Unit 1. Overview of Systems Analysis and Design 4 Hrs.

What is System?

A system is a collection of components (subsystems) that work together to realize some objective. For example, the **library system** contains librarians, books, and periodicals as components to provide knowledge for its members.

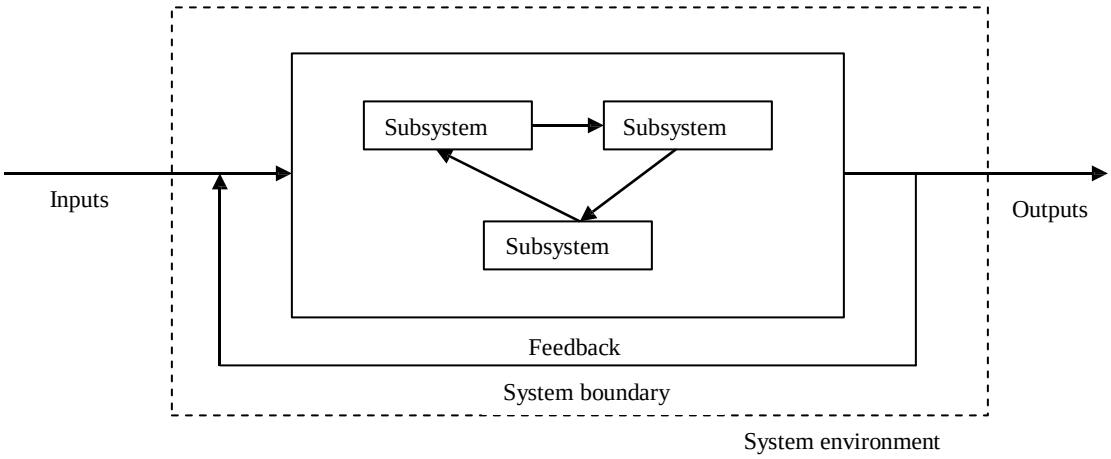


Fig: Basic System Model

Every system has three *activities* or *functions*. These activities are *input*, *processing* and *output*.

- **Input:** It involves capturing and assembling elements that enter the system to be processed. Inputs to the system are anything to be captured by the system from its environment. For example, *raw materials*.
- **Processing:** It involves transformation processes that convert input to output. For example, a *manufacturing process*.
- **Output:** It involves transferring elements that have been produced by a transformation process to their ultimate destinations. Outputs are the things produced by the system and sent into its environment. For example, *finished products*.

The system also includes other two additional activities. These activities include *feedback* and *control*.

- **Feedback:** It is data about the performance of a system. It is the idea of monitoring the current system output and comparing it to the system goal. Any variation from the goal are then fed back in to the system and used to adjust it to ensure that it meets its goal. For example, data about sales performance is feedback to a sales manager.
- **Control:** It involves monitoring and evaluating feedback to determine whether a system is moving toward the achievement of its goals. The control function then makes necessary adjustments to a system's input and processing components to ensure that it produces proper output. For example, a sales manager exercises control

when reassigning salespersons to new sales territories after evaluating feedback about their sales performance.

Theoretical approaches to systems have introduced many generalized *principles*. **Goal setting** is one such principle. It defines exactly what the system is supposed to do. There are principles concerned with system structure and behavior. **System boundary** is one such a principle. This defines the components that make up the system. Anything outside the system boundary is known as **system environment**. A system can be made up of any number of **subsystems**. Each subsystem carries out part of the system function i.e. part of the system goal. The subsystems communicate by passing messages between themselves.

Several systems may share the same environment. Some of these systems may be connected to one another by means of a shared boundary, or **interface**. A system that interacts with other systems in its environment is called **open system**. Finally, a system that has the ability to change itself or environment in order to survive is called an **adaptive system**.

What is an Information System?

In a simplest sense, a system that provides information to people in an organization is called **information system (IS)**.

Information systems in organizations capture and manage data to produce useful information that supports an organization and its employees, customers, suppliers and partners. So, many organizations consider information system to be the essential one.

Information systems produce information by using data about significant *people*, *places*, and *things* from within the organization and/or from the external environment to *make decisions*, *control operations*, *analyze problems*, and *create new products or services*. **Information** is the data shaped into a meaningful form. **Data**, on the other hand, are the collection of raw facts representing events occurring in organizations or the environment before they have been organized and arranged into a form that people can understand and use.

The three activities to produce information in an information system are *input*, *processing*, and *output*. **Input** captures or collects row data from within the organization or from its external environment for processing. **Processing** converts these row data into the meaningful information. **Output** transfers this information to the people who will use it or to the activities for which it will be used. Information systems also require **feedback**, which is used to monitor the current information system output and compare it to the system goal.

The two types of information systems are **formal** and **informal**. **Formal information systems** are based on accepted and fixed definitions of data and procedures for collecting, storing, processing, disseminating, and using these data with predefined rules. **Informal information systems**, in contrast, relay on unstated rules.

Formal information systems can be **manual** as well as **computer based**. **Manual information systems** use paper-and-pencil technology. In contrast, **computer-based information systems (CBIS)** relay on computer hardware and software for processing and disseminating information.

Types of Information Systems

In practice there are several classes of information systems in organizations. Each class serves the needs of different types of users. These are:

1. Transaction processing system (TPS)
2. Management information system (MIS)
3. Decision support system (DSS)
4. Executive information system (EIS)
5. Expert system
6. Communication and collaboration system
7. Office automation system.

(1)Transaction Processing Systems (TPSs)

These are the computerized systems that perform and records the daily routine transactions necessary to conduct business. These systems serve the operational level of the organization. Some examples include sales order entry, hotel reservation systems, payroll, employee record keeping, and shipping.

Transaction processing systems are central to a business. TPS failure for a few hours can cause a firm's demise and perhaps other firms linked to it. Managers need TPS to monitor the status of internal operations and the firm's relations with external environment. TPS are also major producers of information for the other types of systems.

Online transaction processing systems (OLTPS) is an interactive data processing system that involves a direct connection between TPS programs and users. As soon as a single transaction is entered into a computer system, the program interacts immediately with the user for that transaction. It is often known as the live system where there is no time lag between data creation and its processing. A good example of this system is online ticket reservation system.

(2)Management Information Systems (MISs)

These are the information systems at the management level of an organization and serve management-level functions like planning, controlling, and decision-making. These systems provide reports that are usually generated on a predetermined schedule and appear in prearranged format. Typically, these systems use internal data provided by the transaction processing systems. These systems are used for structured decision-making and in some cases for semi-structured decision making as well. Salary analysis and sales reporting are the examples in which MIS can be used.

(3)Decision Support Systems (DSSs)

These systems also serve at the management level of the organization. These systems combine data and sophisticated analytical models or data analysis tools to support semi-structured and unstructured decision-making. These systems use internal information from TPS and MIS, and often information from external sources, such as current stock prices or product prices of competitors. DSS have more analytical power than other systems. Contract cost analysis is an example in which DSS can be used.

(4)Executive Information Systems (EISs)

These systems are also called **executive support systems (ESSs)** and serve the strategic level of the organization. These systems are designed to address unstructured decision making through advanced graphics and communication. These systems incorporate data about external events such as new tax laws or competitors, but they also draw

summarized information from internal MIS and DSS.

These systems are not designed to solve a specific problem but they provide a generalized computing and telecommunication capacity that can be applied to a changing array of problems. 5-year operating plan is an example in which EIS can be used.

(5)Expert Systems

An expert system is an extension of DSS that captures and reproduces the knowledge and expertise of an expert problem solver or decision maker and then simulates the "thinking" or "actions" of that expert. These systems imitate the logic and reasoning of the experts within their respective fields.

Expert systems are implemented with artificial intelligence (AI) technology that captures, stores, and provides access to the reasoning of the experts.

(6)Communication and Collaboration Systems

These systems enable more effective communications between workers, partners, customers and suppliers to enhance their ability to collaborate. These systems use network technology that allows companies to coordinate with other organizations across great distances. These systems create new efficiencies and new relationships between an organization, its customers and suppliers, and business partners redefining organizational boundaries.

(7)Office Automation Systems

Office automation (OA) is more than word processing and spreadsheet applications. Office automation systems support the wide range of business office activities for improved work flow and communication between workers, regardless of whether or not those workers are located in the same office.

Office automation functions include word processing, spreadsheet applications, electronic mails, work group computing, fax processing, work flow management etc.

Office automation systems can be designed to support both individuals and work groups. **Personnel information systems** are those designed to meet the needs of a single user. They are designed to boost an individual's productivity. **Work group information systems**, on the other hand, are designed to meet the needs of a work group. They are designed to boost the group's productivity.

Systems Analysis and Design

System analysis and design is a complex, challenging, and simulating organizational process that a team of business and systems professionals uses to develop and maintain computer-based information systems. It is an organizational improvement process. Information systems are built and rebuilt for organizational benefits.

An important (but not the only) result of system analysis and design is **application software** i.e. software designed to support organizational functions or processes such as inventory management, payroll, or mark-sheet analysis. In addition to application software, the total information system includes the hardware and systems software on which the application software runs, documentation and training materials, the specific

job roles associated with the overall system, controls and the people who use the software along with their work methods.

In systems analysis and design, we use various *methodologies*, *techniques* and *tools* that have been developed, tested, and widely used over the years to assist people during system analysis and design.

Methodologies are comprehensive, multistep approaches to systems development that will guide your work and influence the quality of your final product: the information system. Methodologies use a standard set of steps. A methodology adopted by an organization will be consistent with its general management style. Most methodologies incorporate several development techniques.

Techniques are particular processes that will help to ensure that your work is well thought-out, complete, and comprehensible to other on the project team. Techniques also provide support for a wide range of tasks like conducting interviews, planning and managing the activities in a system development project, diagramming the system's logic, and designing the reports that the system will generate.

Tools are typically computer programs that make it easy to use and benefit from the techniques and to faithfully follow the guidelines of the overall development methodology.

To be effective, both techniques and tools must be consistent with an organizations system development methodology. These make easy for system developers to conduct the steps in methodology.

Importance of Systems Analysis and Design

Systems analysis and design is the collection of important activities that takes place when new information systems are being built or existing ones are changed. All the activities are needed to build good information systems. The systems developed by using systems analysis and design activities fulfill the requirements of organizations' personnel.

Furthermore, we can develop information systems easily and rapidly because there are lots of supporting methodologies, tools, and techniques. The information system can be built in the most effective way. The systems also fit into an existing environment and will be very easy to use and maintain. By following the activities involved in systems analysis and design, we can develop high quality information system within allocated budget and time.

Information System Stakeholders

A stakeholder is any person who has an interest in an existing or proposed information system. She/he may be *technical* or *non-technical* and *internal* or *external* worker. Stakeholders are also called **information workers**. An information worker involves in *creating*, *collecting*, *processing*, *distributing* and *using* information.

There are **six groups** of stakeholders and each group has a different role in the same information system. But in practice, any individual person may play more than one role. For example, a system analyst may also work as a system designer. The six groups are:

1. System owners
2. System users
3. System designers

4. System builders
5. System analysts and project managers
6. Information technology vendors and consultants.

(1)System owners

System owners are the information system's sponsors and chief advocates. They are usually responsible for funding the project of development, operate, and maintain the information system. They are interested with-how much will the system cost? And how much value or what benefit will the system return to the business?

Every information system has one or more system owners. They usually come from the ranks of managers to supervisors.

(2)System Users

These are the people who use or are affected by the information system on a regular basis. They are concerned with the system's functionality related with their jobs and the system's ease of learning and use. A system user may capture, validate, enter, respond, store and exchange data and information. System users are also called **clients**. To know business requirements, discussions with most users need to be kept.

(3)System Designers

These are technology specialists who translate system users' business requirements and constraints into technical solutions. These are interested in information technology choices and the design of systems within the constraints of the chosen technology. They design the computer database, inputs, outputs, screens, networks, and programs that will meet the system users' requirements. These designs guide the construction of the final system.

(4)System Builders

These are also technology specialists who construct information systems and components based on the design specifications generated by the system designer.

(5)Systems Analysts and Project Managers

A. Systems Analyst: Although, many people in organizations are responsible for systems analysis and design, in most organizations the systems analyst has the primary responsibility. The primary role of a systems analyst is to study the problems and needs of an organization in order to determine how people, methods and information technology can best be combined to bring about improvements in the organization. System analysts identify and validate problems and needs and ensure that the technical solution fulfills these problems and needs.

Systems analysts study the system and identify and validate its problems and needs for system owners and users and ensure that the technical solution fulfills the business needs.

B. Project Manager: To build a good information system and applications all the stakeholders must work together as a team. Teams require leadership. For this reason, usually one or more of these stakeholders takes on the role of project manager to ensure that systems are developed on time, within budget and acceptable quality. So, project manager is responsible for planning, monitoring, and controlling projects with

respect to schedule, budget, deliverables, customer satisfaction, technical standards and system quality.

(6)Information Technology Vendors and Consultants

Most information systems are dependent on information technology that must be selected, installed and customized, integrated into business, and technically supported. This technology is developed, sold, and supported by IT vendors.

Similarly, many businesses rely on external consultants to help them develop or acquire information systems and technology. The use of consultants may be driven by the need for specialized knowledge or skills or by an immediate need to complete a project.

System Development Life Cycle (SDLC)

Most organizations use a standard set of steps, called a **systems development methodology** to develop and support their information systems. It is a standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems. And **systems development life cycle (SDLC)** is the traditional methodology used to develop, maintain, and replace information systems. It includes different phases as shown in the figure below. This representation of SDLC is sometimes referred to as the **waterfall model** or **classic life cycle**.

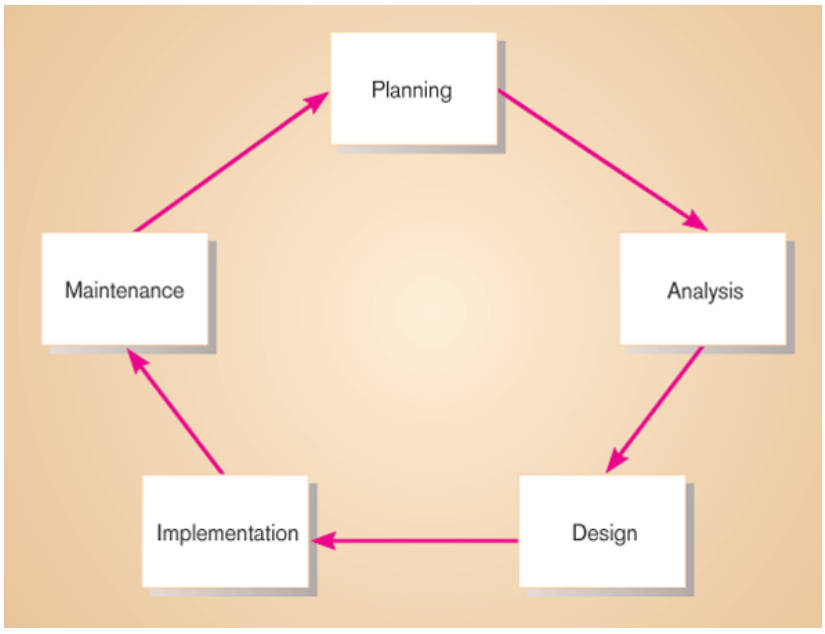


Fig: The systems development life cycle

The first phase is called **planning**. In this phase, someone identifies the need for a new or enhanced system. These needs are then analyzed, prioritized and arranged into a plan for the IS department. Here, a potential information systems project is explained and an argument for continuing or not continuing with the project is presented; a detailed plan is also developed for conducting the remaining phases or the SDLC for the proposed system.

The next phase is called **analysis**. During this phase, the analyst studies the current system and proposes alternative replacement systems. Here, the analyst thoroughly studies the organization's current procedures and the information systems used to perform organizational tasks. The analyst work with users to determine what the users want from a proposed system. The analyst carefully studies any current systems, manual and computerized, that might be replaced or enhanced as part of this project. The analyst studies the requirements and structures them according to their interrelationships and eliminates any redundancies; generates alternative initial designs to match the requirements; compare these alternatives to determine which best meets the requirements within the cost, labor, and technical levels the organization is willing to commit to the development process. The output of this phase is a description of the recommended alternative solution. Once the recommendation is accepted by owners, you can begin to make plans to acquire any hardware and system software necessary to build or operate the system as proposed.

The next phase is called **design**. During this phase, you convert the description of the recommended alternative solution into logical and then physical system specification. Here, you must design all aspects of the system form input and output screens to reports, databases, and computer processes. **Logical design** is the part of the design process that is independent of any specific hardware or software platform. Theoretically, the system could be implemented on any hardware and systems software. **Physical design** is the part of the design phase in which the logical specifications of the system form logical design are transformed into technology-specific details from which all programming and system construction can be accomplished.

The next phase is called **implementation**. In this phase, the information system is coded, tested, installed, and supported in the organization. During coding, programmers write the programs that make up the information system. During testing, programmers and analysts test individual programs and the entire system in order to find and correct errors. During installation, the new system becomes a part of the daily activities of the organization. Implementation activities also include initial user support such as the finalization of documentation, training programs, and ongoing user assistance.

The final phase of SDLC is called **maintenance**. In this phase, information system is systematically repaired and improved. When a system is operating in an organization, users sometimes find problems with how it works and often think of better ways to perform its functions. Also the organization's needs with respect to the system change over time. In maintenance, you make the changes that users ask for and modify the system to reflect changing business conditions.

Waterfall model is the oldest and the most widely used paradigm for information systems development. While it does have weaknesses, it is significantly better than a haphazard approach. This model is suitable for the projects in which user requirements are certain and precise. The problems that are sometimes encountered with the linear sequential model are:

→ Changes can cause confusion as the project team proceeds.

→ It is often difficult for the customer to state all requirements explicitly. The linear sequential model requires this and makes difficulty to respond to changing customer requirements.

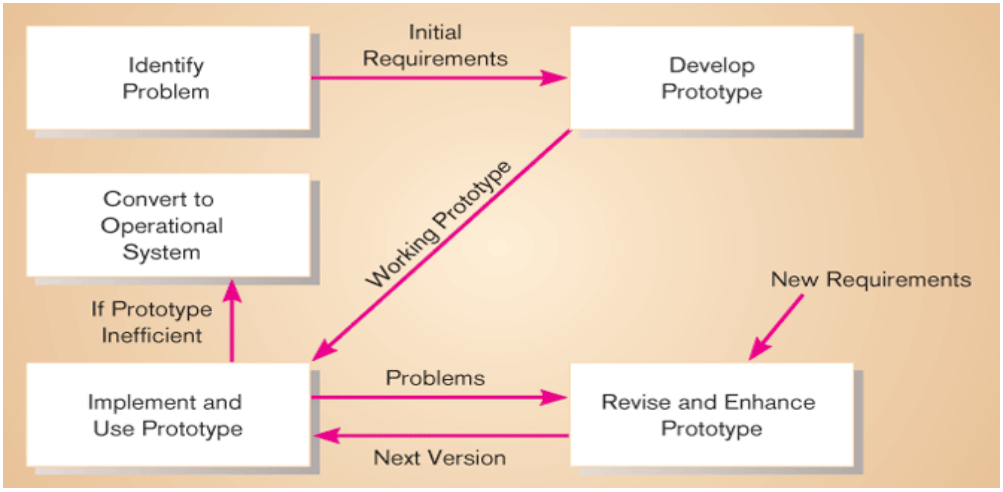
→ A working version of the system will be available to customers late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.
→ The linear nature of the classic life cycle leads to "blocking states" in which some project team members must wait for other members of the team to complete dependent tasks.
→ User involvement is limited.

