATION DESIGN

JAD was developed by two doctors in 1970s

JAD was developed by Chuck Morris and Tony Crawford of IBM in 1977 and has since been proven successful on thousands of software projects across industry sectors and application boundaries. Joint Application Development (JAD) is a user requirements elicitation process that involves the system owner and end users in the design and development of an application through a succession of collaborative workshops called JAD sessions. The JAD approach leads to shorter development lifecycles and greater client satisfaction because it draws users and information systems analysts together to jointly design systems in facilitated group sessions. JAD can also be adapted to developing business processes that do not involve computer automation.

The primary purpose of using JAD in the analysis phase is to collect systems requirements simultaneously from the key people involved with the system. The result is an intense and structured, but highly effective, process. Having all the key people together in one place at one time allows analysts to see the areas of agreement and the areas of conflict. Meeting with all these important people for over a week of intense sessions allows you the opportunity to resolve conflicts or at least to understand why a conflict may not be simple to resolve. JAD sessions are usually conducted in a location away from where the people involved normally work, in order to limit distractions and help participants' better concentrate on systems analysis. A JAD may last anywhere from four hours to an entire week and may consist of several sessions. A JAD employs thousands of dollars of corporate resources, the most expensive of which is the time of the people involved. Other expenses include the costs associated with flying people to a remote site and putting them up in hotels and feeding them for several days. The following is a list of typical JAD participants:

- **JAD session leader:** The JAD leader organizes and runs the JAD. This person has been trained in group management and facilitation as well as in systems analysis. The JAD leader sets the agenda and sees that it is met. He or she remains neutral on issues and does not contribute ideas or opinions, but rather concentrates on keeping the group on the agenda, resolving conflicts and disagreements, and soliciting all ideas.
- **Users:** The key users of the system under consideration are vital participants in a JAD. They are the only ones who clearly understand what it means to use the system on a daily basis.

- **Managers:** Managers of the work groups who use the system in question provide insight into new organizational directions, motivations for and organizational impacts of systems, and support for requirements determined during the JAD.
- **Sponsor:** As a major undertaking, because of its expense, a JAD must be sponsored by someone at a relatively high level in the company such as a vice president or chief executive officer. If the sponsor attends any sessions, it is usually only at the beginning or the end.
- **Systems analysts:** Members of the systems analysis team attend the JAD, although their actual participation may be limited. Analysts are there to learn from users and managers, not to run or dominate the process.
- **Scribe:** The scribe takes notes during the JAD sessions, usually on a personal computer or laptop.
- **IS staff:** Besides systems analysts, other IS staff, such as programmers, database analysts, IS planners and data-center personnel may attend the session. Their purpose is to learn from the discussion and possibly to contribute their ideas on the technical feasibility of proposed ideas or on the technical limitations of current systems.

JAD sessions are usually held in special-purpose rooms where participants sit around horseshoe-shaped tables, as in Figure below. These rooms are typically equipped with whiteboards (possibly electronic, with a printer to make copies of what is written on the board). Other audiovisual tools may be used, such as magnetic symbols that can be easily rearranged on a whiteboard, flip charts, and computer-generated displays. Flip-chart paper is typically used for keeping track of issues that cannot be resolved during the JAD, or for those issues requiring additional information that can be gathered during breaks in the proceedings. Computers may be used to create and display form or report designs or to diagram existing or replacement systems. In general, however, most JADs do not benefit much from computer support. The end result of a completed JAD is a set of documents that detail the workings of the current system and the features of a replacement system. Depending on the exact purpose of the JAD, analysts may gain detailed information on what is desired of the replacement system.

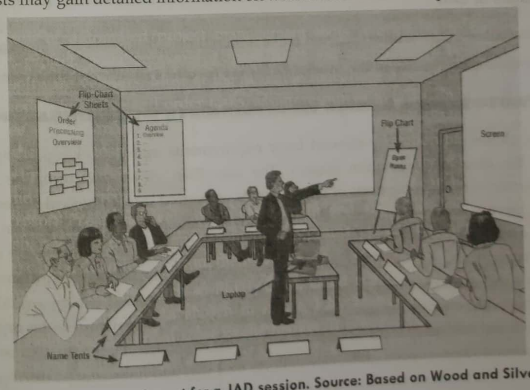


Figure 3.2: A typical room layout for a JAD session. Source: Based on Wood and Silver, 1989

When a JAD is completed, the final result is a set of documents that detail the workings of the current system related to study of a replacement system. These requirements definition document generally includes business activity model and definitions, data model and definition, data input and output requirements. It may also include interface requirements, screen and report layouts, ad hoc query specifications, menus, and security requirements. When used at a later point in the system development life cycle, a JAD session can also be used to refine a system prototype, develop new job profiles for system users, or develop an implementation plan. However, to exploit full potential of JAD, the groupware tools should be applied in JAD workshop sessions. The use of groupware tools to support the joint Application Development technique increases the value of this technique dramatically. When groupware tools are used in an automated JAD workshop, they greatly facilitate the generation, analysis, and documentation of information. This is particularly valuable for JAD workshops conducted to define and build consensus on the requirements for new systems.

resolve conflicts ani tyo existing system ma aako problem haru discuss garerw error free better garaune ani if naya system banauna lako xa vne esle reduce grne yyo system development time ani faster development reduces cost ani team le push grxa kaam garna tw

Advantages of JAD

- JAD allows you to resolve difficulties more simply and produce better, error-free software
- The joint collaboration between the company and the clients lowers all risks
- JAD reduces costs and time needed for project development
- Well-defined requirements improve system quality
- Due to the close communication, progress is faster
- JAD encourages the team to push each other to work faster and deliver on time

Disadvantages of JAD

project development time tw ghatauxa esle tarw aafnae JAD sessions dherae lamo vaidina skxa

- Different opinions within the team make it difficult to align goals and maintain focus
- Depending on the size of the project, JAD may require a significant time commitment

Using Prototyping during Requirements Determination

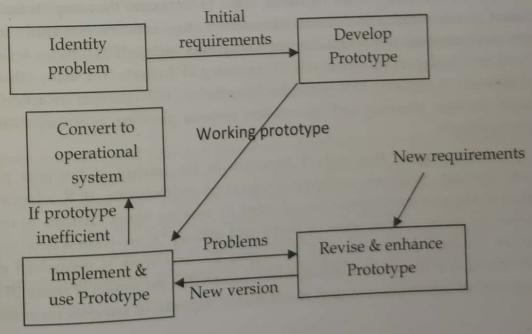
Prototyping allows us to quickly convert basic requirements into a working, though limited, version of the desired information system. The user then views and tests the prototype. Typically, seeing verbal descriptions of requirements converted into a physical system prompts the user to modify existing requirements and generate new ones. For example, in the initial interviews, a user might have said he wanted all relevant utility billing information on a single computer display form, such as the client's name and address, the service record, and payment history. Once the same user sees how crowded and confusing such a design would be in the prototype, he might change his mind and instead ask for the information to be organized on several screens but with easy transitions from one screen to another. He might also be reminded of some important requirements (data, calculations, etc.) that had not surfaced during the initial

interviews. You would then redesign the prototype to incorporate the suggested changes. Once modified, users would again view and test the prototype. Once again, you would incorporate their suggestions for change. Through such a repetitive process, the chances are good that you will be able to better capture a system's requirements. The goal with using prototyping to support requirements determination is to develop concrete specifications for the ultimate system, not to build the ultimate system. Prototyping is most useful for requirements determination when:

- User requirements are not clear or well understood, which is often the case for totally new systems or systems that support decision making.
- One or a few users and other stakeholders are involved with the system.
- Possible designs are complex and require concrete form to evaluate fully.
- Communication problems have existed in the past between users and analysts, and both parties want to be sure that system requirements are as specific as possible.
- Tools (such as form and report generators) and data are readily available to rapidly build working systems.

Prototyping also has some drawbacks as a tool for requirements determination. They include the following:

- A tendency to avoid creating formal documentation of system requirements, which can then make the system more difficult to develop into a fully working system.
- Prototypes can become idiosyncratic to the initial user and difficult to diffuse or adapt to other potential users.
- Prototypes are often built as stand-alone systems, thus ignoring issues of sharing data and interactions with other existing systems.
- Checks in the SDLC are bypassed so that some more subtle, but still important, system requirements might be forgotten (e.g., security, some data-entry controls, or standardization of data across systems).



