Department of Computer Science

St. Joseph's Degree & P.G College

(Autonomous), Affiliated to Osmania University
Re-accredited by NAAC with A Grade with CGPA 3.49
A Catholic Christian Minority Institution
King Koti Road, Hyderabad.

Program	:	B.Sc [MPCs/MSCs/MECs] III Year Semester V
Course	:	System Analysis & Design
Course code	:	BS.07.201.13.T

B.Sc. (Computer Science)

III - YEAR/ V - SEMESTER

THEORY PAPER - V

System Analysis & Design

(w.e.f 2017-18)

Scheme of Instruction	**Scheme of Examination**
Total durations Hrs : 60	Max. Marks : 100
Hours/Week : 06(4T+2P)	Internal Examination :30
Credits : 5	SBT : 10
Instruction Mode: Lecture +practical	External Examination :60
Course Code : BS.07.201.13.T	Exam Duration : 3 Hrs
Course Objectives:	
To prepare the students to develop the skills necessary to handle software projects. To make the students aware of the importance of software engineering principles in designing software projects	
Course Outcomes:	
On completion of the course the student will ➢ Understand the importance of the stages in the software life cycle. ➢ Understand the various process models. ➢ Be able to design software by applying the software engineering principles.	

Unit – I:Introduction to Systemand Approaches to System development

Introduction to System: System, Information System, Types of Information System

Approaches to System development: Software Development Life cycle, Software Development Models:Waterfall model, Iterative Model, RAD model, Incremental model, Spiral model.

Unit - II:Project management and Planning

Project management Concepts: The management Spectrum: People, The Problem, The Process

Software Project Planning: Project planning objectives, Software Scope, Resources, Software Project estimation, The Make-Buy decision, softwarerisks.

Unit - III:Analysis Concepts, Principles and Modeling

Analysis Concepts and Principles: Requirement Analysis, Communication techniques: Initiating the Process, Facilitated Application Specification techniques, Quality Function development.

Analysis Principles: The Information Domain, Modeling, Partitioning, Software Requirement Specification.

Analysis Modeling: Data Modeling: Data objects, Attributes and relationships, cardinality and modality. Data flow diagrams, Entity-Relationship Diagrams, The Data Dictionary.

Unit - IV: Design Concepts & Principles and Effective Modular design

Design Concepts & Principles: Software Design and software Engineering, the design process, the design principles, Design Concepts: Abstraction, Refinement, Modularity, Software architecture, Control hierarchy, Structural Partitioning, Data Structure, Software procedure, Information Hiding.

Effective Modular design: Functional independence, Cohesion, Coupling. User Interface Design: the Golden rules, User Interface and Design Process, Interface analysis.

References:

1) Software Engineering – A Practitioners approach, Fourth Edition, Roger S. Pressman, MGH.
2) An Integrated Approach to Software Engineering,Second Editon, Pankaj Jalote.
3) “System Analysis and Design” by Dennis, Wixon and Roth – John Wiley

System Analysis and Design

UNIT I

Topic 1:

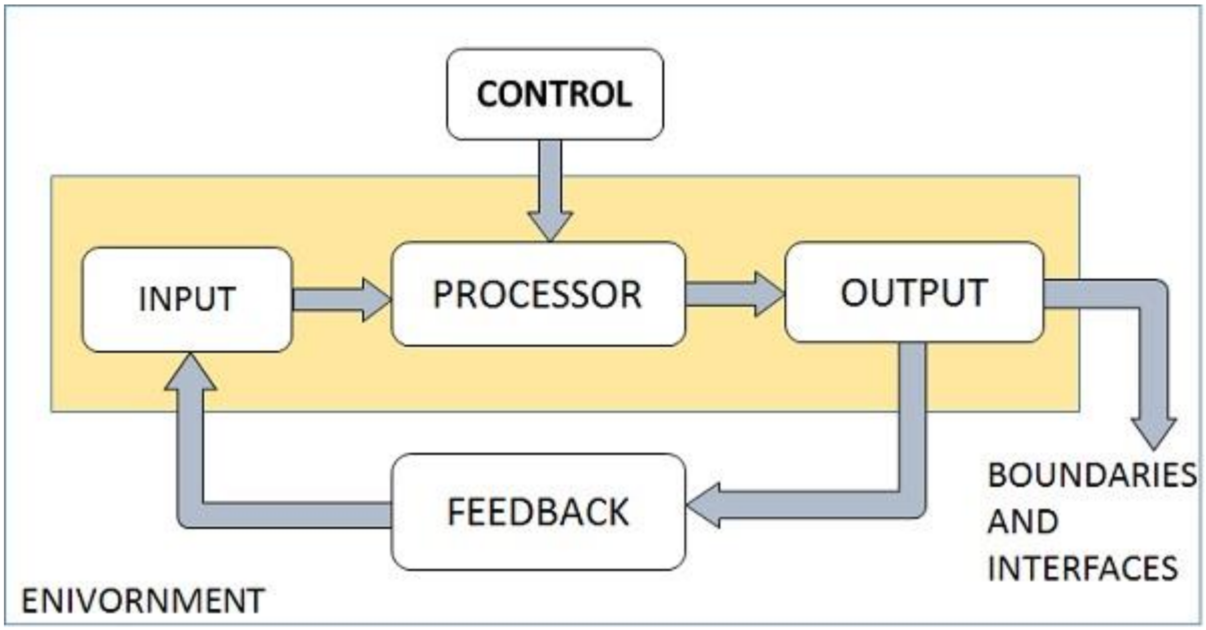
What is a System? Explain its elements

The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is "an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal."

Elements of a System:

The following diagram shows the elements of a system −



Outputs and Inputs

- The main aim of a system is to produce an output which is useful for its user.
- Inputs are the information that enters into the system for processing.
- Output is the outcome of processing.

Processor

- The processor is the element of a system that involves the actual transformation of input into output.

- It is the operational component of a system. Processors may modify the input either totally or partially, depending on the output specification.

Control

- The control element guides the system.
- It is the decision–making subsystem that controls the pattern of activities governing input, processing, and output.

Feedback

- Feedback provides the control in a dynamic system.
- Positive feedback is routine in nature that encourages the performance of the system.
- Negative feedback is informational in nature that provides the controller with information for action.

Environment

- The environment is the “supersystem” within which an organization operates.
- It is the source of external elements that strike on the system.
- It determines how a system must function. For example, vendors and competitors of organization’s environment, may provide constraints that affect the actual performance of the business.

Boundaries and Interface

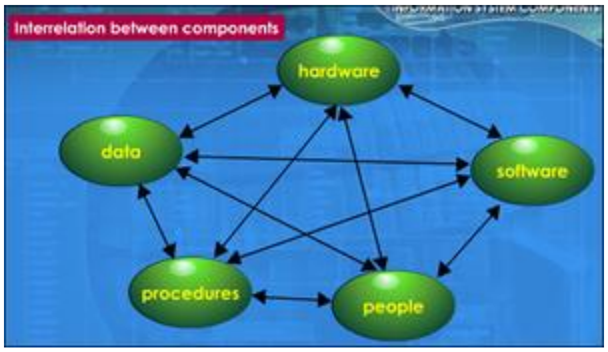
- A system should be defined by its boundaries. Boundaries are the limits that identify its components, processes, and interrelationship when it interfaces with another system.
- The knowledge of the boundaries of a given system is crucial in determining the nature of its interface with other systems for successful design.

Topic 2
Explain Information Systems

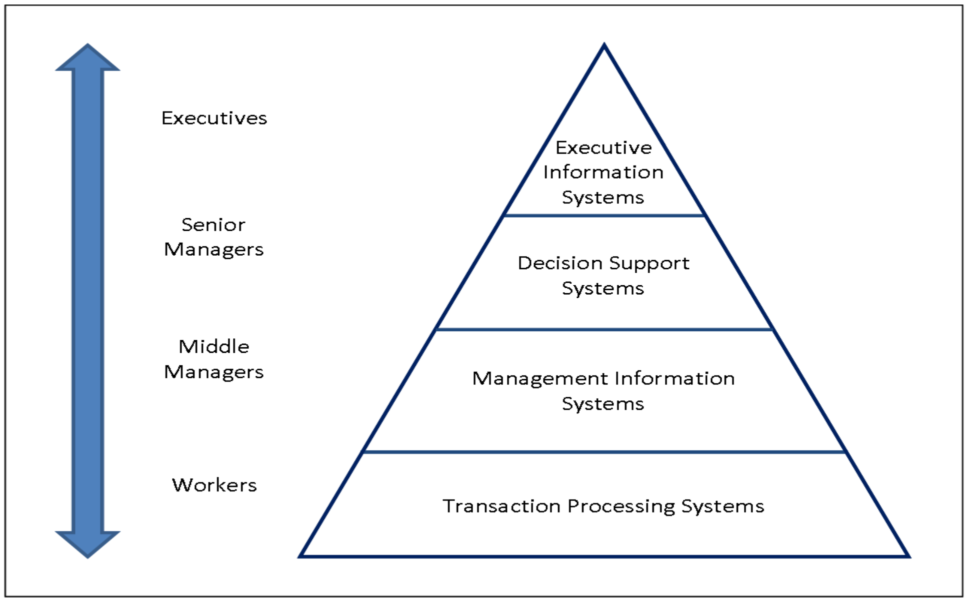
This system includes hardware, software, communication, data, and application for producing information according to the need of an organization.
Information System Components must come together in order to produce a Computer-based IS and they are:

1. Hardware includes all physical devices and materials used in information processing. It includes not only machines, such as computers and other equipment, but also all data media, that is, all tangible objects on which data is recorded, from sheets of paper to magnetic disks.
2. Software includes all sets of information processing instructions. This generic concept of software includes not only the sets of operating instructions called programs, which direct and control computer hardware, but also the sets of information processing instructions needed by people, called procedures.
3. Data is raw facts and can take many forms, including traditional alphanumeric data, composed of numbers and alphabetical and other characters that describe business transactions and other events and entities.
4. Telecommunications Networks consist of computers, communications processors, and other devices interconnected by communications media and controlled by communications software.

5.People are required for the operation of all information systems. They include end users and IS specialists.



Different types of Information Systems:


Executives
Senior Managers
Middle Managers
Workers
Executive Information Systems
Decision Support Systems
Management Information Systems
Transaction Processing Systems

Four level pyramid model based on the different levels of hierarchy in the organization

Using the four level pyramid models above, we can now compare how the information systems in our model differ from each other.

1. Transaction Processing Systems: (TPS)

- ✓ Transaction Processing System is an operational-level system at the bottom of the pyramid.
- ✓ They are usually operated directly by shop floor workers or front line staff, which provide the key data required to support the management of operations.

