2. Computer-aided software engineering (CASE):

- Computer-aided software engineering (CASE) is software tools that assist systems builders in managing the complexities of information system projects and helps ensure that high-quality systems are constructed on time and within budget.
- Visible Analyst (software) is one example of a CASE tool that helps to do graphical planning, analysis, and design in order to build complex client/server applications and databases. Microsoft Visio is a software that also allow users to draw and modify diagrams easily.
- The general types of CASE tools include:
  - ✓ Diagramming tools that enable system process, data, and control structures to be represented graphically.
  - ✓ Display (or form) and report generators make it easier for the systems analyst to identify data requirements and relationships.
  - ✓ A central repository that enables the integrated storage of specifications, diagrams, reports, and project management information.
  - ✓ Documentation generators that help produce both technical and user documentation in standard formats.
  - ✓ Code generators that enable the automatic generation of program and database definition code directly from the design documents, diagrams, forms, and reports.

3. AGILE Vs WATERFALL METHOD

Agile and Waterfall models are two different methods for software development process. Though they are different in their approach, both methods are useful at times, depending on the requirement and the type of the project.

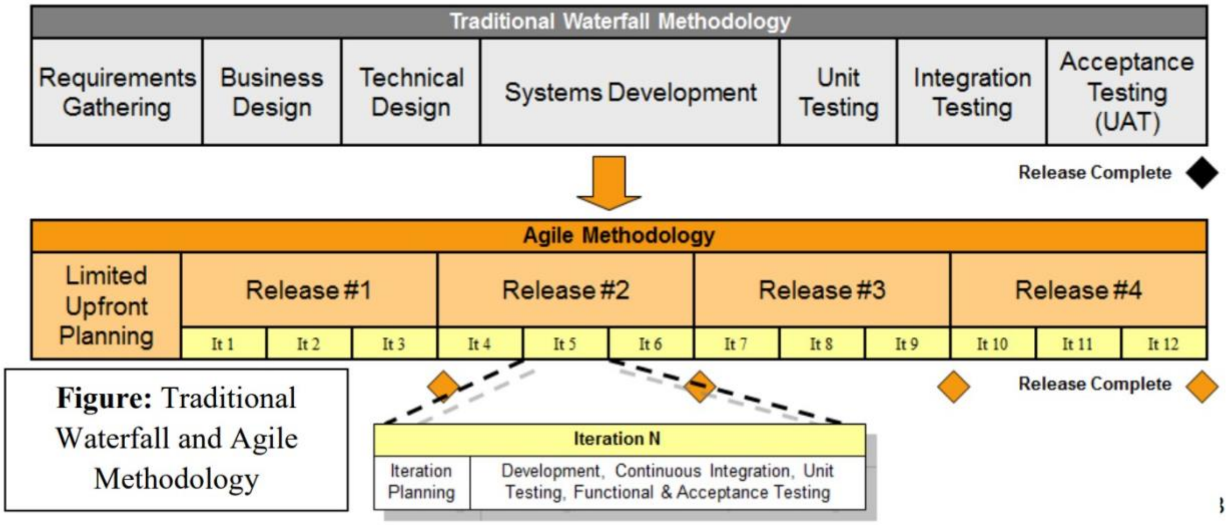
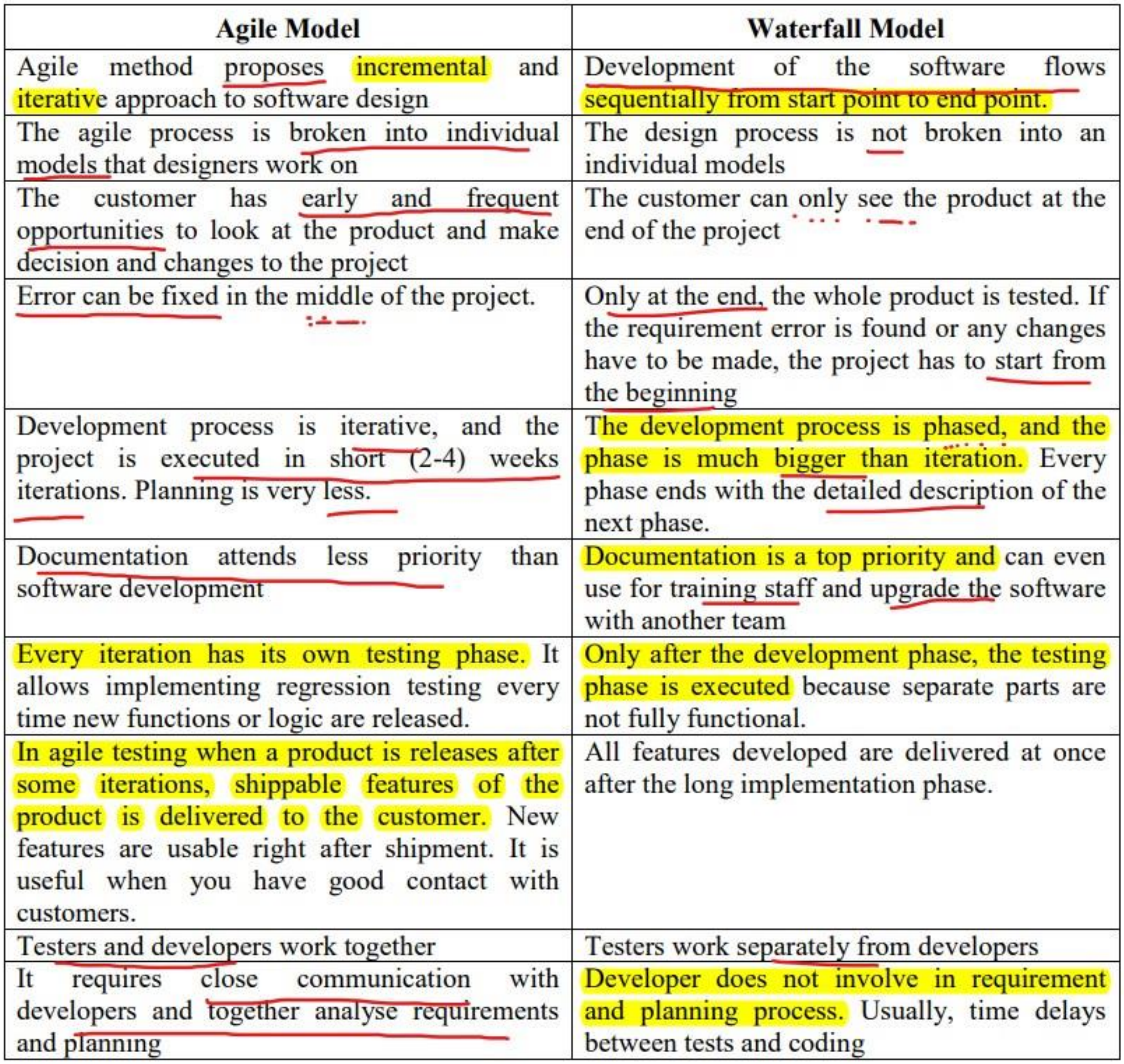


Figure: Traditional Waterfall and Agile Methodology



Agile Model	Waterfall Model
Agile method proposes incremental and iterative approach to software design	Development of the software flows sequentially from start point to end point.
The agile process is broken into individual models that designers work on	The design process is not broken into an individual models
The customer has early and frequent opportunities to look at the product and make decision and changes to the project	The customer can only see the product at the end of the project
Error can be fixed in the middle of the project.	Only at the end, the whole product is tested. If the requirement error is found or any changes have to be made, the project has to start from the beginning
Development process is iterative, and the project is executed in short (2-4) weeks iterations. Planning is very less.	The development process is phased, and the phase is much bigger than iteration. Every phase ends with the detailed description of the next phase.
Documentation attends less priority than software development	Documentation is a top priority and can even use for training staff and upgrade the software with another team
Every iteration has its own testing phase. It allows implementing regression testing every time new functions or logic are released.	Only after the development phase, the testing phase is executed because separate parts are not fully functional.
In agile testing when a product is releases after some iterations, shippable features of the product is delivered to the customer. New features are usable right after shipment. It is useful when you have good contact with customers.	All features developed are delivered at once after the long implementation phase.
Testers and developers work together	Testers work separately from developers
It requires close communication with developers and together analyse requirements and planning	Developer does not involve in requirement and planning process. Usually, time delays between tests and coding

4. FEASIBILITY ANALYSIS

Feasibility analysis guides the organization in determining whether to proceed with the project. Feasibility analysis also identifies the important risks associated with the project that must be managed if the project is approved. As with the system request, each organization has its own process and format for the feasibility analysis, but most include techniques to assess three areas: technical feasibility, economic feasibility, and organizational feasibility. The results of evaluating these six feasibility factors are combined into a feasibility study deliverable that is submitted to the approval committee at the end of project initiation. So, The feasibility study allows us to preview the potential outcome and to decide whether they should continue.

Technical feasibility

- The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organisation and their applicability to the expected needs of the proposed system.
- It is an evaluation of the hardware and software and how it meets the need of the proposed system.

Economic feasibility

- The purpose of the economic feasibility assessment is to determine the positive economic benefits to the organization that the proposed system will provide.
- It includes quantification and identification of all the benefits expected. This assessment typically involves a cost/ benefits analysis.
- Economic feasibility is also called cost-benefit analysis

Operational Feasibility

- Operational feasibility refers to the measure of solving problems with the help of a new proposed system. It helps in taking advantage of the opportunities and fulfills the requirements as identified during the development of the project. It takes care that the management and the users support the project.

Other feasibility analysis:

Schedule feasibility

- A project will fail if it takes too long to be completed before it is useful. Schedule feasibility is a measure of how reasonable the project timetable is.
- Schedule feasibility is to assess the duration of the project whether it is too long to be complete before it is useful. System analysts have to estimate how long the system will take to develop, and whether all potential timeframes and the completion date schedules can be met, as well as whether meeting these date will sufficient for dealing with the needs of the organization.
- When assessing schedule feasibility, a systems analyst must consider the interaction between time and costs.

Legal feasibility

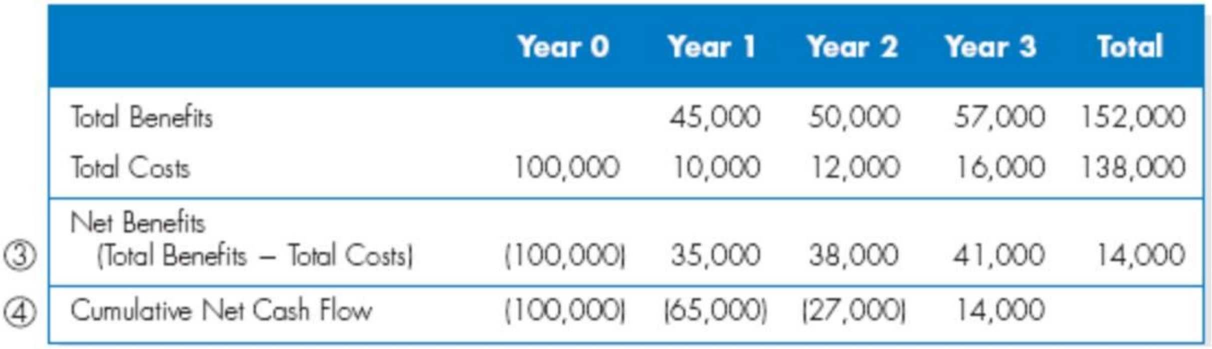
- Determines whether the proposed system conflicts with legal requirements, e.g. a data processing system must comply with the local data protection regulations and if the proposed venture is acceptable in accordance to the laws of the land.

Political Feasibility

- Political feasibility is to gain an understanding of how key stakeholders within the organization view the proposed system. The new information systems may affect the distribution of power and can have political ramification. Therefore, those stakeholders not supporting the project may block or disrupt the project.

5. CASH FLOW ANALYSIS AND MEASURES IN ECONOMIC FEASIBILITY ANALYSIS (COST–BENEFIT ANALYSIS)

IT projects commonly involve an initial investment that produces a stream of benefits over time, along with some ongoing support costs. Therefore, the value of the project must be measured over time. Cash flows, both inflows and outflows, are estimated over some future period. Then, these cash flows are evaluated using several techniques to judge whether the projected benefits justify incurring the costs. A very basic cash flow projection is shown in below figure to demonstrate these evaluation techniques. In this simple example, a system is developed in Year 0 (the current year) costing $100,000. Once the system is operational, benefits and ongoing costs are projected over three years. In row 3 of this figure, net benefits are computed by subtracting each year's total costs from its total benefits. Finally, in row 4, we have computed a cumulative total of the net cash flows.



		Year 0	Year 1	Year 2	Year 3	Total
	Total Benefits		45,000	50,000	57,000	152,000
	Total Costs	100,000	10,000	12,000	16,000	138,000
③	Net Benefits (Total Benefits – Total Costs)	(100,000)	35,000	38,000	41,000	14,000
④	Cumulative Net Cash Flow	(100,000)	(65,000)	(27,000)	14,000	

Figure: Simple Cash Flow Projection

There are some common methods for evaluating a project's worth can now be determined. Each of these calculations will be explained here:

Return on Investment:

- The return on investment (ROI) is a simple calculation that divides the project's net benefits (total benefits - total costs) by the total costs. The ROI formula is:

$$\text{ROI} = \frac{\text{Total Revenue- Total Cost}}{\text{Total Cost}}$$

$$\text{ROI} = \frac{152{,}000 - 138{,}000}{138{,}000}$$

- A high ROI suggests that the project's benefits far outweigh the project's cost.
- ROI is commonly used in practice; however, it is hard to interpret and should not be used as the only measure of a project's worth.