RADICAL METHODS FOR DETERMINING SYSTEM REQUIREMENTS

Whether traditional or modern, the methods for determining system requirements that you have read about in this chapter apply to any requirements determination effort, regardless of its motivation. Yet, most of what you have learned has traditionally been applied to systems development projects that involve automating existing processes. Analysts use system requirements determination to understand current problems and opportunities, as well as what are needed and desired in future systems. Typically, the current way of doing things has a large impact on the new system. In some organizations, though, management is looking for new ways to perform current tasks. These ways may be radically different from how things are done now, but the payoffs may be enormous: Fewer people may be needed to do the same work; relationships with customers may improve dramatically; and processes may become much more efficient and effective, all of which can result in increased profits. The overall process by which current methods are replaced with radically new methods is referred to as business process reengineering (BPR).

To better understand BPR, consider the following analogy. Suppose you are a successful European golfer who has tuned your game to fit the style of golf courses and weather in Europe. You have learned how to control the flight of the ball in heavy winds, roll the ball on wide-open greens, putt on large and undulating greens, and aim at a target without the aid of the landscaping common on North American courses. When you come to the United States to make your fortune on the U.S. tour, you discover that improving your putting, driving accuracy, and sand shots will help, but the new competitive environment is simply not suited to your playing style. You need to reengineer your whole approach, learning how to aim at targets, spin and stop a ball on the green, and manage the distractions of crowds and press. If you are good enough, you may survive, but without reengineering, you will never become a winner. Just as the competitiveness of golf forces good players to adapt their games to changing conditions, the competitiveness of our global economy has driven most companies into a mode of continuously improving the quality of their products and services. Organizations realize that creatively using information technologies can significantly improve most business processes. The idea behind BPR is not just to improve each business process but, in a systems modeling sense, to reorganize the complete flow of data in major sections of an organization to eliminate unnecessary steps, combine previously separate steps, and become more responsive to future changes. Companies such as IBM, Procter & Gamble, Wal-Mart, and Ford have had great success in actively pursuing BPR efforts. Yet, many other companies have found difficulty in applying BPR principles. Nonetheless, BPR concepts are actively applied in both corporate strategic planning and information systems planning as a way to improve business processes radically.

BPR advocates suggest that radical increases in the quality of business processes can be achieved through creatively applying information technologies. BPR advocates also suggest that radical improvement cannot be achieved by making minor changes in existing processes but rather by using a clean sheet of paper and asking, "If we were a new organization, how would we accomplish this activity?" Changing the way work is performed also changes the way information is shared and stored, which means that the results of many BPR efforts are the development of information system maintenance requests, or requests for system replacement.

You likely have encountered or will encounter BPR initiatives in your own organization. A recent survey of IS executives found that they view BPR to be a top IS priority for the coming years.

Identifying Processes to Reengineer

A first step in any BPR effort is to understand what processes need to change, what are the key business processes for the organization. Key business processes are the structured set of measurable activities designed to produce a specific output for a particular customer or market. The important aspect of this definition is that key processes are focused on some type of organizational outcome such as the creation of a product or the delivery of a service. Key business processes are also customer focused. In other words, key business processes would include all activities used to design, build, deliver, support, and service a particular product for a particular customer. BPR, therefore, requires you first to understand those activities that are part of the organization's key business processes and then to alter the sequence and structure of activities to achieve radical improvements in speed, quality, and customer satisfaction. The same techniques you learned to use for system requirements determination can be applied to discovering and understanding key business processes: interviewing key individuals, observing activities, reading and studying organizational documents, and conducting JAD sessions. After identifying key business processes, the next step is to identify specific activities that can be radically improved through reengineering. Michael Hammer and James Champy, two academics who coined the term BPR, suggest systems analysts ask three questions to identify activities for radical change:

1. How important is the activity to delivering an outcome?
2. How feasible is changing the activity?
3. How dysfunctional is the activity?

The answers to these questions provide guidance for selecting which activities to change. Those activities deemed important, changeable, yet functional, are primary candidates for alteration. To identify dysfunctional activities, Hammer and Champy suggest you look for activities that involve excessive information exchanges between individuals, information that is redundantly recorded or needs to be rekeyed, excessive inventory buffers or inspections, and a lot of rework or complexity.

Disruptive Technologies

Once key business processes and activities have been identified, information technologies must be applied to improve business processes radically. Hammer and Champy suggest that organizations think "inductively" about information technology. Induction is the process of reasoning from the specific to the general, which means that managers must learn about the power of new technologies and think of innovative ways to alter the way work is done. This approach is contrary to deductive thinking, in which problems are first identified and solutions then formulated.

Hammer and Champy suggest that managers especially consider disruptive technologies when applying deductive thinking. Disruptive technologies are those that enable the breaking of long-held business rules that inhibit organizations from making radical business changes. For example, Toyota is using production schedule databases and electronic data interchange (EDI)—an information system that allows companies to link their computers directly to suppliers—to work with its suppliers as if they and Saturn were one company. Suppliers do not wait until Saturn sends them a purchase order for more parts but simply monitor inventory levels and automatically send shipments as needed.

Structuring System Process Requirements

Here we understand the logical modeling of processes by studying examples of data flow diagrams.

Process Modeling

Process modeling involves graphically representing the processes, or actions, that capture, manipulate, store, and distribute data between a system and its environment and among components within a system. A common form of a process model is a data-flow diagram (DFD). A data-flow diagram is a graphic that illustrates the movement of data between external entities and the processes and data stores within a system. Although several different tools have been developed for process modeling, we focus solely on data-flow diagrams because they are useful tools for process modeling. Data-flow diagramming is one of several structured analysis techniques used to increase software development productivity. Although not all organizations use each structured analysis technique, collectively, these techniques, like dataflow diagrams, have had a significant impact on the quality of the systems development process.

Modeling a System's Process

The analysis team begins the process of structuring requirements with an abundance of information gathered during requirements determination. As part of structuring, you and the other team members must organize the information into a meaningful representation of the information system that exists and of the requirements desired in a replacement system. In addition to modeling the processing elements of an information system and transformation of data in the system, you must also model the structure of data within the system. Analysts use both process and data models to establish the specification of an information system. With a supporting tool, such as a CASE tool, process and data models can also provide the basis for the automatic generation of an information system.

Deliverables and Outcomes

In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated data-flow diagrams. The following table lists the progression of deliverables that result from studying and documenting a system's process. First, a context data-flow diagram

shows the scope of the system, indicating which elements are inside and outside the system. Second, data-flow diagrams of the current system specify which people and technologies are used in which processes to move and transform data, accepting inputs and producing outputs. The detail of these diagrams allows analysts to understand the current system and eventually to determine how to convert the current system into its replacement. Third, technology-independent, or logical, data-flow diagrams show the dataflow, structure, and functional requirements of the new system. Finally, entries for all of the objects in all diagrams are included in the project dictionary or CASE repository.

1. Context DFD
2. DFDs of current physical system
3. DFDs of new logical system
4. Thorough description of each DFD component

Table: Deliverables for Process Modeling

This logical progression of deliverables helps you to understand the existing system. You can then reduce this system into its essential elements to show the way in which the new system should meet its information processing requirements, as they were identified during requirements determination. In later steps in the systems development life cycle, you and other project team members make decisions on exactly how the new system will deliver these new requirements in specific manual and automated functions. Because requirements determination and structuring are often parallel steps, data-flow diagrams evolve from the more general to the more detailed as current and replacement systems are better understood. Even though data-flow diagrams remain popular tools for process modeling and can significantly increase software development productivity, they are not used in all systems development methodologies. Some organizations, such as EDS, have developed their own type of diagrams to model processes. Some methodologies, such as rapid application development (RAD), do not model processes separately at all. Instead, RAD builds processes—the work or actions that transform data so that they can be stored or distributed-into the prototypes created as the core of its development life cycle. However, even if you never formally use data-flow diagrams in your professional career, they remain a part of systems development's history. DFDs illustrate important concepts about the movement of data between manual and automated steps and are a way to depict work flow in an organization. DFDs continue to benefit information systems professionals as tools for both analysis and communication. For that reason, we devote this entire chapter to DFDs.