Different Approaches to Improving Information Systems Development

Several different approaches have been developed in the continuous effort to improve the systems analysis and design process. The two important approaches are **prototyping** and **joint application development (JAD)**.

Prototyping

Prototyping is a form of **rapid application development (RAD)**. Prototyping is a *rapid*, *iterative*, and *incremental* process of systems development in which requirements are converted to a working system that is continually revised through close work between the development team and the users. We can build a prototype with any computer language or development tool, but special prototyping tools have been developed to simply the process. A prototype can be developed with some fourth-generation language (4GL), with the query and screen and report design tools of a database management system, and with tools called *computer-aided software engineering (CASE)* tools.



In prototyping, the analyst works with users to determine the initial or basic requirements for the system. The analyst then quickly builds a prototype. When the prototype is completed, the users work with it and tell the analyst what they like and do not like about it. The analyst uses this feedback to improve the prototype and takes the new version back to the users. This iterative process continues until the users are relatively satisfied with what they have seen.

Ideally, the prototype serves as a mechanism for identifying information system requirements. In this case, we throw away the prototype (also called **throwaway prototype**) after identifying requirements. The actual information system is developed with an eye toward quality and maintainability based on the requirements.

- **Advantages:**
  - → Useful for projects in which user requirements are uncertain or imprecise.
  - → It encourages active user and management participation.
  - → Projects have higher visibility and support because of the extensive user involvement.
  - → Users and management see working, software based solutions more rapidly.
  - → Errors and omissions tend to be detected earlier in prototypes.
  - → Testing and training are natural by-products.
  - → It is more natural process.
  - → It is most popular for small to medium-size projects.
- **Disadvantages:**
  - → It increases lifetime cost to operate, support and maintain the system.
  - → It can solve the wrong problems since problem analysis is abbreviated or ignored.
  - → The product may have less quality because of speed in development.

Joint Application Development (JAD)

It is used for collecting information system requirements and reviewing system designs. It is a structured process in which users, managers, and analysts work together for several days in a series of intensive structured meetings run by a JAD session leader to specify or review system requirements. Here, people work together to agree on system requirements and design details, time and organizational resources are better managed. Group members are more likely to develop a shared understanding of what the IS is supposed to do.

Computer-aided Software Engineering (CASE) Tools

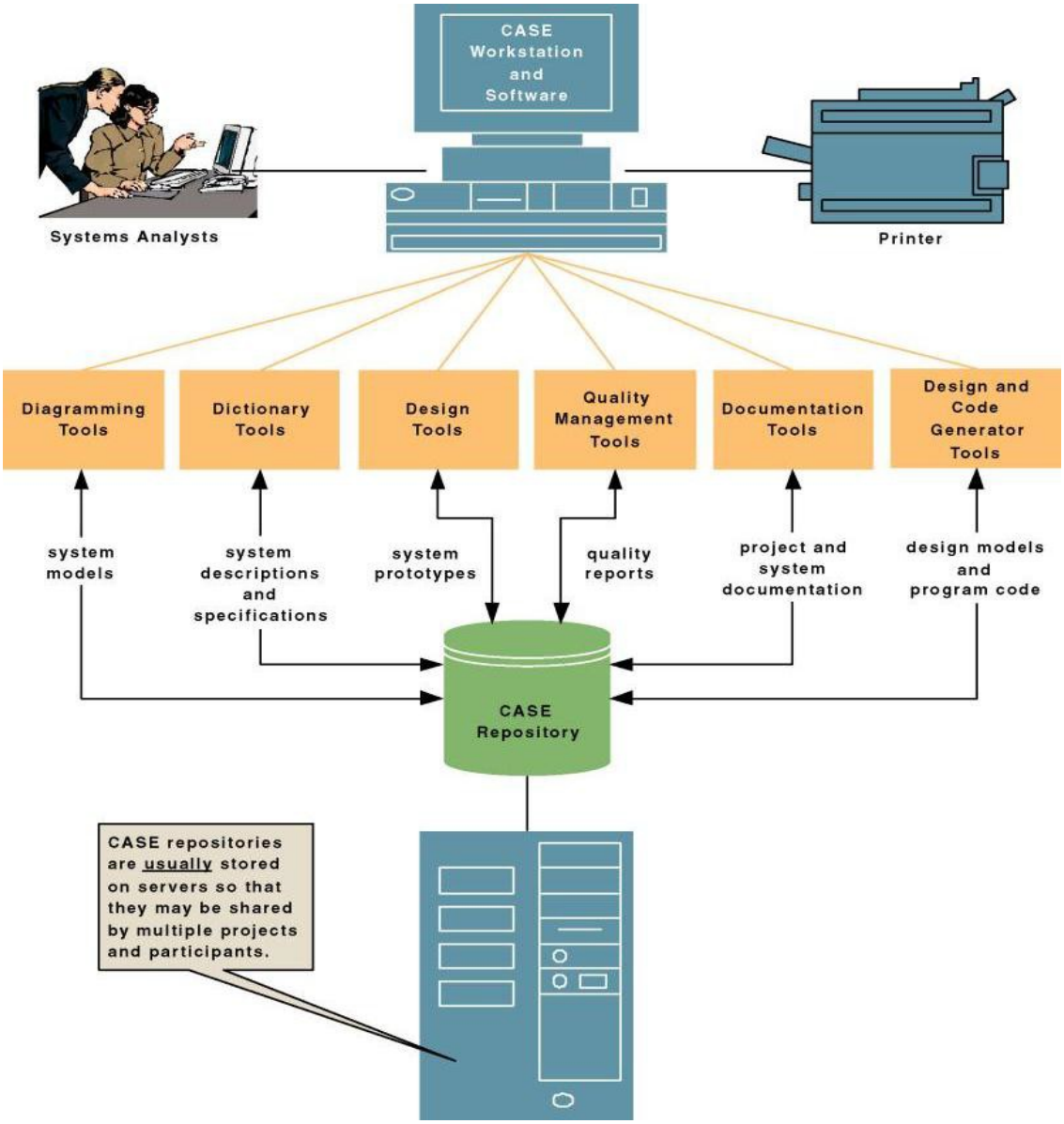
Computer-aided systems engineering (CASE) tools are the software programs that help the development team do their jobs more efficiently and more effectively. These tools support the drawing and analysis of system models. Some CASE tools also provide prototyping and code generation capabilities. Some examples are: Oracle's Designer 2000, Rational's Rose, Platinum's Erwin, Popkin's System Architect 2001, and Visible System's Visible Analyst.

At the center of any CASE tool's architecture is a developer's database called a *CASE repository*. **CASE repository** is a system developer's database where developers can store system models, detailed description and specification, and other products of system development. It is also called **dictionary** or **encyclopedia**.

Around the CASE repository is a collection of tools or facilities for creating system models and documentation. These facilities generally include:

- **Diagramming tools –**These tools are used to draw system models.
- **Dictionary tools** – These tools are used to record, delete, edit, and output detailed documentation and specification.

- **Design tools** – These tools are used to construct system components including system inputs and outputs. These are also called **prototyping tools**.
- **Documentation tools** – These tools are used to assemble, organize, and report on



system models, descriptions and specifications, and prototypes.

- **Quality management tools** – These tools are used to analyze system models, descriptions and specifications, and prototypes for completeness, consistency, and conformance to accepted rules of methodologies.
- **Design and code generator tools** – These tools automatically generate database designs and application programs or significant portions of those programs.

Today's CASE tools provide two distinct ways to develop system models – *forward engineering* and *reverse engineering*. **Forward engineering** requires the system analyst to draw system models, either from scratch or from templates. The resulting models are

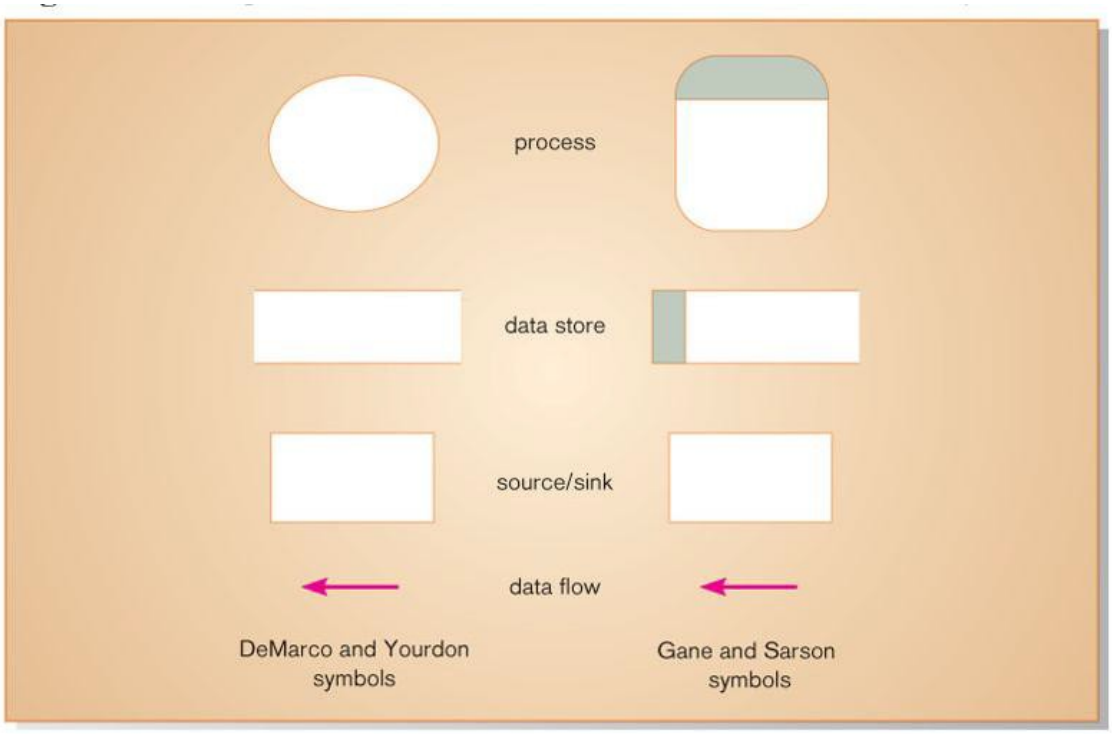
subsequently transformed into program code. **Reverse engineering**, on the other hand, allows a CASE tool to read existing program code and transform that code into a representative system model that can be edited and refined by the systems analyst. CASE tools that allow for bi-directional, forward and reverse engineering are said to provide for "round-trip engineering". The figure below shows CASE tool architecture.

Unit 2. Modeling Tools for Systems Analyst **5 Hrs.**

Data Flow Diagram (DFD)

Process modeling involves graphically representing the functions or processes that capture, manipulate, store, and distribute data between a system and its environment and between components within a system. A common form of a process model is a **data flow diagram (DFD)**.

A **data flow diagram (DFD)** is a tool that depicts the flow of data through a system and the work or processing performed by that system. It is also called **bubble chart**, **transformation graph**, or **process model**. There are two different sets of data flow diagram symbols, but each set consists of four symbols that represent the same things: **data flows**, **data stores**, **processes**, and **sources/sinks** (or **external entities**). The figure below shows two different sets of symbols developed by DeMacro and Yourdon and Gane and Sarson. The set of symbols we will use here was deviced by Gane and Sarson.



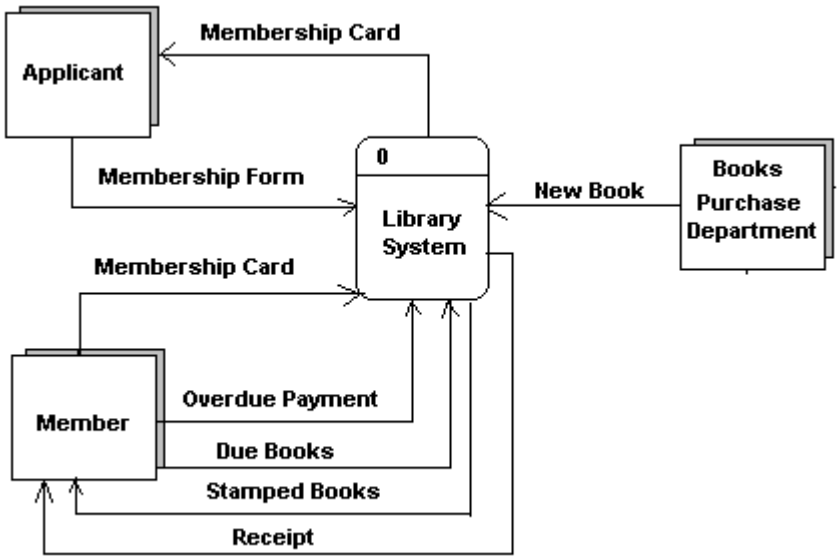
According to *Gane and Sarson*, **rounded rectangles** represent *process* or *work* to be done, **squares** represent *external agents*, **open-ended boxes** represent *data store* (sometimes called *files* or *databases*), and **arrows** represent *data flows* or *inputs* and *outputs* to and from the processes.

Process is the work or actions performed on data so that they are transformed, stored or distributed. *Data store* is the data at rest (inside the system) that may take the form of many different physical representations. External entity (source/sink) is the origin and/or destination of data. Data flow represents data in motion, moving from one place in a system to another.

Developing DFDs

The highest-level view of a system is called **context diagram** or **context DFD**. A context DFD is used to document the scope of the project of development. It is also called **environmental model**.

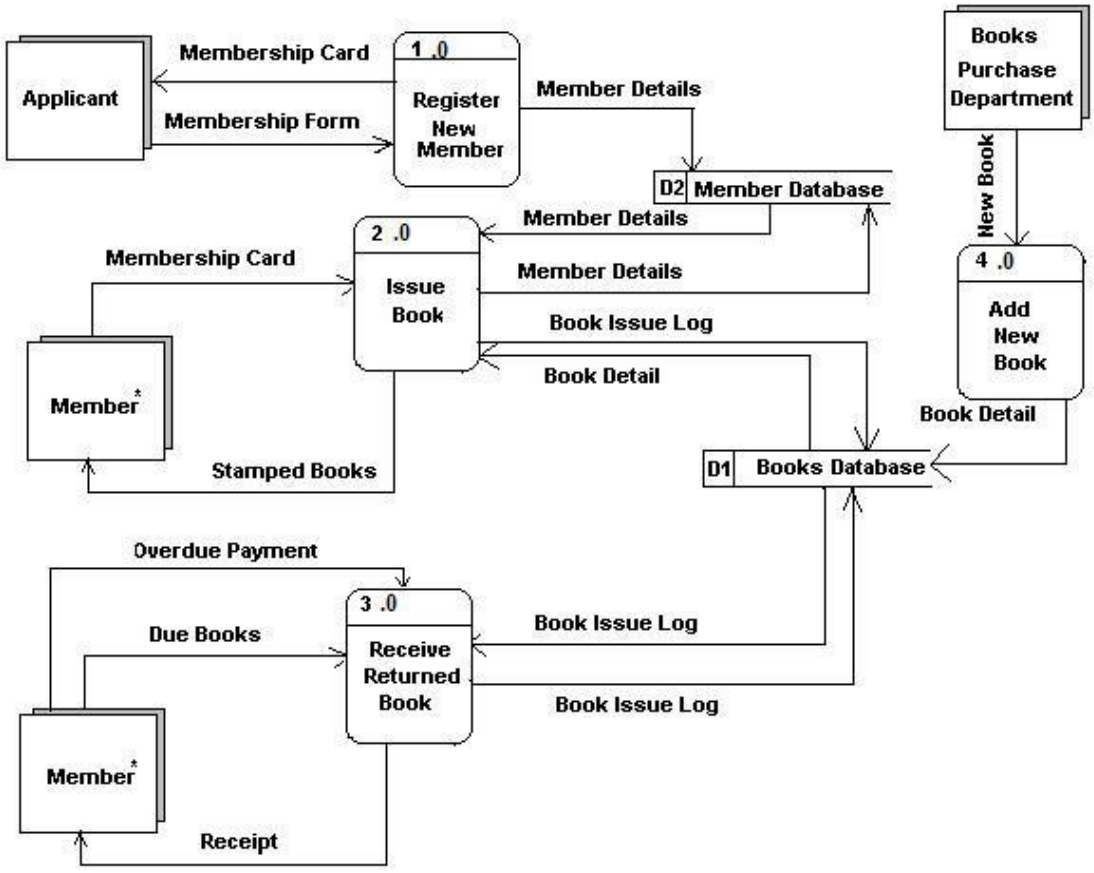
A project's scope defines what aspect of the business, an information system or application is supposed to support and how the system being modeled must interact with other systems and the business as a whole. Because the scope of any project is always subject to change, the context diagram is also subject to constant change. The context DFD contains one and only one process and it has no data storage. Sometimes this process is identified by the number "0". The figure below shows the context diagram of library system.



The next step is to draw a DFD with major processes that are represented by the single process in the context diagram. These major processes represent the major functions of the system. This diagram also includes data stores and called **level-0 diagram** (or **level-0 DFD**). A level-0 diagram is a DFD that represents a system's major processes, data flows, and data stores at a high level of detail. In this diagram, sources/sinks should be same as in the context diagram. Each process has a number that ends in .0. The figure below shows level-0 DFD.

Here, we started with a high-level context diagram. Upon thinking more about the system, we saw that the larger system consisted of four processes. This process is called **functional decomposition**. Functional decomposition is an iterative process of breaking the description or perspective of a system down into finer and finer detail. When we repeat the decomposition until we reached the point where no subprocesses can logically be broken down any further, we get the lowest level of DFD called **primitive DFD**.

When we decompose level-0 DFD, we get level-1 DFD. In level-1 DFD, we label subprocesses of process 1.0 to 1.1, 1.2, and so on. Similarly, subprocesses of process 2.0 to 2.1, 2.2, and so on. In general, a **level-n diagram** is a DFD that is generated from **n** nested decompositions from a level-0 diagram.