Some examples of TPS

- ✓ Payroll systems
- ✓ Order processing systems
- ✓ Reservation systems
- ✓ Stock control systems
- ✓ Systems for payments and funds transfers

2. Management Information Systems

- ✓ Management Information Systems are management-level systems that are used by middle managers to help ensure the smooth running of the organization in the short to medium term.

- ✓ The highly structured information provided by these systems allows managers to evaluate an organization's performance by comparing current with previous outputs.

Some examples of MIS

- ✓ Sales management systems
- ✓ Inventory control systems
- ✓ Budgeting systems
- ✓ Management Reporting Systems (MRS)
- ✓ Personnel (HRM) systems

3. Decision Support Systems

- ✓ A Decision Support System can be seen as knowledge based system, used by senior managers, which facilitates the creation of knowledge and allow its integration into the organization.
- ✓ These systems are often used to analyze existing structured information and allow managers to project the potential effects of their decisions into the future.

Some examples of DSS

- ✓ Group Decision Support Systems (GDSS)
- ✓ Computer Supported Co-operative work (CSCW)
- ✓ Logistics systems
- ✓ Financial Planning systems
- ✓ Spreadsheet Models
- ✓ Are concerned with predicting the future

4. Executive Information Systems

- ✓ Executive Information Systems are strategic-level information systems that are found at the top of the Pyramid.
- ✓ They help executives and senior managers analyze the environment in which the organization operates, to identify long-term trends, and to plan appropriate courses of action. Functions of an EIS

Some examples of EIS

Executive Information Systems tend to be highly individualized and are often custom made for a particular client group; however, a number of off-the-shelf EIS packages do exist and many enterprise level systems offer a customizable EIS module.

Expert Systems:

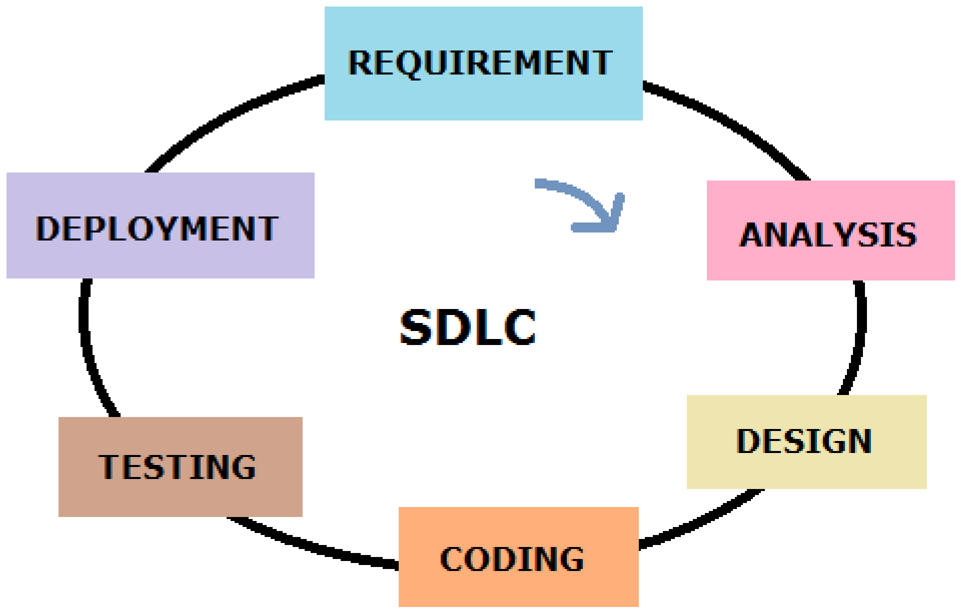
- ✓ An expert system is a computer program that uses artificial intelligence (AI) technologies to simulate the judgment and behavior of a human or an organization that has expert knowledge and experience in a particular field.The expert systems are the computer applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise.

Topic 3

Explain Software Development life cycle(SDLC)

- ✓ The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process. SDLC is a structure followed by a development team within the software organization. It consists of a detailed plan describing how to develop, maintain and replace specific software.
- ✓ Software life cycle models describe phases of the software cycle and the order in which those phases are executed.
- ✓ Each phase produces deliverables required by the next phase in the life cycle.
- ✓ There are following seven phases in every Software development life cycle model:

1. Requirement gathering
2. System Analysis
3. Design
4. Implementation or coding
5. Testing
6. Deployment
7. Maintenance



1) Requirement gathering:

- ✓ Business requirements are gathered in this phase.
- ✓ This phase is the main focus of the project managers and stake holders.
- ✓ Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system?
- ✓ These are general questions that get answered during a requirements gathering phase.

2) System Analysis:

- ✓ After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.
- ✓ Finally, a Requirement Specification (SRS) document is created which serves the purpose of guideline for the next phase of the model. The testing team follows the Software Testing Life Cycle and starts the Test Planning phase after the requirements analysis is completed.

3) Design:

- ✓ In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase.
- ✓ System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- ✓ In designing System designers will draw various diagrams like UML (Unified Modeling) Diagrams, ER Diagrams(Entity Relationship) Diagrams, Data Flow Diagrams.
- ✓ The system design specifications serve as input for the next phase of the model.
- ✓ In this phase the testers comes up with the Test strategy, where they mention what to test, how to test.

4) Implementation / Coding:

- ✓ On receiving system design documents, the work is divided in modules/units and actual coding is started.
- ✓ Since, in this phase the code is produced so it is the main focus for the developer.
- ✓ Developers will use various technologies to develop the source code.
- ✓ This is the longest phase of the software development life cycle.

5) Testing:

- ✓ After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase.

- ✓ During this phase all types of functional testing like unit testing, integration testing, system testing, acceptance testing are done as well as non-functional testing are also done.

6) Deployment:

- ✓ After successful testing the product is delivered / deployed to the customer for their use.
- ✓ As soon as the product is given to the customers they will first do the beta testing. If any changes are required or if any bugs are caught, then they will report it to the engineering team. Once those changes are made or the bugs are fixed then the final deployment will happen.

7) Maintenance:

- ✓ Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time.
- ✓ This process where the care is taken for the developed product is known as maintenance.

SOFTWARE DEVELOPMENT PROCESS MODELS:

A process model specifies a general process, usually as a set of stages. This model will be suitable for a class of projects. i.e. a model provides generic structure of the process that can be followed by some projects to achieve their goals.

Topic 4

Water fall Model:

- ✓ The **Waterfall Model** was first Process Model to be introduced. It is very simple to understand and use. In a *Waterfall* model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- ✓ *Waterfall* model is the earliest **SDLC** approach that was used for software development.
- ✓ In "**The Waterfall**" approach, the whole process of *software development* is divided into separate phases.
- ✓ The outcome of one phase acts as the input for the next phase sequentially.
- ✓ This means that any phase in the development process begins only if the previous phase is complete.

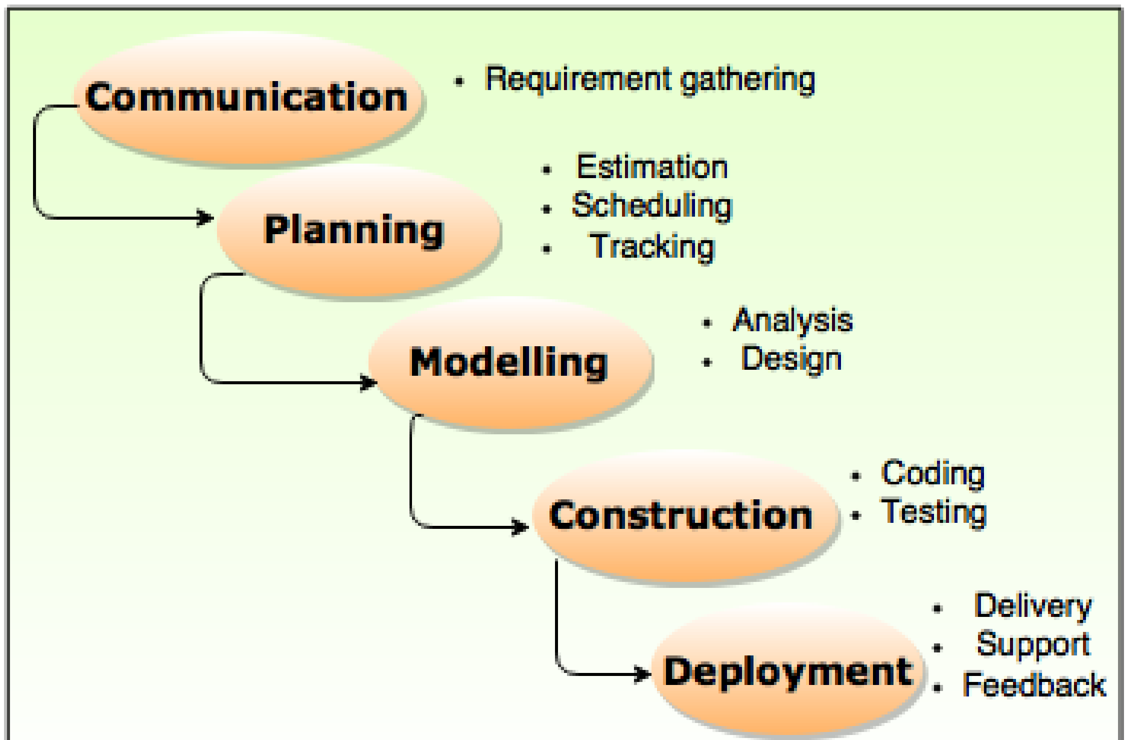


Fig. - The Waterfall model

The sequential phases in Waterfall model are −

1. Communication

In communication phase, the major task performed is requirement gathering which helps in finding out the exact need of the customer. Once all the needs of the customer are gathered the next step is planning.

2. Planning

In planning major activities like planning for schedule, keeping tracks on the processes and the estimation related to the project are done. Planning is even used to find the types of risks involved throughout the projects. Planning describes how technical tasks are going to take place and what resources are needed and how to use them.

3. Modeling

This is one the important phases of the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed. Different models for developing software are created depending on the requirements gathered in the first phase and the planning done in the second phase.

4. Construction

The actual coding of the software is done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase software is actually developed and tested.

5. Deployment

In this last phase the product is actually rolled out or delivered & installed at customer's end and support is given if required. A feedback is taken from the customer to ensure the quality of the product.

Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

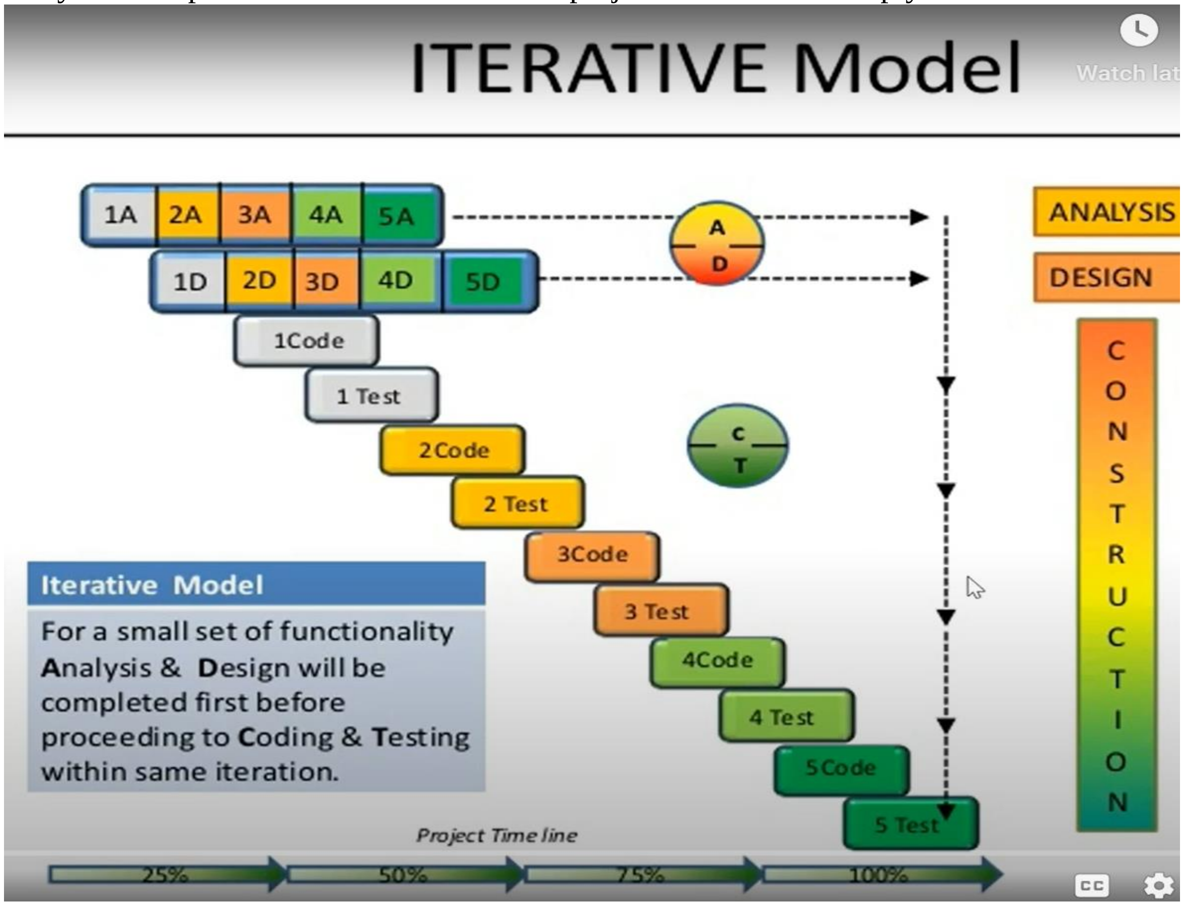
Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

UNIT I

Topic 5: Explain Iterative Model:

- ✓ The iterative model combines the benefits of both waterfall model and prototype model. The basic idea of this model is to develop increments and each increment adding some functional capability to the system until the full version is of the system is implemented.
- ✓ It can be viewed as a sequence of waterfalls. The feedback from one iteration is used in the future iterations.
- ✓ In the first step of this model, a simple initial implementation is done for a subset of the overall problem.
- ✓ This subset is one that contains some key aspects of the problem that are easy to understand and implement.
- ✓ A "project control list " is created that contains all the tasks that must be performed to obtain the final implementation.
- ✓ Each step in the iteration consisting of removing the next task from the list, designing the implementation for the selected task, coding and testing, performing an analysis of the partial system obtained after this step, and updating the list as a result of the analysis. This process is continued until the project control list is empty.



Iterative Model

For a small set of functionality Analysis & Design will be completed first before proceeding to Coding & Testing within same iteration.

When to use iterative model:

- Requirements of the complete system are clearly defined and understood.
- When the project is big.
- Major requirements must be defined; however, some details can evolve with time.

Advantages of Iterative model:

- ✓ In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- ✓ In iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.
- ✓ In iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- ✓ In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative model:

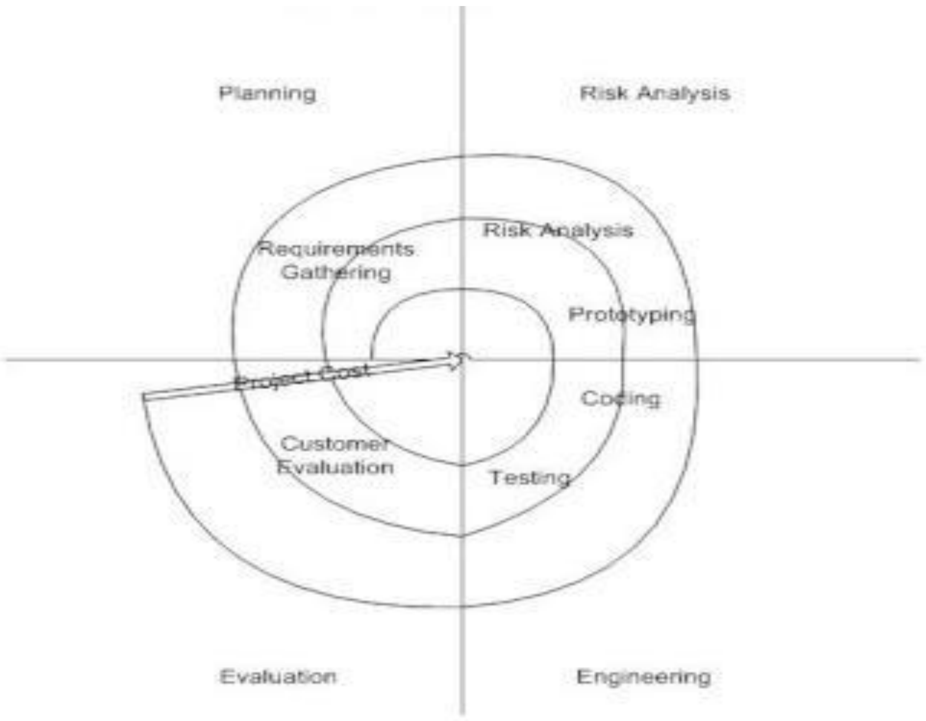
- ✓ Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle
- ✓ More resources may be required.
- ✓ Although cost of change is lesser, but it is not very suitable for changing requirements.
- ✓ More management attention is required.
- ✓ System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- ✓ Management complexity is more.
- ✓ End of project may not be known which is a risk.
- ✓ Highly skilled resources are required for risk analysis.
- ✓ Projects progress is highly dependent upon the risk analysis phase.

UNIT I

Topic 6: Explain Spiral Model in detail.

What is Spiral Model?

- ✓ The spiral model was first mentioned by Barry Boehm in his 1986 paper.
- ✓ The spiral model is similar to the incremental model, with more emphasis placed on risk analysis.
- ✓ The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.
- ✓ A software project repeatedly passes through these phases in iterations (called Spirals in this model).
- ✓ The baseline spiral, starting in the planning phase, requirements is gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.
- ✓ Each phase in spiral model begins with a design goal and ends with the client reviewing the progress.



Spiral Model Phases Explanation:

Spiral Model Phases	Activities performed during phase
Planning	• It includes estimating the cost, schedule and resources for the iteration. • It also involves understanding the system requirements for continuous communication between the system analyst and the customer • Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
Risk Analysis	• Identification of potential risk is done while risk mitigation strategy is planned and finalized • During the second quadrant all the possible solutions are evaluated to select the best possible solution. • Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. • At the end of this quadrant, Prototype is built for the best possible solution.
Engineering	• It includes testing, coding and deploying software at the customer site • During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
Evaluation	• Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun. • In the fourth quadrant, the Customers evaluate the so far developed version of the software. • In the end, planning for the next phase is started.

When to use Spiral Methodology?

- When project is large
- When releases are required to be frequent.
- When risk and costs evaluation is important
- For medium to high-risk projects
- When requirements are unclear and complex
- When changes may require at any time

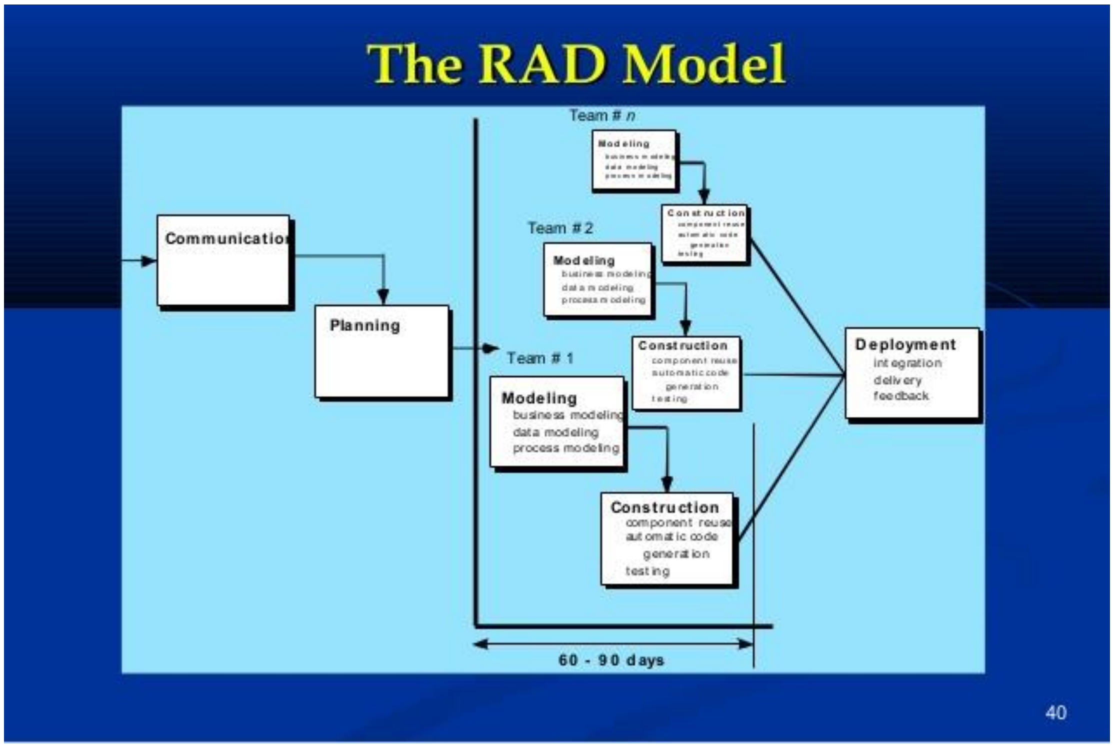
Advantages and Disadvantages of Spiral Model

Advantages	Disadvantages
• Additional functionality or changes can be done at a later stage	• Risk of not meeting the schedule or budget
• Cost estimation becomes easy as the prototype building is done in small fragments	• It works best for large projects only also demands risk assessment expertise
• Continuous or repeated development helps in risk management	• For its smooth operation spiral model protocol needs to be followed strictly
• Development is fast and features are added in a systematic way	• Documentation is more as it has intermediate phases
• There is always a space for customer feedback	• It is not advisable for smaller project, it might cost them a lot

UNIT I

Topic 7: Explain RAD Model in detail.