Data Flow Diagrams

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expands it to a hierarchy of detailed diagrams. DFD has often been used due to the following reasons:

- Logical information flow of the system
- Determination of physical system construction requirements
- Simplicity of notation
- Establishment of manual and automated systems requirements

There are four basic symbols that are used to represent a data-flow diagram.

Process

A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database, or it can be complex as producing a report containing monthly sales of all retail stores in the northwest region. Every process has a name that identifies the function it performs. The name consists of a verb, followed by a singular noun.

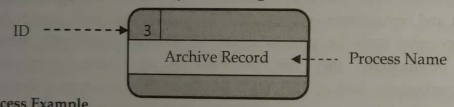
Example:

- Apply Payment
- Calculate Commission
- Verify Order

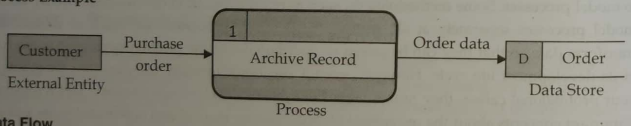
Notation

A rounded rectangle represents a process

Processes are given IDs for easy referencing



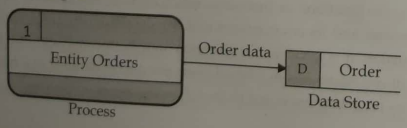
Process Example



Data Flow

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure). Because every process changes data from one form into another, at least one data-flow must enter and one data-flow must exit each process symbol.

Data flow Example:

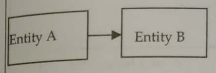
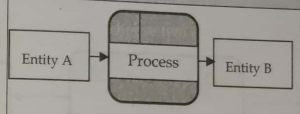
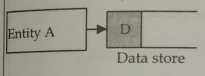
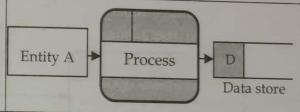
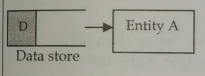
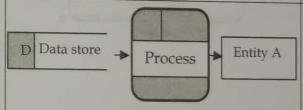
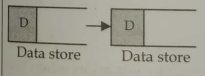
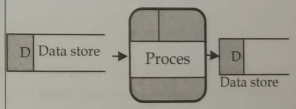


Notation

- Straight lines with incoming arrows are input data flow
- Straight lines with outgoing arrows are output data flows

Rule of Data Flow

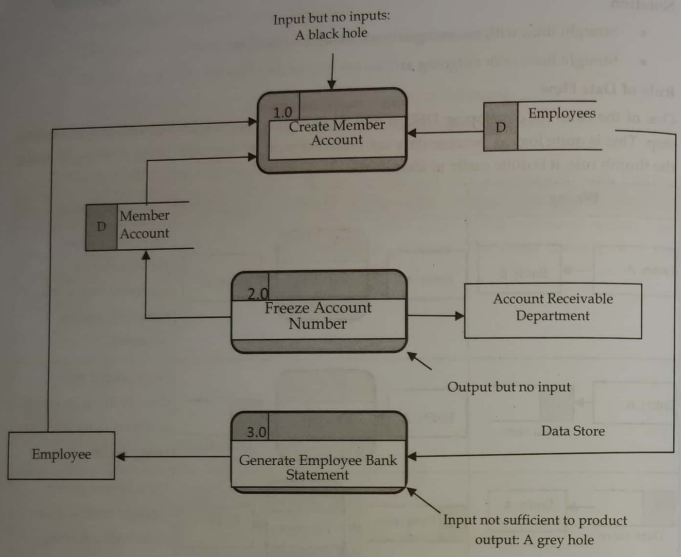
One of the rules for developing DFD is that all flow must begin with and end at a processing step. This is quite logical, because data can't transform on its own with being process. By using the thumb rule, it is quite easily to identify the illegal data flows and correct them in a DFD.

Wrong	Right	Description
		An entity cannot provide data to another entity without some processing occurred.
		Data cannot move directly from an entity to a data story without being processed.
		Data cannot move directly from a data store without being processed.
		Data cannot move directly from one data store to another without being processed.

Other frequently-made mistakes in DFD

A second class of DFD mistakes arises when the outputs from one processing step do not match its inputs and they can be classified as:

- **Black holes** - A processing step may have input flows but no output flows.
- **Miracles** - A processing step may have output flows but no input flows.
- **Grey holes** - A processing step may have outputs that are greater than the sum of its input

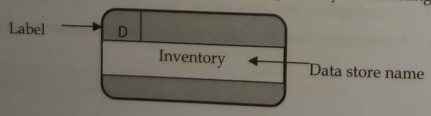


Data Store

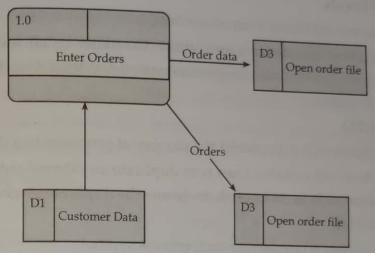
A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data because one or more processes need to use the stored data in a later time. A data store must be connected to a process with a data-flow. Each data store must have at least one input data-flow and at least one output data-flow (even if the output data-flow is a control or confirmation message).

Notation

- Data can be written into the data store, which is depicted by an outgoing arrow
- Data can be read from a data store, which is depicted by an incoming arrow.



Data Store Example



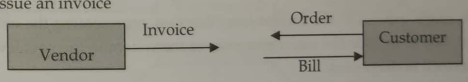
External Entity

An external entity is a person, department, outside organization, or other information system that provides data to the system or receives outputs from the system. External entities are components outside of the boundaries of the information systems. They represent how the information system interacts with the outside world. External entities also are called terminators because they are data origins or final destinations. An external entity must be connected to a process through a data-flow.

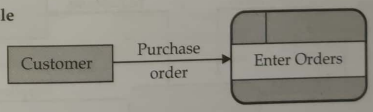
- A rectangle represents an external entity
- They either supply data or receive data
- They do not process data

Notation

- A customer submitting an order and then receive a bill from the system
- A vendor issue an invoice



External Entity Example



Guideline for Developing Data-Flow Diagram

1. Context Diagram - Level 0

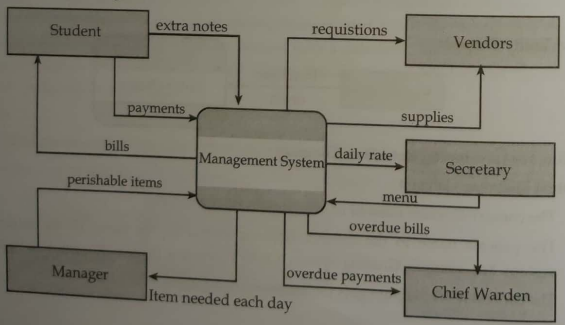
- The context diagram must fit in one page.
- The process name in the context diagram should be the name of the information system. For example, Grading System, Order Processing System, Registration System.
- The context level diagram gets the number 0 (level zero).

2. **Unique Name for Levels**
   - Use unique names within each set of symbols.
   - For example, there can be only one entity Customer in all levels of the data-flow diagrams; or here can be only one process named Calculate Overtime among all levels of data-flow diagrams.
3. **No Cross Line in DFD**
   - One way to achieve this is to restrict the number of processes in a data-flow diagram.
   - Another way to avoid crossing lines is to duplicate an external entity or data store. Use a special notation such as an asterisk, to denote the duplicate symbol.
4. **Numbering Convention**
   - Use a unique reference number for each process symbol.
   - Other process numbers are in the hierarchy of:
     - (1, 2, 3...);
     - (1.1, 1.2, 1.3, ..., 2.1, 2.2, 2.3,...);
     - (1.1.1, 1.1.2, 1.1.3...)