Break-Even Point:

- The break-even point (also called the payback method) is defined as the number of years it takes a firm to recover its original investment in the project from net cash flows.
- As shown in row 4 of above figure, the project's cumulative cash flow figure becomes positive during Year 3, so the initial investment is "paid back" over two years plus some fraction of the third year.

$$\text{BEP} = \begin{array}{l}\text{Number of}\\ \text{years of}\\ \text{negative}\\ \text{cash flow}\end{array} + \frac{\text{That year's Net Cash Flow} - \text{That year's Cumulative Cash Flow}}{\text{That year's Net Cash Flow}}$$

Using the values in Figure 1-8, the BEP calculation is:

$$\text{BEP} = 2 + \frac{41{,}000 - 14{,}000}{41{,}000} = 2 + \frac{28{,}000}{41{,}000} = 2.68 \text{ years}$$

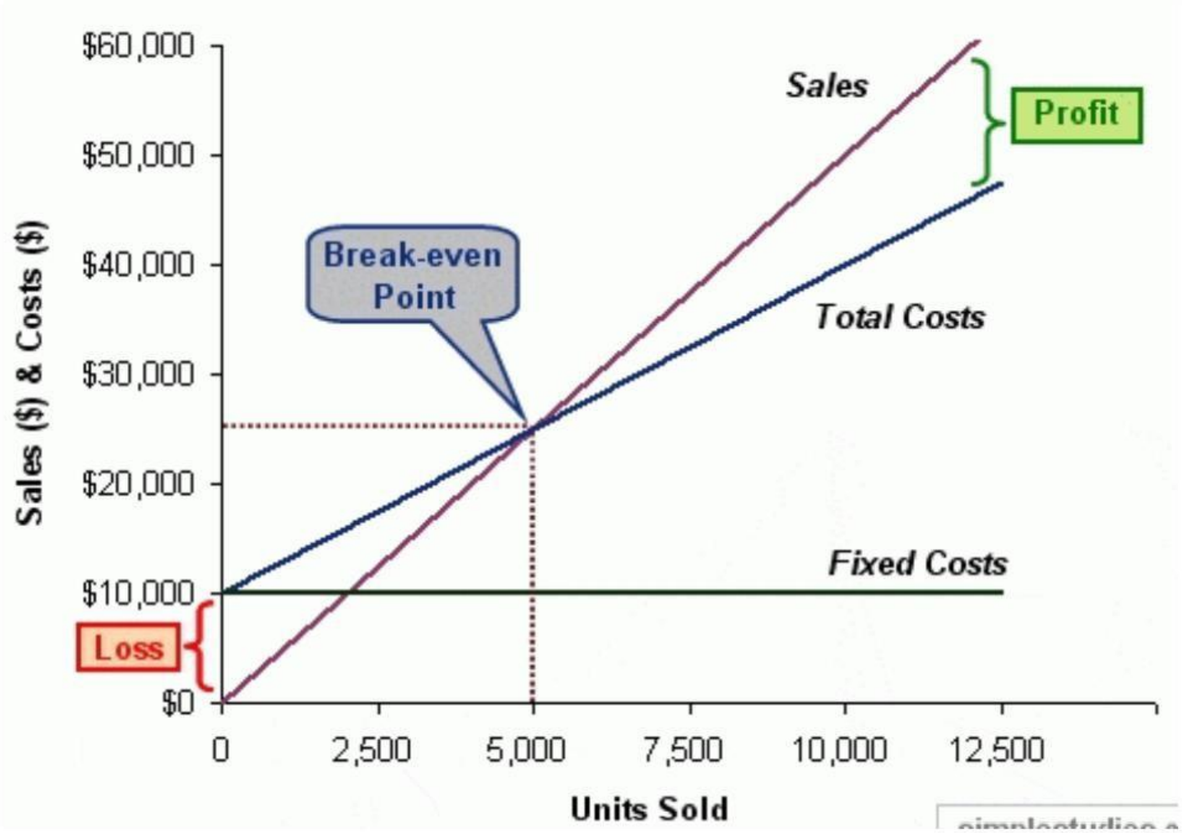


Figure: Break-Even Analysis

6. Guidelines for Drawing DFDs:

When you draw a context diagram and other DFDs, you should follow several guidelines:

- Draw the context diagram so it fits on one page.
- Use the name of the information system as the process name in the context diagram. For example, the process name in Figure 9 is GRADING SYSTEM. Notice that the process name is the same as the system name. For processes in lower-level DFDs, you would use a verb followed by a descriptive noun, such as ESTABLISH GRADEBOOK, ASSIGN FINAL GRADE, or PRODUCE GRADE REPORT.
- Use unique names within each set of symbols. For instance, the diagram in below figure shows only one entity named STUDENT and only one data flow named FINAL GRADE. Whenever you see the entity STUDENT on any other DFD in the grading system, you know that you are dealing with the same entity. Whenever the FINAL GRADE data flow appears, you know that you are dealing with the same data flow. The naming convention also applies to data stores.
- Do not cross lines. One way to achieve that goal is to restrict the number of symbols in any DFD. On lower-level diagrams with multiple processes, you should not have more than nine process symbols. Including more than nine symbols usually is a signal that your diagram is too complex and that you should reconsider your analysis.
- Provide a unique name and reference number for each process. Because it is the highest-level DFD, the context diagram contains process 0, which represents the entire information system, but does not show the internal workings. To describe the next level of detail inside process 0, you must create a DFD named diagram 0, which will reveal additional processes that must be named and numbered (See figure 10). As you continue to create lower-level DFDs, you assign unique names and reference numbers to all processes, until you complete the logical model.
- Obtain as much user input and feedback as possible. Your main objective is to ensure that the model is accurate, easy to understand, and meets the needs of its users.

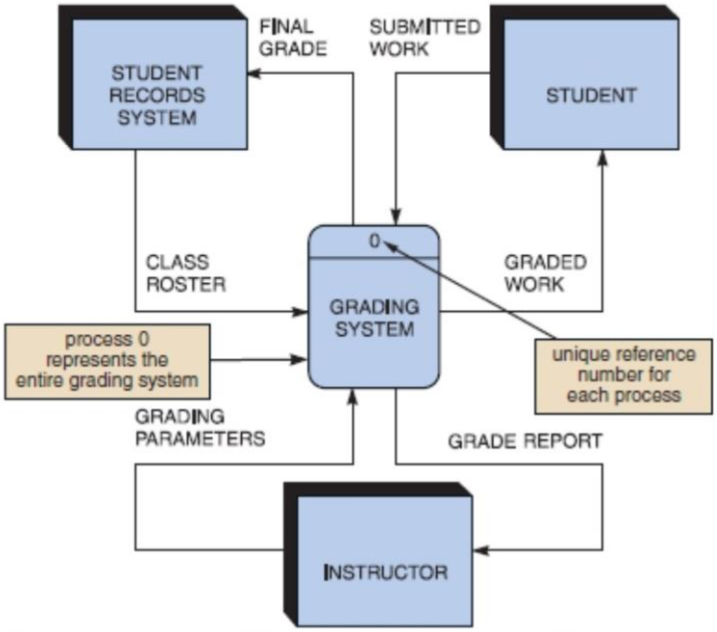


Figure 9: Context diagram DFD for a grading system.

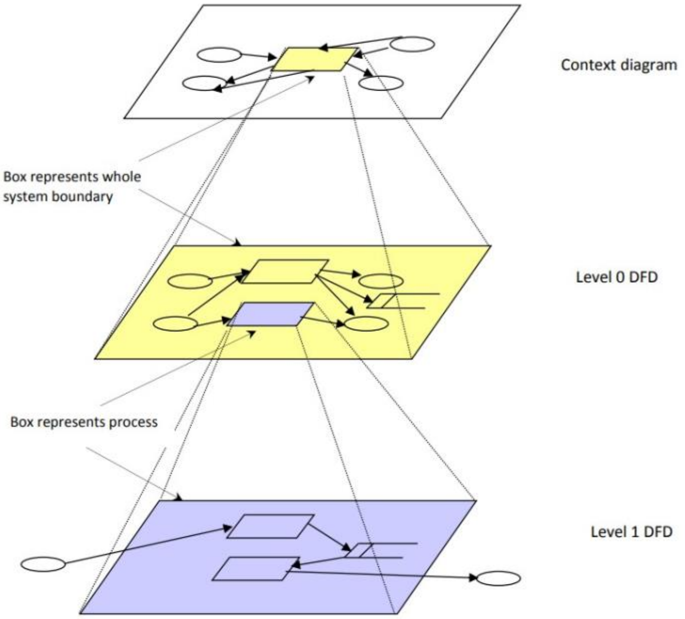


Figure 10: Different levels of DFD

Developing Data Flow Diagrams

Step 1: Draw a Context Diagram

A context diagram is a top-level view of an information system that shows the system's boundaries and scope. To draw a context diagram, you have to use a single process symbol, and you identify it as process 0.

Example:

context diagram for an order system The context diagram for an order system is shown in Figure 11. Notice that the ORDER SYSTEM process is at the center of the diagram and five entities surround the process. Three of the entities, SALES REP, BANK, and ACCOUNTING, have nine data flow.

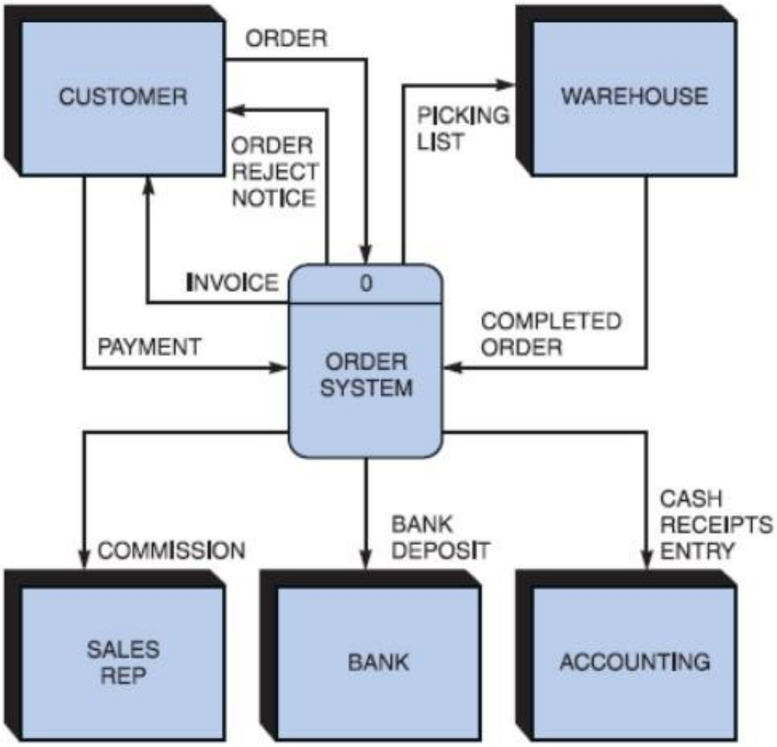


Figure 11: Context diagram DFD for an order system.

Step 2: Draw a Diagram 0 DFD

To show the detail inside the black box, you create DFD diagram 0. Diagram 0 zooms in on the system and shows major internal processes, data flows, and data stores. Diagram 0 also repeats the entities and data flows that appear in the context diagram. When you expand the context diagram into DFD diagram 0, you must retain all the connections that flow into and out of process 0.

Example: Diagram 0 DFD for an order system

1. A CUSTOMER submits an ORDER. Depending on the processing logic, the FILL ORDER process either sends an ORDER REJECT NOTICE back to the customer or sends a PICKING LIST to the WAREHOUSE.

2. A COMPLETED ORDER from the WAREHOUSE is input to the CREATE INVOICE process, which outputs an INVOICE to both the CUSTOMER process and the ACCOUNTS RECEIVABLE data store.

3… A CUSTOMER makes a PAYMENT that is processed by APPLY PAYMENT. APPLY PAYMENT requires INVOICE DETAIL input from the ACCOUNTS RECEIVABLE data store along with the PAYMENT. APPLY PAYMENT also outputs PAYMENT DETAIL back to the ACCOUNTS RECEIVABLE data store and outputs COMMISSION to the SALES DEPT, BANK DEPOSIT to the BANK, and CASH RECEIPTS ENTRY to ACCOUNTING.

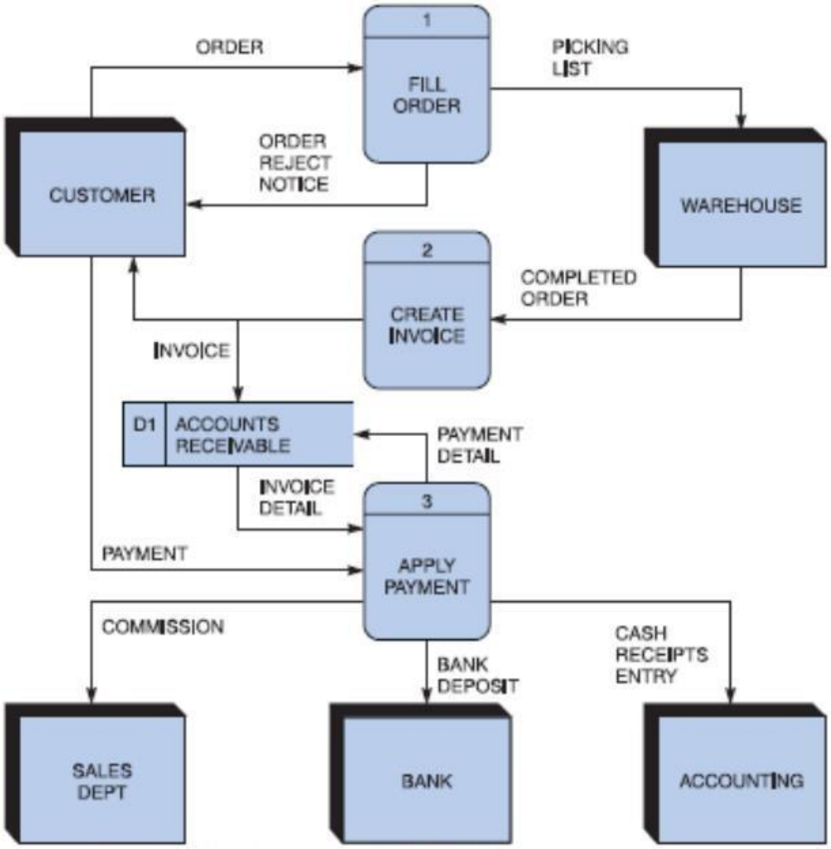


Figure 12: Diagram 0 DFD for the order system.

7. Leveling examples

Leveling uses a series of increasingly detailed DFDs to describe an information system. For example, a system might consist of dozens, or even hundreds, of separate processes. Using leveling, an analyst starts with an overall view, which is a context diagram with a single process symbol. Next, the analyst creates diagram 0, which shows more detail. The analyst continues to create lower-level DFDs which is the parent. When you explode the FILL ORDER process into diagram 1 DFD, however, you see that three processes (1.1, 1.2, and 1.3) interact with the two data stores, which now are shown.