- ✓ RAD model is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects.
- ✓ The developments are time boxed, delivered and then assembled into a working prototype.
- ✓ This can quickly give the customer something to see and use and to provide feedback regarding the delivery and their requirements.



1. Communication

In communication phase, the major task performed is requirement gathering which helps in finding out the exact need of the customer. Once all the needs of the customer are gathered the next step is planning.

2. Planning

In planning major activities like planning for schedule, keeping tracks on the processes and the estimation related to the project are done. Planning is even used to find the types of risks involved throughout the projects. Planning describes how technical tasks are going to take place and what resources are needed and how to use them.

3. Modeling

This is one the important phases of the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed.

Different types of modeling:

Type	Description
Business Modeling	• On basis of the flow of information and distribution between various business channels, the product is designed
Data Modeling	• The information collected from business modeling is refined into a set of data objects that are significant for the business
Process Modeling	• The data object that is declared in the data modeling phase is transformed to achieve the information flow necessary to implement a business function
Application Generation	• Automated tools are used for the construction of the software, to convert process and data models into prototypes
Testing and Turnover	• As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

4. Construction
The actual coding of the software is done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase software is actually developed and tested. Here we will use automatic code generator tools and from existing projects we will reuse the components.

5. Deployment
In this last phase the product is actually rolled out or delivered & installed at customer's end and support is given if required. A feedback is taken from the customer to ensure the quality of the product.

When to use RAD Methodology?

- When a system needs to be produced in a short span of time (2-3 months)
- When the requirements are known
- When the user will be involved all through the life cycle
- When technical risk is less
- When there is a necessity to create a system that can be modularized in 2-3 months of time

Advantages and Disadvantages of RAD Model

Advantages	Disadvantages
• Flexible and adaptable to changes	• It can't be used for smaller projects

• It is useful when you have to reduce the overall project risk	• Not all application is compatible with RAD
• It is adaptable and flexible to changes	• When technical risk is high, it is not suitable
• It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used	• If developers are not committed to delivering software on time, RAD projects can fail
• Due to code generators and code reuse, there is a reduction of manual coding	• Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period
• Due to prototyping in nature, there is a possibility of lesser defects	• Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application

UNIT I
Topic 8: Incremental Process model:

- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for any suggested modifications.
- The next increment implements on the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is finished.
  For example, the word-processing software is developed using the incremental model.

1. Communication

In communication phase, the major task performed is requirement gathering which helps in finding out the exact need of the customer. Once all the needs of the customer are gathered the next step is planning.

2. Planning

In planning major activities like planning for schedule, keeping tracks on the processes and the estimation related to the project are done. Planning is even used to find the types of risks involved throughout the projects. Planning describes how technical tasks are going to take place and what resources are needed and how to use them.

3. Modeling

This is one the important phases of the architecture of the system is designed in this phase. Analysis is carried out and depending on the analysis a software model is designed. Different models for developing software are created depending on the requirements gathered in the first phase and the planning done in the second phase.

4. Construction

The actual coding of the software is done in this phase. This coding is done on the basis of the model designed in the modeling phase. So in this phase software is actually developed and tested.

5. Deployment

In this last phase the product is actually rolled out or delivered & installed at customer's end and support is given if required. A feedback is taken from the customer to ensure the quality of the product.

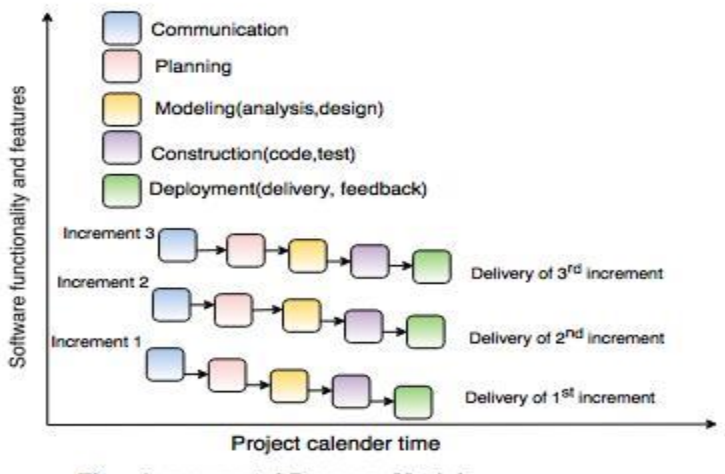

Communication
Planning
Modeling(analysis,design)
Construction(code,test)
Deployment(delivery, feedback)
Increment 3
Increment 2
Increment 1
Delivery of 3rd increment
Delivery of 2nd increment
Delivery of 1st increment
Software functionality and features
Project calender time

Fig. - Incremental Process Model

Advantages of incremental model

- This model is flexible because the cost of development is low and initial product delivery is faster.
- It is easier to test and debug during the smaller iteration.
- The working software generates quickly and early during the software life cycle.
- The customers can respond to its functionalities after every increment.

Disadvantages of the incremental model

- The cost of the final product may cross the cost estimated initially.
- This model requires a very clear and complete planning.
- The planning of design is required before the whole system is broken into small increments.
- The demands of customer for the additional functionalities after every increment causes problem during the system architecture.

St. Joseph's Degree & PG College
Department of Computer Science

Subject: System Analysis & Design
Paper – V

Unit – II
Project Management Concepts: The Management Spectrum: People, Process and Product
Software Project Planning: Project planning objectives, Software Scope, Resources, Software Project Estimation, The Make-Buy Decision, Software Risks

Topic 1: Software Project Management: (The Management Spectrum)

- Software Project management involves planning, monitoring and control of people, process and event that occurs as software evolves from preliminary concept to fully operational deployment.
- The Software Project management Spectrum t focuses on the four P's:
  1. People
  2. Product
  3. Process
  4. Project

The People

- The most important contributor to a successful software project is People.
- The manager who forgets that software engineering work is an intensely human endeavor will never have success in project management.
- Every organization needs to continually improve its ability to attract, develop, motivate, organize, and retain the workforce needed to accomplish its strategic business objectives.
- The software process (and every software project) is populated by stakeholders who Can be categorized into one of five constituencies:

1. **Senior managers** who define the business issues that often have a significant influence on the project.
2. **Project (technical) managers** who must plan, motivate, organize, and control the practitioners who do software work.
3. **Practitioners** who deliver the technical skills that are necessary to engineer a product or application.
4. **Customers** who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. **End users** who interact with the software once it is released for production use.

The Product

- Before a project can be planned, product objectives and scope should be established, alternative solutions should be considered, and technical and management constraints should be identified.
- Without this information, it is impossible to define reasonable (and accurate) estimates of the cost, an effective assessment of risk, a realistic breakdown of project tasks, or a manageable project schedule that provides a meaningful indication of progress.
- Software developer, Project Manager and other stakeholders must meet to define product objectives and scope.

- Objectives identify the overall goals for the product (from the stakeholders' points of view) without considering how these goals will be achieved.

The Process:

A software process provides the framework from which a comprehensive plan for software development can be established. A number of different tasks sets— tasks, milestones, work products, and quality assurance points—enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.

The Project:

The project includes all and everything of the total development process and to avoid project failure the manager has to take some steps, has to be concerned about some common warnings etc.

- Some projects are simple and routine, but most projects are complex and unique.
- We conduct planned and controlled software projects to manage complexity.
- The manager who embarks without a solid project plan jeopardizes the success of the project.

Reasons for the failure of software project

1. Software people don't understand their customer's needs.
2. The product scope is poorly defined.
3. Changes are managed poorly.
4. The chosen technology changes.
5. Business needs change.
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost.
9. The project team lacks people with appropriate skills.
10. Managers [and practitioners] avoid best practices and lessons learned.

Topic 2: Explain Software Project Planning and objectives

- Project planning is the process of identifying and planning the activities required to build a software product
- The objective of software project planning is to provide a framework that enables the manager to make reasonable estimates of resources, cost, and schedule.
- Planning will improve a project's outcome.
- Planning requires you to make an initial commitment
- Effective planning is needed to resolve problems upstream [early in the project] at low cost, rather than downstream [late in the project] at high cost

Task Set for Project Planning

1. Establish Project Scope.
2. Determine Feasibility.
3. Analyze risks
4. Define required resources.
    a. Determine required human resources.

    b. Define reusable software resources.
    c. Identify environmental resources.

5. Estimate cost and effort.
    a. Decompose the problem.
    b. Develop two or more estimates using size, function points, process tasks, or use cases.
    c. Reconcile the estimates.
6. Develop a project schedule
    a. Establish a meaningful task set.
    b. Define a task network.
    c. Use scheduling tools to develop a time-line chart.
    d. Define schedule tracking mechanisms.

Topic 3: Explain SOFTWARE SCOPE

- Scope identifies the primary data, functions, and behaviors that characterize the product, and more important, attempts to bound these characteristics in a quantitative manner.
- Software scope describes
    - The functions and features that are to be delivered to end users
    - The data that are input and output
    - The "content" that is presented to users as a consequence of using the software;
    - The performance, constraints, interfaces, and reliability that bound the system.
- Scope is defined using one of two techniques:
    1. A narrative description of software scope is developed after communication with all stakeholders.
    2. A set of use cases is developed by end users.
- Functions described in the statement of scope (or within the use cases) are evaluated and in some cases refined to provide more detail prior to the beginning of estimation.
- Performance considerations encompass processing and response time requirements.
- Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.
- Scope is defined by answering the following questions:
  Context. How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?
  Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input?
  Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?
- Software project scope must be unambiguous and understandable at the management and technical levels.
- A statement of software scope must be bounded.