Context-Level Diagram

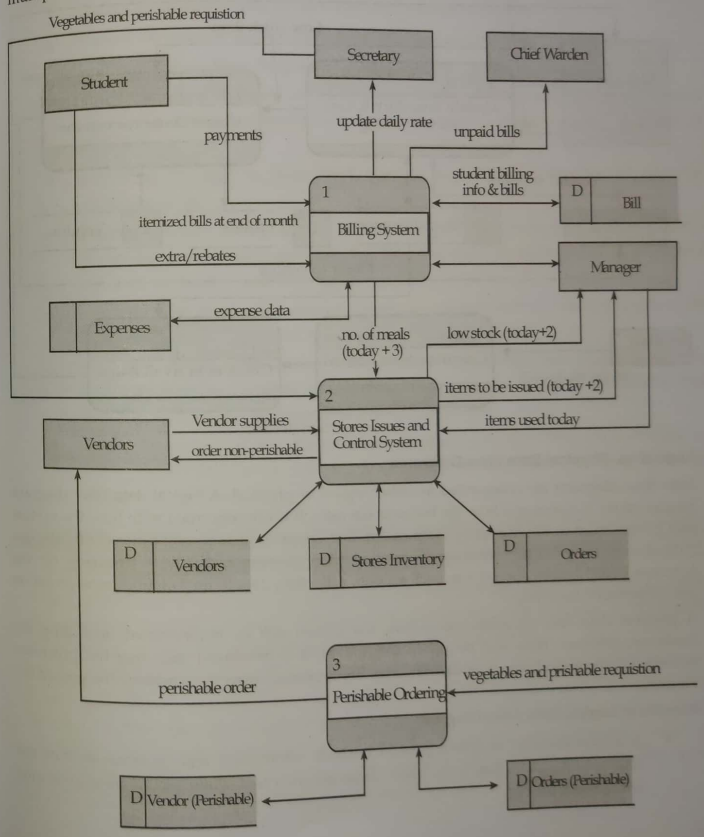
A context diagram gives an overview and it is the highest level in a data flow diagram, containing only one process representing the entire system. It should be split into major processes which give greater detail and each major process may further split to give more detail.

- All external entities are shown on the context diagram as well as major data flow to and from them.
- The diagram does not contain any data storage.
- The single process in the context-level diagram, representing the entire system, can be exploded to include the major processes of the system in the next level diagram, which is termed as diagram 0.



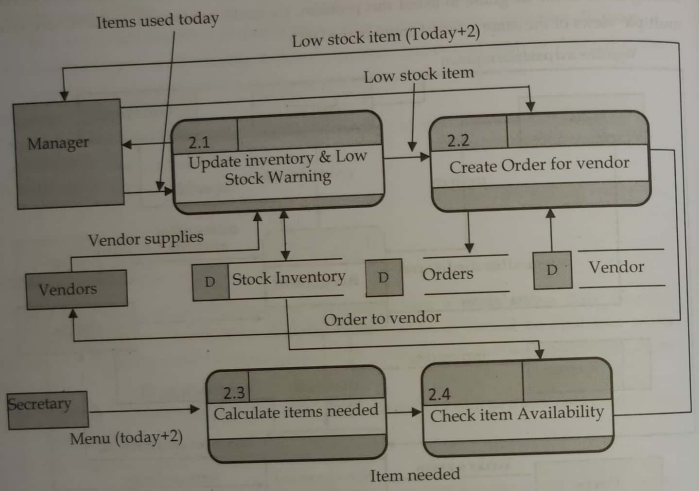
Level 1 DFD

Processes in diagram 0 (with a whole number) can be exploded further to represent details of the processing activities. Example below shows the next level (Diagram 1) of process explosion. Although the following level 1 DFD only has three processes, there are quite a few input and input from the processes to the external entities and that could end up to be a few cross lines among them in the diagram; to avoid this problem, we could use (master and auxiliary view) multiple views of the same external entity in the DFD.



Level 2 DFD

If a process with a lot of data flow linking between a few external entities, we could first extract that particular process and the associated external entities into a separate diagram similar to a context diagram, before you refine the process into a separate level of DFD; and by this way you can ensure the consistency between them much easier.



Logical vs. Physical Data Flow Diagrams

Data flow diagrams are categorized as either logical or physical. A logical data flow diagram focuses on the business and how the business operates. It is not concerned with how the system will be constructed. We can ignore implementation specifics such as, computer configuration; data storage technology, communication or message passing methods by focusing on the functions performed by the system, such as, data collection, data to information transformation and information reporting.

A physical data flow diagram shows how the system will be implemented, including the hardware, software, files, and people in the system. It is developed such that the processes described in the logical data flow diagrams are implemented correctly to achieve the goal of the business.

Benefits of Logical Data Flow Diagram

- A logical diagram is drawn present business information and centered on business activities, which makes it an ideal communication tool when use in communicating with project users.

- Logical DFD is based on business events and independent of particular technology or physical arrangement, which makes the resulting system more stable.
- Logical DFD allows analyst to understand the business being studied and to identify the reason behind implementation plans.
- Systems implemented based on logical DFD will be easier to maintain because business functions are not subject to frequent change.
- Very often, logical DFD does not contain data stores other than files or a database, making less complex than physical DFD and is easier to develop.
- Physical DFD can be easily formed by modifying a logical DFD.

Benefits of Physical Data Flow Diagram

- Clarifying which processes are manual and which are automated: Manual processes require detailed documentation and automated process require computer programs to be developed.
- Describing processes in more detail than do logical DFDs: Describes all steps for processing of data.
- Sequencing processes that have to be done in a particular order: Sequence of activities that lead to a meaningful result are described. For example, update must be performed before a producing a summary report.
- Identifying temporary data storage: Temporary storage such as a sales transaction file for a customer receipt (report) in a grocery store, are described.
- Specifying actual names of files and printouts: Logical data flow diagrams describe actual filenames and reports, so that the programmers can relate those with the data dictionary during the developmental phase of the system.
- Adding controls to ensure the processes are done properly: These are conditions or validations of data that are to be met during input, update, delete, and other processing of data.

LOGIC MODELING

A logic model is a graphic depiction (road map) that presents the shared relationships among the resources, activities, outputs, outcomes, and impact for your program. It depicts the relationship between your program's activities and its intended effects.

Data flow diagrams do not show the logic inside the processes – what occurs within a process? How input data is converted into output information. Logic modeling involves representing internal structure and functionality of processes depicted on a DFD. Processes must be clearly described before translating them into programming language. Logic modeling can also be used to show when processes on a DFD occur. Logic modeling will be generic without taking syntax of a particular programming language. A logic model has four components:

1. **Needs** are about the problem and why it's important to address it. What issue are we trying to address?
2. **Inputs** are the things that contribute to addressing the need (usually a combination of money, time, expertise and resources). What resources are we investing?
3. **Activities** describe the things that the inputs allow to happen. What are we doing with these resources?
4. **Outcomes** are usually expressed in terms of measures of success. What difference are we hoping to make?

Each process on the lowest level DFD will be represented by one or more of the following:

- Structured English
- Decision Tables
- Decision Trees
- State-transition diagrams
- Sequence diagrams
- Activity diagrams

Modeling Logic with Decision Tables

Decision tables are a concise visual representation for specifying which actions to perform depending on given conditions. They are algorithms whose output is a set of actions. A decision table is an excellent tool to use in both testing and requirements management. Essentially it is a structured exercise to formulate requirements when dealing with complex business rules. Decision tables are used to model complicated logic. They can make it easy to see that all possible combinations of conditions have been considered and when conditions are missed, it is easy to see this.

Let's take an example scenario for an ATM where a decision table would be of use.

A customer requests a cash withdrawal. One of the business rules for the ATM is that the ATM machine pays out the amount if the customer has sufficient funds in their account or if the customer has the credit granted. Already, this simple example of a business rule is quite complicated to describe in text. A decision table makes the same requirements clearer to understand:

Conditions	R1	R2	R3
Withdrawal amount<=balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

In a decision table, conditions are usually expressed as true (T) or false (F). Each column in the table corresponds to a rule in the business logic that describes the unique combination of circumstances that will result in the actions. The table above contains three different business rules, and one of them is the "withdrawal is granted if the requested amount is covered by the balance." It is normal to create at least one test case per column, which results in full coverage of all business rules.

Decision tables can be used in all situations where the outcome depends on the combinations of different choices, and that is usually very often. In many systems there are tons of business rules where decision tables add a lot of value.

Steps to create decision tables

- **Step 1 – Analyze the requirement and create the first column**

  Requirement: "Withdrawal is granted if requested amount is covered by the balance or if the customer is granted credit to cover the withdrawal amount". Express conditions and resulting actions in a list so that they are either TRUE or FALSE. In this case there are two conditions, "withdrawal amount ≤ balance" and "credit granted". There is one result, the withdrawal is granted.