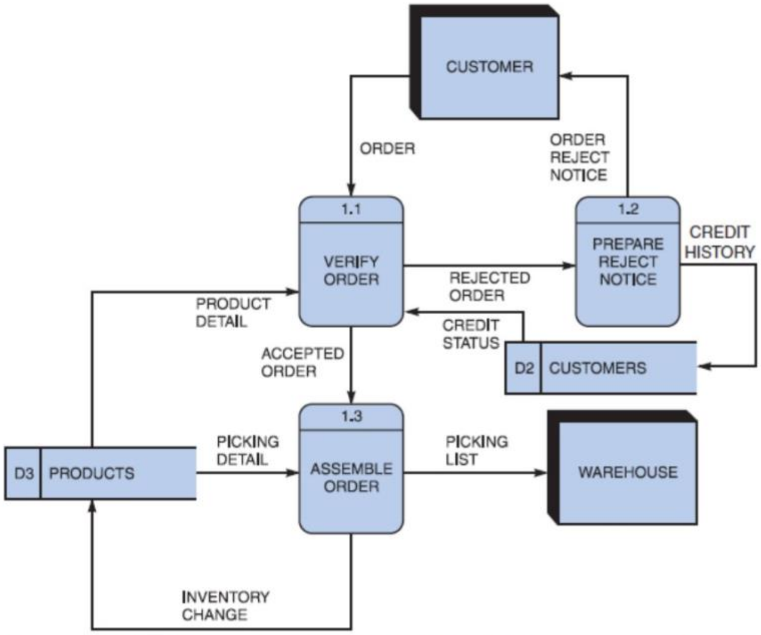


Figure 13: Diagram 1 DFD shows details of the FILL ORDER process in the order system.

Figures 12 and 13 provide an example of leveling. Figure 12 shows diagram 0 for an order system, with the FILL ORDER process labeled as process 1. Now consider Figure 5-17, which provides an exploded view of the FILL ORDER process. Notice that FILL ORDER (process 1) actually consists of three processes: VERIFY ORDER (process 1.1), PREPARE REJECT NOTICE (process 1.2), and ASSEMBLE ORDER (process 1.3).

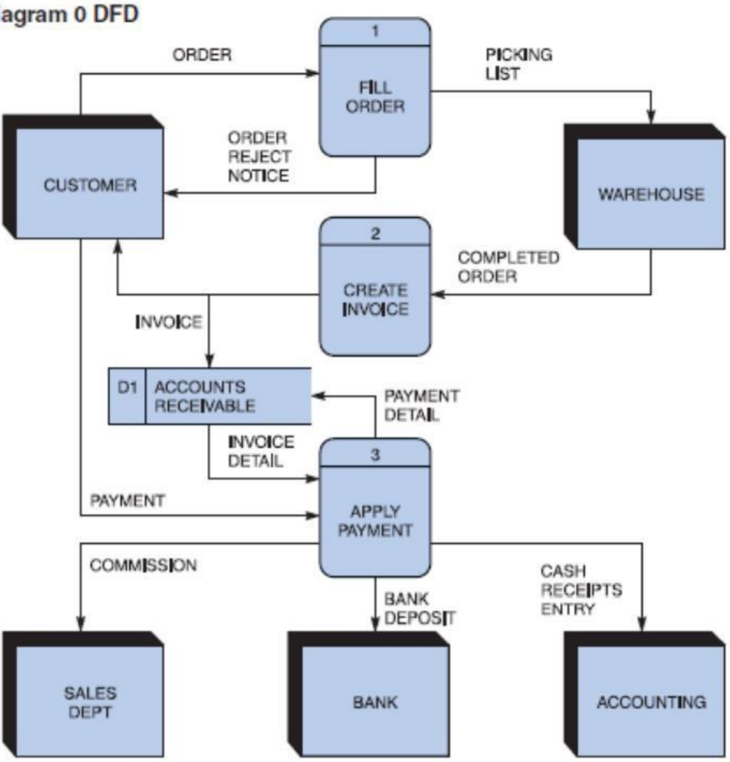
When you compare Figures 12 and 13, you will notice that Figure 13 (the exploded FILL ORDER process) shows two data stores (CUSTOMERS and PRODUCTS) that do not appear on Figure 5-16, which is the parent DFD. Why not? The answer is based on a simple rule: When drawing DFDs, you show a data store only when two or more processes use that data store. The CUSTOMERS and PRODUCTS data stores were internal to the FILL ORDER process, so the analyst did not show them on diagram 0, which is the parent. When you explode the FILL ORDER process into diagram 1 DFD, however, you see that three processes (1.1, 1.2, and 1.3) interact with the two data stores, which now are shown.

8. BALANCING EXAMPLES

Balancing ensures that the input and output data flows of the parent DFD are maintained on the child DFD. For example, Figure 14 shows two DFDs: The order system diagram 0 is shown at the top of the figure, and the exploded diagram 3 DFD is shown at the bottom.

The two DFDs are balanced, because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top. To verify the balancing, notice that the parent process 3, APPLY PAYMENT, has one incoming data flow from an external entity, and three outgoing data flows to external entities. Now examine the child DFD, which is diagram 3. Now, ignore the internal data flows and count the data flows to and from external entities. You will see that the three processes maintain the same one incoming and three outgoing data flows as the parent process.

Order System Diagram 0 DFD



Next

Order System Diagram 3 DFD

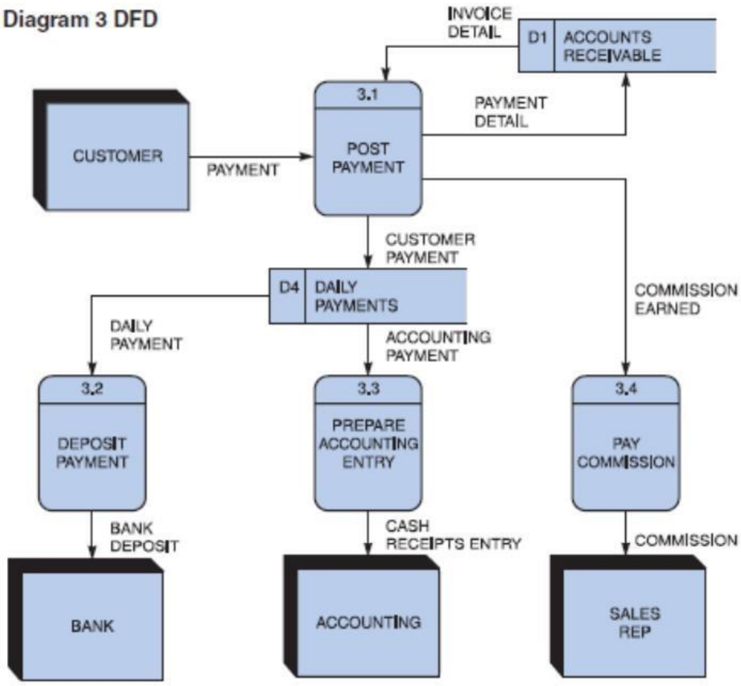


FIGURE 14: The order system diagram 0 is shown at the top of the figure, and exploded diagram 3 DFD (for the APPLY PAYMENT process) is shown at the bottom. The two DFDs are balanced, because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top.