Topic 4: Explain SOFTWARE FEASIBILITY and its types

Once scope has been identified project feasible analysis need to be made.
Feasibility study is the initial design stage of any project, which brings together the elements of knowledge that indicate if a project is possible or not. Feasibility study means an analysis of a project to determine if it is practical or not.

- **Technical Feasibility** – Does the company have the technological resources to undertake the project? Are the processes and procedures conducive to project success?

- **Schedule Feasibility** – Does the company currently have the time resources to undertake the project? Can the project be completed in the available time?
- **Economic Feasibility** – Given the financial resources of the company, is the project something that can be completed? The economic feasibility study is more commonly called the cost/benefit analysis.
- **Cultural Feasibility** – What will be the impact on both local and general cultures? What sort of environmental implications does the feasibility study have?
- **Legal/Ethical Feasibility** – What are the legal implications of the project? What sort of ethical considerations are there? You need to make sure that any project undertaken will meet all legal and ethical requirements before the project is on the table.
- **Resource Feasibility** – Do you have enough resources, what resources will be required, what facilities will be required for the project, etc.
- **Operational Feasibility** – This measures how well your company will be able to solve problems and take advantage of opportunities that are presented during the course of the project
- **Marketing Feasibility** – Will anyone want the product once its done? What is the target demographic? Should there be a test run? Is there enough buzz that can be created for the product?

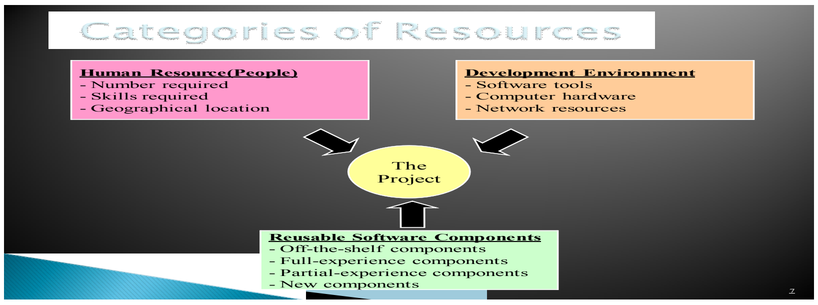
Topic 5: Explain various software resources.

- The important planning task is estimation of the resources required to accomplish the software development effort.
- Each resource is specified with
    - A description of the resource
    - A statement of availability
    - The time when the resource will be required
    - The duration of time that the resource will be applied
- Three major categories of software engineering resources

  a. Determine required human resources.

  b. Define reusable software resources.

  c. Identify environmental resources.



Categories of Resources

Human Resource(People)
- Number required
- Skills required
- Geographical location

Development Environment
- Software tools
- Computer hardware
- Network resources

The Project

Reusable Software Components
- Off-the-shelf components
- Full-experience components
- Partial-experience components
- New components

2

Human Resources

- The Planners need to select the number and the kind of people skills needed to complete the project
- They need to specify the organizational position and job specialty for each person
- Small projects of a few person-months may only need one individual

- Large projects spanning many person-months or years require the location of the person to be specified also
- The number of people required can be determined only after an estimate of the development effort

Reusable Software Resources

- Component-based software engineering emphasizes reusability—that is, the creation and reuse of software building blocks. Such building blocks, often called components, must be cataloged for easy reference, standardized for easy application, and validated for easy integration.
- Reusable software components can be classified into the following categories

  Off-the-shelf components
  - Components are from a third party or were developed for a previous project
  - Ready to use; fully validated and documented; virtually no risk

  Full-experience components
  - Components are similar to the software that needs to be built
  - Software team has full experience in the application area of these components
  - Modification of components will incur relatively low risk

  Partial-experience components
  - Components are related somehow to the software that needs to be built but will require substantial modification
  - Software team has only limited experience in the application area of these components
  - Modifications that are required have a fair degree of risk

  New components
  - Components must be built from scratch by the software team specifically for the needs of the current project
  - Software team has no practical experience in the application area
  - Software development of components has a high degree of risk

Environmental Resources.

- A software engineering environment (SEE) incorporates hardware, software, and network resources that provide platforms and tools to develop and test software work products.
- Most software organizations have many projects that require access to the SEE provided by the organization

Planners must identify the time window required for hardware and software and verify that these resources will be available

Topic 6: Explain in detail about Software PROJECT ESTIMATION

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.
Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.
There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

- **Putnam Model**

This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

- **COCOMO**

  COCOMO stands for COnstructive COst MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Topic 7: Explain various categories of SOFTWARE RISKS

- Risk Analysis and management are activities that help a software team to understand uncertainty.
- Risk exhibits two characteristics
  1. Uncertainty: The occurrence of risk is uncertain that is risk may or may not occur.
  2. Loss: If the risk becomes a reality, unwanted consequences or losses will occur.
- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.

Different categories of risks are:

Project risks:

- They threaten the project plan.
- That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase.
- Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

Schedule Risk:

Project schedule get slip when project tasks and schedule release risks are not addressed properly.
Schedule risks mainly affect a project and finally on company economy and may lead to project failure.
Schedules often slip due to the following reasons:

- Wrong time estimation
- Resources are not tracked properly. All resources like staff, systems, skills of individuals, etc.
- Failure to identify complex functionalities and time required to develop those functionalities.

.

Budget Risk

- Wrong budget estimation.
- Cost overruns
- Project scope expansion

Operational Risks: Risks of loss due to improper process implementation failed system or some external events risks. Causes of Operational Risks:

- Failure to address priority conflicts
- Failure to resolve the responsibilities
- Insufficient resources
- No proper subject training
- No resource planning
- No communication in the team.

Technical risks:

- ✓ They threaten the quality and timeliness of the software to be produced.
- ✓ If a technical risk becomes a reality, implementation may become difficult or impossible.
- ✓ Technical risks identify potential design, implementation, interface, verification, and Maintenance problems.

Technical risks generally lead to failure of functionality and performance.
Causes of Technical Risks are:

- Continuous changing requirements
- No advanced technology available or the existing technology is in the initial stages.
- The product is complex to implement.
- Difficult project modules integration.

Programmatic Risks: These are the external risks beyond the operational limits. These are all uncertain risks are outside the control of the program. These external events can be:

- Running out of the fund.
- Market development
- Changing customer product strategy and priority
- Government rule changes.

Predictable risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).

Unpredictable risks are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

Topic 8: Explain Make-or-Buy Decision in detail.

Introduction

- ✓ Outsourcing is closely related to make or buy decision. The corporations made decisions on what to make internally and what to buy from outside in order to maximize the profit margins.
- ✓ As a result of this, the organizational functions were divided into segments and some of those functions were outsourced to expert companies, who can do the same job for much less cost.

- ✓ Make or buy decision is always a valid concept in business. No organization should attempt to make something by their own, when they stand the opportunity to buy the same for much less price.
- ✓ This is why most of the electronic items manufactured and software systems developed in the Asia, on behalf of the organizations in the USA and Europe.

Four Numbers You should know to make make/buy decision

When you are supposed to make a make-or-buy decision, there are four numbers you need to be aware of. Your decision will be based on the values of these four numbers. Let's have a look at the numbers now. They are quite self-explanatory.

- The volume
- The fixed cost of making
- Per-unit direct cost when making
- Per-unit cost when buying

Now, there are two formulas that use the above numbers. They are 'Cost to Buy' and 'Cost to Make'. The higher value loses and the decision maker can go ahead with the less costly solution.

Cost to Buy(CTB)=Volume x Per-unit cost when buying

Cost to Make(CTM)=Fixed costs +(Per-unit direct cost x volume)

Reasons for Making

There are number of reasons a company would consider when it comes to making in-house. Following are a few:

- Cost concerns
- Need of direct control over the product
- Supplier unreliability
- Lack of competent suppliers
- Reduction of logistic costs (shipping etc.)
- To maintain a backup source
- Political and environment reasons
- Organizational pride

Reasons for Buying

Following are some of the reasons companies may consider when it comes to buying from a supplier:

- Lack of technical experience
- Supplier's expertise on the technical areas and the domain
- Cost considerations
- Need of small volume
- Insufficient capacity to produce in-house
- Brand preferences
- Strategic partnerships

Conclusion:

- Make-or-buy decision is one of the key techniques for management practice. Due to the global outsourcing, make-or-buy decision making has become popular and frequent.
- If you make a make-or-buy decision that can create a high impact, always use a process for doing that. When such a process is followed, the activities are transparent and the decisions are made for the best interest of the company.

More Information on MAKE/BUY DECISION

- It is often more cost effective to acquire rather than develop software
- Managers have many acquisition options
  - Software may be purchased (or licensed) off the shelf
  - “Full-experience” or “partial-experience” software components may be acquired and integrated to meet specific needs
  - Software may be custom built by an outside contractor to meet the purchaser’s specifications
- The make/buy decision can be made based on the following conditions
  - Will the software product be available sooner than internally developed software?
  - Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  - Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

What is ER Diagram?

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

What is ER Model?

ER Model stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database.