Conditions
Withdrawal amount<=balance
Credit granted
Actions
Withdrawal granted

- **Step 2: Add Columns**

  Calculate how many columns are needed in the table. The number of columns depends on the number of conditions and the number of alternatives for each condition. If there are two conditions and each condition can be either true or false, you need 4 columns. If there are three conditions there will be 8 columns and so on. Mathematically, the number of columns is $2^{conditions}$. In this case $2^2 = 4$ columns.

  Number of columns that is needed:

Number of Conditions	Number of Columns
1	2
2	4
3	8
4	16
5	32

The bottom line is that you should create more smaller decision tables instead of fewer larger ones, otherwise you run the risk of the decision tables being so large as to be unmanageable. Test the technique by picking areas with fewer business rules.

Now is the time to fill in the T (TRUE) and F (FALSE) for the conditions. How do you do that? The simplest is to say that it should look like this:

- Row 1: TF
- Row 2: TTFF
- Row 3: TTTTFFFF

For each row, there is twice as many T and F as the previous line.

Repeat the pattern above from left to right for the entire row. In other words, for a table with 8 columns, the first row will read TFTFTFTF, the second row will read TTFFTTFF and the third row will read TTTTFFFF.

Conditions				
Withdrawal amount<=balance	T	F	F	F
Credit granted	T	T	F	F
Actions				
Withdrawal granted				

- **Step 3: Reduce the table**

Mark insignificant values with "-". If the requested amount is less than or equal to the account balance it does not matter if credit is granted. In the next step, you can delete the columns that have become identical.

Conditions				
Withdrawal amount<=balance	T	F	T	F
Credit granted	-	T	-	F
Actions				
Withdrawal granted				

Check for invalid combinations. Invalid combinations are those that cannot happen, for example, that someone is both an infant and senior. Mark them somehow, e.g. with "X". In this example, there are no invalid combinations.

Finish by removing duplicate columns. In this case, the first and third column are equal, therefore one of them is removed.

- **Step 4: Determine actions**

Enter actions for each column in the table. You will be able to find this information in the requirement. Name the columns (the rules). They may be named R1/Rule 1, R2/Rule 2 and so on, but you can also give them more descriptive names.

Conditions			
Withdrawal amount<=balance	T	F	F
Credit granted	-	T	F
Actions			
Withdrawal granted	T	T	F

- **Step 5: Write test cases**

Write test cases based on the table. At least one test case per column gives full coverage of all business rules.

- **Test case for R1:** balance = 200, requested withdrawal = 200. Expected result: withdrawal granted.
- **Test case for R2:** balance = 100, requested withdrawal = 200, credit granted. Expected result: withdrawal granted.
- **Test case for R3:** balance = 100, requested withdrawal = 200, no credit. Expected Result: withdrawal denied.

Advantages and disadvantages of decision tables

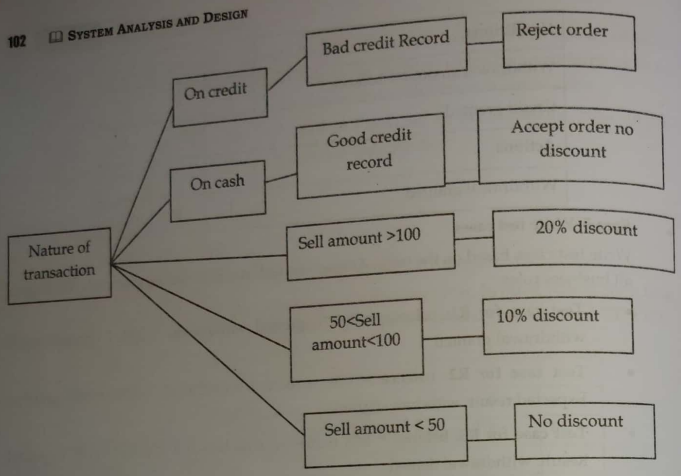
One advantage of using decision tables is that they make it possible to detect combinations of conditions that would otherwise not have been found and therefore not tested or developed. The requirements become much clearer and you often realize that some requirements are illogical, something that is hard to see when the requirements are only expressed in text.

A disadvantage of the technique is that a decision table is not equivalent to complete test cases containing step-by-step instructions of what to do in what order. When this level of detail is required, the decision table has to be further detailed into test cases.

Decision Trees

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on. Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



Advantages of Decision tree

- It expresses the logic of if then else in pictorial form.
- It is useful to express the logic when a value is variable or an action is dependent on nested decision i.e. the outcome of another decision.
- It helps the analyst to identify the actual decision to be made.
- It is used to verify logic and problems that involve a few complex decision and limited number of actions.

Disadvantages of Decision tree

- The lack of decision tree is that there is absence of information in its format to take what other combinations of conditions to test.
- A large number of branches with many paths will confuse rather than help in analysis.

PSEUDO-CODES

Pseudo-code is an informal way of programming description that does not require any strict programming language syntax or underlying technology considerations. It is used for creating an outline or a rough draft of a program. Pseudo-code summarizes a program's flow, but excludes underlying details. System designers write pseudo-code to ensure that programmers understand a software project's requirements and align code accordingly.

It's simply an implementation of an algorithm in the form of annotations and informative text written in plain English. It has no syntax like any of the programming language and thus can't be compiled or interpreted by the computer.

Example: Program to print Fibonacci up to n numbers.

```
Void function Fibonacci
Get value of n;
Set value of a to 1;
Set value of b to 1;
Initialize i to 0
For (i=0; i< n; i++)
{
        If a greater than b
        {
                Increase b by a;
                Print b;
        }
        Else if b greater than a
        {
                Increase a by b;
                Print a;
        }
}
```

Advantages of Pseudo-code

- Improves the readability of any approach. It's one of the best approaches to start implementation of an algorithm.
- Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that's where a pseudo-code proves vital.
- The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

STRUCTURING SYSTEM DATA REQUIREMENTS

INTRODUCTION

Structuring system and database requirements concentrates on the definition, structure and relationships within data. The characteristics of data captured during data modeling are crucial in the design of databases, programs, computer screens and printed reports. This information is essential in ensuring data integrity in an information system. Moreover data are often the most complex aspects of many modern information systems and are reasonably permanent (in contrast to data flows, which are rather dynamic).

The most common format used for data modeling is entity-relationship diagramming. Data modeling explains the characteristics and structure of data independent of how the data may be stored in computer memories and is usually developed iteratively.

Conceptual Data Modeling

A conceptual schema or conceptual data model is a map of concepts and their relationships used for databases. This describes the semantics of an organization and represents a series of assertions about its nature. A conceptual data model is a representation of organizational data. Its purpose is to show as many rules about the meaning and interrelationships among data as possible. It is important that the process, logic and data model descriptions of a system are consistent and complete since each describes different but complementary views of the same information system.

Conceptual data modeling is typically done in parallel with other requirements analysis and structuring steps during systems analysis. On larger systems development teams, a subset of the project team concentrates on data modeling while other team members focus attention on process or logic modeling. Analysts develop a conceptual data model for the current system and then build or refine a purchased conceptual data model that supports the scope and requirements for the proposed or enhanced system. Conceptual data modeling, using either the ER or UML approach, is particularly useful in the early steps of the database life cycle, which involve requirements analysis and logical design. These two steps are often done simultaneously, particularly when requirements are determined from interviews with end users and modeled in terms of data-to-data relationships and process-to-data relationships. The conceptual data modeling step (ER approach) involves the classification of entities and attributes first, then identification of generalization hierarchies and other abstractions, and finally the definition of all relationships among entities. Relationships may be binary (the most common), ternary, and higher-level n-ary. Data modeling of individual requirements typically involves creating a different view for each end user's requirements. Then the designer must integrate those views into a global schema so that the entire database is pictured as an integrated whole. This helps to eliminate needless redundancy—such elimination is particularly important in logical design. Controlled redundancy can be created later, at the physical design level, to enhance database performance.

The process of conceptual data modeling

This process usually begins with developing a conceptual data model for the existing system. Then a new CDM, which includes all of the data requirements for the new system is built. E-R diagrams can be translated into wide variety of technical architectures for data, such as relational, network and hierarchical.

Deliverables and outcomes

The primary deliverable from the conceptual data modeling is the entity-relationship diagram. Another deliverable from CDM is a full set of entries about data objects to be stored in the project dictionary or repository. The repository is the mechanism to link data, process and logic models of an information system.