Entity Relationship Diagram (E-R Diagram)

During *analysis* phase, a systems analyst uses **entity relationship data model (E-R model)** as a **conceptual data model**. A **conceptual data model** is a detailed model that captures the overall structure of organizational data while being independent of any database management system or other implementation consideration. And an **E-R model** is a detailed, logical representation of the data for an organization or for a business area. The E-R model is expressed in terms of entities in the business environment, the relationships or associations among those entities, and the attributes or properties of both the entities and their relationships. An E-R model is normally expressed as an **entity relationship diagram (E-R diagram)**, which is a graphical representation of an E-R model. It has three basic concepts: **entities**, **attributes**, and **relationships**.

Entities:

An entity is a person, place, object, event, or concept in the user environment about which the organization wishes to capture and store data. An **entity type** (sometimes called an **entity class** or **entity set**) is a collection of entities that share common properties or characteristics. An **entity instance** (or **instance**) is a single occurrence of an entity type.

Each entity type in E-R diagram is given a name. When naming entity types, we should use the following guidelines:

- An entity type name is a *singular noun* like CUSTOMER, STUDENT, or AUTOMOBILE.

- An entity type name should be *descriptive and specific to the organization* like PURCHASE ORDER for orders placed with suppliers to distinguish it from CUSTOMER ORDER for orders placed by customers.
- An entity type name should be *concise* like REGISTRATION for the event of a student registering for a class rather than STUDENT REGISTRATION FOR CLASS.
- *Event entity types* should be named for the *result of the event*, not the activity or process of the event like the event of a project manager assigning an employee to work on a project results in an ASSIGNMENT.

Each entity type in E-R diagram should be defined. When defining entity types, we should use the following guidelines:

- An entity type definition should include a statement of *what the unique characteristic(s) is (are) for each instance*.
- An entity type definition should make clear *what entity instances are included and not included* in the entity type.
- An entity type definition often includes a description of *when an instance of the entity type is created or deleted*.
- For some entity types the definition must specify w*hen an instance might change into an instance of another entity type*. For example, a bid for a construction company becomes a contract once it is accepted.
- For some entity types the definition must specify *what history is to be kept about entity instances*.

An entity type in E-R diagram is drawn using **rectangle**. This shape represents all instances of the named entity. We place entity type name inside the rectangle. For example, the figure below shows STUDENT entity type.

Attributes:

Each entity type has a set of attributes associated with it. An attribute is a property or characteristic of an entity that is of interest to the organization. For example, STUDENT entity type may have Student_ID, Student_Name, Home_Address, Phone_Number, and Major as its attributes. Similarly, EMPLOYEE entity type may have Employee_ID, Employee_name, and Skill, Address as its attributes.

Each attribute in E-R diagram is given a name. When naming attributes, we should use the following guidelines:

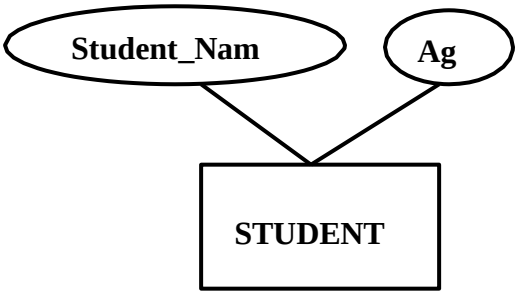
- An attribute is a *noun* like Customer_ID, Age, or Skill.
- An attribute name should be *unique*. No two attributes of the same entity type may have the same name, and it is desirable, for clarity purposes, that no two attributes across all entity types have the same name.
- To make an attribute unique and clear, *each attribute name should follow a standard form*. For example, Student_GPA as opposed to GPA_of_Students.
- Similar attributes of different entity types should use the similar but distinguishing names. For example, Student_Residence_City_Name for STUDENT entity type and Faculty_Residence_City_Name for FACULTY entity type.

Each attribute in E-R diagram should be defined. When defining attributes, we should use the following guidelines:

- An attribute definition states *what the attribute is and possibly why it is important*.

- An attribute definition should make it clear *what is included and what is not included* in the attributes value. For example, Employee_Monthly_Salary_Amount is the amount paid each month exclusive of any benefits, bonuses, or special payments.
- An attribute definition may contain any *aliases* or alternative names.
- An attribute definition may state *the source of values for the attribute to make the meaning clearer*.
- An attribute definition should indicate *if a value for the attribute is required or optional* (to maintain data integrity).
- An attribute definition may indicate *if a value for the attribute may change* (to maintain data integrity).
- An attribute definition may also indicate any *relationships that an attribute has with other attributes*. For example, Age is determined from for Date_of_Birth.

An attribute in E-R diagram is drawn using an **ellipse**. We place attribute name inside the ellipse with a line connecting it to the associated entity type. For example, the figure below shows Student_Name and Age attributes of STUDENT entity type.



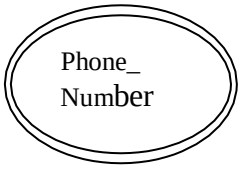
Every entity type must have an attribute or set of attributes that distinguishes one instance from other instances of the same type. A **candidate key** is an attribute or combination of attributes that uniquely identifies each instance of an entity type. For example, a candidate key for a STUDENT entity type might be Stident_ID.

Some entity type may have more than one candidate key. In such a case, we must choose one of the candidate keys as the identifier. An **identifier** (or **primary key**) is a candidate key that has been selected to be used as the unique characteristic for an entity type. We can use the following selection rules to select identifiers:

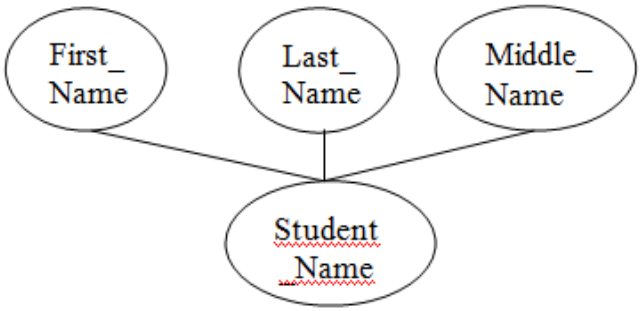
- Choose a candidate key that will not change its value over the life of each instance of the entity type.
- Choose a candidate key that will never be null.
- Avoid using *intelligent keys*.
- Consider substituting single value *surrogate keys* for large composite keys.

The name of the identifier is underlined on an E-R diagram. For example, the figure below shows Student_Id as an identifier for a STUDENT entity type.

A **multivalued attribute** may take more than one value for each entity instance. For example Phone_Number attribute of STUDENT entity type. We use a double-lined ellipse to represent multivalued attribute.



An attribute that has meaningful component parts is called **composite attribute**. For example, Student_Name attribute of STUDENT entity type has First_Name, Middle_Name, and Last_Name as its component parts.



An attribute whose value can be computed from related attribute values is called **derived attribute**. For example, value of Age attribute is computed from Date_of_Birth attribute. We use dashed ellipse to denote derived attribute.

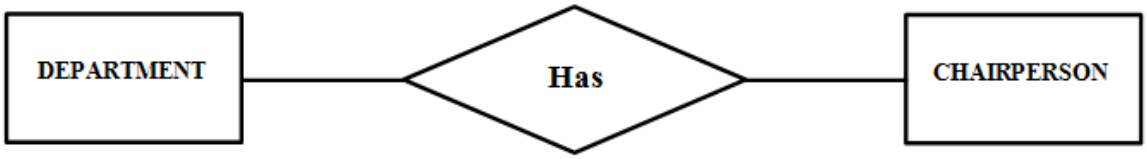
Relationships

A **relationship** is an association between the instances of one or more entity types that is of interest to the organization. We use **diamond** to denote relationships. Relationships are labeled with *verb phrases*. For example, if a training department in a company is interested in tracking with training courses each of its employees has completed, this leads to a relationship (called Completes) between the EMPLOYEE and COURSE entity types as shown in the figure below.

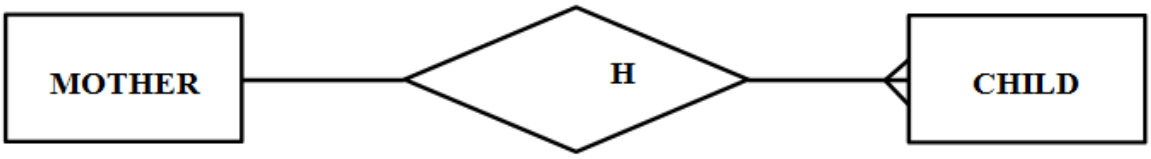
As indicated by arrows, the *cardinality* of this relationship is many-to-many since each employee may complete more than one course, and each course may be completed by more than one employee.

The **cardinality** of a relationship is the number of instances of one entity type that can (or must) be associated with each instance of another entity type. The cardinality of a relationship can be in one of the following four forms: *one-to-one*, *one-to-many*, *many-to-one*, and *many-to-many*.

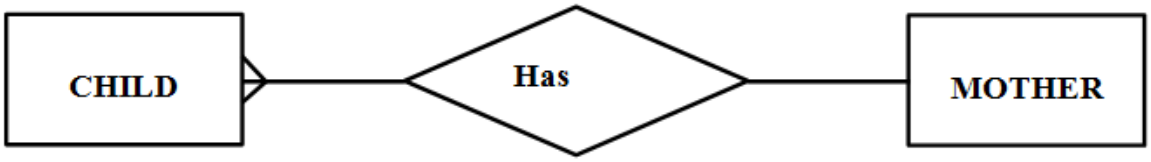
- **One-to-one:** An instance in entity type **A** is associated with at most one instance in entity type **B**, and an instance in entity type **B** is associated with at most one instance in entity type **A**. For example, cardinality between DEPARTMENT and CHAIRPERSON.



- **One-to-many:** An instance in entity type **A** is associated with any number (zero or more) of instances in entity type **B**, and an instance in entity type **B**, however, can be associated with at most one instance in entity type **A**. For example, cardinality between MOTHER and CHILD.



- **Many-to-one:** An instance in entity type **A** is associated with at most one instance in entity type **B**, and an instance in entity type **B**, however, can be associated with any number (zero or more) of instances in entity type **A**. For example, cardinality between CHILD and MOTHER.



- **Many-to-many:** An instance in entity type **A** is associated with any number (zero or more) of instances in entity type **B**, and an instance in entity type **B** is associated with any number (zero or more) of instances in entity type **A**. For example, cardinality between STUDENT and COURSE.

The **degree** of a relationship is the number of entity types that participate in the relationship. The three most common relationships in E-R models are *unary* (degree one), *binary* (degree two), and *ternary* (degree three). Higher degree relationships are also possible, but they are rarely encountered.

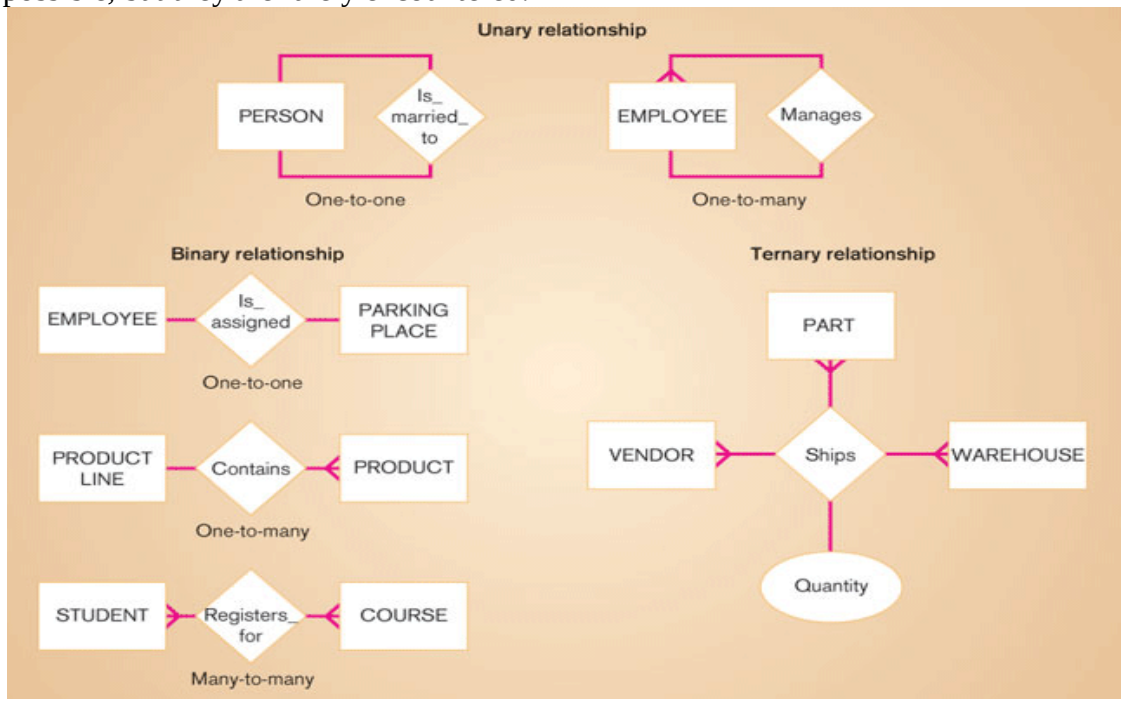


Fig: Example relationships of different degrees

A **unary relationship** is a relationship between the instances of one entity type. It is also called a **recursive relationship**. A **binary relationship** is a relationship between instances of two entity types. This is the most common type of relationship encountered in data modeling. A **ternary relationship** is a simultaneous relationship among the instances of three entity types.

We can also use the concept *minimum* and *maximum* cardinality when we draw E-R diagrams. The **minimum cardinality** of a relationship is the minimum number of instances of an entity type that may be associated with each instance of another entity type. The **maximum cardinality** of a relationship is the maximum number of instances of an entity type that may be associated with each instance of another entity type. For example, suppose a portion of banking database as shown in the figure below. Here, the minimum number of accounts for a customer is one and the maximum number of accounts is many (unspecified number greater than one). Similarly, the minimum number of customers associated with an account is one and the maximum number of customers is many.

When the minimum cardinality of a relationship is zero, then we say that the entity type is an **optional participation** or **partial participation** in the relationship. When the minimum cardinality of a relationship is one, then we say that the entity type is a **mandatory participation** or **total participation** in the relationship. In the figure above, the entity type ACCOUNT is a mandatory participation in the relationship. Similarly, the entity type CUSTOMER is a mandatory participation.

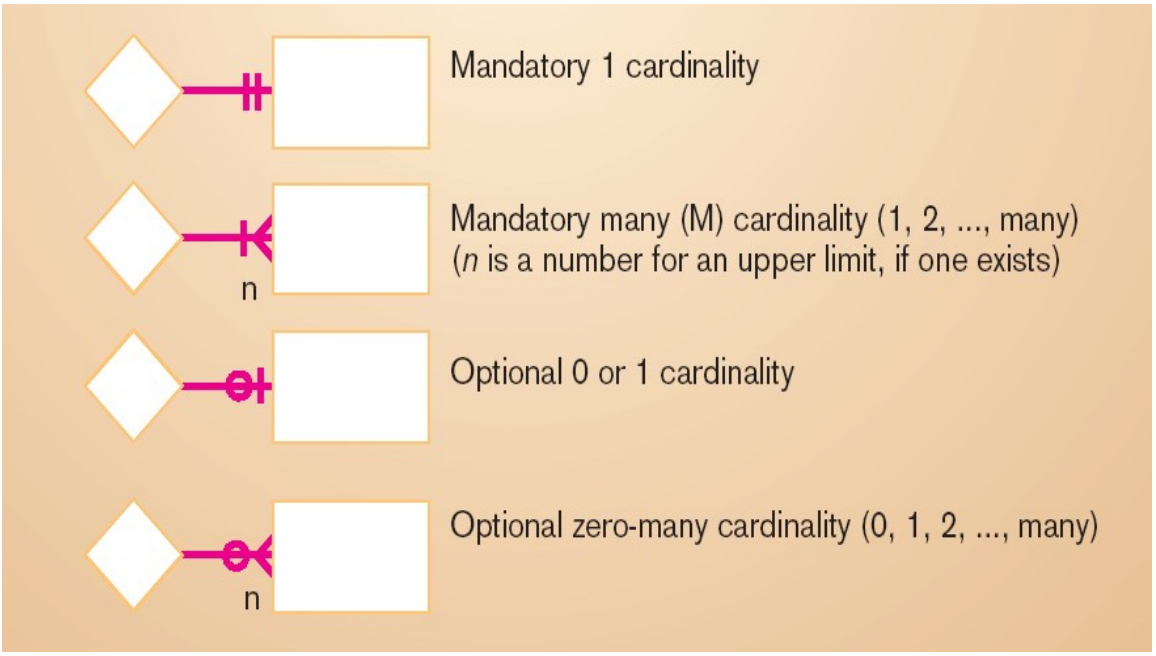


Fig: Cardinality symbols

Each relationship in E-R diagram is given a name. When naming relationships, we should use the following guidelines:

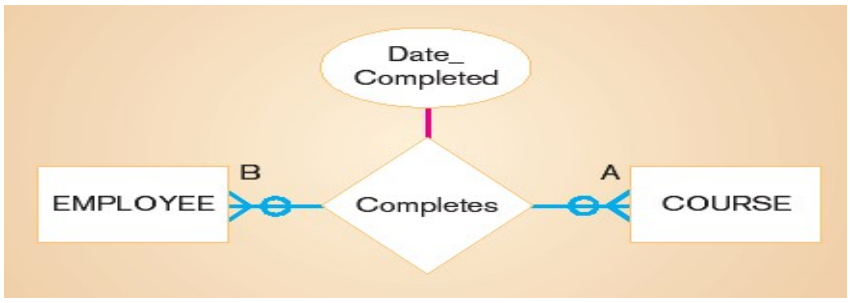
- A relationship name is *verb phrase* like Assigned_to, Supplies, or Teaches. This name represents action taken, not the result of the action, usually in the present tense.
- A relationship name should *avoid vague names*, such as Has or Is_related_to.

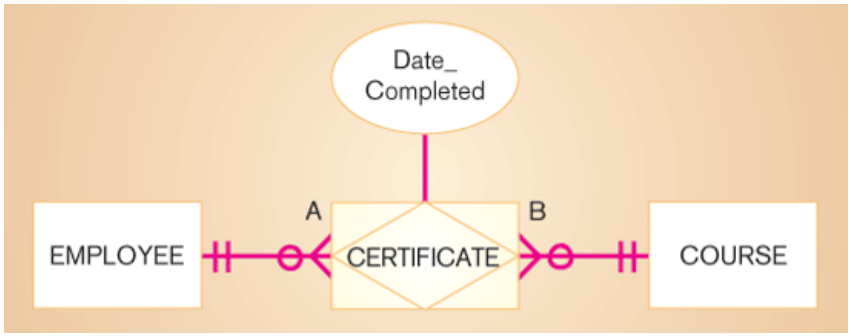
Each relationship in E-R diagram should be defined. When defining relationships, we should use the following guidelines:

- A relationship definition should explain *what action is being taken and possibly why it is important*.
- It may be important to *give examples to clarify the action*.
- The definition should explain any *optional participation*.
- A relationship definition should *explain any restrictions on participation in the relationship*.
- A relationship definition should *explain the extent of history that is kept in the relationship*.
- A relationship definition should *explain the reason for any explicit maximum cardinality* other than many.
- A relationship definition should *explain whether an entity instance* involved in a relationship instance *can transfer participation to another relationship instance*.