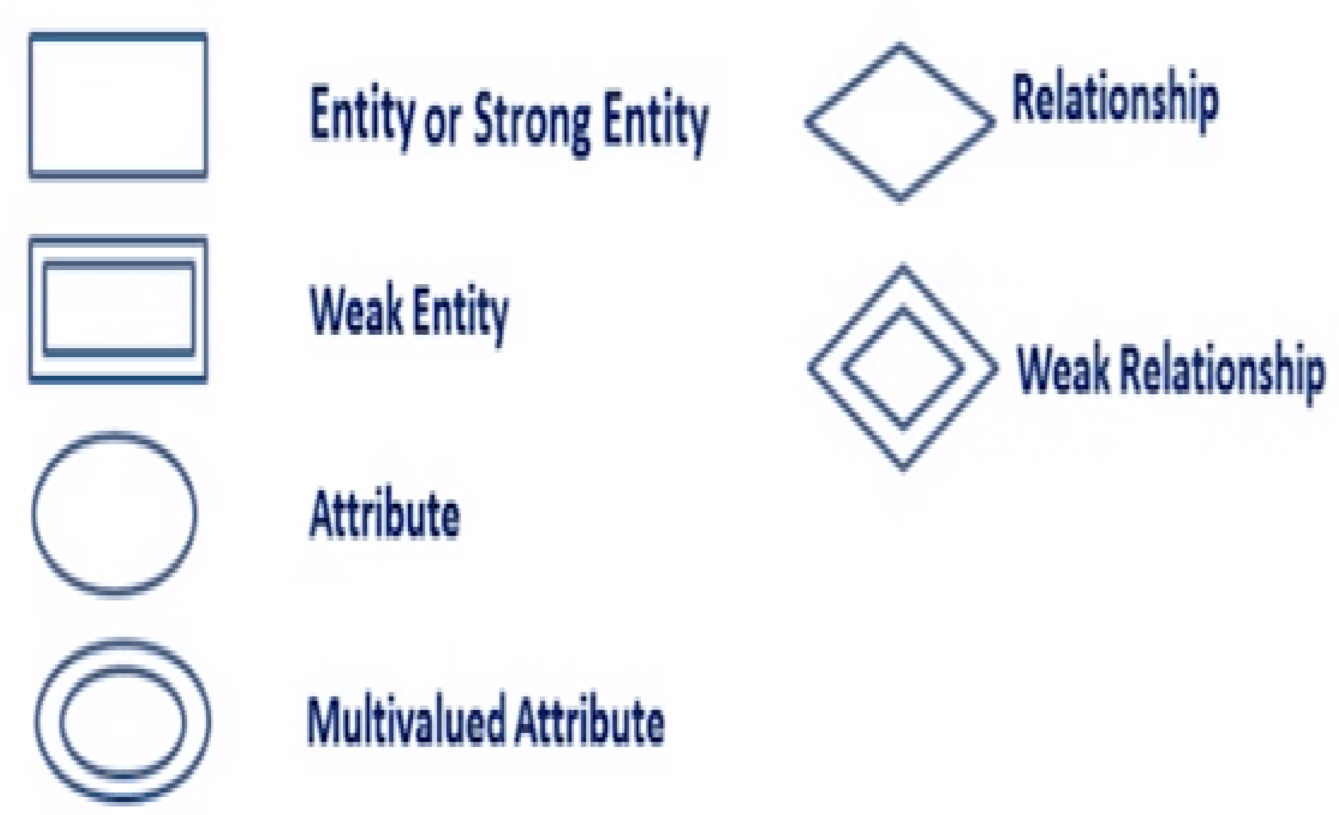
Here, are prime reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD Diagram allows you to communicate with the logical structure of the database to users

Entity Relationship Diagram Symbols & Notations mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations.

Following are the main components and its symbols in ER Diagrams:

- **Rectangles:** This Entity Relationship Diagram symbol represents entity types
- **Ellipses :** Symbol represent attributes
- **Diamonds:** This symbol represents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes



Components of the ER Diagram

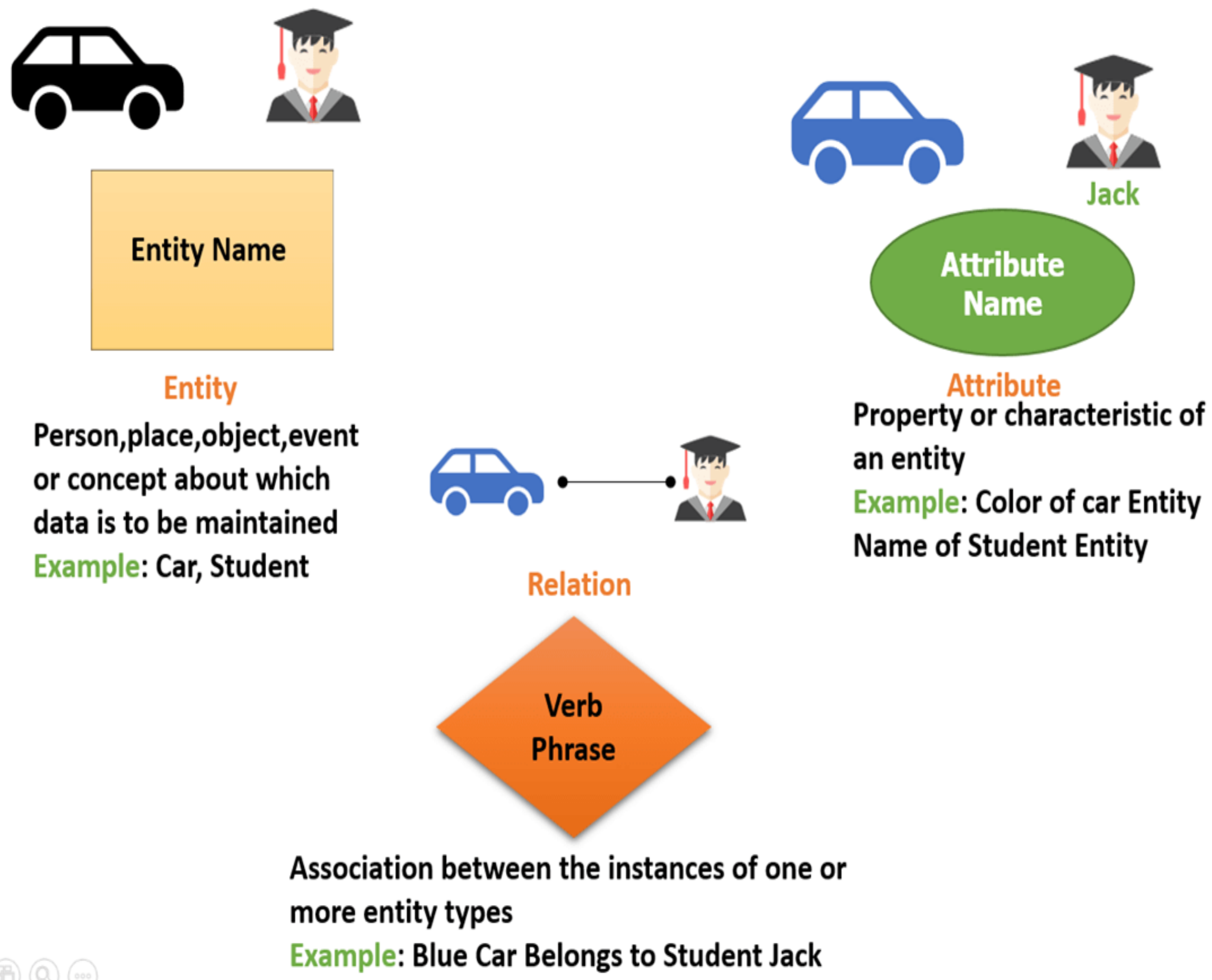
This model is based on three basic concepts:

- Entities

- Attributes
- Relationships

ER Diagram Examples

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.


Entity Name
Entity
Person,place,object,event or concept about which data is to be maintained
Example: Car, Student
Jack
Attribute Name
Attribute
Property or characteristic of an entity
Example: Color of car Entity Name of Student Entity
Relation
Verb Phrase
Association between the instances of one or more entity types
Example: Blue Car Belongs to Student Jack

Components of the ER Diagram

WHAT IS ENTITY?

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Examples of entities:

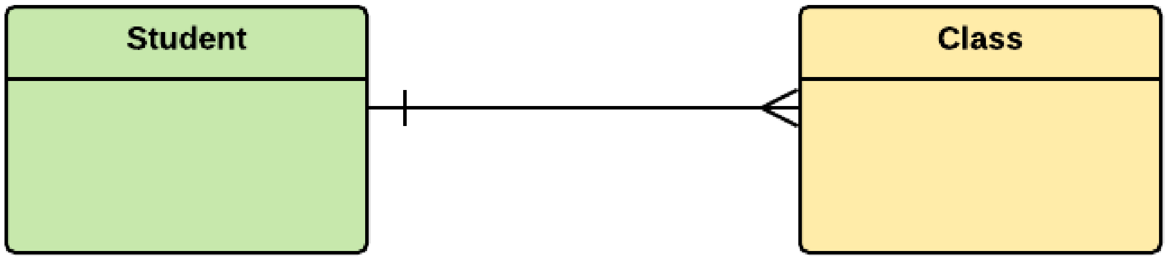
- **Person:** Employee, Student, Patient
- **Place:** Store, Building
- **Object:** Machine, product, and Car
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course, book, machine

Notation of an Entity

Entity set:

Student

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.



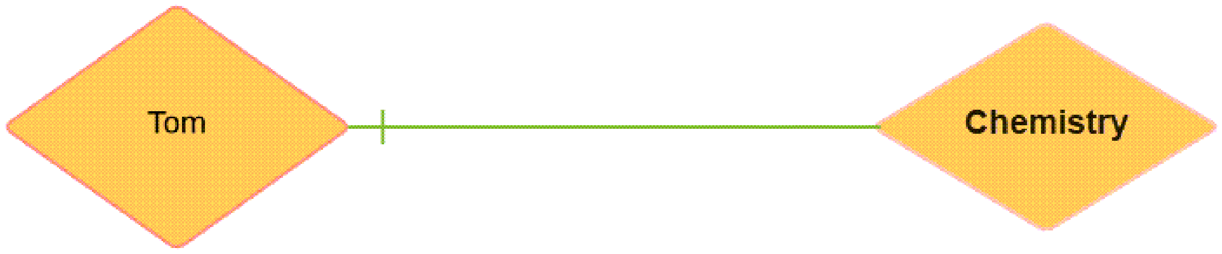
Example of Entities:

A university may have some departments. All these departments employ various lecturers and offer several programs.

Some courses make up each program. Students register in a particular program and enroll in various courses. A lecturer from the specific department takes each course, and each lecturer teaches a various group of students.

Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.



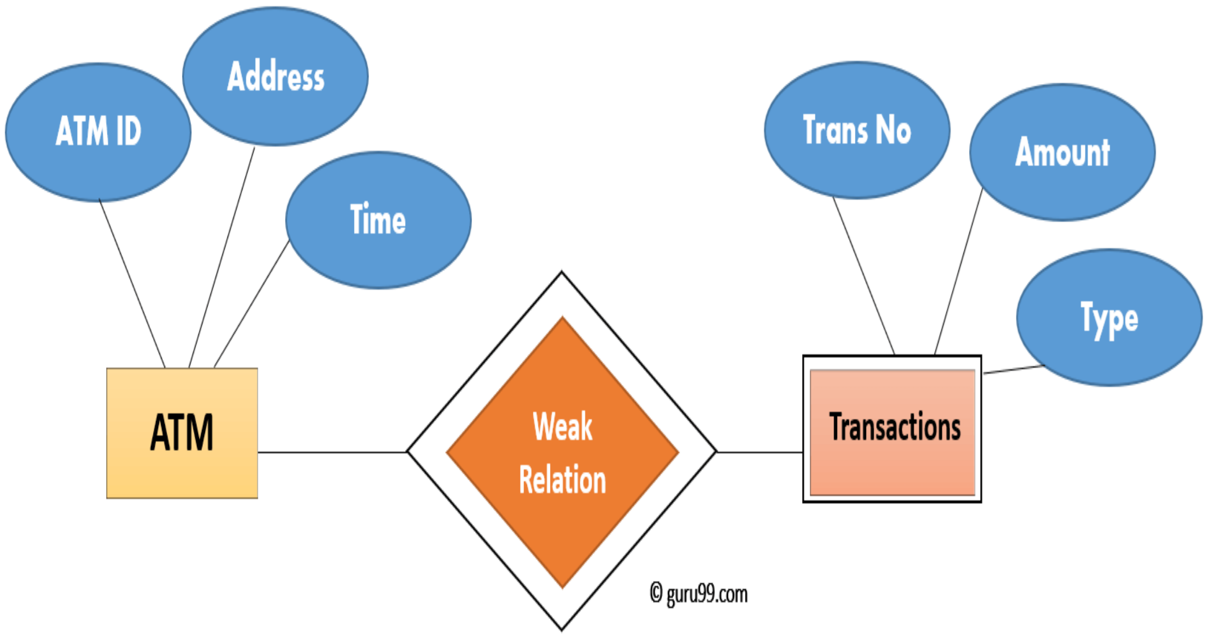
Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

For example:

- You are attending this lecture
- I am giving the lecture
- Just loke entities, we can classify relationships according to relationship-types:
- A student attends a lecture
- A lecturer is giving a lecture.