Gathering Information for Conceptual Data Modeling

A data model explains what the organization does and what rules govern how work is performed in the organization. You typically do data modeling from a combination of perspectives. The first perspective is generally called the top-down approach. This perspective derives the business rules for a data model from an intimate understanding of the nature of the business, rather than from specific information requirements in the computer displays, reports or business forms. The bottom-up approach is on the contrary a process of gaining an understanding of data by reviewing specific business documents handled within the system.

Requirements determination methods must include questions and investigations that take a data, not only a process and logic, focus. For example, during interviews with potential system users—during Joint Application Design (JAD) sessions or through requirements interviews—you must ask specific questions in order to gain the perspective on data that you need to develop or tailor a purchased data model. In later sections of this chapter, we will introduce some specific terminology and constructs used in data modeling. Even without this specific data modeling language, you can begin to understand the kinds of questions that must be answered during requirements determination. These questions relate to understanding the rules and policies by which the area to be supported by the new information system operates. That is, a data model explains what the organization does and what rules govern how work is performed in the organization. You do not, however, need to know (and often can't fully anticipate) how or when data are processed or used to do data modeling.

You typically do data modeling from a combination of perspectives. The first perspective is generally called the top-down approach. This perspective derives the business rules for a data model from an intimate understanding of the nature of the business, rather than from any specific information requirements in computer displays, reports, or business forms. It is this perspective that is typically the basis for a purchased data model.

Introduction to E-R Modeling

The basic E-R modeling notation uses three main constructs: data entities, relationships and their associated attributes. An E-R data model is a detailed, logical representation of the data for an organization or for a business area. An entity is a person, place, object, event, or concept in the user environment about which the organization wishes to maintain data. An entity has its own identity, which distinguishes it from other entities. There is an important distinction between entity types and entity instances. An entity type is a collection of entities that share common properties or characteristics. An entity instance is a single occurrence of an entity type. For example there is usually one EMPLOYEE type but there may be hundreds of instances of this entity type stored in a database. Each entity type has a set of attributes associated with it. For example: for an entity STUDENT we have such attributes like: STUDENT NO, NAME, ADDRESS, PHONE NO. Every entity must have an attribute or set of attributes that distinguishes one instance from other instances of the same type – and that is called a candidate key. A primary key is a candidate key that has been selected to be used as the identifier for the entity type. For each entity the name of the primary key is underlined on an E-R diagram.

Entity Relationship Modeling (ER Modeling) is a graphical approach to database design. It uses Entity/Relationship to represent real world objects. An **Entity** is a thing or object in real world that is distinguishable from surrounding environment. For example each employee of an organization is a separate entity.

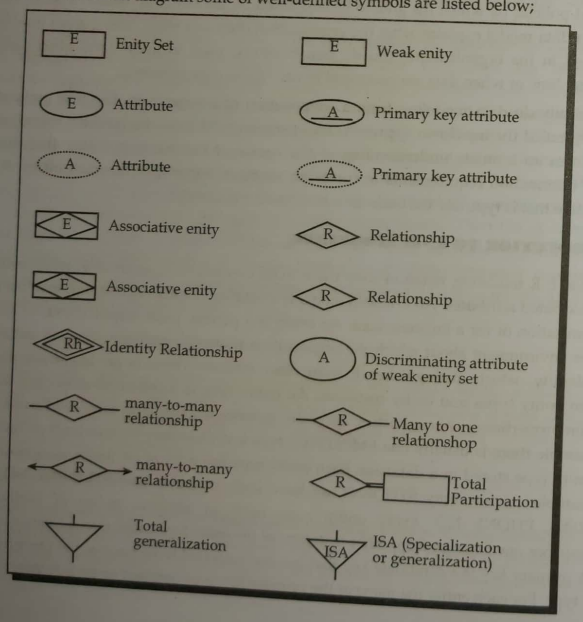
It is a technique used in database design that helps describe the relationship between various entities of an organization. Terms used in E-R model

- **Entity** - It specifies distinct real world items in an application. For example: vendor, item, student, course, teachers, etc.
- **Relationship** - they are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and teaches are relationship.

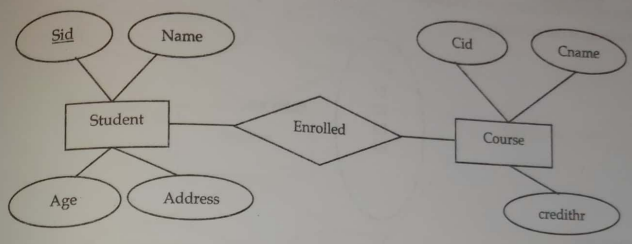
association between two entity sets is called relationship entities.

- **Attributes** - It specifies the properties of relationships. For example, vendor code, student name.

Entity-relationship diagrams (ERD) are essential to modeling anything from simple to complex databases, but the shapes and notations used can be very confusing. There are various types of symbols are used for ER diagram some of well-defined symbols are listed below;



Example:

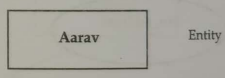


ELEMENTS OF ER DIAGRAM/ER DESIGN ISSUES

ENTITY

An entity is a real-world thing either living or non-living that is easily recognizable and non-recognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world. An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity. Simply, it is something which has real existence. Like tuple1 contains information about Aarav (id, name and Age) which has existence in real world. So the tuple1 is an entity. So we may say each tuple is an entity.

Example: let "Aarav" is a particular member of entity type student.

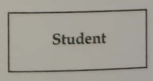


Entity

Entity type

It is collection of entities having common attribute. As in Student table each row is an entity and has common attributes. So STUDENT is an entity type which contains entities having attributes id, name and Age. Also each entity type in a database is described by a name and a list of attribute. So we may say a table is an entity type.

Example: Collection of entities with similar properties

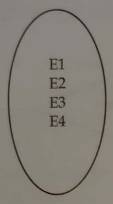


Entity type

Entity set

It is same as an entity type, but defined at a particular point in time, such as students enrolled in a class on the first day. Other examples: Customers who purchased last month, cars currently registered in Florida. A related term is instance, in which the specific person or car would be an

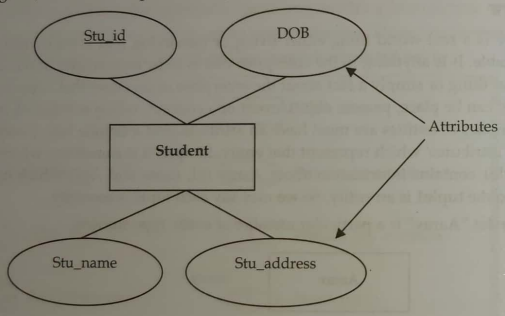
instance of the entity set. **Example**: let E1 is an entity having entity type Student and set of all students is called entity set.



Entity set

Attributes

Attributes are the properties which define the entity type. For example, Student_id, student_name, DOB, Age, Address, Mobile_No etc. are the attributes which defines entity type Student. In ER diagram, attribute is represented by an oval.



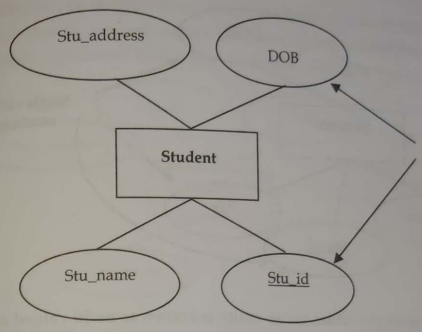
Types of Attributes

Attributes of an entity type can be further divided into following types

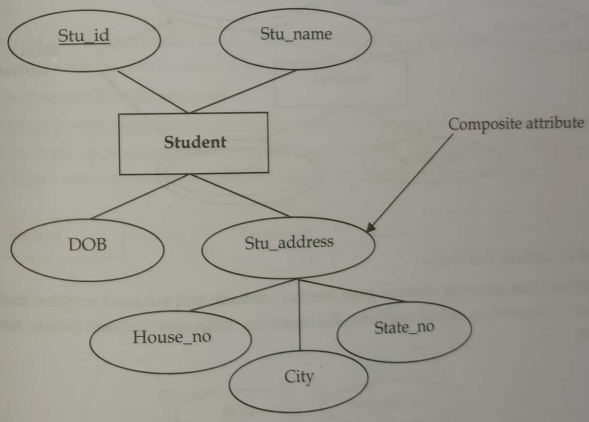
- Atomic vs. composite attributes
- Single valued vs. multi valued attributes
- Stored vs. derived attributes
- NULL value attribute
- Key attribute

Atomic vs. composite attributes

An attribute that cannot be divided into smaller independent attribute is known as atomic attribute. **For example**, assume Student is an entity and its attributes are Name, Age, DOB, Address and Phone no. Here the Stu_id, DOBattributes of student (entity) cannot further divide. In this example Stu_id and DOB are atomic attribute.