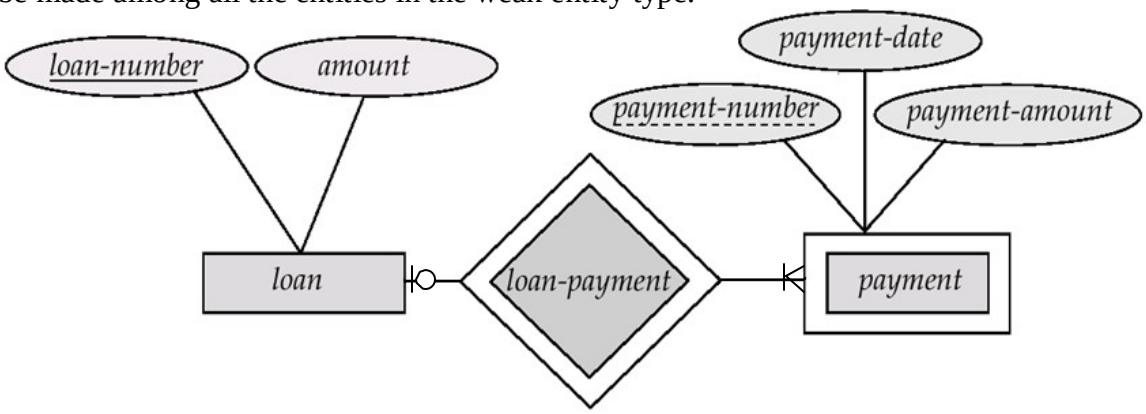
Other E-R Notations

- **Associative Entity:** An entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances is called **associative entity**. Associative entity is also called a **gerund**. For example, the first figure below shows a relationship with an attribute and the second figure shows an associative entity.





- **Weak Entity Type:** An entity type may not have sufficient attributes to form a primary key. Such an entity type is termed as a weak entity type. An entity type that has a primary key is termed as a strong entity type. For a weak entity type to be meaningful, it must be associated with another entity type, called the identifying or owner entity type, using one of the key attribute of owner entity type. The weak entity type is said to be existence dependent on the identifying entity type. The relationship associating the weak entity type with the identifying entity type is called the identifying relationship. The identifying relationship is many-to-one form the weak entity type to the identifying entity type, and the participation of the weak entity type in the relationship is total. Although a weak entity type does not have a primary key, we use discriminator (or partial key) as a set of attributes that allows the distinction to be made among all the entities in the weak entity type.



Supertypes and Subtypes

- **Subtype:** Subtype is a subgrouping of the entities in an entity type that is meaningful to the organization and that shares common attributes or relationships distinct from other subgroupings. For example, an entity type STUDENT to GRADUATE STUDENT and UNDERGRADUATE SUTDENT.
- **Supertype:** Supertype is a generic entity type that has a relationship with one or more subtypes. For example PATIENT with OUTPATIENT and RESIDENTPATIENT.

There are several important business rules for supertype/subtype relationships.

- **Total specialization rule:** Specifies that each entity instance of the supertype must be a member of some subtype in the relationship.
- **Partial specialization rule:** Specifies that an entity instance of the supertype does not have to belong to any subtype. May or may not be an instance of one of the subtypes.
- **Disjoint rule:** Specifies that if an entity instance of the supertype is a member of one subtype, it cannot simultaneously be a member of any other subtype.
- **Overlap rule:** Specifies that an entity instance can simultaneously be a member of two (or more) subtypes.

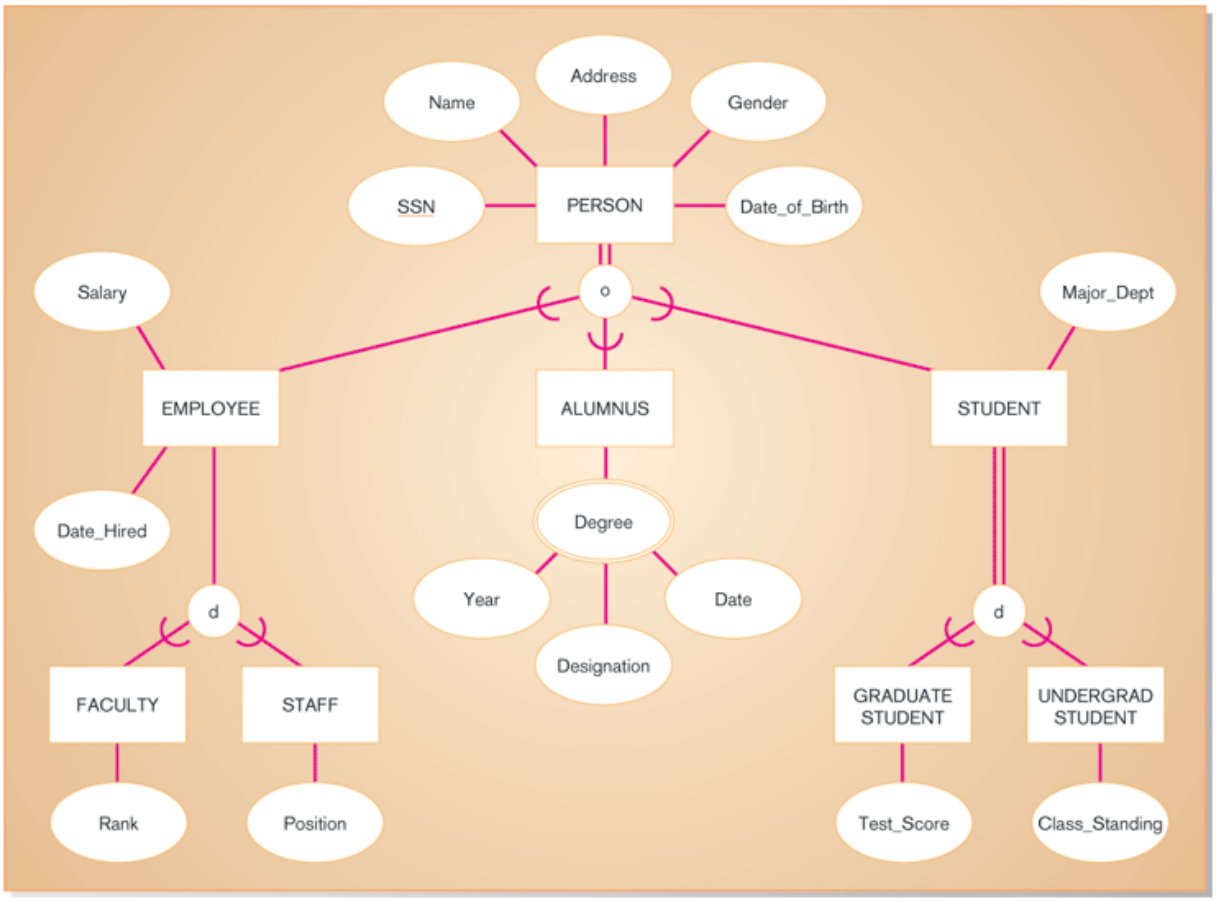


Fig: Example of supertype/subtype hierarchy

In the figure above, PERSON is supertype and EMPLOYEE, ALUMNUS, and STUDENT are subtypes. Similarly, EMPLOYEE is supertype for FACULTY and STAFF subtypes and STUDENT is supertype for GRADUATE STUDENT and UNDERGRADUATE STUDENT subtypes. Supertype is connected with a line to a circle, which in turn is connected by a line to the subtypes. The U-shaped symbol indicates that the subtype is a subset of the supertype. Total specialization is shown by a double line from the supertype to the circle and partial specialization is shown by a single line. Disjoint versus overlap is shown by a "d" or an "o" in the circle.

Some Exercises

1. Construct an ER diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
2. Construct an ER diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examinations conducted.
3. Construct an ER diagram of the library system in your college.
4. Construct an ER diagram to maintain data about students, instructors, semester, and courses in a college.
5. Construct an ERD to record the marks that students get in different exams of different course offerings.

Unit 3. Structured Methodologies **6 Hrs.**

Structured Methodologies

Structured methodologies (or structured systems analysis and design) have been used to document, analyze, and design information systems since the 1970s. **Structured** refers to the fact that the techniques are step by step, with each step building on the previous one. Structured methodologies are top-down, progressing from the highest, most abstract level to the lowest level of detail – from the general to the specific.

Structured development methods are process-oriented, focusing primarily on modeling the processes, or actions that capture, store, manipulate, and distribute data as the data flow through a system. These methods separate data from processes.

The primary tool for representing a system's component processes and the flow of data between them is the **data flow diagram (DFD)**. The data flow diagram offers a logical graphic model of information flow, partitioning a system into modules that show manageable levels of detail. It rigorously specifies the processes or transformations that occur within each module and the interfaces that exist between them.

Another tool for structured analysis is a **data dictionary**, which contains information about individual pieces of data and data groupings within a system. The data dictionary defines the contents of data flows and data stores so that systems builders understand exactly what pieces of data they contain. **Process specification** is another tool that describes the transformation occurring within the lowest level of data flow diagrams. They express the logic for each process. We can express the logic using structured English, decision table, decision tree etc.

In structured methodology, software design is modeled using hierarchical structure charts. The **structure chart** is a top-down chart, showing each level of design, its relationship to other levels, and its place in the overall design structure. The design first considers the main function of a program or system, then breaks this function into sub functions, and decomposes each subfunction until the lowest level of detail has been reached. A structure chart may document one program, one system (a set of programs) or part of one program.

Need for Structured Methodology

Most information systems development organizations use structured methodology because it offers the following advantages:

- Improve project management & control
- Make more effective use of experienced and inexperienced development staff
- Develop better quality systems with low cost
- Make projects resilient to the loss of staff
- Enable projects to be supported by computer-based tools such as computer-aided software engineering (CASE) systems
- Establish a framework for good communications between participants in a project
- Enable projects to deliver the product on time because it allows to plan, manage and control a project well
- Respond to the changes in the business environment while project progresses
- Improves the overall productivity by encouraging on-time delivery, meeting business requirements, ensuring better quality, and using human resources effectively.

Advantages and Disadvantages of Modeling and Data Dictionaries

Advantages:

- It minimizes planning overhead because all the phases are planned up front.
- Requirements analysis tends to be more thorough and better documented.
- Business requirements and system designs are easier to validate with pictures.
- It is easier to identify, conceptualize and analyze alternative technical solutions.
- Design specifications tend to be sound, stable, adaptable, and flexible because of models.
- Systems can be constructed more correctly the first time from complete and clear model based specifications
-

Disadvantages:

- It is time consuming because it takes time to collect facts, draw models and validate those models.
- The models can only be as good as the users' understanding of those requirements.
- It reduces the users' role in project to passive participation because most users want to see working software instead of models.
- It is considered to be inflexible; users must fully specify requirements before design; design must fully document technical specifications before construction; and so forth.

Role of CASE in Data Modeling

Data models are stored in a repository. CASE (computer-aided systems engineering) provides the repository for storing the data models and its detail description. Most CASE tools support computer-assisted data modeling and database design. CASE supports in drawing and maintaining the data models and their underlying details.

Using CASE product, you can easily create professional, readable data models without the use of paper, pencil, erasers, and templates. The models can be easily modified to reflect corrections and changes suggested by end users – you don't have to start over. Also, most CASE products provide powerful analytical tools that can check your models for mathematical errors, completeness, and consistency. Some CASE products can even help you analyze the data models for consistency, completeness, and flexibility.

Also, some CASE tools support reverse engineering of existing files and database structures into data models. The resulting data models represent physical data models that can be revised and reengineered into a new file or database, or they may be translated into their equivalent logical models. The logical data models could then be edited and forward engineered into a revised physical data model and subsequently a file or database implementations.

Unit 4. Systems Analysis **8 Hrs.**

Systems Planning and Initial Investigation

The primary purpose of systems planning is to identify problem's nature and its scope. It also includes preliminary (or initial) investigation and feasibility study. Initial investigation is used to understand the problem, define the project scope and constraints, identify the benefits, estimate the time and costs, and interact with managers and users. Feasibility study is used to determine some feasibility (economic, operational, technical feasibility etc) of the system.

The purpose of this phase is twofold. First, it answers the question, "Is this project worth looking at?" Second, assuming that the problem is worth looking at, it establishes the size and boundaries of the project, the project vision, any constraints or limitations, the required project participants, and the budget and schedule.

Information Gathering Techniques

To develop an information system, we first must be able to correctly identify, analyze, and understand what the users' requirements are or what the user wants the system to do. To know users' requirements, we use information gathering techniques. Information gathering techniques are also called **requirements discovery techniques** or **fact finding techniques** or **data collection techniques**. Information gathering includes those techniques to be used by system analysts to identify or extract system problems and solution requirements from the user community. Systems analysts need an organized method for information gathering. Some information gathering techniques are discussed below:

Sampling of Existing Documentation, Forms, and Files

When we are studying an existing system, we can develop a good feel for the system by studying existing documentation, forms, and files. A good analyst always gets facts from the existing documentation rather than from people.

Because it would be impractical to study every occurrence of every form or record in a file or database, system analysts normally use sampling techniques to determine what can happen in the system.

Sampling is the process of collecting a representative sample of documents, forms, and records. The most commonly used sampling techniques are **randomization** and **stratification**.

Randomization is a sampling technique in which there is no predetermined pattern or plan for selecting sample data.

Stratification is a systematic sampling technique that attempts to reduce the variance of the estimates by spreading out the sampling.

Research and Site Visits

In this technique, we perform site visits at systems they know have experienced similar problems. If these systems are willing to share, valuable information can be obtained that may save tremendous time and cost in the development process.

Computer trade journals and reference books are also a good source of information. They can provide us with information how others have solved similar problems. Also, through the Internet we can collect immeasurable amounts of information.

Observation of the Work Environment

Observation is an effective data-collection technique for obtaining an understanding of a

system. In this technique, the systems analyst either participates in or watches a person performing activities to learn about the system.

Advantages:

- → Data gathered can be highly reliable. Sometimes observations can be conducted to check the validity of data obtained directly from individuals.
- → The systems analyst is able to see exactly what is being done.
- → It is relatively inexpensive compared to other fact-finding techniques, because other techniques usually require more employees.
- → It allows the system analyst to do work measurement.

Disadvantages:

- → People usually feel uncomfortable when being watched to their work.
- → Some system activities may take place at odd times, causing a scheduling inconvenience for the system analyst.
- → It may cause interruption.
- → Some tasks may not always be performed by observation.

Questionnaires

This technique is used to conduct surveys through questionnaires. **Questionnaires** are special purpose documents that allow the analyst to collect information and opinions from the respondents. The document can be mass-produced and distributed to respondents, who can then complete the questionnaire on their own time. Questionnaires allow the analyst to collect facts from a large number of people.

Types of Questionnaires

There are two formats of questionnaire, **free-format** and **fixed-format**. Free-format questionnaire offer the respondent to record the answer in the space provided after the questionnaire.

Fixed-format questionnaire contain questions that require selection of predefined responses. In this format, the respondent must choose from the available answer. There are three types of fixed-format questions:

- → **Multiple Choice Questions: -** The respondent is given several answers of a question. The respondent should be told if more than one answer can be selected.
- → **Rating Questions: -** The respondent is given a statement and asked to use supplied responses to state an opinion.
- → **Ranking questions: -** The respondent is given several possible answers, which are to be ranked in the order of preference or experience.

Advantages

- → Most questionnaires can be answered quickly.
- → Inexpensive for gathering data from a large number of individuals.
- → Responses can be tabulated and analyzed quickly.

Disadvantages

- → There is no guarantee that an individual will answer or expand on all the questions.
- → Questionnaires tend to be inflexible.
- → It not possible for the systems analyst to observe and analyze the respondent's body language.
- → There is no immediate opportunity to clarify a vague or incomplete answer to questions.
- → Good questionnaires are difficult to prepare.
- → The number of respondents is often low.

Interviews

The personal **interview** is generally recognized as the most often used fact-finding technique. Interviews are the fact-finding techniques whereby the systems analysts collect information from individuals through face-to-face interaction.

There are two roles assumed in an interview. The systems analyst is the **interviewer**, responsible for organizing and conducting the interview. The system user or system owner is the **interviewee**, who is asked to respond to a series of questions.

Types of Interviews

There are two types of interviews: **unstructured** and **structured**. Unstructured interviews are conducted with only a general goal or subject in mind and with few, if any, specific questions. Structured interviews on the other hand are conducted with a set of specific questions to ask the interviewee.

Types of Questions

There are two types of questions in interview: **open-ended** and **closed-ended**. Open-ended questions allow the interviewee to respond in any way that seems appropriate. But, closed-ended questions restrict answers to either specific choices or short, direct responses.

Advantages

- → Interviews give the analyst an opportunity to motivate the interviewee to respond freely and openly to questions.
- → Interviews allow more feedback from the interviewee.
- → Interviews give the analyst an opportunity to observe the interviewee's nonverbal communication.

Disadvantages

- → Interviewing is a very time-consuming and therefore costly fact-finding approach.
- → Success of interviews is highly dependent on the systems analyst's human relational skills.
- → Interviewing may be impractical due to the location of interviewees.

Discovery Prototyping

Discovery prototyping is the act of building a small-scale, representative or working model of the users' requirements to discover or verify the requirements.

Usually only the areas where the requirements are not clearly understood are prototyped. Creating discovery prototypes enables the developers as well as the users to better understand and refine the requirements involved with developing the system.

Advantages

- → Allows users and developers to experiment with the software and develop an understanding of how the system might work.
- → Aids in determining the feasibility and usefulness of the system before high development costs are incurred.
- → Serves as a training mechanism for users.
- → Aids in building system test plans and scenarios to be used last in the system testing process.
- → May minimize the time spent for fact-finding and help to define more stable and reliable requirements.

Disadvantages

- → Users and developers may need to be trained in the prototyping approach.
- → Doing prototyping may extend the development schedule and increase the development costs
- → Users may develop unrealistic expectations.

Joint Requirements Planning (JRP)

It is a process whereby highly structured group meeting is conducted to analyze problems and define requirements. JRP is a subset of a more comprehensive ***joint application development (JAD)***. The JRP participants are:

- → **Sponsor: -** This person is normally an individual who is in top management and has authority that spans the different departments and users who are to be involved in the systems project.
- → **Facilitator: -** The JRP facilitator or leader is usually responsible for leading all sessions that are held for a systems project.
- → **Users and Managers: -** Users devote themselves to the JRP sessions to effectively communicate business rules and requirements, review design prototypes and make acceptance decisions. Managers approve project objectives, establish project priorities, approve schedules and costs, and approve identified training needs and implementation plans.
- → **Scribe(s): -** Scribes are responsible for keeping records pertaining to everything discussed in the meeting.
- → **IT Staff: -** IT personnel listen and take notes regarding issues and requirements voiced by the users and managers. Normally, IT personnel do not speak unless invited to do so.