9. ER Diagram:

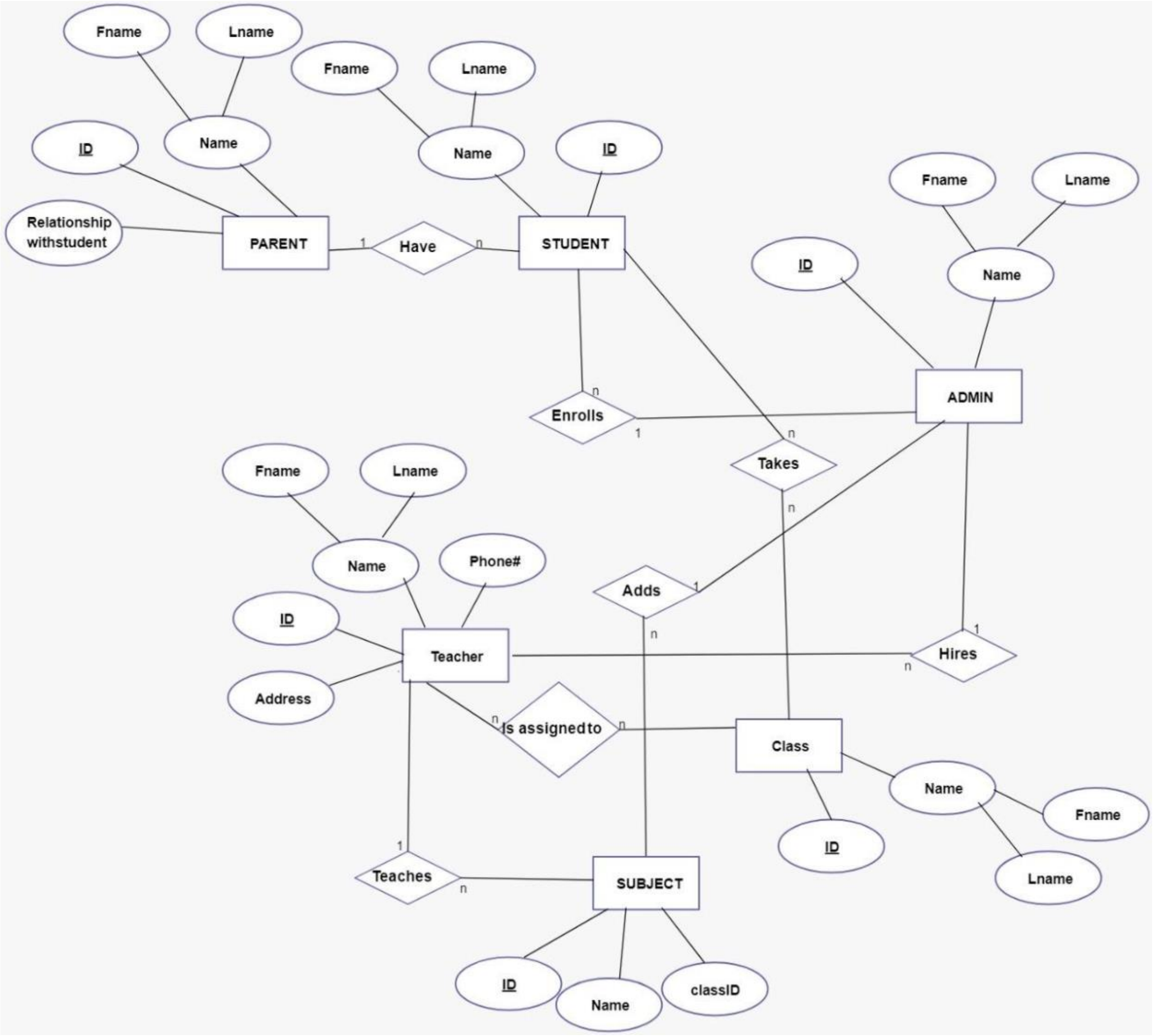


Figure: Hospital Management System

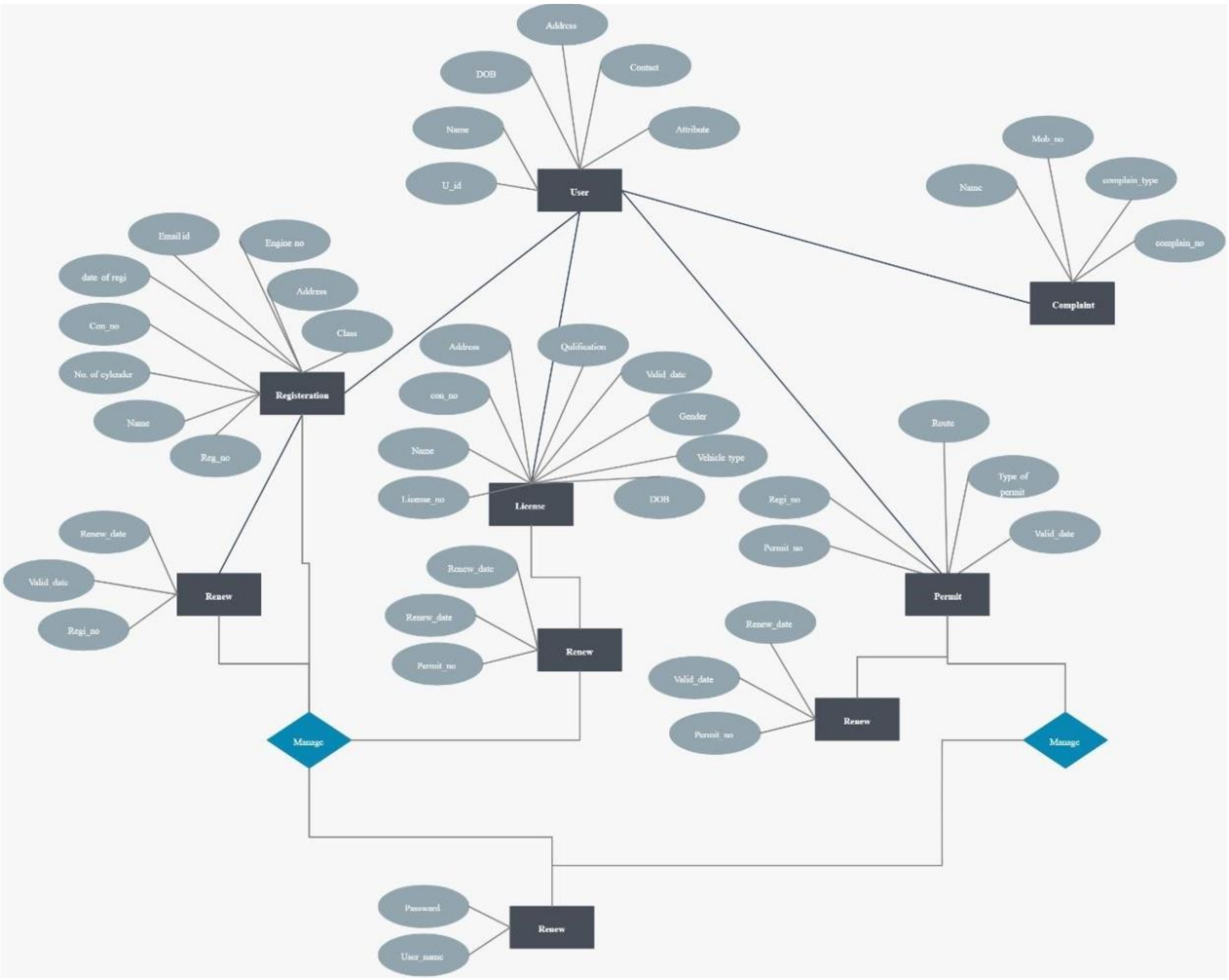


Figure:
Transport Management System

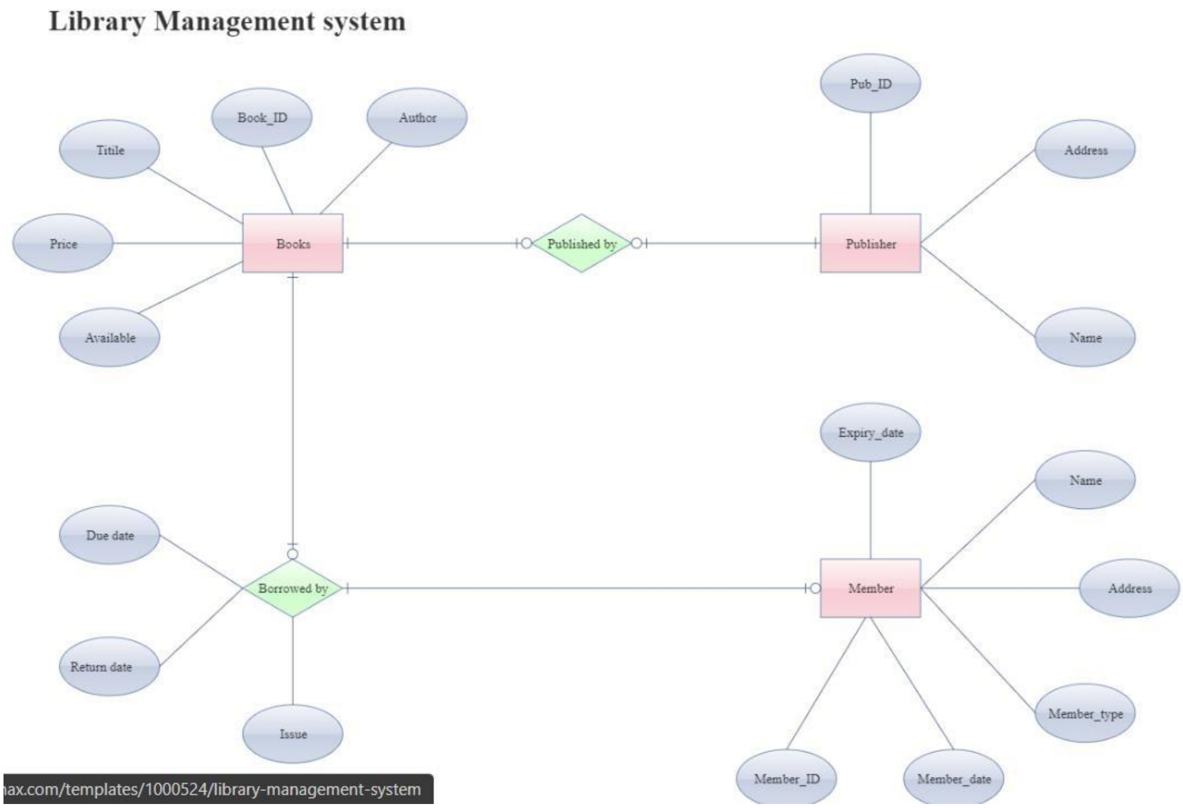


Figure:
Library Management System

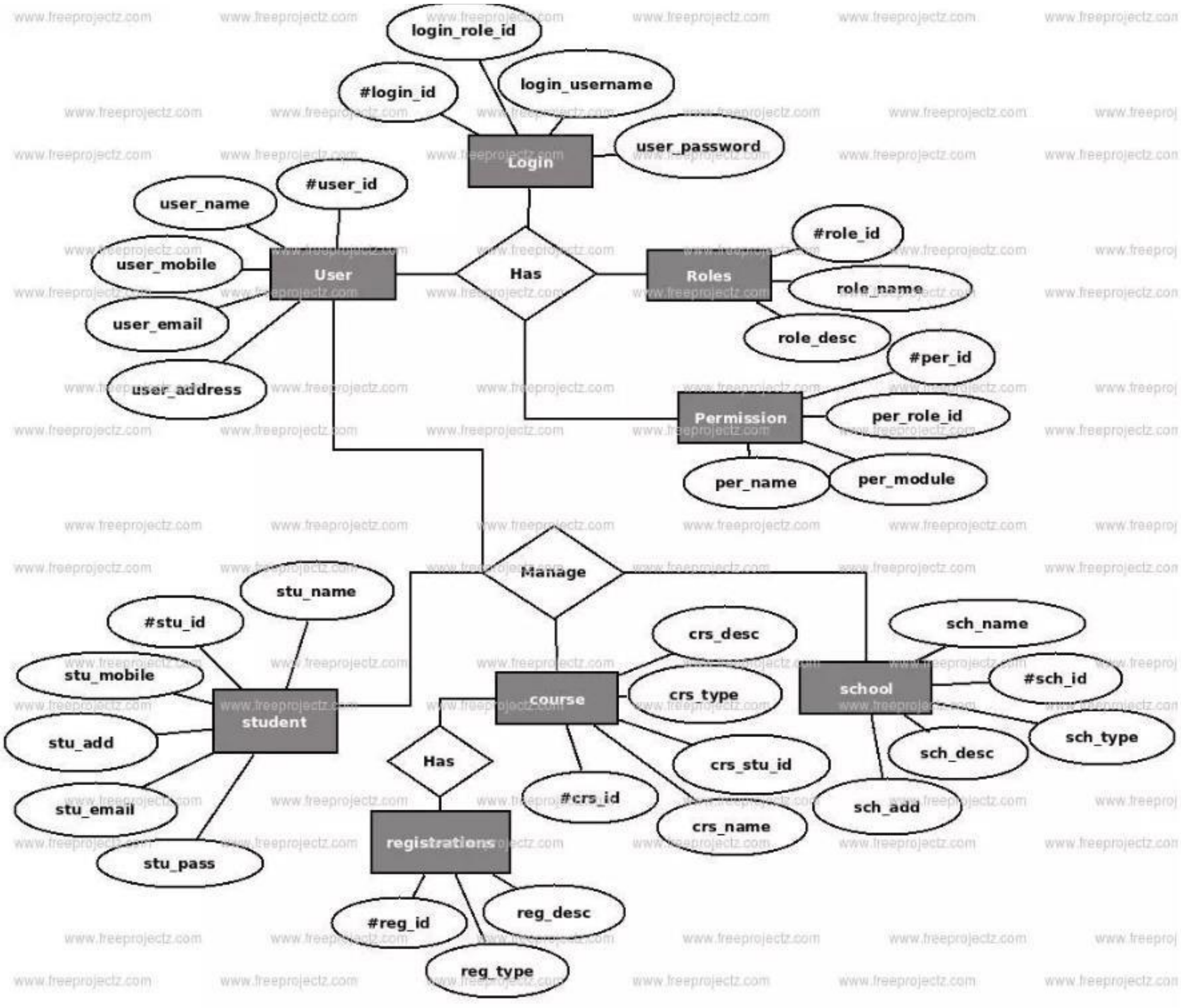


Figure: School Management System

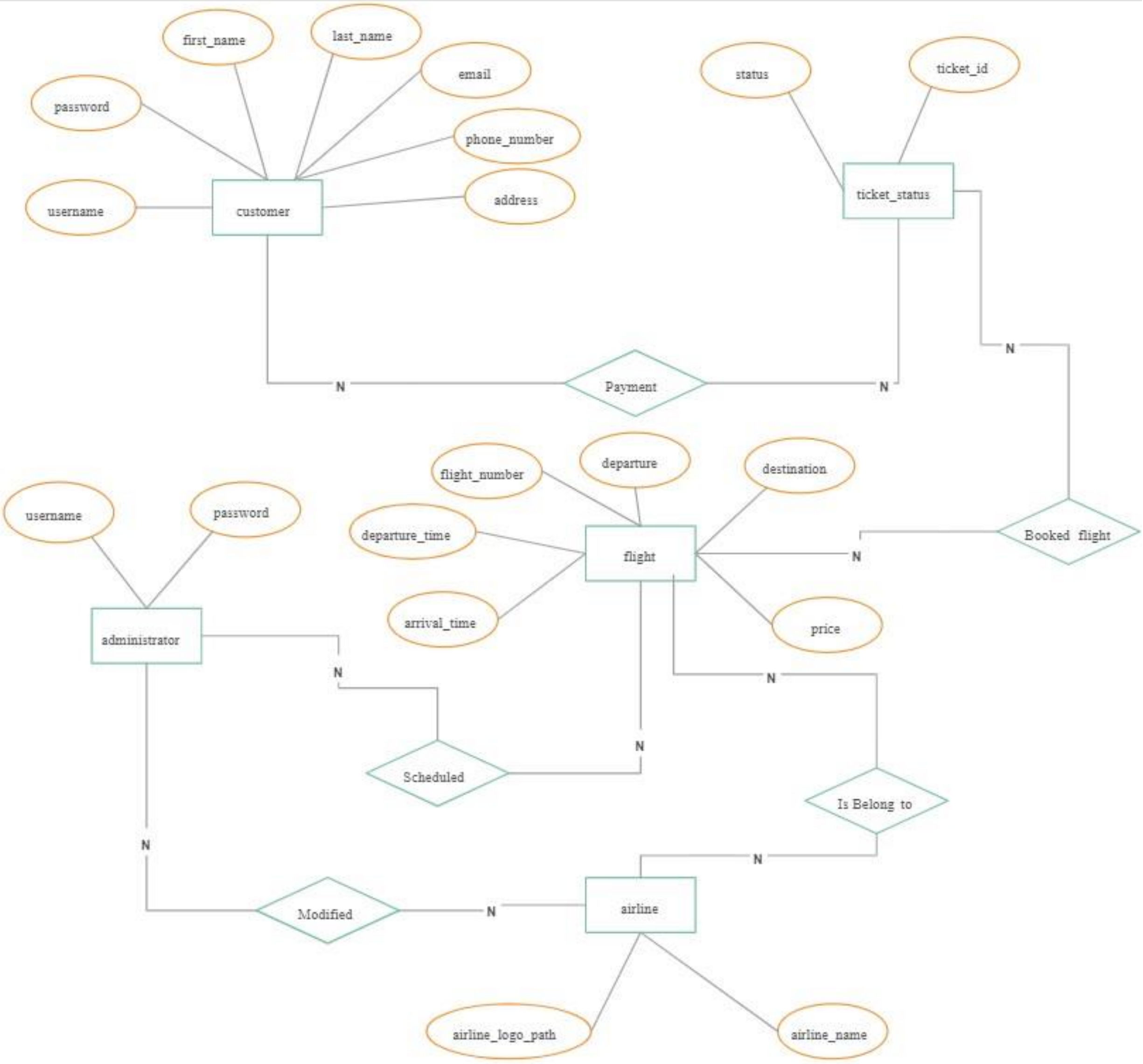


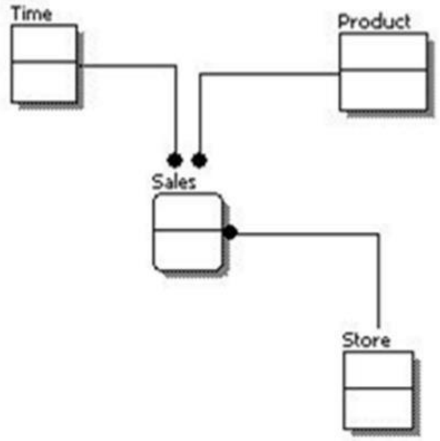
Figure: Airline Reservation System

10. Data Modeling

- Data modeling is often the first step in database design as the designers first create a conceptual model of how data items relate to each other.
- Data modeling involves a progression from conceptual model to logical model to physical schema.
- The Data Model typically evolves through the following three general stages:
  A) conceptual data model
  B) logical data model
  C) physical data model

Conceptual Data Modeling

- A conceptual data model identifies the highest-level relationships between the different entities.
- Designed and developed to be independent of DBMS, data storage locations or technologies.
- Features of conceptual data model include:
  - ✓ Important entities and the relationships among them.
  - ✓ No attribute is specified.
  - ✓ No primary key is specified The figure below is an example of a conceptual data model.



From the figure above, we can see that the only information shown via the conceptual data model is the entities that describe the data and the relationships between those entities. No other information is shown through the conceptual data model.

Logical data model

A logical data model is a fully-attributed data model that is independent of DBMS, technology, data storage or organizational constraints. It typically describes data requirements from the business point of view.
A logical data model describes the data in as much detail as possible. Features of a logical data model include:

- ✓ All entities and relationships among them.
- ✓ All attributes for each entity are specified.
- ✓ The primary key for each entity is specified.
- ✓ Foreign keys (keys identifying the relationship between different entities) are specified. ✓ Normalization occurs at this level.

The steps for designing the logical data model are as follows:

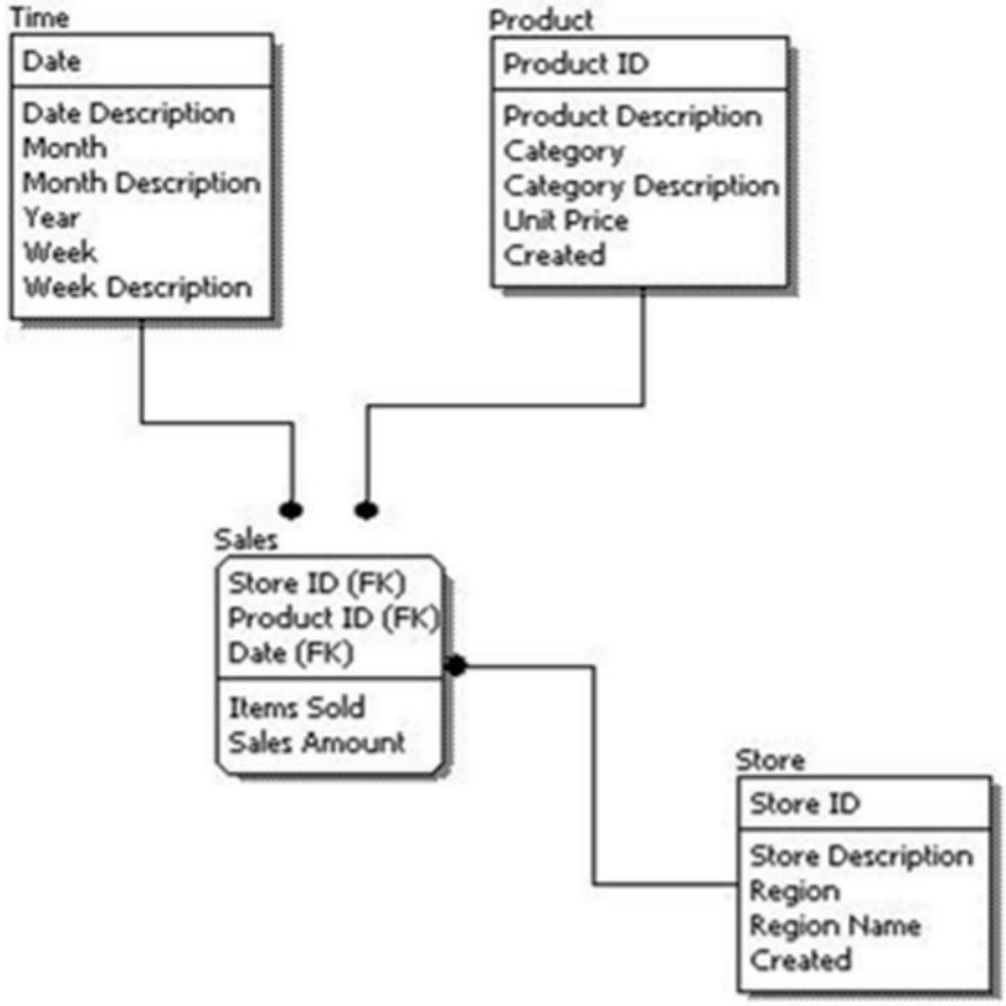
1. Specify primary keys for all entities.