Weak Entities

A weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.



In above ER Diagram examples, "Trans No" is a discriminator within a group of transactions in an ATM.

Let's learn more about a weak entity by comparing it with a Strong Entity

Strong Entity Set	**Weak Entity Set**
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.

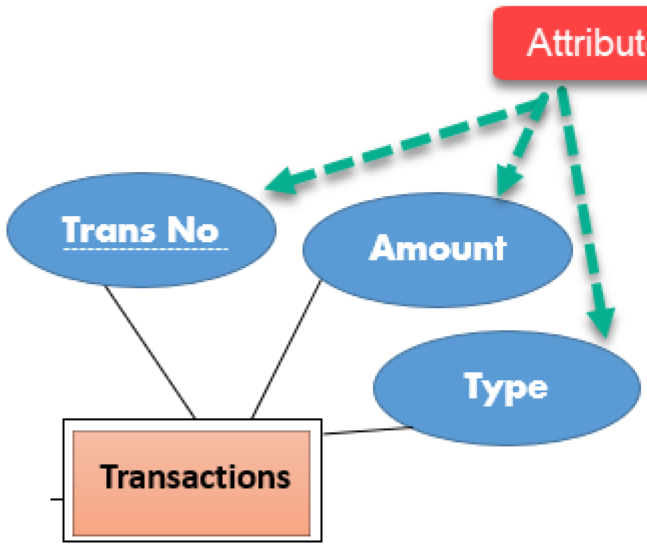
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

Attributes

It is a single-valued property of either an entity-type or a relationship-type.

For example, a lecture might have attributes: time, date, duration, place, etc.

An attribute in ER Diagram examples, is represented by an Ellipse



Types of Attributes	Description

Simple attribute	Simple attributes can't be divided any further. For example, a student's contact number. It is also called an atomic value.
Composite attribute	It is possible to break down composite attribute. For example, a student's full name may be further divided into first name, second name, and last name.
Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age should not be stored directly. Instead, it should be derived from the DOB of that employee.
Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

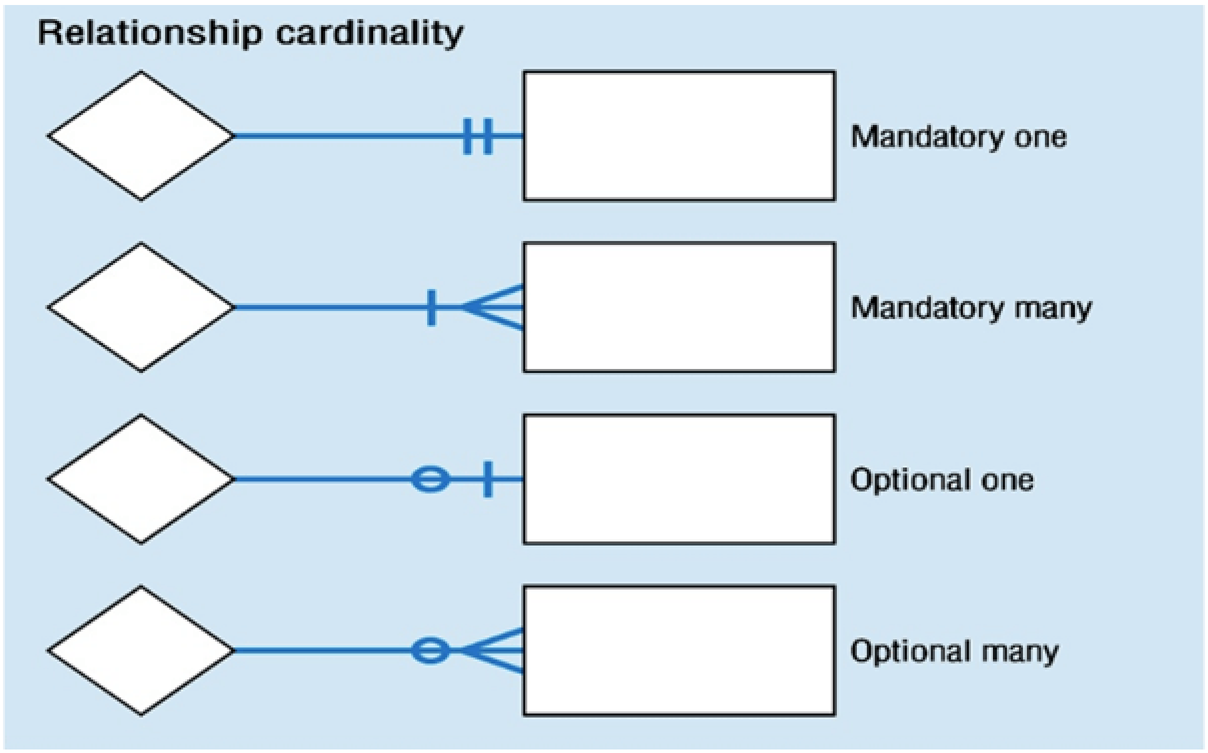
Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships

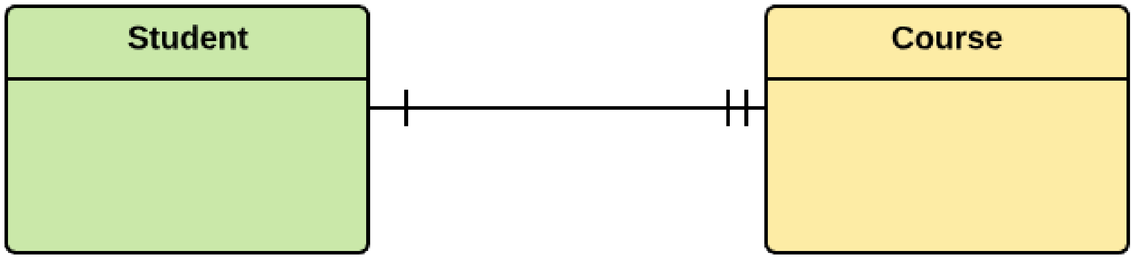
- Many-to-Many Relationships



1.One-to-one:

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

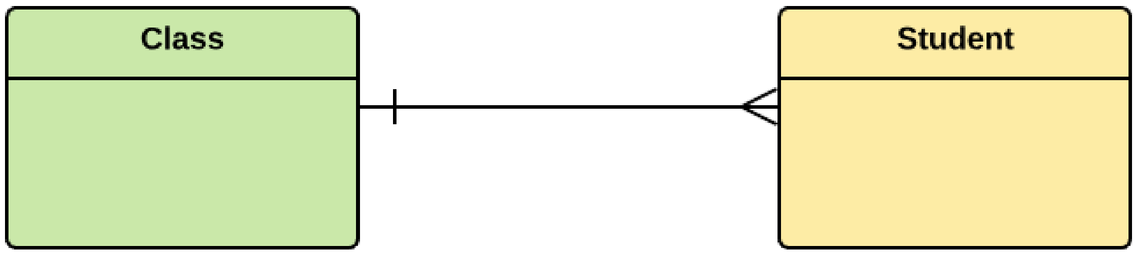
Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.



2.One-to-many:

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

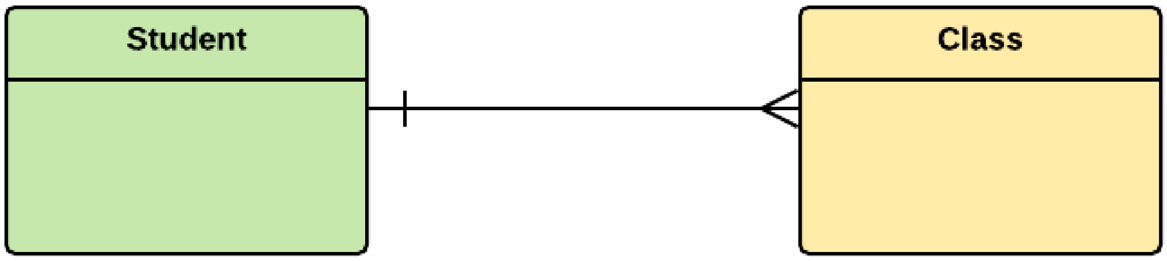
For example, one class is consisting of multiple students.



3. Many to One

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

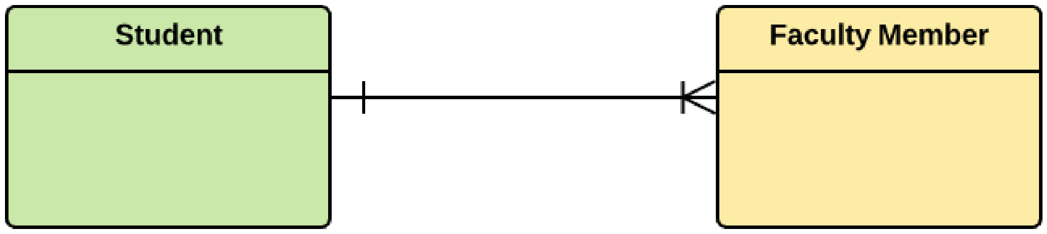
For example, many students belong to the same class.



4. Many to Many:

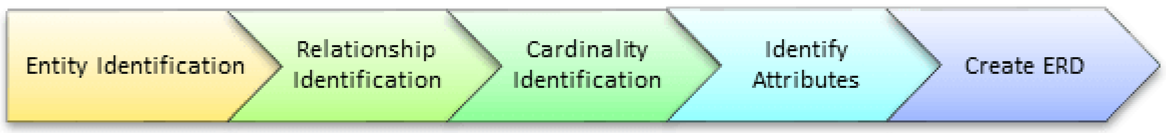
One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.