Benefits

- → It actively involves users and managers in the development project.
- → It reduces the amount of time required to develop systems.
- → When JRP incorporates prototyping as a means for conforming requirements and obtaining design approvals, the benefits of prototyping are realized.

Structured Analysis Tools

Structured analysis includes different tools like DFD (data flow diagram), ERD (entity relationship diagram), data dictionary, structured English, decision table, and decision tree.

Feasibility Study

Feasibility is the measure of how beneficial or practical the development of information system will be to an organization. **Feasibility study** is the process by which feasibility is measured. Feasibility analysis is appropriate to the systems analysis phase.

Four Tests for Feasibility

During systems analysis phase, the system analyst identifies different alternate solutions and analyzes those solutions for feasibility. To analyze different alternative solutions, most analysts use four categories of feasibility tests:

- Operational feasibility
- Technical feasibility
- Schedule feasibility and
- Economic feasibility

1. **Operational Feasibility:** It is a measure of how well the solution will work in an organization. It is also a measure of how people feel about the system/project. So, this feasibility is people oriented. Operational feasibility addresses two major issues:
    a. *Is the problem worth solving, or will the solution to the problem work?*

b. *How do end users and management feel about the problem (solution)?*

When determining operational feasibility, *usability analysis* is often performed with a working prototype of the proposed system. **Usability analysis** is a test of system's user interfaces and is measured in how easy they are to learn and to use and how they support the desired productivity levels of the users. Usability is measured in terms of *ease of learning*, *ease of use*, and *satisfaction*.

2. **Technical Feasibility:** It is a measure of practically of a specific technical solution and availability of technical resources and expertise. Technical feasibility is computer oriented. This feasibility addresses three major issues:
   a. *Is the proposed technology or solution practical?*
   b. *Do we currently possess the necessary technology?*
   c. *Do we possess the necessary technical expertise, and is the schedule reasonable?*
3. **Schedule Feasibility:** It is a measure of how reasonable the project timetable is. Schedule feasibility is the determination of whether the time allocated for a project seems accurate. Projects are initiated with specific deadlines. It is necessary to determine whether the deadlines are mandatory or desirable. If the deadlines are desirable rather than mandatory, the analyst can propose alternative schedules.
4. **Economic Feasibility:** It is the measure of the cost-effectiveness of a project or solution. This feasibility deals with costs and benefits of the information system. The bottom-line in many projects is economic feasibility. During the early phases of the project, economic feasibility analysis amounts to little more than judging whether the possible benefits of solving the problem are worthwhile. However, as soon as specific requirements and alternative solutions have been identified, the analyst can determine the costs and benefits of each alternative.

Some other feasibility tests are also possible. These are *legal and contractual feasibility* and *political feasibility*. **Legal and contractual feasibility** is the process of assessing potential legal and contractual ramifications due to the construction of a system. **Political feasibility** is the process of evaluating how key stakeholders within the organization view the proposed system.

Cost-Benefit Analysis Techniques

Economic feasibility has been defined as a cost-benefit analysis. Most schools offer coerces like *financial management*, *financial decision analysis*, and *engineering economics and analysis* for cost benefit analysis. The cost benefit analysis techniques include:

- *How much will the system costs?*
- *What benefits will the system provide?*
- *Is the proposed system cost-effective?*

How much will the system costs?

Costs fall into two categories: *costs associated with developing the system* and *costs associated with operating the system*. The former costs can be estimated from the start of the project and should be refined at the end of each phase of the project. The later can be estimated only after specific computer-based solution has been defined.

System development costs are usually onetime costs that will not recur after the project has been completed. Many organizations have standard cost categories that must be evaluated. In the absence of such categories, we use the following list:

- **Personnel cost –** The salaries of systems analysts, programmers, consultants, data entry personnel, computer operators, secretaries, and the like who work on the

project.

- **Computer usage –** The cost in the use of computer resources.
- **Training –** Expenses for the training of computer personnel or end-users.
- **Supply, duplication, and equipment costs**.
- **Cost of any new computer equipment and software**.

The operating costs tend to recur throughout the lifetime of the system. The costs in this case can be classified as *fixed* or *variable*.

- **Fixed costs –** Fixed costs occur at regular intervals but at relatively fixed rates. Some examples include: lease payments and software license payments, salaries of IS operators and support personnel etc.
- **Variable costs –** Variable costs occur in proportion to some usage factor. Some examples include: costs of computer usage (e.g., CPU time used, storage used), supplies (e.g., printer paper, floppy disks), overhead costs (e.g., utilities, maintenance, and telephone service) etc.

What benefits will the system provide?

Benefits normally increase profits or decrease costs. Benefits can be classified into two categories: *tangible* and *intangible*.

- **Tangible benefits –** Tangible benefits are those that can be easily quantified. These benefits are usually measured in terms of monthly or annual savings or of profit to the firm. Alternatively, these benefits might be measured in terms of unit cost savings or profit. Some examples of tangible benefits are: fewer processing errors, increased throughput, decreased response time, elimination of job steps, increased sales, reduced credit losses, and reduce expenses.
- **Intangible benefits –** Intangible benefits are believed to be difficult or impossible to quantify. Unless these benefits are at least identified, it is entirely possible that many projects would not be feasible. Some examples of intangible benefits are: improved customer goodwill, improved employee morale, better service to community, and better decision making.

Unfortunately, if a benefit cannot be quantified, it is difficult to accept the validity of an associated cost-benefit analysis that is based on incomplete data.

Is the proposed system cost-effective?

There are three popular techniques to assess economic feasibility, also called cost-effectiveness: *payback analysis*, *return to investment*, and *net present value*.

One concept that is shared by all three techniques is the **time value of money** – a dollar today is worth more than a dollar one-year from now.

Some of the costs of the system will be accrued in after implementation. Before cost-benefit analysis, these costs should be brought back to the current dollars. **Present value** is the current value of a dollar at any time in the future. It is calculated using the formula:

$$PV_n = 1/(1 + i)^n$$

Where PV_n is the present value of \$1.00 n years from now and i is the discount rate.

- **Payback analysis –** It is a technique for determining if and when an investment will pay for itself. Because system development costs are incurred long before benefits begin to occur, it will take some time for the benefits to overtake the costs. After implementation, there will be additional operating expenses that must be recovered. *Payback analysis* determines how much time will lapse before accrued benefits overtake accrued and continuing costs. This period of time is called **payback period**, that is, the period of time that will lapse before accrued benefits overtake accrued costs.

- **Return-on-investment analysis –** This technique compares the lifetime profitability of the solution. It is a percentage rate that measures the relationship between the amounts the business gets back from an investment and the amount invested. It is calculated as follows:

  Lifetime ROI = (Estimated lifetime benefits – Estimated lifetime costs)/Estimated lifetime costs

- **Net present value –** It is an analysis technique that compares costs and benefits for each year of the system's lifetime. Many managers consider it the preferred cost-benefit analysis technique.

Systems Design **8 Hrs.**

The Processes and Stages of System Design

Information systems design is defined as those tasks that focus on the specification of a detailed computer-based solution. It is also called **physical design**. Hence, systems design focuses on the technical or implementation concerns of the information system.

The purpose of design is to transform the requirements represented in analysis phase into physical design specifications that will guide system construction. The design specification is used to write computer programs during implementation phase. In other words, the design phase addresses how technology will be used in the new system. Design process represents a specific technical solution of the information system. The design phase is concerned with designing **files and databases**, **forms and reports**, **dialogues and interfaces** and **system and program structure**.

Designing Forms and Reports

Forms are used to present or collect information on a single item such as a customer, product, or event. Forms can be used for both input and output. **Reports**, on the other hand, are used to convey information on a collection of items. Forms and report design is a key ingredient for successful systems. As users often equate the quality of a system to the quality of its input and out methods, the design process of for forms and reports is an especially important activity.

Forms and reports are identified during requirements structuring. The kinds of forms and reports the system will handle are established as a part of the design strategy formed at the end of the analysis phase of the system development process. Forms and reports are integrally related to various diagrams developed during requirements structuring. For example, every input form will be associated with a data flow entering a process on a DFD, and every output form or report will be a data flow produced by a process on a DFD. Further, the data on all forms and reports must consists of data elements on data stores and on the E-R data model for the application, or must be computed from these data elements.

The Process of Designing Forms and Reports

Designing forms and reports is a user-focused activity that typically follows a prototyping approach. First, you must gain an understanding of the intended user and task objectives by collecting initial requirements during requirements determination. During this process, several questions must be answered. These questions attempt to answer the "who, what, when, where, and how" related to the creation of all forms and reports as given below.

1. Who will use the form or report?
2. What is the purpose of the form or report?
3. When is the form or report needed or used?
4. Where does the form or report need to be delivered and used?
5. How many people need to use or view the form or report?

Gaining an understanding of these questions is a required first step in the creation of any form or report.

After collecting the initial requirements, you structure and refine this information into an initial prototype. Structuring and refining the requirements are completely independent of the users, although you may need to occasionally contact users in order to clarify some issue overlooked during analysis. Finally, you ask users to review and evaluate the prototype. After reviewing the prototype, users may accept the design or request that changes be made. If changes are needed, you will repeat the construction-evaluate-

refinement cycle until the design is accepted. Usually, several iterations of this cycle occur during the design of a single form or report. As with any prototyping process, you should make sure that these iterations occur rapidly in order to gain the greatest benefits from this design approach.

The initial prototype may be constructed in numerous environments. The obvious choice is to use CASE tool or the standard development tools used within your organization. Often, initial prototypes are simply mock screens that are not working modules or systems. Mock screens can be produced from a word processor, computer graphics design package, or electronic spreadsheet.

Deliverables and Outcomes

Design specifications are the major deliverables and are inputs to the system implementation phase. Design specifications have three sections:

1. **Narrative overview –** This section contains general overview of the characteristics of the target users, tasks, system, and environmental factors in which the form or report will be used. The purpose is to explain to those who will actually develop the final form, why this form exists, and how it will be used so that they can make the appropriate decisions.
2. **Sample design –** This section provides a sample design of the form. This design may be hand drawn using a coding sheet although, in most instances, it is developed using CASE or standard development tool. Using actual development tools allows the design to be more thoroughly tested and assessed.
3. **Testing and usability assessment –** This section provides all testing and usability assessment information. Assessing usability depends on speed, accuracy, and satisfaction.

Formatting Forms and Reports

A wide variety of information can be provided to users of information systems and, as technology continues to evolve, a greater variety of data types will be used. There are numerous guidelines for formatting information.

General Formatting Guidelines

Over the past several years, industry and academic researchers have spent considerable effort investigating how information formatting influences individual task, performance, and perceptions of usability. Through this work, several guidelines for formatting information have emerged as given below:

1. **Meaningful titles –** Titles should be clear and specific describing content and use. They should include version information and current date.
2. **Meaningful information –** Forms should include only necessary information with no need to modification.
3. **Balance of layout –** Adequate spacing and margins should be used. All data and entry fields should be clearly labeled.
4. **Easy navigation system –** Clearly show how to move forward and backward. Clearly show where you are (e.g., page 1 of 3) currently. Notify the user when on the last page of a multipaged sequence.

Highlighting Information

As the display technologies continue to improve, there will be a greater variety of methods available to you for highlighting information. Most commonly used methods for highlighting information are:

1. Blinking and audible tones

2. Color differences
3. Intensity differences
4. Size differences
5. Font differences
6. Reverse video
7. Boxing
8. Underlining
9. All capital letters
10. Offsetting the position of nonstandard information

There are several situations when highlighting can be a valuable technique for conveying special information:

1. Notifying users of errors in data entry or processing
2. Providing warnings to users regarding possible problems such as unusual data values or an unavailable device
3. Drawing attention to keywords, commands, high priority messages, and data that have changed or gone outside normal operating ranges

Color versus No-color

Color is a powerful tool for the designer in influencing the usability of a system. When applied appropriately, color provides many potential benefits to forms and reports as given below:

1. Soothes and strikes the eye
2. Accents an uninteresting display
3. Facilitates subtle discriminations in complex displays
4. Emphasizes the logical organization of information
5. Draws attention to warnings
6. Evokes more emotional reactions

Color also provides problems as given below:

1. Color pairings may wash out or cause problems for some users (e.g., color blindness)
2. Resolution may degrade with different displays
3. Color fidelity may degrade on different displays
4. Printing or conversion to other media may not easily translate

Displaying Text

In business-related systems, textual output is becoming increasingly important. Some of the guidelines that have emerged from past research are:

1. **Case –** Display text in mixed upper and lower case and use conventional punctuations.
2. **Spacing –** Use double spacing if space permits. If not, place a blank line between paragraphs.
3. **Justification –** Left-justify text and leave a ragged right margin.
4. **Hyphenation –** Do not hyphenate words between lines.
5. **Abbreviations –** Use abbreviations and acronyms only when they are widely understood by users and are significantly shorter than the full text.

Designing Tables and Lists

The context and meaning of tables and lists are significantly derived from the format of the information. Consequently, the usability of information displayed in tables and alphanumeric lists is likely to be much more influenced by effective layout than most other types of information display. As with the display of textual information, tables and

lists can also be greatly enhanced by following a few simple guidelines.

1. **Use meaningful labels –** All columns and rows should have meaningful labels. Labels should be separated from other information by using highlighting. Re-display labels when the data extend beyond a single screen or page.
2. **Formatting columns, rows, and text –** Sort in a meaningful order (e.g., ascending, descending, or alphabetic). Place a blank line between every five rows in long columns. Similar information displayed in multiple columns should be sorted vertically (that is, read from top to bottom, not left to right). Columns should have at least two spaces between them. Allow white space on printed reports for user to write notes. Use a single typeface, except for emphasis. Use same family of typefaces within and across displays and reports. Avoid overly fancy fonts.
3. **Formatting numeric, textual, and alphanumeric data –** Right justify numeric data and align columns by decimal points or other delimiter. Left justify textual data. Use short line length, usually 30 to 40 characters per line. Break long sequences of alphanumeric data into small groups of three to four characters each.

When you design the display of numeric information, you must determine whether a table or a graph should be used. A considerable amount of research focusing on this topic has been conducted. In general, tables are best when the user's task is related to finding an individual data value from a larger data set whereas line and bar graphs are more appropriate for an understanding of data changes over time. Some guidelines for selecting tables versus graphs are given below:

Use tables for

- Reading individual data values

Use graphs for

- Providing quick summary of data
- Displaying trends over time
- Comparing points and patterns of variables
- Forecasting activities
- Reporting of vast amounts of information when relatively simple impressions are to be drawn

Paper versus Electronic Reports

When a report is produced on paper rather than on a computer display, you must consider type of printer to use. For example, laser printers (especially color laser printers) and ink jet printers allow you to produce a report that looks exactly as it does on the display printer. However, other types of printers are not able to closely reproduce the display screen image onto paper. For example, many business reports are produced using high spend impact printers that produce characters and a limited range of graphics by printing a fine pattern of dots.

Unit-6

System Implementation

System Implementation is the way of carrying out a developed system into working condition.

The process of Coding, Testing and Installation

Coding is the process whereby the physical design specifications created by the analysis team are turned into working computer code by the programming team. Depending on the size and complexity of the system, coding can be an involved, intensive activity.

Once coding has begun, the testing process can begin and proceed in parallel. As each program module is produced, it can be tested individually, then as part of a larger program, and then as part of a larger system. Although testing is done during implementation, we must begin planning for testing earlier in the project. Planning involves determining what needs to be tested and collecting test data. This is often done during the analysis phase because testing requirements are related to system requirements.

Installation is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, documentation, and work procedures to those consistent with the new system.

The process of Documenting the system, Training Users and Supporting Users.

Although the process of documentation proceeds throughout the life cycle, it receives, formal attention during the implementation phase because the end of implementation largely marks the end of the analysis team's involvement in systems development.

Larger organizations also tend to provide training and support to computer users throughout the organization. Some of the training and support is very specific to particular application systems, whereas the rest is general to particular operating systems or off-the-shelf software packages.

Software Application Testing

Software testing is a method of assessing the functionality of a software program.

Different types of Tests

1. Inspections: A testing technique in which participants examine program code for predictable language specific errors. Syntax, grammar, and some other routine errors can be checked by automated inspection software. 60 to 90 percent of all software effects as well as provide programmers with feedback that enables them to avoid making the same types of errors in future work.
2. Desk checking: A testing technique in which the programmer or someone else who understands the logic of the program works through the code with a paper and pencil. The reviewer acts as the computer, mentally checking each step and its results for the entire set of computer instructions.
3. Unit testing: Automated technique whereby each module is tested alone in an attempt to discover any errors that may exist in the module's code.
4. Integration testing: The process of bringing together more than one modules that a program comprises for testing purposes.

5. System testing: The process of bringing together of all of the programs that a system comprises for testing purposes. Programs are typically integrated in a top-down incremental fashion. The system can be tested in two ways:
    i. **Black box testing**: In Black box test (also called functional test) internal code of the program are tested. It is called black box testing because the test cases are totally hidden for the general users.
    ii. **White box testing**: In white box test (also called glass box test) structure of the program is tested. It called white box testing because the test cases are totally visible to the general users and they can also make test cases.
6. Stub testing: A technique used in testing modules, especially where modules are written and tested in a top down fashion, where a few lines of codes are used to substitute for subordinate modules. Top-level modules contain many call to subordinate modules, we may wonder how they can be tested if the lower-level modules haven't been written yet. This is called stub testing.
7. User acceptance testing: Once the system tests have been satisfactorily completed, the system is ready for acceptance testing, which is testing the system in the environment where it will eventually be used.
    i. **Alpha testing:**

    User testing of a completed information system using simulated data. The types of tests performed during alpha testing include the following:

    a. Recovery testing: Forces the software to fail in order to verify that recovery is properly performed.
    b. Security testing: Verifies that protection mechanisms built into the system will protect it from improper penetration.
    c. Stress testing: Tries to break the system (eg: what happens when a record is written to the database with incomplete information or what happens under extreme online transaction loads or with a large number of concurrent users).
    d. Performance testing: Determines how the system performs in the range of possible environments in which it may be used (eg: different hardware configurations, networks, operating systems).

    iii. **Beta testing:**

    User testing of a completed information system using real data in the real user environment. The intent of the beta test is to determine whether the software, documentation, technical support and training activities work as intended. Beta testing can be viewed as a rehearsal of the installation phase.