2. Find the relationships between different entities.

3. Find all attributes for each entity.

4. Resolve many-to-many relationships.

5. Normalization.



Physical data model

- A physical data model is a fully-attributed data model that is dependent upon a specific version of a data persistence technology.
- The target implementation technology may be a relational DBMS, an XML document, a NoSQL data storage component, a spreadsheet or any other data implementation option.
- Features of a physical data model include:
  - Specification all tables and columns.
  - Foreign keys are used to identify relationships between tables.
  - Denormalization may occur based on user requirements.
  - Physical considerations may cause the physical data model to be quite different from the logical data model.
  - Physical data model will be different for different RDBMS. For example, data type for a column may be different between Oracle, DB2 etc.

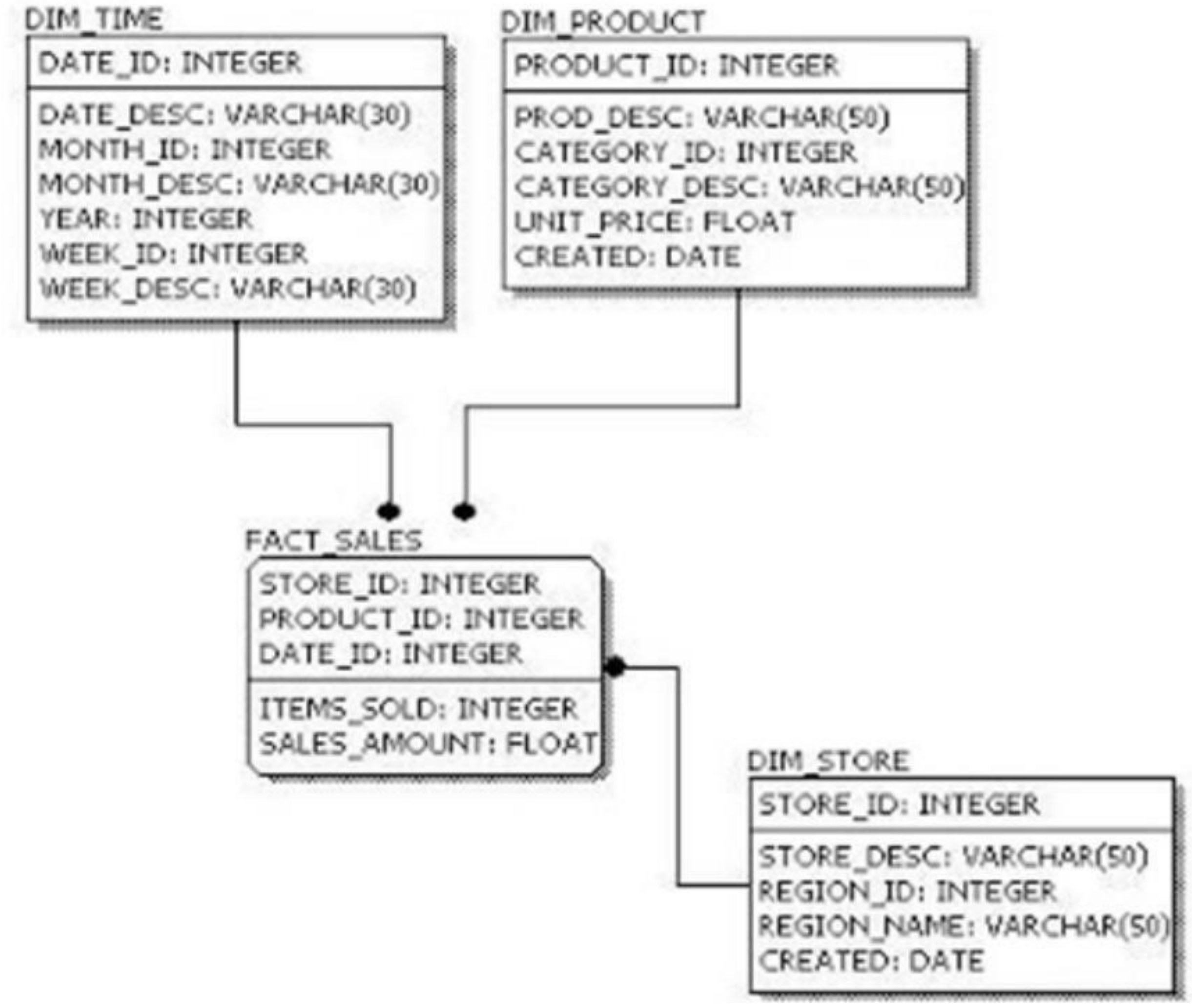
The steps for physical data model design are as follows:

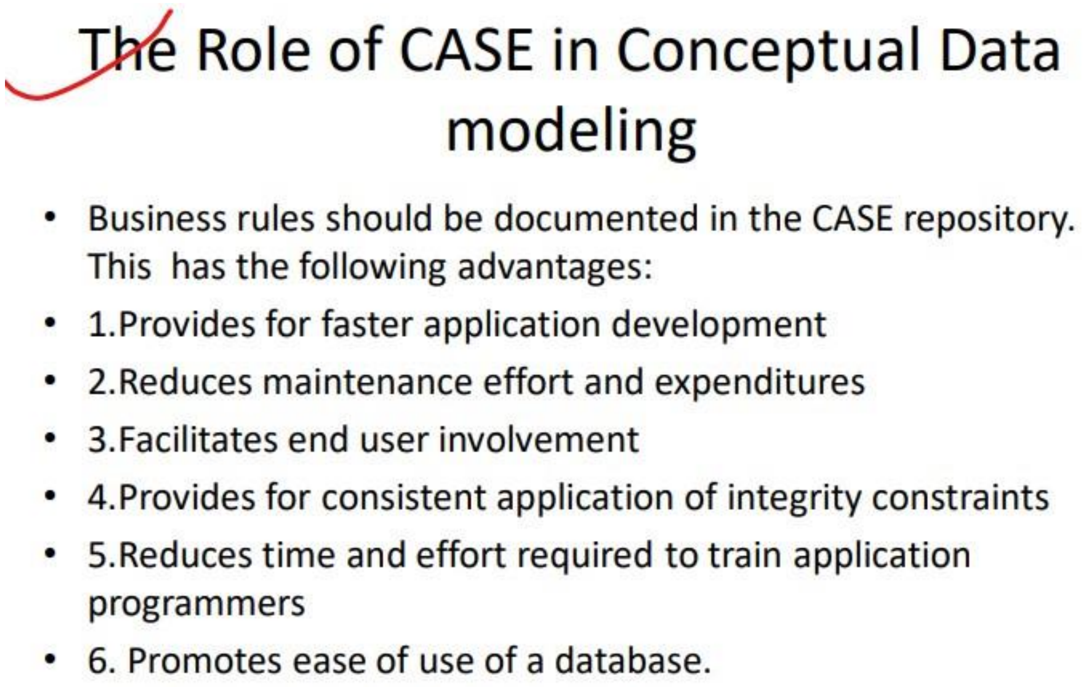
1. Convert entities into tables.

2. Convert relationships into foreign keys.

3. Convert attributes into columns.

4. Modify the physical data model based on physical constraints / requirements.





The Role of CASE in Conceptual Data modeling

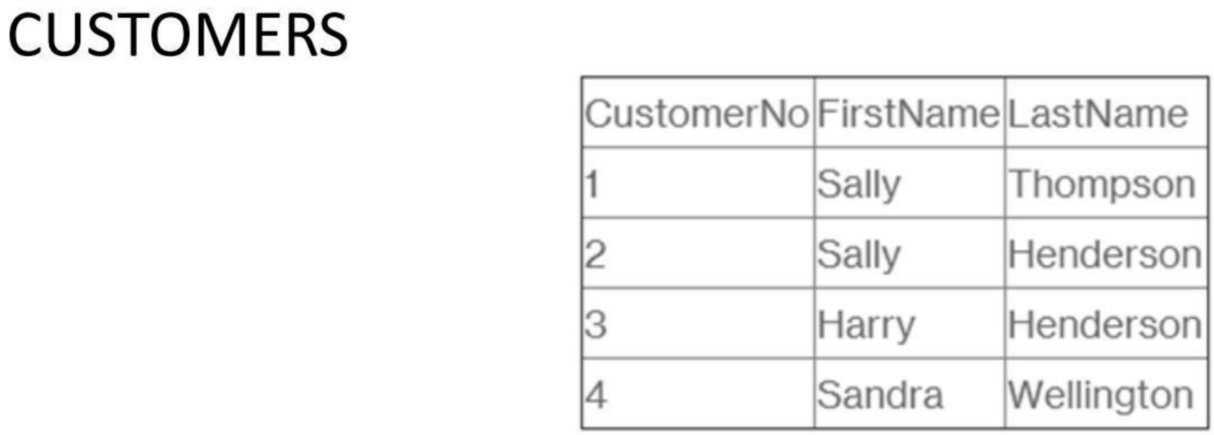
- Business rules should be documented in the CASE repository. This has the following advantages:
- 1.Provides for faster application development
- 2.Reduces maintenance effort and expenditures
- 3.Facilitates end user involvement
- 4.Provides for consistent application of integrity constraints
- 5.Reduces time and effort required to train application programmers
- 6. Promotes ease of use of a database.

Entities

- An entity is a person, place ,object, event or concept about which an organization wishes to maintain data.
- Some Examples: EMPLOYEE, STUDENT,STORE, STATE, MACHINE, SALE, ACCOUNT
- Entity type (or entity class) collection of entities with common characteristics (Eg:STUDENT)
- Entity instance: A single occurrence of an entity type (E.g Eugene Ching)
- We use capital letters in naming an entity type and in an ER diagram, the name is placed inside a rectangle representing the entity.

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.
- Example: STUDENT: Student_ID, Student_Name, Home_ Address, Phone_Number,Major **Primary key**
- A primary key is a column (or columns) in a table that uniquely identifies the rows in that table.



CUSTOMERS

CustomerNo	FirstName	LastName
1	Sally	Thompson
2	Sally	Henderson
3	Harry	Henderson
4	Sandra	Wellington

- ✓ For example, in the table above, CustomerNo is the primary key.
- ✓ The values placed in primary key columns must be unique for each row: no duplicates can be tolerated. In addition, nulls are not allowed in primary key columns.

Multivalued Attributes

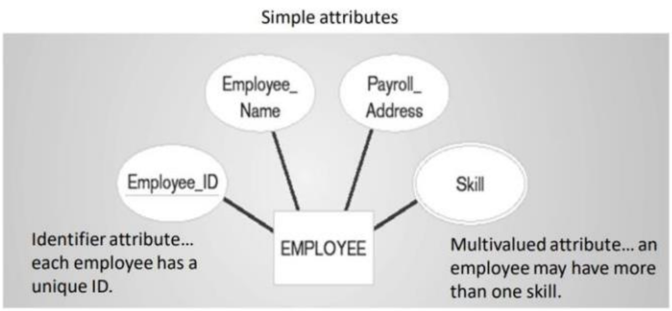
• An attribute that may take on more than one value for each entity instance

• Represented on E-R Diagram in two ways:

– double-lined ellipse

– weak entity

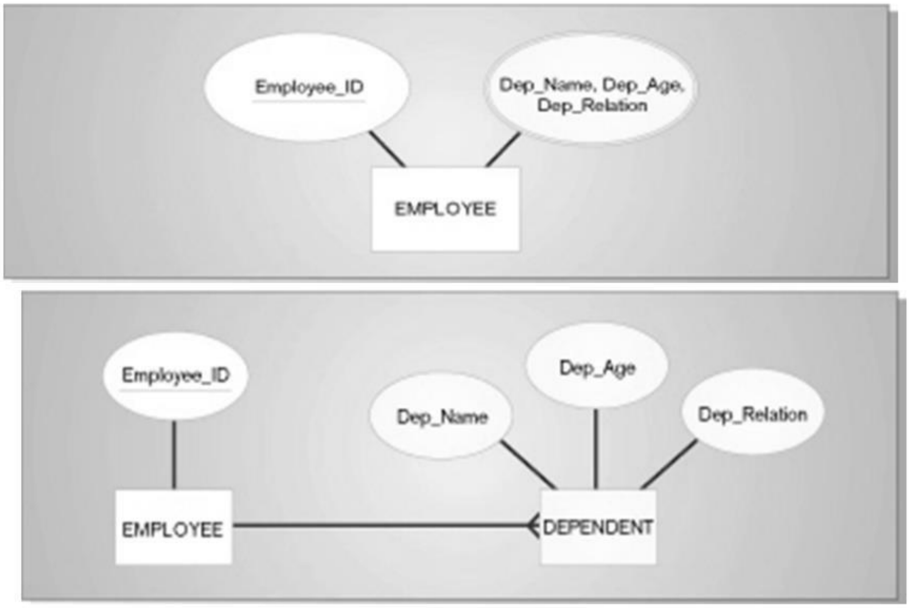
Example: Multivalued attribute
(shown as double-lined ellipse)


Simple attributes
Employee_ Name
Payroll_ Address
Employee_ID
Skill
EMPLOYEE
Identifier attribute... each employee has a unique ID.
Multivalued attribute... an employee may have more than one skill.

weak entity

• In a relational database, a weak entity is an entity that cannot be uniquely identified by its attributes alone; therefore, it must use a foreign key in conjunction with its attributes to create a primary key

Example: Multivalued attribute (shown as weak entity)



Supertypes and Subtypes

- A 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.
- Ex: STUDENT is an entity type in a university. Two subtypes of STUDENT are GRADUATE STUDENT and UNDERGRADUATE STUDENT
- A Supertype is a generic entity type that has a relationship with one or more subtypes.