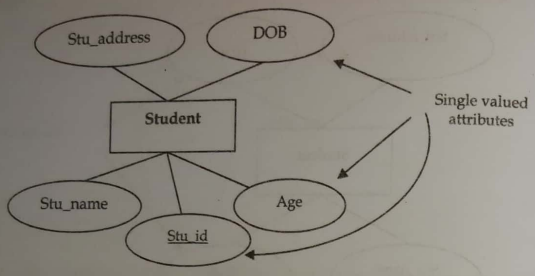
An attribute that can be divided into smaller independent attribute is known as composite attribute. **For example**, assume Student is an entity and its attributes are Stu_id, Name, DOB, Address and Phone no. Here the address (attribute) of student (entity) can be further divide into House no, city and so on. In this example address is **composite** attribute.



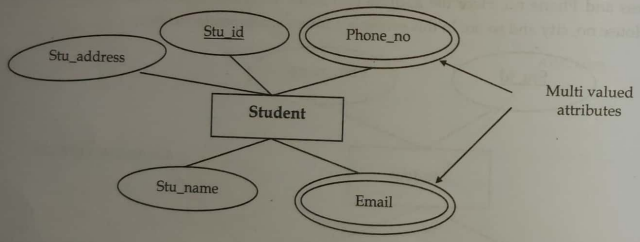
Single valued vs. multi valued attributes

An attribute that has only single value for an entity is known as single valued attribute.

For example, assume Student is an entity and its attributes are Stu_id, Name, DOB, age, Address and Phone no. Here the age (attribute) of student (entity) can have only one value. Here, age is single valued attribute.

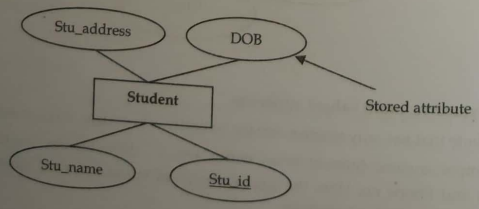


An attribute that can have multiple values for an entity is known as multi valued attribute. **For example**, assume Student is an entity and its attributes are Stu_id, Name, Age, Address and Phone no. Here the Phone no (attribute) of student (entity) can have multiple value because a student may have many phone numbers. Here, Phone no is multi valued attribute.

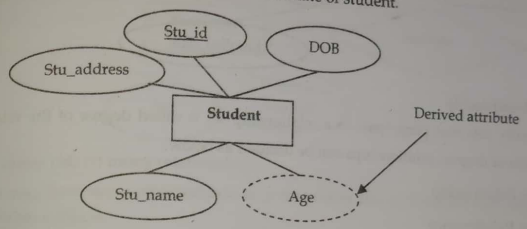


Stored vs. derived Attributes

An attribute that cannot be derived from another attribute and we need to store their value in database is known as stored attribute. For example, birth date cannot derive from age of student.



An attribute that can be derived from another attribute and we do not need to store their value in the database due to dynamic nature is known as derived attribute. It is denoted by doted oval. For example, age cannot derive from birth date of student.



NULL value attribute

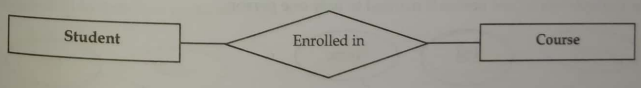
An attribute, which has not any value for an entity is known as null valued attribute. For example, assume Student is an entity and its attributes are Name, Age, Address and Phone no. There may be chance when a student has no phone no. In that case, phone no is called null valued attributes.

Key attribute

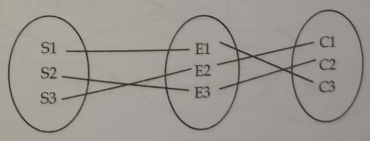
An attribute that has unique value of each entity is known as key attribute. For example, every student has unique roll no. Here roll no is key attribute.

Relationship type and Relationship set

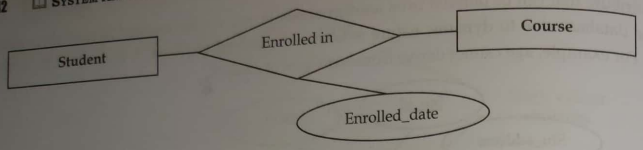
A relationship type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, relationship type is represented by a diamond and connecting the entities with lines.



A set of relationships of same type is known as relationship set. The following relationship set depicts S1 is enrolled in C2, S2 is enrolled in C1 and S3 is enrolled in C3.



In another way, we can say that association between two entity sets is called relationship set. A relationship set may also have attributes called descriptive attributes. For example, the Enrolled_in relationship set between entity sets student and course may have the attribute enrolled_date.



Degree of a Relationship

Number of entity sets that participate in a relationship set is called degree of the relationship set. On the basis of degree, relationships can be divided as below:

- Unary Relationship
- Binary Relationship
- N-ary Relationship

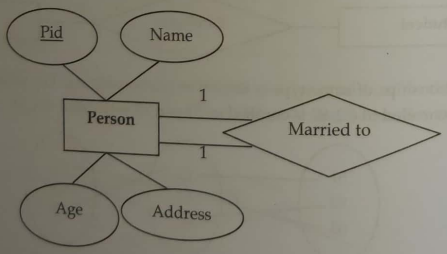
Unary Relationship

If only one entity set participates in a relation, the relationship is called as unary relationship. Here same entity set participates in relationship twice with different roles. Role names are specified above the link joining entity set and relationship set. This type of relationship set is sometimes called a recursive relationship set. There are three types of unary relationships:

- 1:1 unary relationship
- 1:M unary relationship
- M:N unary relationship

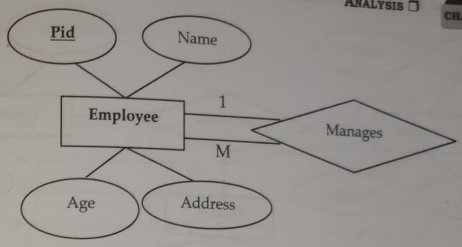
One to one (1:1) unary relationship

In the example below, one person is married to only one person.



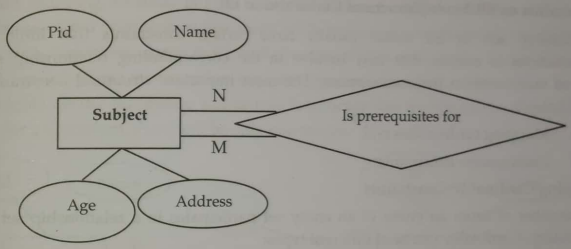
One to many (1: M) unary relationship

An employee may manage many employees but an employee is managed by only one employee. This types of relationship with employee relationship set itself is called 1:M unary relationship as shown in below;



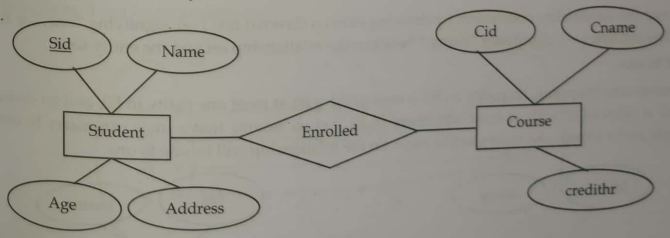
Many to many (M: N) unary relationship

A subject may have many other subjects as prerequisites and each subject may be a prerequisite to many other subjects.



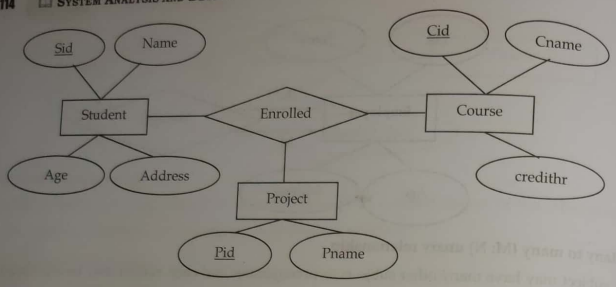
Binary relationship

When there are two entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course. This is the most common type of relationship in database systems.