Installation

The process of moving from the current information system to the new one is called installation. Different ways of installation are:

1. **Direct installation**: Changing over from the old information system to a new one by turning off the old system when the new one is turned on. Any errors resulting from the new system will have a direct impact on the users. If the new system fails, considerable delay may occur until the old system can again be made operational and business transactions are reentered to make the database up to date. Direct installation can be very risky. Direct installation requires a complete installation of the whole system. For a large system, this may mean a long time until the new system can be installed, thus delaying

system benefits or even missing the opportunities that motivated the system request. It is the least expensive installation method, and it creates considerable interest in making the installation a success.

2. **Parallel installation:** Running the old information system and the new one at the same time until management decides the old system can be turned off. All of the work done by the old system is concurrently performed by the new system. Outputs are compared to help determine whether the new system is performing as well as the old. Errors discovered in the new system do not cost the organization much, if anything, because errors can be isolated and the business can be supported with the old system. Because all work is essentially done twice, a parallel installation can be very expensive, running two systems implies employing two staffs to operate and maintain. A parallel approach can also be confusing to users because they must deal with both systems. A parallel approach may not be feasible, especially if the users of the system cannot tolerate redundant effort or if the size of the system is large.
3. **Pilot installation:** It is also known as single-location installation. Rather than converting all of the organization at once, single location installation involves changing form the current to the new system in only one place or in a series of separate sites over time. The single location may be a branch office, a single factory, or one department, and the actual approach used for installation in that location may be any of the other approaches. The key advantage to single location installation is that it limits potential damage and potential cost by limiting the effects to a single site. Once management has determined that installation has been successful at one location, the new system may be deployed in the rest of the organization, possibly continuing with installation at one location at a time. Problems with the system can be resolved before deployment to other sites. Even though the single location approach may be simpler for users, it still places a large burden on information system staff to support two versions of the system. Problems are isolated at one site at a time. IS staff members can devote all of their efforts to success at the pilot site. If different locations require sharing of data, extra programs will need to be written to synchronize the current and new systems.
4. **Phased installation:** It is also called staged installation. Different parts of the old and new systems are used in cooperation until the whole new system is installed. By converting gradually, the organization's risk is spread out over time and place. Also phased installation allows for some benefits from the new system before the whole system is ready. For example, a new data-capture methods can be used before all reporting modules are ready. For a phased installation, the new and replaced systems must be able to coexist and probably share data. Thus bridge programs connecting old and new databases and programs often must be built. Sometimes the new and old systems are so incompatible that pieces of the old system cannot be incrementally replaced so this strategy is not feasible. A phased approach requires careful version control, repeated conversions at each phase, and a long period of change, which may be frustrating and confusing to users.

Documenting the system

Documentation is the process of collecting, organizing, storing and maintaining a complete record of system and other documents used or prepared during the different phases of the life cycle of system. System cannot be considered to be complete, until it is properly documented. Proper documentation of system is necessary due to the following reasons:

1. It solves the problem of indispensability of an individual for an organization. Even if the person, who has designed or developed the system, leaves the organization, the documented knowledge remains with the organization, which can be used for the continuity of that software.

2. It makes system easier to modify and maintain in the future. The key to maintenance is proper and dynamic documentation. It is easier to understand the concept of a system from the documented records.
3. It helps in restarting a system development, which was postponed due to some reason. The job need not be started from scratch, and old ideas may still be easily recapitulated from the available documents, which avoids duplication of work, and saves lot of times and effort.

Types of documentation

a. System documentation
   System documentation records detailed information about a system's design specification, its internal workings and its functionality. System documentation is intended primarily for maintenance programmers. It contains the following information:
   1. A description of the system specifying the scope of the problem, the environment in which it functions, its limitation, its input requirement, and the form and type of output required.
   2. Detailed diagram of system flowchart and program flowchart.
   3. A source listing of all the full details of any modifications made since its development.
   4. Specification of all input and output media required for the operation of the system.
   5. Problem definition and the objective of developing the program.
   6. Output and test report of the program.
   7. Upgrade or maintenance history, if modification of the program is made.

   There are two types of system documentation. They are:

   i. Internal documentation
      Internal documentation is part of the program source code or is generated at compile time.
   ii. External documentation
      External documentation includes the outcome of structured diagramming techniques such as dataflow and entity-relationship diagrams.
b. User documentation
   User documentation consists of written or other visual information about an application system, how it works and how to use it. User documentation is intended primarily for users. It contains the following information:
      1. Set up and operational details of each system.
      2. Loading and unloading procedures.
      3. Problems which could arise, their meaning reply and operation action.
      4. Special checks and security measures.
      5. Quick reference guides about operating a system in a short, concise format.

Training and supporting users

The type of training needed will vary by system type and user expertise. Types of training methods are:

a. Resident expert
b. Traditional instructor-led classroom training
c. E-learning/distance learning
d. Blended learning (combination of instructor-led and e-learning)
e. Software help components
   - Electronic performance support system: component of a software package or an application in which training and educational information is embedded.
f. External sources, such as vendors

Computing supports for users has been provided in one of a few forums:

Automating support: online support forums provide users access to information on new releases, bugs and tips for more effective users access to information on new releases, bugs and tips form more effective usage. Forums are offered over the internet or over company intranets.

Providing support through a help desk: A help desk is an information systems department function and is staffed by IS personnel. The help desk is the first place users should call when they need assistance with an information system. The help desk staff members either deal with the users questions or refer the users to the most appropriate person.

Quality assurance

A process or procedure for minimizing errors and ensuring quality in products. Poor quality can result from inaccurate requirements, design problems, coding errors, faulty documentation, and ineffective testing. A quality assurance (QA) team reviews and tests all applications and systems changes to verify specifications and software quality standards. A successful organization must improve quality in every area, including its information systems. Top management must provide the leadership, encouragement, and support needed for high-quality IT resources. The main objective of quality assurance is to avoid problems or to detect them as soon as possible. In an effort to achieve high standards of quality, software systems developers should consider software engineering concepts, internationally recognized quality standards, and careful project management techniques.

Software Engineering: Because quality is so important, you can use an approach called software engineering to manage and improve the quality of the finished system. Software engineering is a software development process that stresses solid design, accurate documentation, and careful testing.

International Organization for Standardization (ISO): International Organization for Standardization (ISO) is a worldwide body that establishes quality standards for products and services. ISO standards include everything from internationally recognized symbols, to the ISBN numbering system that identifies textbook. In addition, ISO seeks to offer a global consensus of what constitutes good management practices — practices that can help firms deliver consistently high-quality products and services. Because software is so important to a company's success, many firms seek assurance that software systems either purchased or developed in-house, will meet rigid quality standards. In 1991, ISO established a set of guidelines called ISO 9000-3, which provided a quality assurance framework for developing and maintaining software. A company can specify ISO 9000-3 standards when it purchases software from a supplier or use ISO guidelines for in-house software development to ensure that the final result measures up to ISO standards. ISO requires a specific development plan, which outlines a step-by-step process for transforming user requirements into a finished product. ISO standards can be quite detailed. For example, ISO requires that a software supplier document all testing and maintain records of test results. If problems are found, they must be resolved, and any modules affected must be retested. Additionally, software and hardware specifications of all test equipment must be documented and included in the test records.

Levels of assurance: Analysts use four levels of quality assurance: testing, verification, validation, and certification.

a. **Testing:** Systems testing is an expensive but critical process that can take as much as 50 percent of the budget for program development. The common view of testing held by users is that it is performed to prove that there are no errors in a program. However, this is virtually impossible, since analysts cannot

prove that software is free and clear of errors. Therefore, the most useful and practical approach is with the understanding that testing is the process of executing a program with explicit intention of finding errors that is, making the program fail. The tester, who may be an analyst, programmer, or specialist trained in software testing, is actually trying to make the program fail. A successful test, then, is one that finds an error. Analysts know that an effective testing program does not guarantee systems reliability. Reliability is a design issue. Therefore, reliability must be designed into the system. Developers cannot test for it.

b. **Verification and Validation**: Like testing, verification is also intended to find errors. Executing a program in a simulated environment performs it. Validation refers to the process of using software in a live environment in order to find errors. When commercial systems are developed with the explicit intention of distributing them to dealers for sale or marketing them through company – owned field offices, they first go through verification, some-times called alpha testing. The feedback from the validation phase generally produces changes in the software to deal with errors and failures that are uncovered. Then a set of user sites is selected that puts the system into use on a live basis. These beta test sites use the system in day- to - day activities; they process live transactions and produce normal system output. The system in live is very sense of the word, except that the users are aware they are using a system that can fail. But the transactions that are entered and the persons using the system are real. Validation many continue for several months. During the course of validating the system, failure may occur and the software will be changed. Continued use may produce additional failures and the need for still more change.

c. **Certification**: Software certification is an endorsement of the correctness of the program, an issue that is rising in importance for information systems applications. There is an increasing dependence on the purchase or lease of commercial software rather than on its in-house development. However, before analysts are willing to approve the acquisition of a package, they often require certification of the software by the developer or an unbiased third party. For example, selected accounting firms are now certifying that a software package in fact does what the vendor claims it does and in a proper manner. To so certify the software, the agency appoints a team of specialists who of specialists who carefully examine the documentation for the system to determine what the vendor claims the system does and how it is accomplished. Then they test the software against those claims. If no serious discrepancies or failures are encountered, they will certify that the software does what the documentation claims. They do not, however, certify that the software is the right package for a certain organization. That responsibility remains with the organization and its team of analysts.

Maintenance

Correcting and upgrading process of the system is called system maintenance. Maintenance is necessary to eliminate errors in the working system during its working life and to tune the system to any variations in its working environment. Different types of maintenance are:

1. Corrective maintenance: Corrective maintenance refers to changes made to repair defects in the design , coding or implementation of the system. For example if we had recently purchased a new home , corrective maintenance would involve repairs made to things that had never worked as designed, such as a faulty electrical outlet or misaligned door. More corrective maintenance problems surface soon after installation. When corrective maintenance problems surface, they are typically urgent and need to be resolved to curtail possible interruptions in normal business activities. Of all types of maintenance, corrective accounts for as much as 75 percent of all maintenance activity. It simply focuses on removing defects from an existing system without adding new functionality.
2. Adaptive maintenance: Adaptive maintenance involves making changes to an information system to evolve its functionality to changing business needs or to migrate it to a different operating environment. Within a home, adaptive maintenance might be adding storm windows to improve the cooling performance of an air conditioner. Adaptive maintenance is less urgent than corrective maintenance

because business and technical changes typically occur over some period of time. Adaptive maintenance is generally a small part of an organizations maintenance effort, but it adds value to the organization.

3. Perfective maintenance: Perfective maintenance involves making enhancements to improve processing performance or interface usability or to add desired, but not necessarily required, system features. In home perfective maintenance would be adding a new room. Many systems professionals feel that perfective maintenance is not really maintenance but rather new development. Perfective maintenance usually is cost effective during the middle of the system's operational life. Early in systems operation, perfective maintenance usually is not needed. Later, perfective maintenance might be necessary, but have a high cost.
4. Preventive maintenance: Preventive maintenance involves changes made to a system to reduce the chance of future system failure. An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalize how a system sends a report information to a printer so that the system can easily adapt to changes in printer technology. In our home example, preventive maintenance could be painting the exterior to better protect the home from severe weather conditions.

The process of maintaining information system

Maintenance phase is the last phase of SDLC. The major activity occur within maintenance:

1. Obtaining maintenance requests
   Obtaining maintenance requests requires that a formal process be established whereby users can submit system change requests. When developing the procedures for obtaining maintenance requests, organizations must also specify an individual within the organization to collect these requests and manage their dispersal to maintenance personnel.
2. Transforming requests into changes
   Once a request is received, analysis must be conducted to gain an understanding of the scope of the request. It must be determined how the request will affect the current system and how long such a project will take. As with the initial development of a system, the size of a maintenance request can be analyzed for risk and feasibility.
3. Designing changes
   Change request can be transformed into a formal design change, which can then be fed into the maintenance implementation phase.
4. Implementing changes

The cost of Maintenance

Information systems maintenance costs are a significant expenditure. For some organizations, as much as 60 to 80 percent of their information systems budget is allocated to maintenance activities. On average, 52 percent of a company's programmers are assigned to maintain existing software. Only 3 percent are assigned to new application development.

Numerous factors influence the maintainability of a system. Of these factors, three are most significant; the number of latent defects, the number of customers, and documentation quality. The others are personnel, tools and software structure.

- Latent defects: This is the number of unknown errors existing in the system after it is installed. Because corrective maintenance accounts for most maintenance activity, the number of latent defects in a system influences most of the cost associated with maintaining a system.
- Number of customers for a given system: Greater the number of customers, the greater the maintenance costs. For example, if a system has only one customer, problem and change requests will come from only one source. Training, error reporting and support will be simpler.
- Quality of system documentation: Without quality documentation, maintenance efforts can increase exponentially. Quality documentation makes it easier to find code that needs to be changed and to understand how the code needs to be changed. Good documentation also explains why a system does what it does and why alternatives were not feasible.
- Maintenance personnel: In some organizations, best programmers are assigned to maintenance. Highly skilled programmers are needed because the maintenance programmer is typically not the original programmer and must quickly understand and carefully change the software.
- Tools: Tools that can automatically produce system documentation where none exists can also lower maintenance costs. Tools that can automatically generate new code based on system specification changes can dramatically reduce maintenance time and costs.
- Well structured programs: Well designed programs are easier to understand and fix.

Managing Maintenance

Maintenance activities consume more and more of the systems development budget, maintenance management has become increasingly important.

Managing Maintenance Personnel

Many organizations had a “maintenance group” that was separate from the “development group”. With the increased number of maintenance personnel, the development of formal methodologies and tools, changing organizational forms, end user computing, and the widespread use of very high level languages for the development of some systems, organizations have rethought the organization of maintenance and development personnel. One key issue is that many systems professionals don’t want to perform maintenance because they feel that it is more exciting to build something new than change an existing system.

Measuring Maintenance Effectiveness

To measure effectiveness, we must measure the following factors:

- Number of failures
- Time between each failure
- Type of failure

Measuring the number of and time between failures will provide us with the basis to calculate a widely used measure of system quality. This metric is referred to as the mean time between failures (MTBF).

A more revealing method of measurement is to examine the failures that are occurring. Over time, logging the types of failures will provide a very clear picture of where, when and how failures occur. Tracking the types of

failures also provides important management information for future projects. For example, if a higher frequency of errors occurs when a particular development environment is used, such information can help guide personnel assignments, training courses, or the avoidance of a particular package, language, or environment during future development. Without measuring and tracking maintenance activities, we cannot gain the knowledge to improve or know how well we are doing relative to the past. To effectively manage and to continuously improve, we must measure and assess performance over time.

Hardware/Software selection and the Computer Contract

Selecting hardware and software for implementing information system in an organization is a serious and time-consuming process that passes through several phases. The main steps of the selection process are listed below:

1. **Requirement analysis: -** System configuration requirements are clearly identified and a decision to acquire the system is taken in this step.

2. **Preparation of tender specifications: -** After studying the feasibility and deciding upon the configuration, tender documents are prepared for the benefit of vendors to clarify the details of various specifications, as listed below.

   i. Purchase procedure and schedule: it includes

      a) Date of tender submission
      b) Evaluation criteria
      c) Scope for negotiations, if any and
      d) Expected usage environment and load pattern

   ii. Equipment specification: Detailed technical specifications of each item required for both mandatory and optional items.

   iii. Quotation format:

      a) Format for stating technical details and quoting prices
      b) Whether deviations from specifications should be specifically listed
      c) Prices and levies (duties, taxes etc.) could be quoted as lumpsum or required separately.
      d) Required validity of the quotation.
      e) Earnest money deposit required, if any.

   iv. Proposed terms of contract
      a) Expected delivery schedule.
      b) Uptime warranties required
      c) Penalty clause, if any
      d) Payment terms (Whether advance payment acceptable)
      e) Arbitrary clauses
      f) Training needs.
      g) Post warranty maintenance terms expected.

   v. Any additional information required

3. **Inviting tenders**: After the preparation of tender specifications, tenders are invited. Invitation of tenders may depend upon the magnitude of purchase (estimate equipment cost). It may be through

   i) Open tender (through newspaper advertisement)

   ii) Limited tender (queries sent to a few selected vendors)

iii) Propriety purchase (applies mostly to upgrade requirements)

iv) Direct purchase from market. (applies mostly to consumables)

4. **Technical scrutiny and short listing**

   i) All tendered bids are opened on a pre-defined date and time.

   ii) Deviations from the specifications, if any, in each bid are noted.

   iii) A comparative summary is prepared against the list of tendered technical features.

   Additional factors to considered are:

   i) Financial health of the vendor (from balance sheets)

   ii) Nature and extent of support (from information provided on number of support staff per installed site and cross-check with selected customers)

   iii) Engineering quality pf products (factory inspection of product facilities, QA procedures and R&D)

5. **Detailed evaluation of short listed vendors: -** This step primarily involves getting any finer technical clarifications. Visits to customer sites and factory inspections may be planned. If any specific performance requirement is stipulated, the offered product is to be examined at this stage through suitable benchmark tests. For benchmark tests, standard benchmarks may be used as adequate performance indicators.

6. **Negotiation and procurement decision: -** Because of the extensive competition, computer system vendors may offer significant concessions. Negotiations are held to maximize these concessions. However, price negotiations are often not permitted by some organizations.

   When price negotiations are permitted, the committee members should have a good knowledge of the prevailing market prices, current trends, and also the duty/tax structure.

   i) Computer magazines
   ii) Vendor directories.
   iii) Contact with other users
   iv) Past personal experience.

7. **Delivery and installation: -** In this step, the vendor delivers the hardware/software to the buyer's organization, where it is matched with the specifications mentioned in the purchase order. If conforms to these specifications, the vendor installs the system in the premises of the organization.