Cardinality

- The number of instances of entity B that can or must be associated with each instance of entity A •

  Minimum Cardinality

  – The minimum number of instances of entity B that may be associated with each instance of

  entity A • Maximum Cardinality

  – The maximum number of instances of entity B that may be associated with each instance of entity A
- Mandatory vs. Optional Cardinalities

  – Specifies whether an instance must exist or can be absent in the relationship

Strong Entity

A strong entity is **complete by itself and is not dependent on any other entity type**. It possess a primary key which describes each instance in the strong entity set uniquely. That means any element in the strong entity set can be uniquely identified.

What is User Interface Design?

User interface (UI) design is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable. UI design refers to graphical user interfaces and other forms—e.g., voice-controlled interfaces.

Designing User Interfaces for Users

User interfaces are the access points where users interact with designs. They come in three formats:

1. Graphical user interfaces (GUIs)—Users interact with visual representations on digital control panels. A computer's desktop is a GUI.

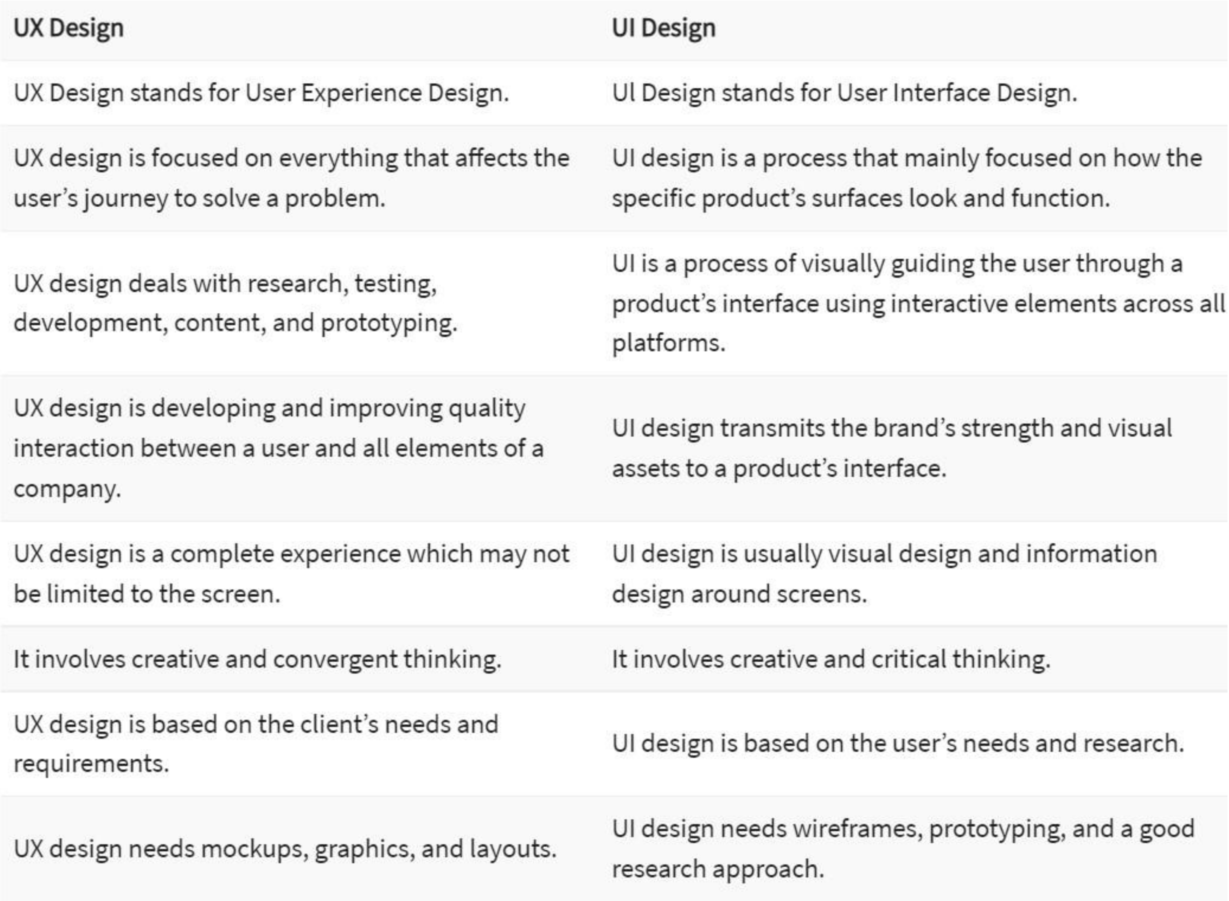
2. Voice-controlled interfaces (VUIs)—Users interact with these through their voices. Most smart assistants—e.g., Siri on iPhone and Alexa on Amazon devices—are VUIs.

3. Gesture-based interfaces—Users engage with 3D design spaces through bodily motions: e.g., in virtual reality (VR) games

UI vs. User Experience (UX) Design

Often confused with UX design, UI design is more concerned with the surface and overall feel of a design. UI design is a craft where you the designer build an essential part of the user experience. UX design covers the entire spectrum of the user experience. One analogy is to picture UX design as a car with UI design as the driving console.

"Interfaces get in the way. I don't want to focus my energies on an interface. I want to focus on the job." — Don Norman, Grand old man of UX design



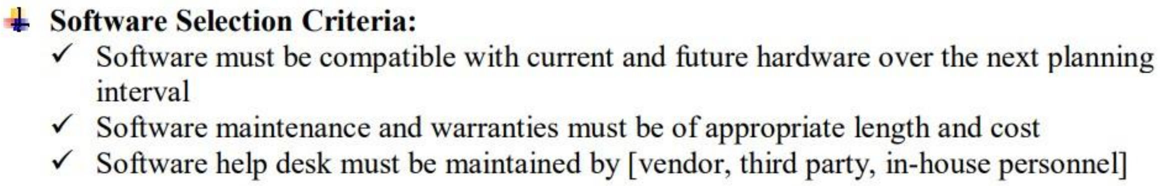
UX Design	UI Design
UX Design stands for User Experience Design.	Ul Design stands for User Interface Design.
UX design is focused on everything that affects the user's journey to solve a problem.	UI design is a process that mainly focused on how the specific product's surfaces look and function.
UX design deals with research, testing, development, content, and prototyping.	UI is a process of visually guiding the user through a product's interface using interactive elements across all platforms.
UX design is developing and improving quality interaction between a user and all elements of a company.	UI design transmits the brand's strength and visual assets to a product's interface.
UX design is a complete experience which may not be limited to the screen.	UI design is usually visual design and information design around screens.
It involves creative and convergent thinking.	It involves creative and critical thinking.
UX design is based on the client's needs and requirements.	UI design is based on the user's needs and research.
UX design needs mockups, graphics, and layouts.	UI design needs wireframes, prototyping, and a good research approach.

How to make Great UIs

To deliver impressive GUIs, remember—**users are humans, with needs such as comfort and a limit on their mental capacities.** You should follow these guidelines:

1. **Make buttons and other common elements perform predictably** (including responses such as pinchto-zoom) **so users can unconsciously use them everywhere**. Form should follow function.
2. **Maintain high discoverability.** Clearly label icons and include wellindicated affordances: e.g., shadows for buttons.
3. **Keep interfaces simple (with only elements that help serve users' purposes) and create an "invisible" feel.**
4. **Respect the user's eye and attention regarding layout.** Focus on hierarchy and readability:
    1. **Use proper alignment.** Typically choose edge (over center) alignment.
    2. **Draw attention to key features using:**
    - Color, brightness and contrast. Avoid including colors or buttons excessively.
    - Text via font sizes, bold type/weighting, italics, capitals and distance between letters. Users should pick up meanings just by scanning.

5. **Minimize the number of actions for performing tasks but focus on one chief function per page.** Guide users by indicating preferred actions. Ease complex tasks by using progressive disclosure.
6. **Put controls near objects that users want to control.** For example, a button to submit a form should be near the form.
7. **Keep users informed regarding system responses/actions with feedback.**
8. **Use appropriate UI design patterns to help guide users and reduce burdens** (e.g., pre-fill forms). Beware of using dark patterns, which include hard-to-see prefilled opt-in/opt-out checkboxes and sneaking items into users’ carts.
9. **Maintain brand consistency.**
10. **Always provide next steps which users can deduce naturally, whatever their context**



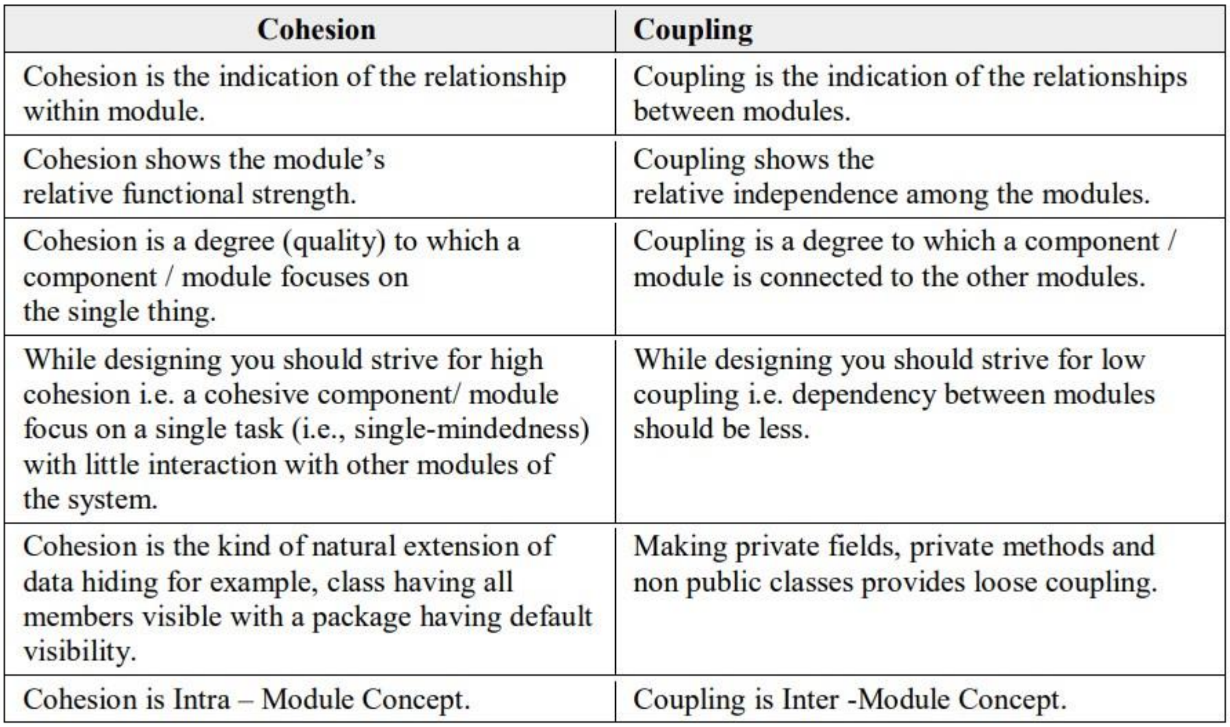
Software Selection Criteria:

- ✓ Software must be compatible with current and future hardware over the next planning interval
- ✓ Software maintenance and warranties must be of appropriate length and cost
- ✓ Software help desk must be maintained by [vendor, third party, in-house personnel]

Functional independency:

Functional independence is evaluated using two criteria:

1. Cohesion
2. Coupling



Cohesion	Coupling
Cohesion is the indication of the relationship within module.	Coupling is the indication of the relationships between modules.
Cohesion shows the module's relative functional strength.	Coupling shows the relative independence among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the single thing.	Coupling is a degree to which a component / module is connected to the other modules.
While designing you should strive for high cohesion i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system.	While designing you should strive for low coupling i.e. dependency between modules should be less.
Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility.	Making private fields, private methods and non public classes provides loose coupling.
Cohesion is Intra – Module Concept.	Coupling is Inter -Module Concept.

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.

II) 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.

III) 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: – This step involves the following activities.

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 summery 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 an 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.

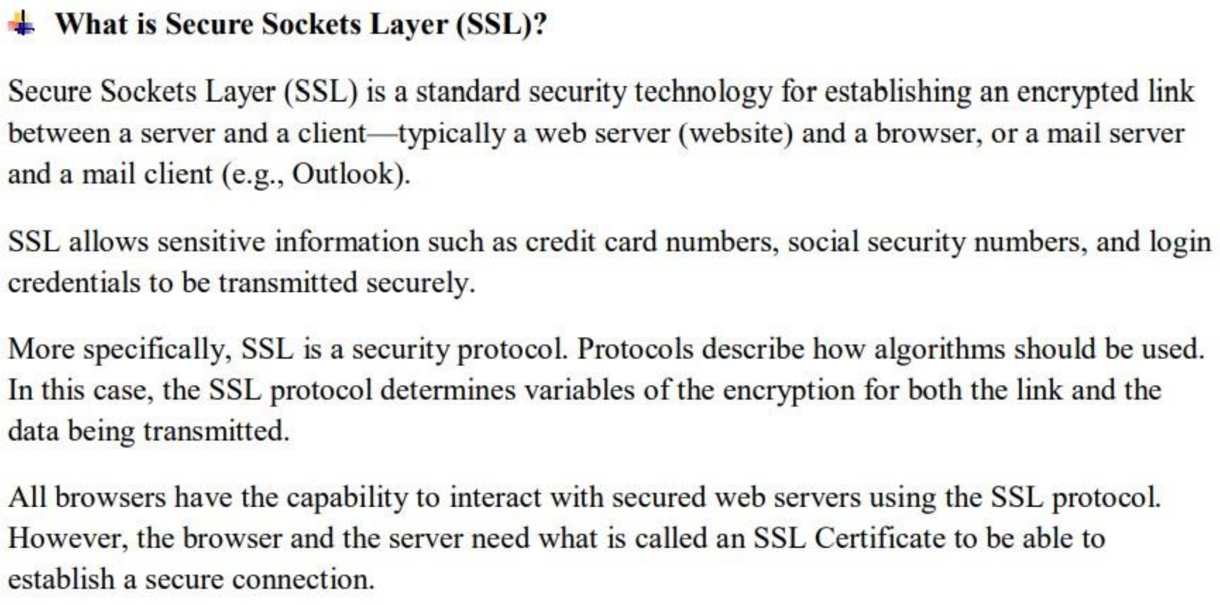
6. Negotiation and procurement decision: – Because of the extensive competition, computer system vendors may offer significant concessions. Negotiations are held to maximize these concessions. However, price negotiations are often not permitted by some organizations.