N-ary relationship

When there are n entities set participating in a relation, the relationship is called as n-ary relationship. If n=1 then it is called unary relationship, if n=2 then it is called binary relationship. Generally in N-ary relationship there are more than two entities participating with a single relationship i.e. n > 2.



Constraints on ER Model/Structural Constraint in ER

Relationship sets in ER model usually have certain constraints that limit the possible combinations of entities that may involve in the corresponding relationship set. Database content must confirm these constraints. The most important structural constraints in ER are listed below:

- Mapping cardinalities and
- Participation constraints

Mapping Cardinality Constraints

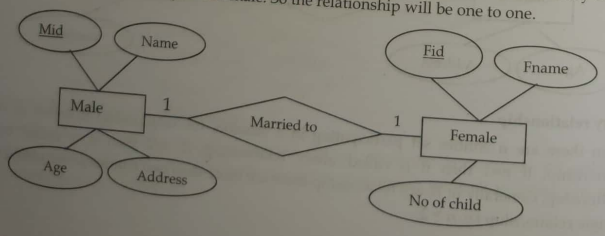
The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

- One-to- One
- One-to- Many
- Many-to-One
- Many-to-Many

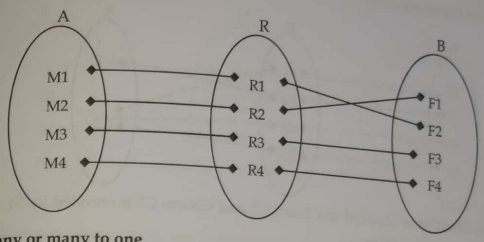
We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.

One to one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1. Let us assume that a male can marry to one female and a female can marry to one male. So the relationship will be one to one.

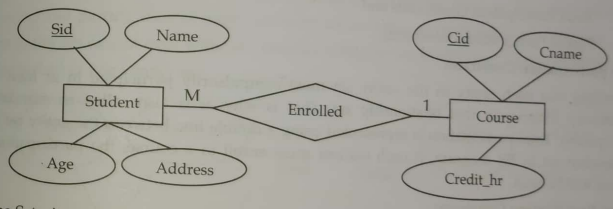


Using Sets, it can be represented as:

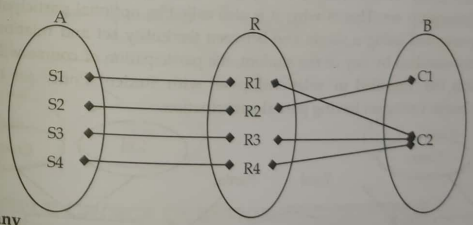


One to many or many to one

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1. In many-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.



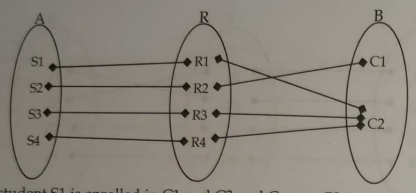
Using Sets, it can be represented as:



Many to many

In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

Using Sets, it can be represented as:



In this example, student S1 is enrolled in C1 and C3 and Course C2 is enrolled by S1, S2 and S4. So it is many to many relationships.

Participation Constraints

Participation Constraint is applied on the entity participating in the relationship set. Constraint on ER model that determines whether all or only some entity occurrences participate in a relationship is called participation constraint. It specifies whether the existence of an entity depends on its being related to another entity via the relationship type. There are two types of participation constraints:

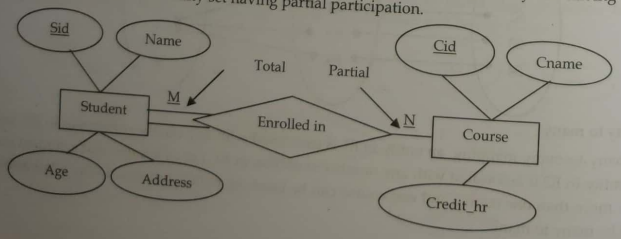
- Total Participation Constraints and
- Partial Participation Constraints.

Total participation Constraint

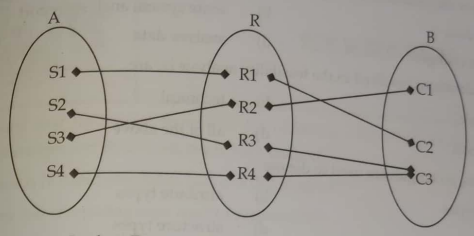
It specifies that each entity in the entity set must compulsorily participate in at least one relationship instance in that relationship set. That is why; it is also called as mandatory participation. Total participation is represented using a double line between the entity set and relationship set in ER diagram. If each student must enroll in a course, the participation of student will be total.

Partial Participation Constraint

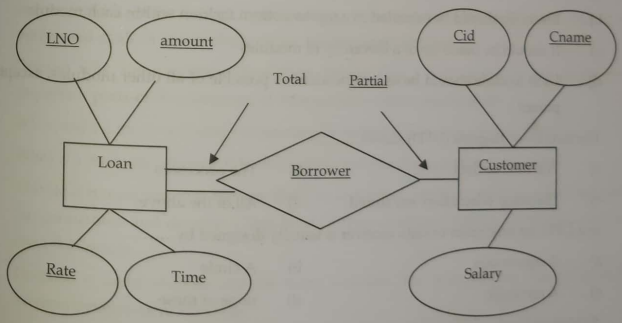
It specifies that each entity in the entity set may or may not participate in the relationship instance in that relationship set. That is why; it is also called as optional participation. Partial participation is represented using a single line between the entity set and relationship set. If some courses are not enrolled by any of the student, the participation of course will be partial. The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



Using set, it can be represented as,



Every student in Student Entity set is participating in relationship but there exists a course C3 which is not taking part in the relationship. Thus participation of student relation with relationship 'Enrolled in' is called total and participation of course relation with given relationship is called partial. Similarly, let's take an another example, consider Customer and Loan entity sets in a banking system, and a relationship set borrower between them indicates that only some of the customers have Loan but every Loan should be associated with some customer. Therefore there is total participation of entity set Loan in the relationship set borrower but participation of entity set customer is partial in relationship set borrower. Here, Loan entity set cannot exist without Customer entity set but existence of Customer entity set is independent of Loan entity set.



MULTIPLE CHOICE QUESTIONS

1. In a DFD external entities are represented by a .....
   a) Rectangle
   b) ellipse
   c) Diamond shaped box
   d) circle

EXERCISE

1. Name four traditional techniques for collecting information during analysis. Give the advantages and disadvantages of each method.
2. What is JRP and who takes part in a JRP session?
3. How has computing been used to support JRP?
4. What benefits do GSS provide?
5. Which types of CASE tools are appropriate for use during requirements determination?
6. What are the advantages and disadvantages of discovery prototyping?
7. Explain how conceptual data modeling is different when you start with a prepackaged data model rather than a clean sheet of paper.
8. List the deliverables from the conceptual data modeling part of the analysis phase of the systems development.
9. How are E-R diagrams similar to and different from decision trees? In what ways are data and logic modeling techniques complementary? What problems might be encountered if either data or logic modeling techniques were not performed well or not performed at all as part of the systems development process?
10. Discuss why some systems developers believe that a data model is one of the most important parts of the statement of information system requirements.
11. When must a many-to-many relationship be modeled as an associative entity?
12. What is a business process? Why is business process diagramming important?
13. Choose a transaction that you are likely to encounter, perhaps ordering a cap and gown for graduation, and develop a high-level DFD or a context diagram. Decompose this to a level-0 diagram.
14. Describe systems analysis and the major activities that occur during this phase of the systems development life cycle.
15. Describe four traditional techniques for collecting information during analysis. When might one be better than another?
16. What is JAD? How is it better than traditional information gathering techniques? What are its weaknesses?

17. Describe how prototyping can be used during requirements determination. How is it better or worse than traditional methods?
18. What are disruptive technologies and how do they enable organizations to radically change their business processes?
19. Why is continual user involvement a useful way to discover system requirements? Under what conditions might it be used? Under what conditions might it not be used?
20. Suppose you were asked to lead a JAD session. List 10guidelines you would follow to assist you in playing the proper role of a JAD session leader.

❑❑❑