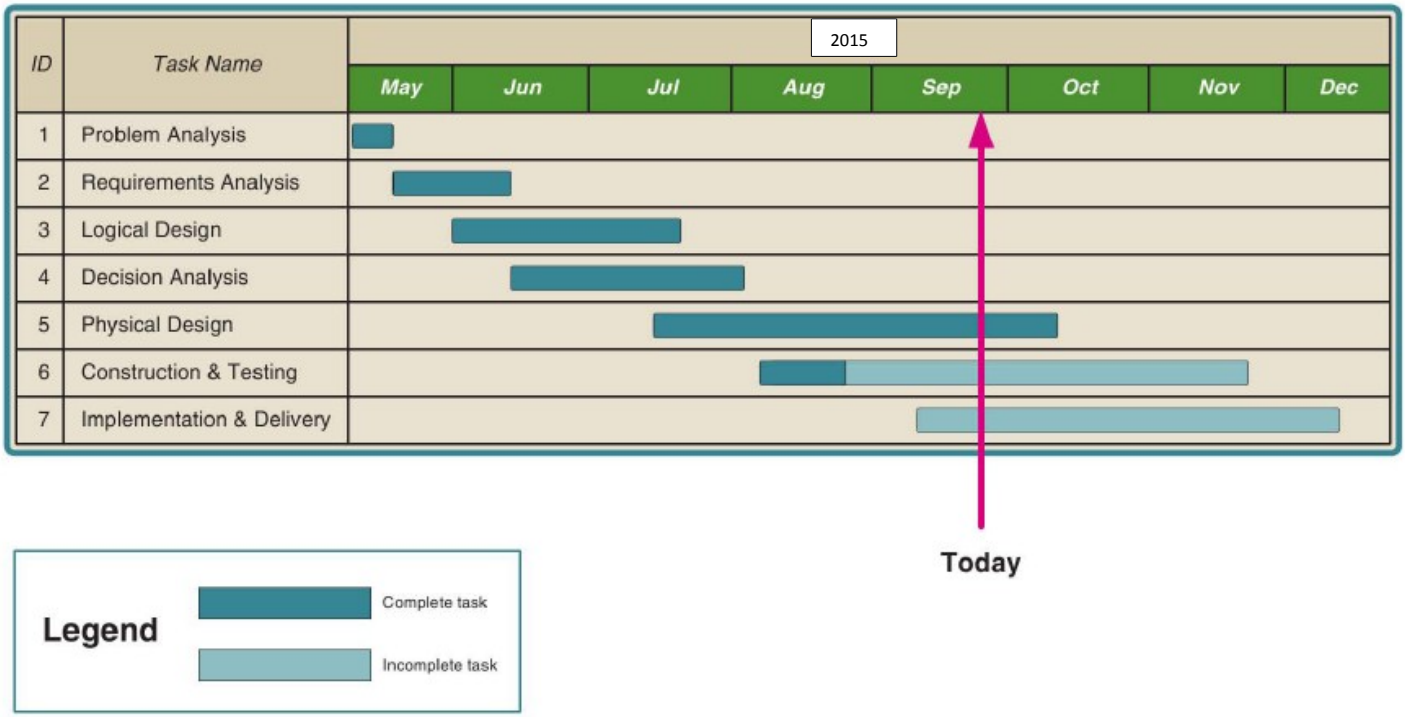
8. **Post-installation review: -** After the system is installed, a system evaluation is made to determine how closely the new system conforms to the plan. A post-installation review, in which system specifications and user requirements are audited, is made. The feedback obtained in this step helps in taking corrective decision.

Project Scheduling and Software

A project schedule is a specific timetable, usually in the form of charts that show tasks, task dependencies, and critical tasks that might delay the project. Project scheduling also involves selecting and staffing the project team, assigning specific tasks to team members, and arranging for other necessary resources.
When scheduling a project, the project manager must know the duration of each task, the order in which the activities will be performed, the start and end times for each task, and the person(s) assigned to each specific task. Two tools use for scheduling projects are GANTT chart and PERT chart.

GANTT chart

Gantt charts were developed by Henry L. Gantt, a mechanical engineer and management consultant. His goal was to design a chart that could show planned and actual progress on a complex project.
A **Gantt chart** is a horizontal bar chart that represents a series of tasks. The chart displays time on the horizontal axis and arranges the tasks vertically, from top to bottom. The position of the bar shows the planned start and end of the task, and the length of the bar indicates its duration. On the horizontal axis, time can be shown as elapsed time from a fixed starting point, or as actual calendar dates.



Gantt charts offer the advantage of clearly showing *overlapping* tasks, that is, tasks that can be performed at the same time. The bars can be shaded to clearly indicate percentage completion and project progress. Gantt charts are more effective when you are seeking to communicate schedule.

PERT chart

A PERT (Program Evaluation Review Technique) chart is a graphical network model that depicts a project's tasks and the relationships between those tasks. PERT was developed to make clear the *interdependence* between project tasks before those tasks are scheduled. The arrows indicate that one task is dependent on the start or completion of another task. PERT charts are more effective when you want to study the relationships between tasks.

Scope Defination	
5-3-2015	N/A
5-3-2015	N/A

Problem Analysis	
5-3-2015	5-12-2015
5-3-2015	5-11-2015

Requirement Analysis	
5-12-2015	6-12-2015
5-12-2015	6-14-2015

Logical Design	
5-28-2015	7-15-2015
5-30-2015	7-18-2015

Decision Analysis	
6-13-2015	7-30-2015
6-13-2015	8-3-2015

Physical Design	
7-3-2015	6-12-2015
7-5-2015	6-14-2015

Construction and Testing	
7-19-2015	11-13-2015
7-20-2015	In progress

Implement and delivery	
9-10-2015	12-14-2015
TBD	TBD

Legend

Task	
Scheduled Start	Scheduled Finish
Actual Start	Actual Finish

Intertask Dependency

Task	
Scheduled Start	Scheduled Finish
Actual Start	Actual Finish

TBD: To Be Determined

N/A: Not Applicable

University Exam questions

1. Explain the process of maintaining the information system with example. (2069) (10 marks).
2. What are the two important things to remember about testing systems? (2069) (5 marks).
3. Differentiate between system documentation and user documentation. (2069) (5 marks).
4. What are the different types of maintenance? (2069) (5 marks).
5. What do you mean by quality assurance? Explain with example. (2070) (5 marks).
6. Comparison between corrective, adaptive, perfective and preventive maintenance. (2070) (5 marks).
7. What are the main deliverable from testing and installation? (2070) (5 marks).
8. Explain the various types of system testing with example. (2071)(5 marks).
9. Explain the process of software maintenance. (2071)(5 marks).

Unit-7

Object Oriented Analysis and Design

Introduction

Object-oriented analysis is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.
Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design.

Object Oriented Development life cycle

The Object Oriented Methodology of Building Systems takes the objects as the basis. For this, first the system to be developed is observed and analyzed and the requirements are defined as in any other method of system development. Once this is done, the objects in the required system are identified. For example in case of a Banking System, a customer is an object, a chequebook is an object, and even an account is an object.
Object oriented development life cycle contains:

1. **System Analysis** As in any other system development model, system analysis is the first phase of development in case of Object Modeling too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning. Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized.
2. **System Design** System Design is the next development stage where the overall architecture of the desired system is decided. The system is organized as a set of sub systems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user. As the basic philosophy of Object-Oriented method of system analysis is to perceive the system as a set of interacting objects, a bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system as in the case of traditional Waterfall Model where the processes form the important part of the system.
3. **Object Design** In this phase, the details of the system analysis and system design are implemented. The Objects identified in the system design phase are designed. Here the implementation of these objects is decided as the data structures get defined and also the interrelationships between the objects are defined. Object Oriented Philosophy is very much similar to real world and hence is gaining popularity as the systems here are seen as a set of interacting objects as in the real world. To implement this concept, the process-based structural programming is not used; instead objects are created using data structures. Just as every programming language provides various data types and various variables of that type can be created, similarly, in case of objects certain data types are predefined. For example, we can define a data type called pen and then create and use several objects of this data type. This concept is known as creating a class.
   Class: A class is a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined. The class defines the basic attributes and the operations of the objects of that type. Defining a class does not define any object, but it only creates a template. For objects to be actually created instances of the class are created as per the requirement of the case.
   Abstraction: Classes are built on the basis of abstraction, where a set of similar objects are observed and their common characteristics are listed. Of all these, the characteristics of concern to the system under observation are picked up and the class definition is made. The attributes of no concern to the

system are left out. This is known as abstraction. The abstraction of an object varies according to its application. For instance, while defining a pen class for a stationery shop, the attributes of concern might be the pen color, ink color, pen type etc., whereas a pen class for a manufacturing firm would be containing the other dimensions of the pen like its diameter, its shape and size etc.

Inheritance: Inheritance is another important concept in this regard. This concept is used to apply the idea of reusability of the objects. A new type of class can be defined using a similar existing class with a few new features. For instance, a class vehicle can be defined with the basic functionality of any vehicle and a new class called car can be derived out of it with a few modifications. This would save the developers time and effort as the classes already existing are reused without much change. Coming back to our development process, in the Object Designing phase of the Development process, the designer decides onto the classes in the system based on these concepts. The designer also decides on whether the classes need to be created from scratch or any existing classes can be used as it is or new classes can be inherited from them.

4. **Implementation** During this phase, the class objects and the interrelationships of these classes are translated and actually coded using the programming language decided upon. The databases are made and the complete system is given a functional shape.

The complete OO methodology revolves around the objects identified in the system. When observed closely, every object exhibits some characteristics and behavior. The objects recognize and respond to certain events. For example, considering a Window on the screen as an object, the size of the window gets changed when resize button of the window is clicked. Here the clicking of the button is an event to which the window responds by changing its state from the old size to the new size.

While developing systems based on this approach, the analyst makes use of certain models to analyze and depict these objects. The methodology supports and uses three basic Models:

Object Model - This model describes the objects in a system and their interrelationships. This model observes all the objects as static and does not pay any attention to their dynamic nature.

Dynamic Model - This model depicts the dynamic aspects of the system. It portrays the changes occurring in the states of various objects with the events that might occur in the system.

Functional Model - This model basically describes the data transformations of the system. This describes the flow of data and the changes that occur to the data throughout the system.

While the Object Model is most important of all as it describes the basic element of the system, the objects, all the three models together describe the complete functional system. As compared to the conventional system development techniques, OO modeling provides many benefits. Among other benefits, there are all the benefits of using the Object Orientation. Some of these are:

Reusability - The classes once defined can easily be used by other applications. This is achieved by defining classes and putting them into a library of classes where all the classes are maintained for future use. Whenever a new class is needed the programmer looks into the library of classes and if it is available, it can be picked up directly from there.

Inheritance - The concept of inheritance helps the programmer use the existing code in another way, where making small additions to the existing classes can quickly create new classes.

Programmer has to spend less time and effort and can concentrate on other aspects of the system due to the reusability feature of the methodology.

Data Hiding - Encapsulation is a technique that allows the programmer to hide the internal functioning of the objects from the users of the objects. Encapsulation separates the internal functioning of the object from the external functioning thus providing the user flexibility to change the external behaviour of the object making the programmer code safe against the changes made by the user.

The systems designed using this approach are closer to the real world as the real world functioning of the system is directly mapped into the system designed using this approach .

Advantages of object oriented methodology

- Object Oriented Methodology closely represents the problem domain. Because of this, it is easier to produce and understand designs.
- The objects in the system are immune to requirement changes. Therefore, allows changes more easily.
- Object Oriented Methodology designs encourage more re-use. New applications can use the existing modules, thereby reduces the development cost and cycle time.
- Object Oriented Methodology approach is more natural. It provides nice structures for thinking and abstracting and leads to modular design.

The Unified Modeling Language

The **Unified Modeling Language** (**UML**) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system. It was created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh.

UML is designed to enable users to develop an expressive, ready to use visual modeling language. In addition, it supports high level development concepts such as frameworks, patterns and collaborations. UML includes a collection of elements such as:

- Programming Language Statements
- Actors: specify a role played by a user or any other system interacting with the subject.
- Activities: These are tasks, which must take place in order to fulfill an operation contract. They are represented in activity diagrams.
- Business Process: includes a collection of tasks producing a specific service for customers and is visualized with a flowchart as a sequence of activities.
- Logical and Reusable Software Components

UML diagrams can be divided into two categories. The first type includes six diagram types representing structural information. The second includes the remaining seven representing general types of behavior. Structure diagrams are used in documenting the architecture of software systems and are involved in the system being modeled. Different structure diagrams are:

1. Class Diagram: represents system class, attributes and relationships among the classes.
2. Component Diagram: represents how components are split in a software system and dependencies among the components.
3. Deployment Diagram: describes the hardware used in system implementations.

4. Composite Structure Diagram: describes internal structure of classes.

5. Object Diagram: represents a complete or partial view of the structure of a modeled system.

6. Package Diagram: represents splitting of a system into logical groupings and dependency among the grouping.

Behavior diagrams represent functionality of software system and emphasize on what must happen in the system being modeled. The different behavior diagrams are:

- Activity Diagram: represents step by step workflow of business and operational components.
- Use Case Diagram: describes functionality of a system in terms of actors, goals as use cases and dependencies among the use cases.
- UML State Machine Diagram: represents states and state transition.
- Communication Diagram: represents interaction between objects in terms of sequenced messages.
- Timing Diagrams: focuses on timing constraints.
- Interaction Overview Diagram: provides an overview and nodes representing communication diagrams.
- Sequence Diagram: represents communication between objects in terms of a sequence of messages.

UML diagrams represent static and dynamic views of a system model. The static view includes class diagrams and composite structure diagrams, which emphasize static structure of systems using objects, attributes, operations and relations. The dynamic view represents collaboration among objects and changes to internal states of objects through sequence, activity and state machine diagrams.

Use Case Diagram

Use case diagram is used to capture the dynamic nature of a system. It consists of use cases, actors and their relationships. Use case diagram is used at a high level design to capture the requirements of a system. So it represents the system functionalities and their flow. Although the use case diagrams are not a good candidate for forward and reverse engineering but still they are used in a slightly differently way to model it.

Purpose of Use Case Diagram

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors.

Use Case Diagram objects

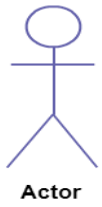
Use case diagrams consist of 4 objects.

- Actor

- Use case
- System
- Package

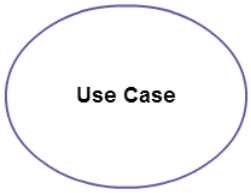
The objects are further explained below.

Actor
Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.

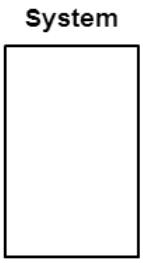


Use Case

A use case **represents a function or an action within the system**. Its drawn as an oval and named with the function.

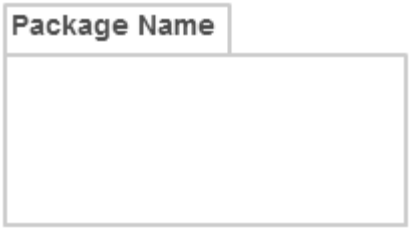


System
System is used to **define the scope of the use case** and drawn as a rectangle. This an optional element but useful when you are visualizing large systems. For example you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.

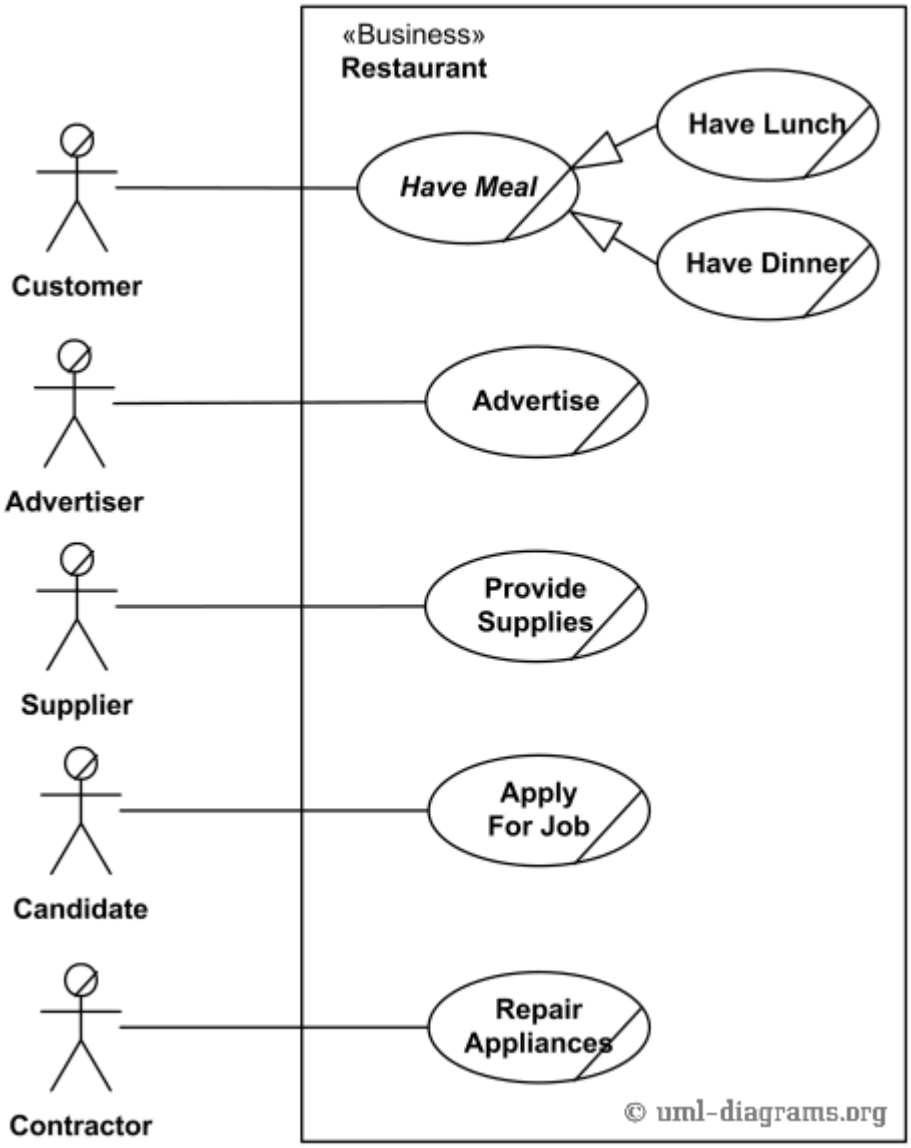


Package

Package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are **used to group together use cases**. They are drawn like the image shown below.

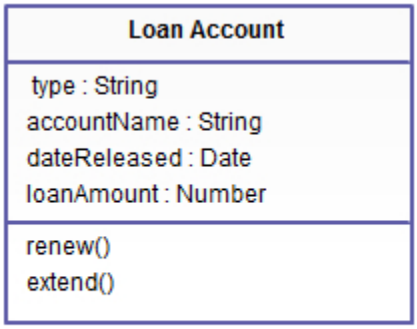


Example of use case diagram for a restaurant



Object modeling: Class Diagram

The class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams which can be mapped directly with object oriented languages. The class diagram shows a collection of classes, interfaces, associations, collaborations and constraints. It is also known as a structural diagram. The purpose of the class diagram is to model the static view of an application. The class diagrams are the only diagrams which can be directly mapped with object oriented languages and thus widely used at the time of construction.



Simple class diagram with attributes and operations

In the example, a class called "loan account" is depicted. Classes in class diagrams are represented by boxes that are partitioned into three:

1. The top partition contains the name of the class.

2. The middle part contains the class's attributes.

3. The bottom partition shows the possible operations that are associated with the class.

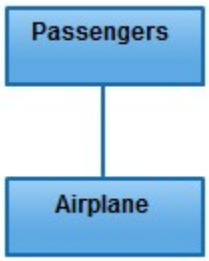
Those should be pretty easy to see in the example: the class being described is a loan account, some of whose attributes include the type of loan, the name of the borrower/loaner, the specific date the loan was released and the loan amount. As in the real world, various transactions or operations may be implemented on existing loans such as renew and extend. The example shows how class diagrams can encapsulate all the relevant data in a particular scenario in a very systematic and clear way.