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.



What is Secure Sockets Layer (SSL)?

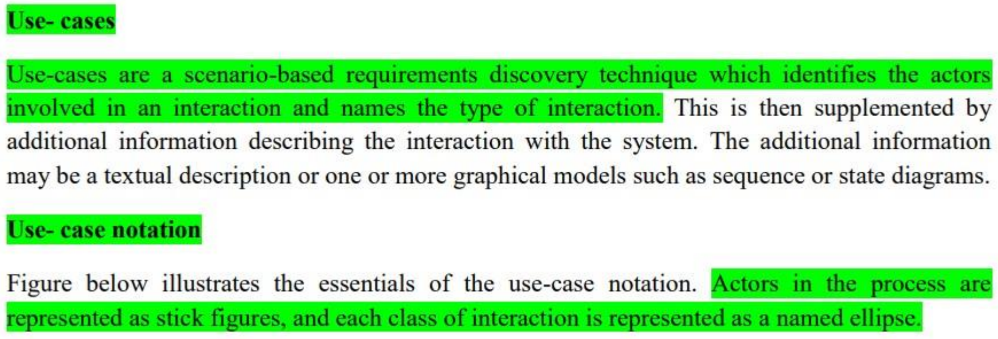
Secure Sockets Layer (SSL) is a standard security technology for establishing an encrypted link between a server and a client—typically a web server (website) and a browser, or a mail server and a mail client (e.g., Outlook).

SSL allows sensitive information such as credit card numbers, social security numbers, and login credentials to be transmitted securely.

More specifically, SSL is a security protocol. Protocols describe how algorithms should be used. In this case, the SSL protocol determines variables of the encryption for both the link and the data being transmitted.

All browsers have the capability to interact with secured web servers using the SSL protocol. However, the browser and the server need what is called an SSL Certificate to be able to establish a secure connection.

Use – Case



Use- cases

Use-cases are a scenario-based requirements discovery technique which identifies the actors involved in an interaction and names the type of interaction. This is then supplemented by additional information describing the interaction with the system. The additional information may be a textual description or one or more graphical models such as sequence or state diagrams.

Use- case notation

Figure below illustrates the essentials of the use-case notation. Actors in the process are represented as stick figures, and each class of interaction is represented as a named ellipse.

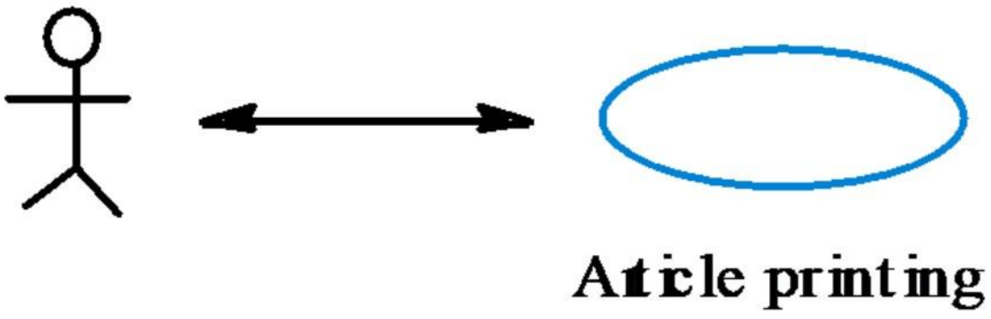


Figure: A simple use-case for article printing

The set of use-cases represents all of the possible interactions to be represented in the system requirements. Figure below develops the LIBSYS example and shows other use-cases in that environment.

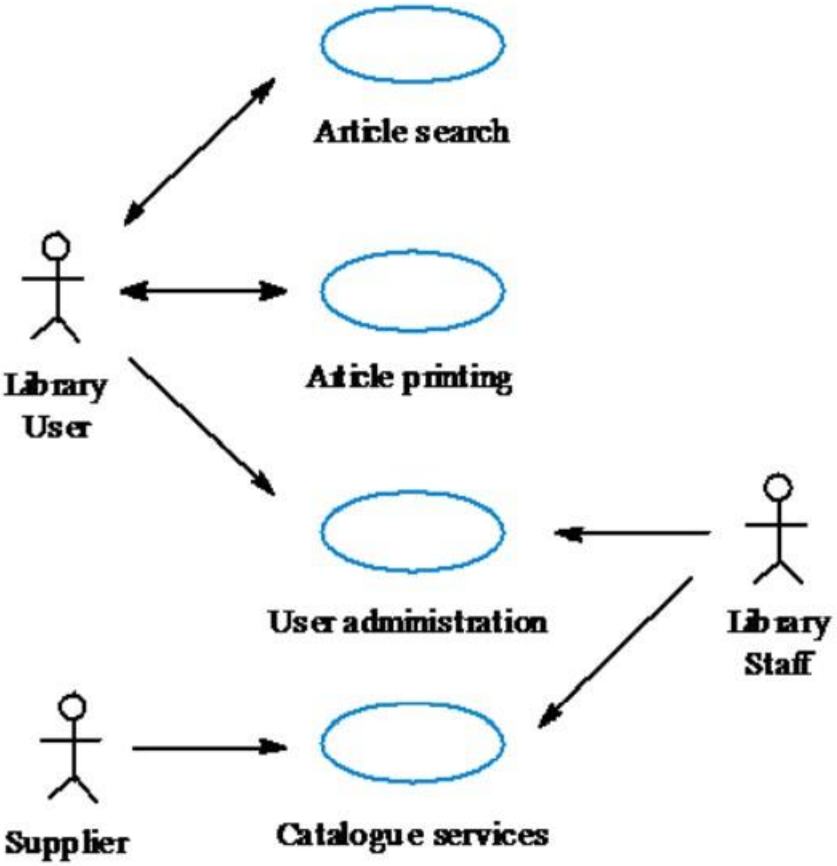


Figure: Use cases for the library system

Use-cases identify the individual interactions with the system. They can be documented with text or linked to graphical models that develop the scenario in more detail. Sequence diagrams are often used to add information to a use-case. These sequence diagrams show the actors involved in the interaction, the objects they interact with and the operations associated with these objects. As an illustration of this figure below shows the interactions involved in using LIBSYS for downloading and printing article. In this figure there are four objects of classes-Article, Form, Workspace and Printer-involved in this interaction. The sequence of actions is from top to bottom, and the labels on the arrows between the actors and objects indicate the names of operations. Essentially, a user request for an article triggers a request for a copyright form. Once the user has completed the form, the article is downloaded and sent to the printer. Once printing is completed, the article is deleted from the LIBSYS workspace.

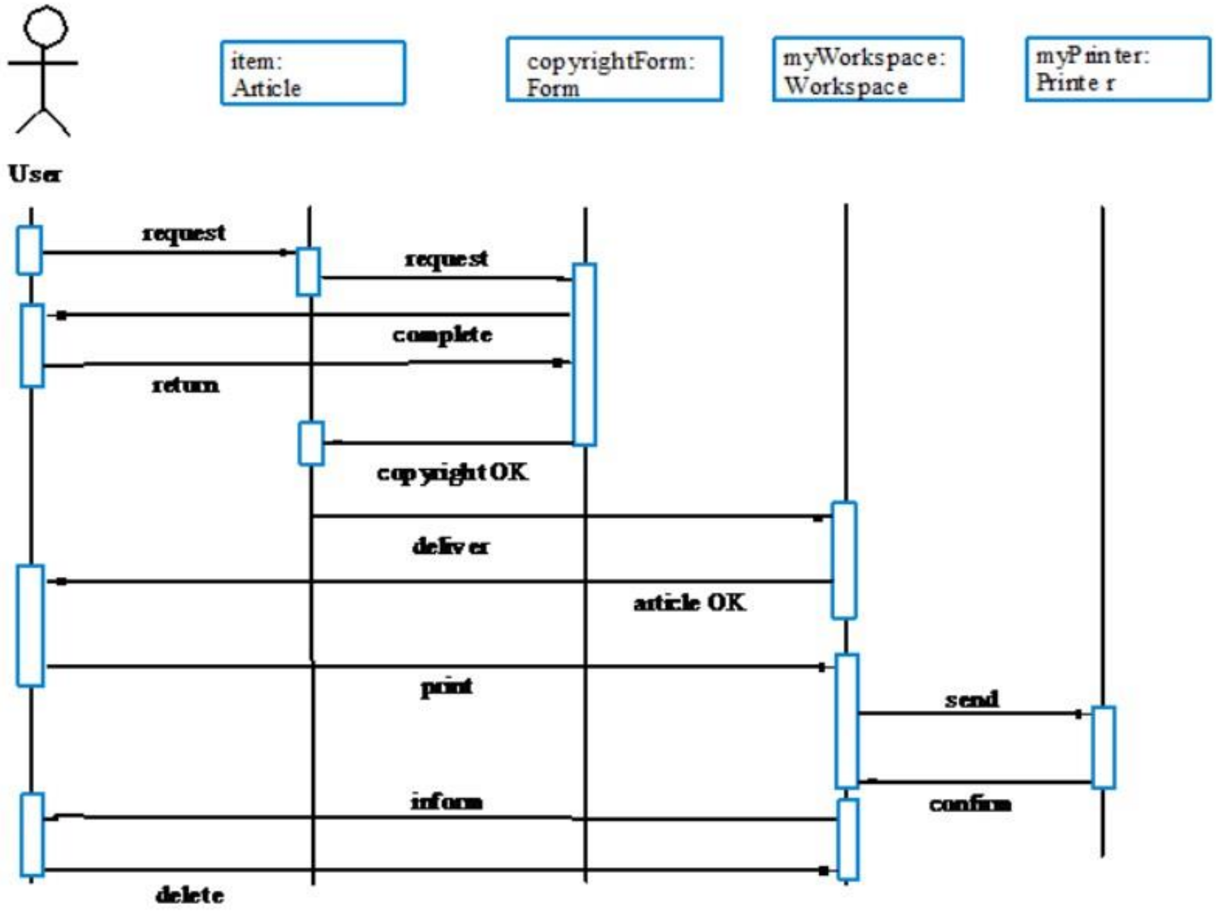
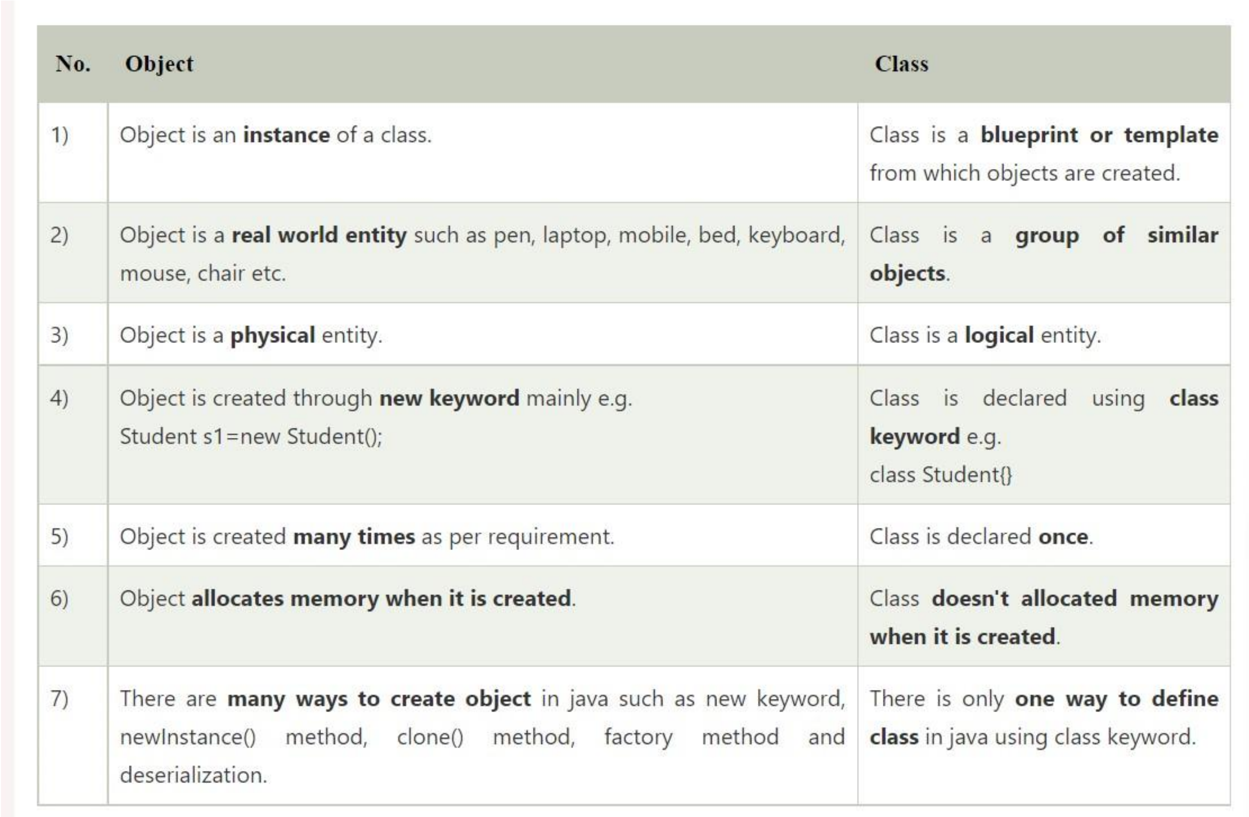