In object-oriented modeling, class diagrams are considered the key building blocks that enable information architects, designers, and developers to show a given system's classes, their attributes, the functions or operations that are associated with them, and the relationships among the different classes that make up a system.

Relationships in Class Diagrams

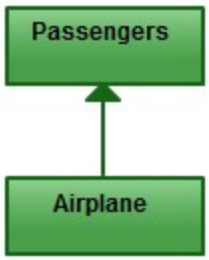
Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections.

Association



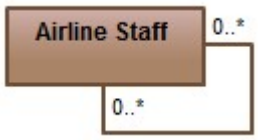
Association is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as above.

Directed Association



Directed Association refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

Reflexive Association

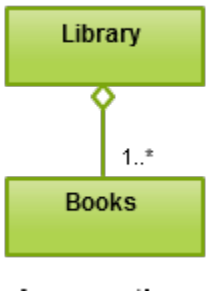


Reflexive Association occurs when a class may have multiple functions or responsibilities. For example, a staff working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

Multiplicity

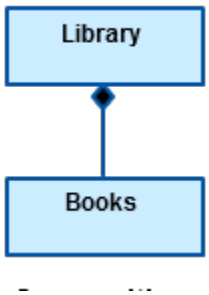
Multiplicity is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..* in the diagram means “zero to many”.

Aggregation



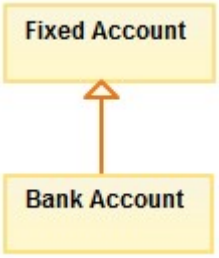
Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the life cycle of the container. In the same example, books will remain so even when the library is dissolved. To render aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

Composition



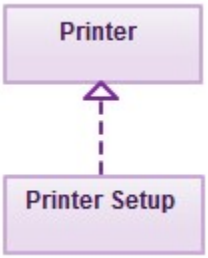
Composition is very similar to the aggregation relationship, with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag’s side pocket will also cease to exist once the shoulder bag is destroyed. To depict a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

Inheritance / Generalization

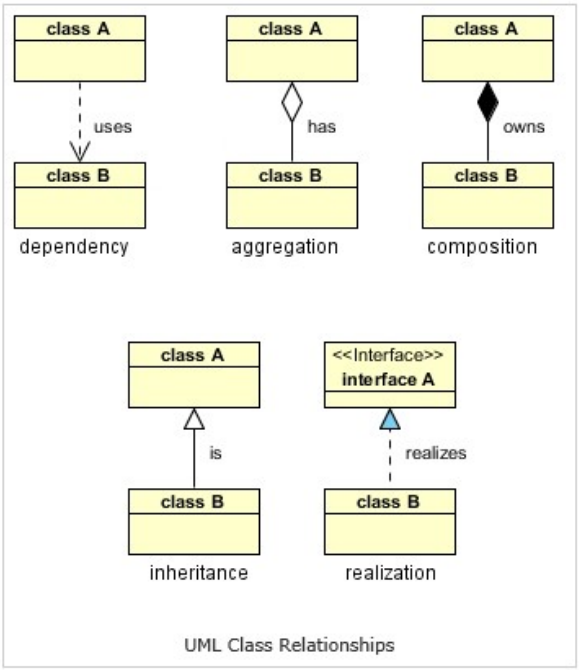


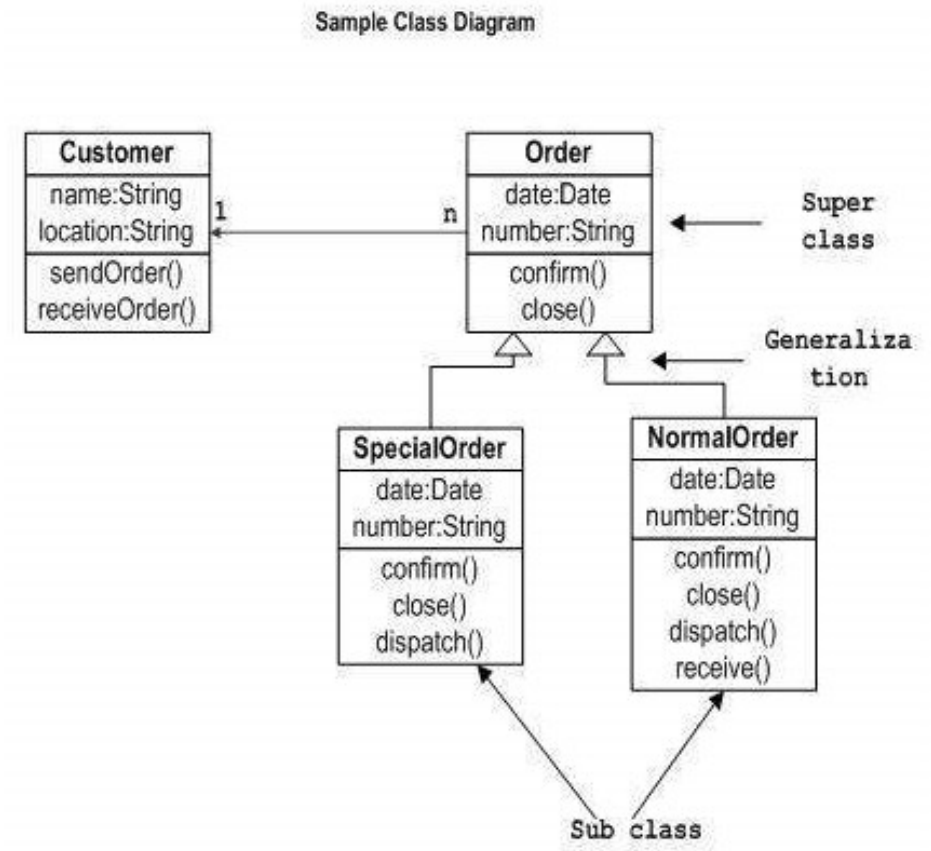
Inheritance refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To depict inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead

Realization



Realization denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality to the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.





Dynamic Modeling: State Diagram

State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and track the different states of its objects through the system. A state is a specific condition or situation of an element during its lifecycle.

Simple States

A simple state indicates a condition or situation of an element. For example, the project management system may be in one of the following simple states:

Inactive: Indicates that the project management system is not available to its users, because it is not started or has been shut down.

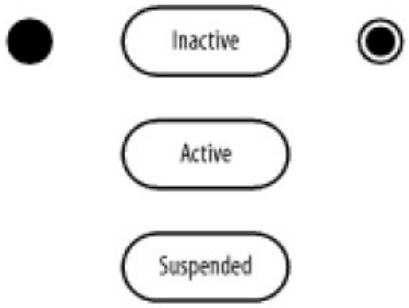
Active: Indicates that the project management system has been started and is available to its users.

Suspended: Indicates that the project management system has encountered some severe error, perhaps because it is running low on secondary storage and requires user intervention before becoming active again.

Initial and Final States

An initial state indicates the state of an element when it is created. In the UML, an initial state is shown using a small solid filled circle. A final state indicates the state of an element when it is destroyed. In the UML, a final state is shown using a circle surrounding a small solid filled circle.

A state diagram may have only one initial state, but may have any number of final states **initial, simple and final states**



Events

An event is an occurrence, including the reception of a request. There are a number of different types of events within the UML.

- CallEvent. Associated with an operation of a class, this event is caused by a call to the given operation. The expected effect is that the steps of the operation will be executed.
- SignalEvent. Associated with a signal, this event is caused by the signal being raised.
- TimeEvent. An event cause by expiration of a timing deadline.
- ChangeEvent. An event caused by a particular expression (of attributes and associations) becoming true.

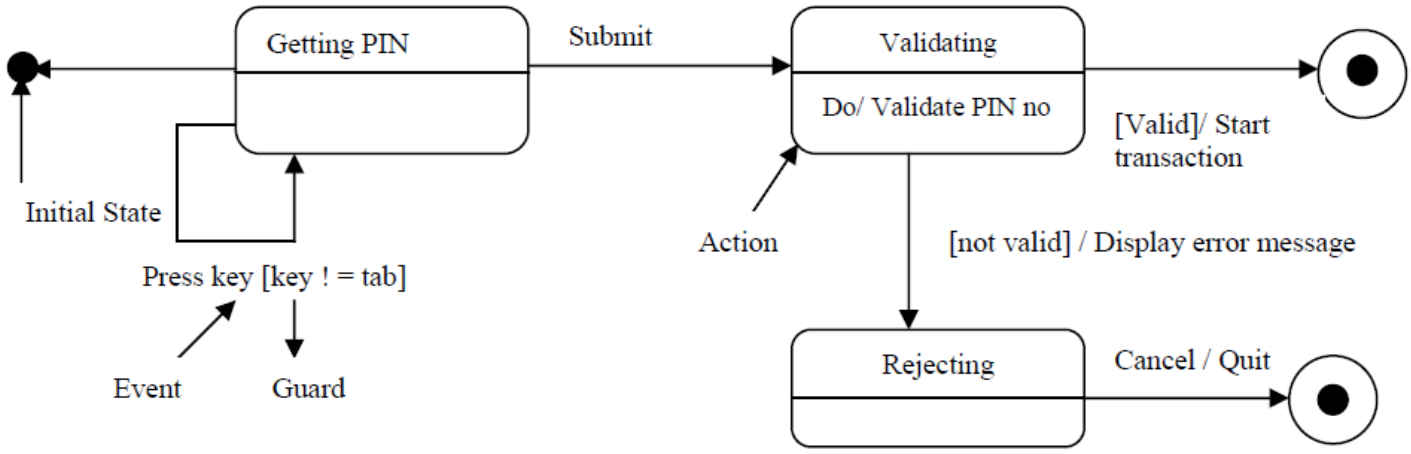


Figure: State diagram of online banking system

Logging in can be factored into four non-overlapping states: **Getting PIN**, **Validating**, and **Rejecting**. From each state comes a complete set of **transitions** that determine the subsequent state.

States are rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows. Our diagram has self-transition, on **Getting PIN**.

The initial state (black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form action. While in its **Validating** state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.

Dynamic Modeling: Sequence Diagram

A sequence diagram shows elements as they interact over time, showing an interaction or interaction instance. Sequence diagrams are organized along two axes: the horizontal axis shows the elements that are involved in the interaction, and the vertical axis represents time proceeding down the page. The elements on the horizontal axis may appear in any order.

Class Roles or Participants

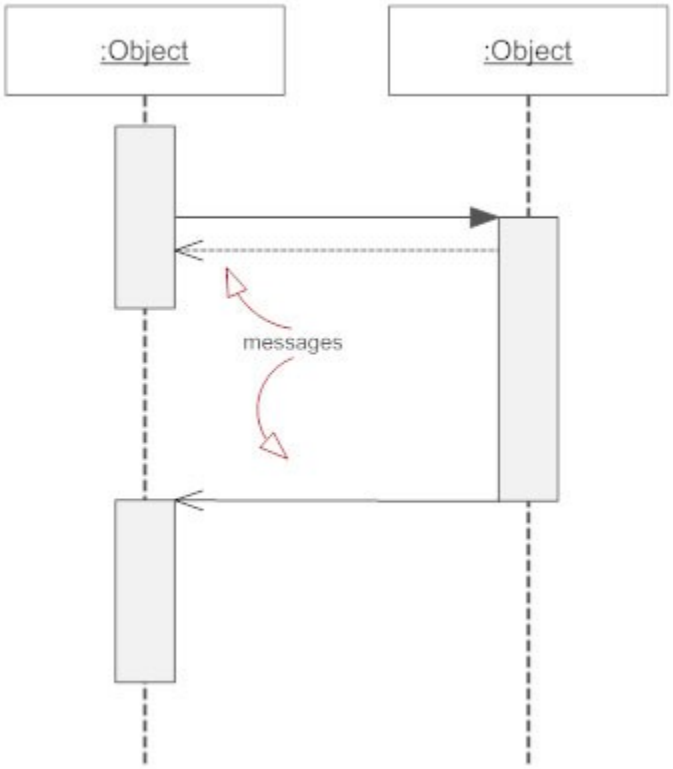
Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation or Execution Occurrence

Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

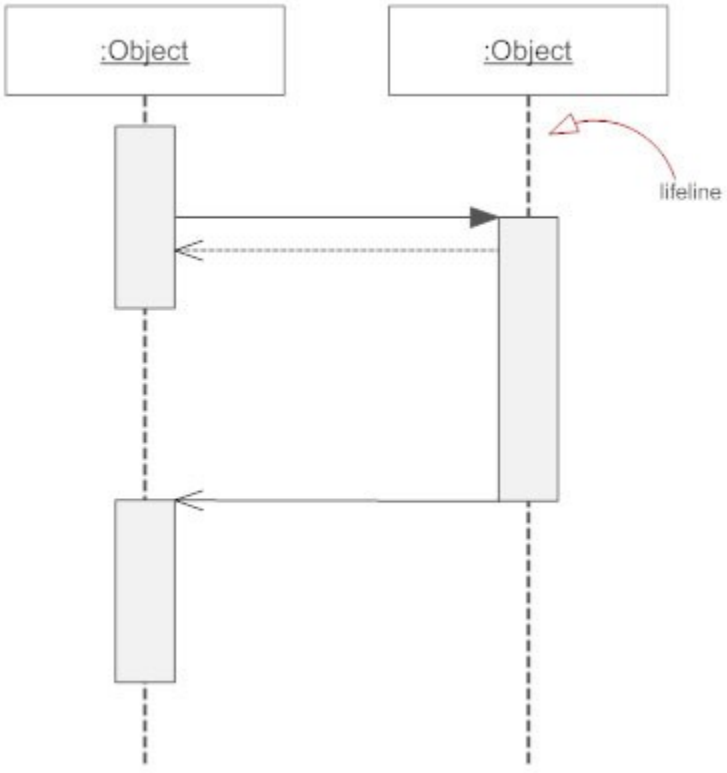
Messages

Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.



Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This

object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

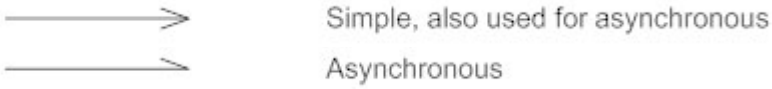
Types of messages in Sequence Diagram

Synchronous Message

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

Asynchronous Message

Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

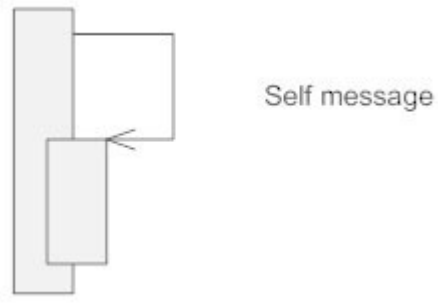


Reply or Return Message

A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

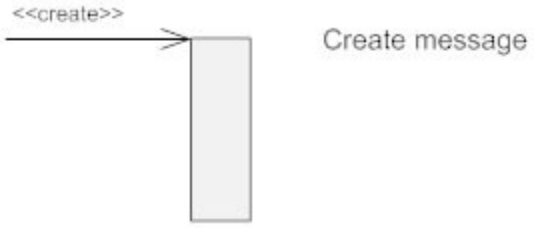
Self Message

A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.



Create Message

This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.



Delete Message

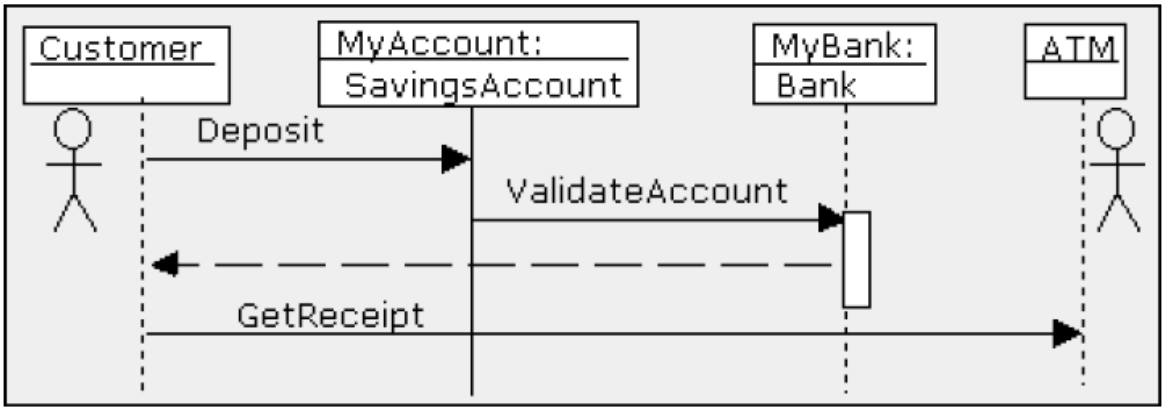
This is a message that destroys an object. It can be shown by an arrow with an x at the end.

Found Message

A message sent from an unknown recipient, shown by an arrow from an endpoint to a lifeline.

Lost Message

A message sent to an unknown recipient. It's shown by an arrow going from a lifeline to an endpoint, a filled circle or an x.



The Sequence diagram allows the person reading the documentation to ***follow the flow of messages*** from each object. The vertical lines with the boxes on top represent instances of the classes (or objects). The label to the left of the colon is the instance and the label to the right of the colon is the class. The horizontal arrows are the messages passed between the instances and are read from top to bottom. Here

- a customer (user) depositing money into MyAccount which is an instance of Class SavingsAccount.
- Then MyAccount object Validates the account by asking the Bank object, MyBank to ValidateAccount.
- Finally, the Customer Asks the ATM object for a Receipt by calling the ATM‘s operation GetReceipt.

The white rectangle indicate the scope of a method or set of methods occurring on the Object My Bank. The dotted line is a return from the method ValidateAccount.

Analysis verses Design

Analysis	Design
1. Analysis focus on determining what the business needs is.	1. Design takes those business needs and determines how they will be met through a specific system implementation.
2. The analysis phase includes activities designed to discover and document the features and functions the system must have.	2. The focus in design phase is to figure out how to create a system technically that will provide all those needed features and functions.
3. Analyst thoroughly studies the organization's current procedures and the information systems used to perform organizational task.	3. Analyst convert the description of the recommended alternative solution into logical and then physical system specifications.
4. In this phase analysts work with users to determine what the users want from a proposed system.	4. In this phase analysts design all aspects of the system, from input and output screens to reports, databases and computer processes.
5. This phase comes before design phase.	5. This phase comes after analysis phase.
6. The output of the analysis phase is a description of the alternative solution recommended by the analysis team.	6. The output of the design phase is the physical system specifications in a form ready to be turned over to programmers and other system builders for construction.

University Exam questions

1. Explain the Unified Modeling Language with example. (2069)(5 marks)
2. Differentiate between object modeling and dynamic modeling (2070)(2071)(5 marks)
3. What are the major differences between analysis and design? (2071)(5 marks)