Figure: System interaction for article printing

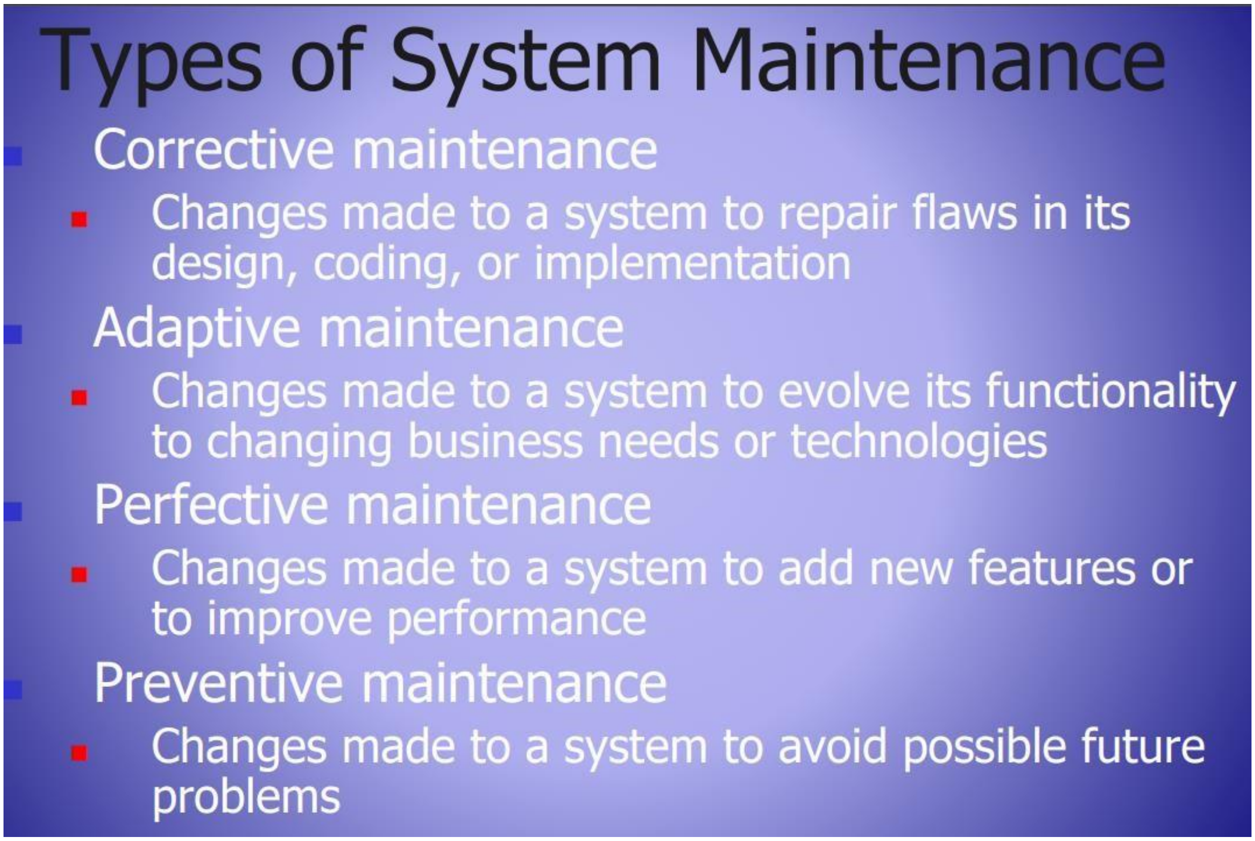
Data:

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

Difference between object and class:

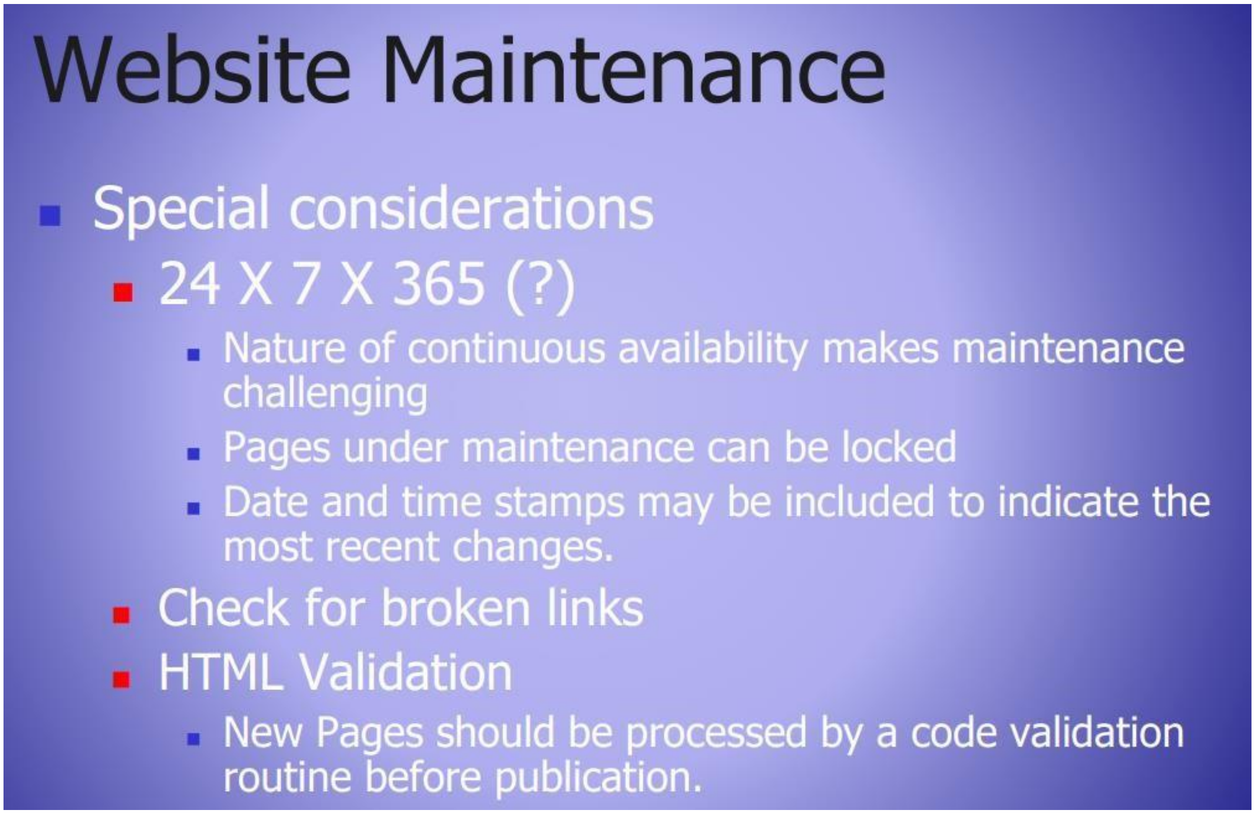


No.	Object	Class
1)	Object is an **instance** of a class.	Class is a **blueprint or template** from which objects are created.
2)	Object is a **real world entity** such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a **group of similar objects**.
3)	Object is a **physical** entity.	Class is a **logical** entity.
4)	Object is created through **new keyword** mainly e.g. Student s1=new Student();	Class is declared using **class keyword** e.g. class Student{}
5)	Object is created **many times** as per requirement.	Class is declared **once**.
6)	Object **allocates memory when it is created**.	Class **doesn't allocated memory when it is created**.
7)	There are **many ways to create object** in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only **one way to define class** in java using class keyword.



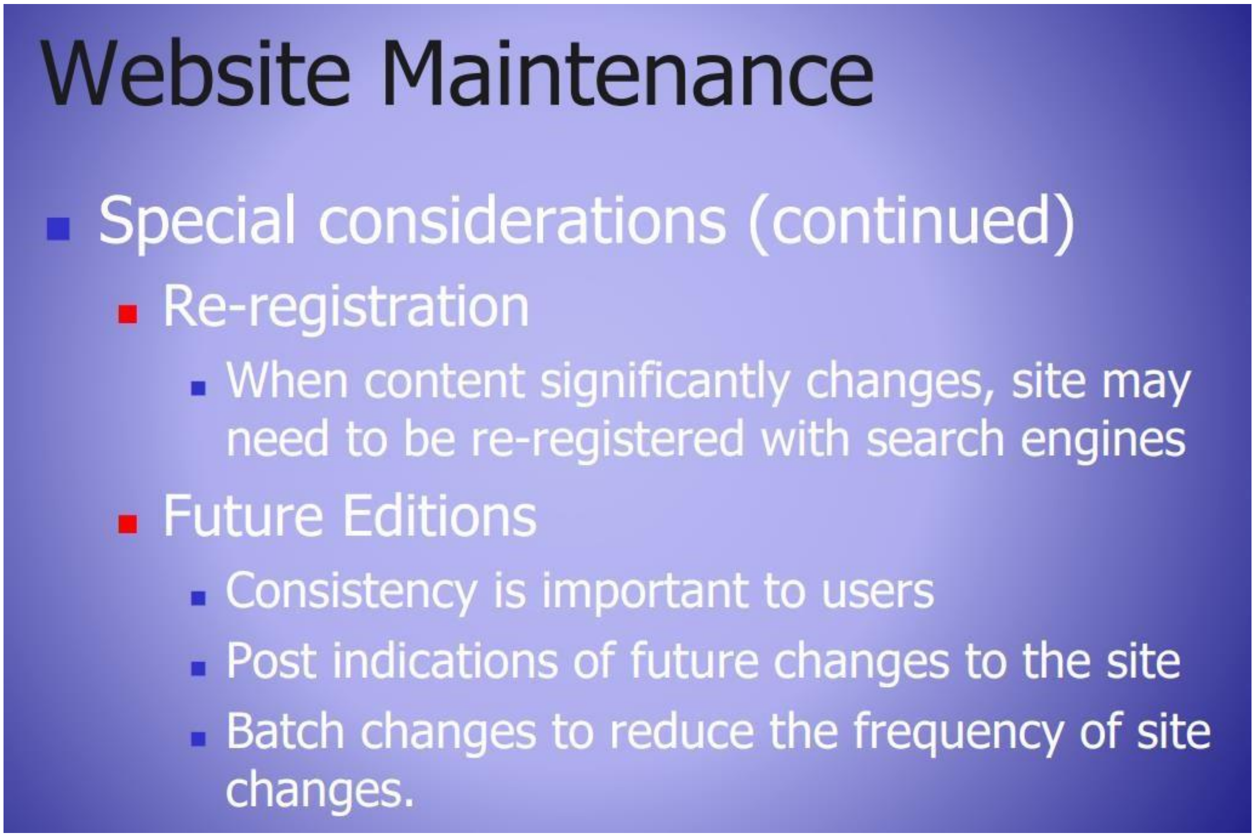
Types of System Maintenance

- Corrective maintenance
  - Changes made to a system to repair flaws in its design, coding, or implementation
- Adaptive maintenance
  - Changes made to a system to evolve its functionality to changing business needs or technologies
- Perfective maintenance
  - Changes made to a system to add new features or to improve performance
- Preventive maintenance
  - Changes made to a system to avoid possible future problems



Website Maintenance

- Special considerations
  - 24 X 7 X 365 (?)
    - Nature of continuous availability makes maintenance challenging
    - Pages under maintenance can be locked
    - Date and time stamps may be included to indicate the most recent changes.
  - Check for broken links
  - HTML Validation
    - New Pages should be processed by a code validation routine before publication.

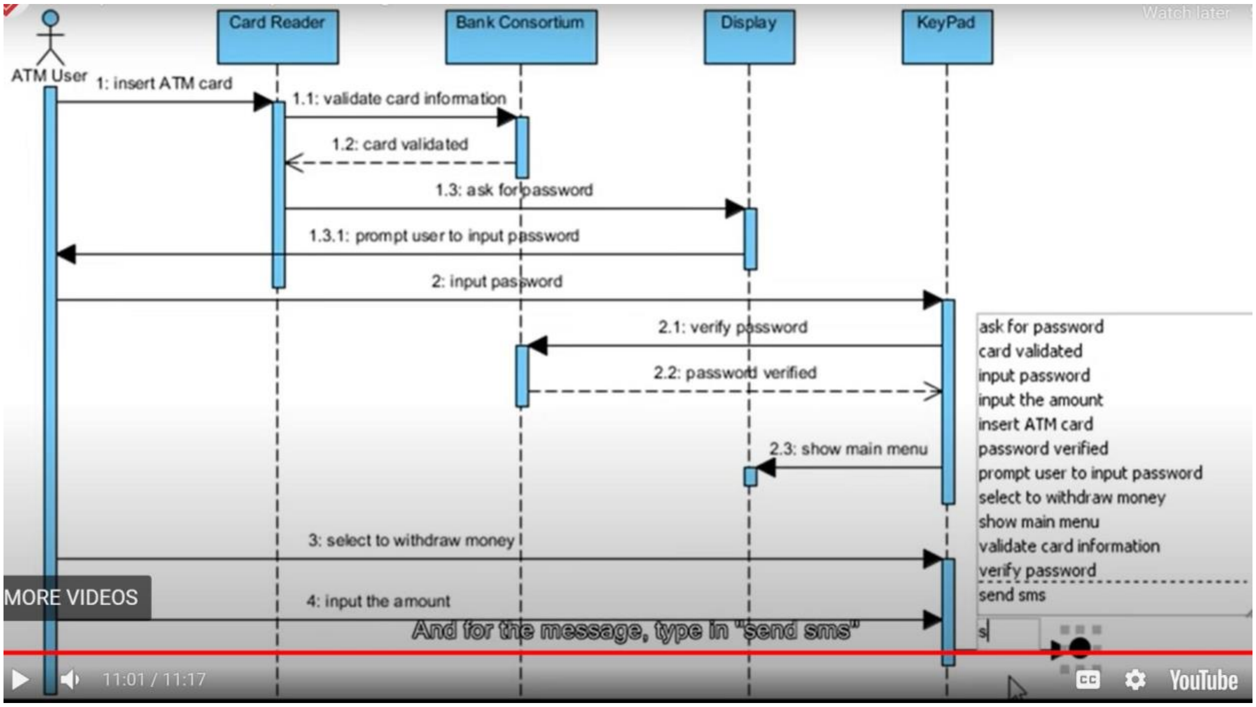


Website Maintenance

- Special considerations (continued)
  - Re-registration
    - When content significantly changes, site may need to be re-registered with search engines
  - Future Editions
    - Consistency is important to users
    - Post indications of future changes to the site
    - Batch changes to reduce the frequency of site changes.

What is sequence diagram?

A sequence diagram is **a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction**. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction.



And for the message, type in "send sms"

Concept of case tools:

Tools designed to support specific business modeling techniques are often referred to as computer aided software engineering (CASE) tools. A CASE tool is **a product that helps to analyze, model and document business processes**. It is a tool or a toolset that supports the underlying principles and methods of analysis.

Segment – 6 full revision