How to Create an Entity Relationship Diagram (ERD)

Now in this ERD Diagram Tutorial, we will learn how to create an ER Diagram. Following are the steps to create an ER Diagram:



Steps to Create an ER Diagram

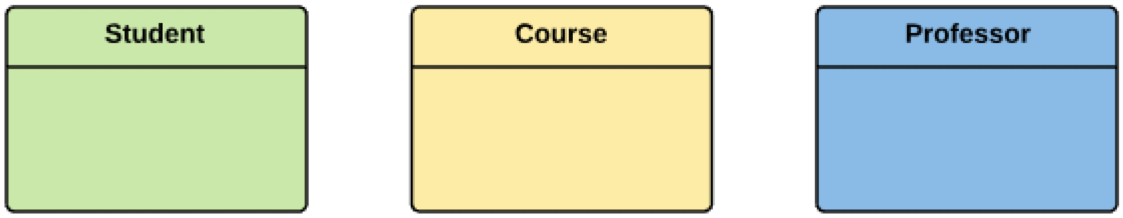
Let's study them with an Entity Relationship Diagram Example:

```
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course
```

Step 1) Entity Identification

We have three entities

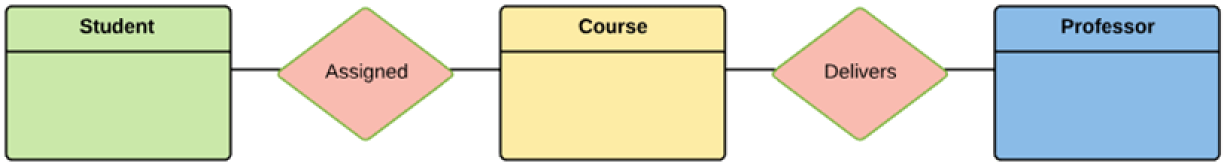
- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

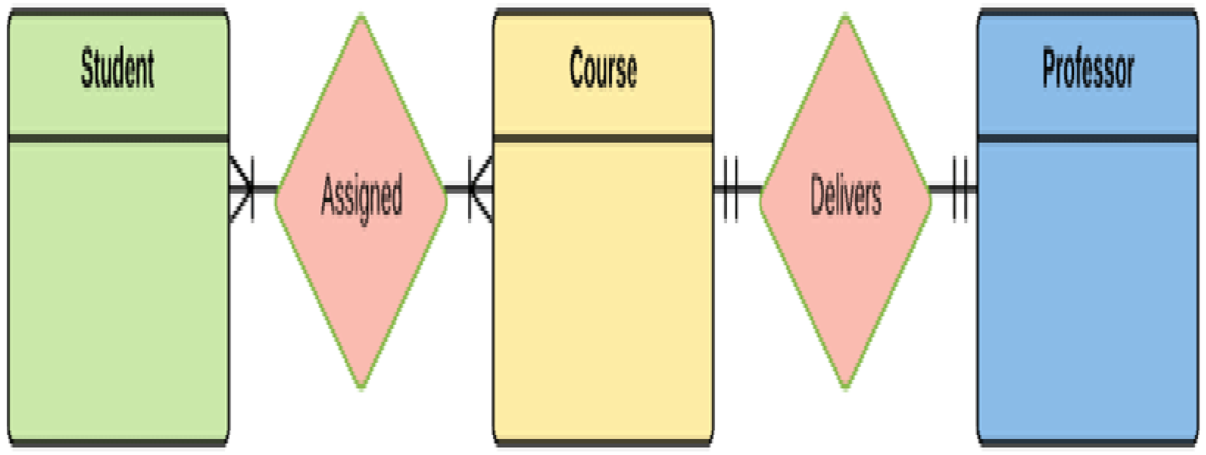
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



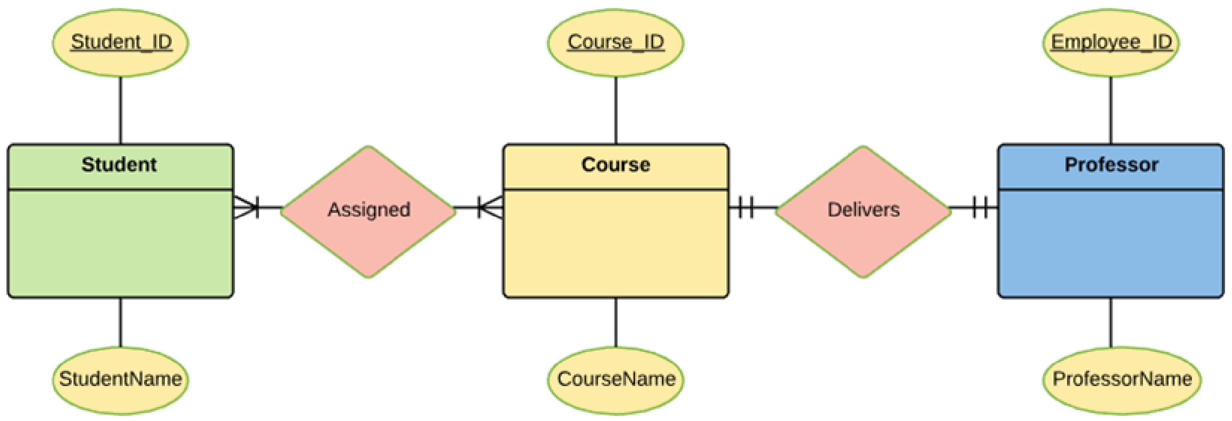
Step 4) Identify Attributes

You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

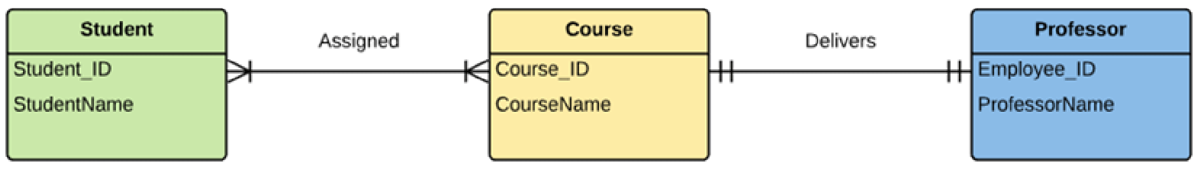
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD Diagram

A more modern representation of Entity Relationship Diagram Example



Best Practices for Developing Effective ER Diagrams

Here are some best practice or example for Developing Effective ER Diagrams.

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other
- You should use colors to highlight important portions of the ER diagram

What is ER Diagram?

ER Diagram stands for Entity Relationship Diagram, also known as ERD is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities, attributes and relationships.

What is ER Model?

ER Model stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database.

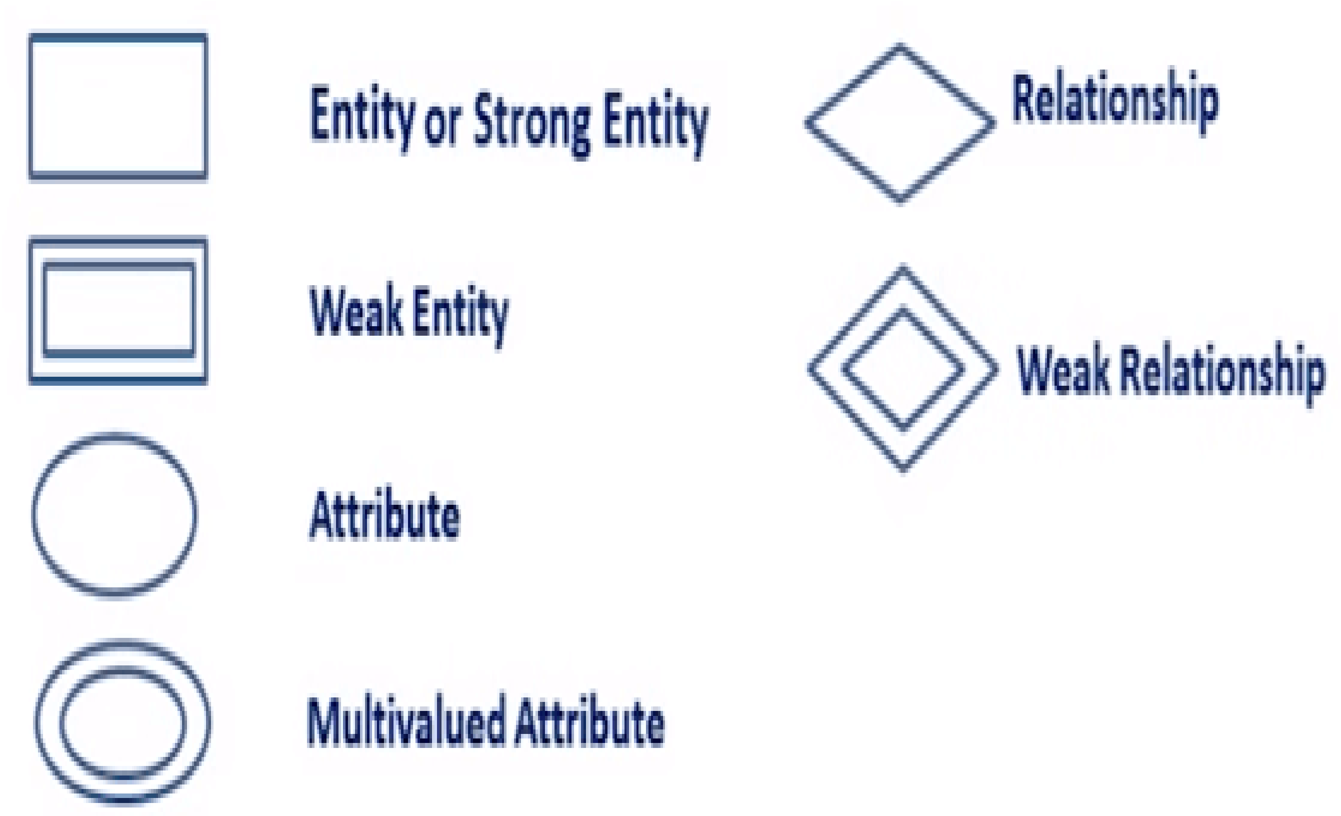
Here, are prime reasons for using the ER Diagram

- Helps you to define terms related to entity relationship modeling
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD Diagram allows you to communicate with the logical structure of the database to users

Entity Relationship Diagram Symbols & Notations mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations.

Following are the main components and its symbols in ER Diagrams:

- **Rectangles:** This Entity Relationship Diagram symbol represents entity types
- **Ellipses :** Symbol represent attributes
- **Diamonds:** This symbol represents relationship types
- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes



Components of the ER Diagram

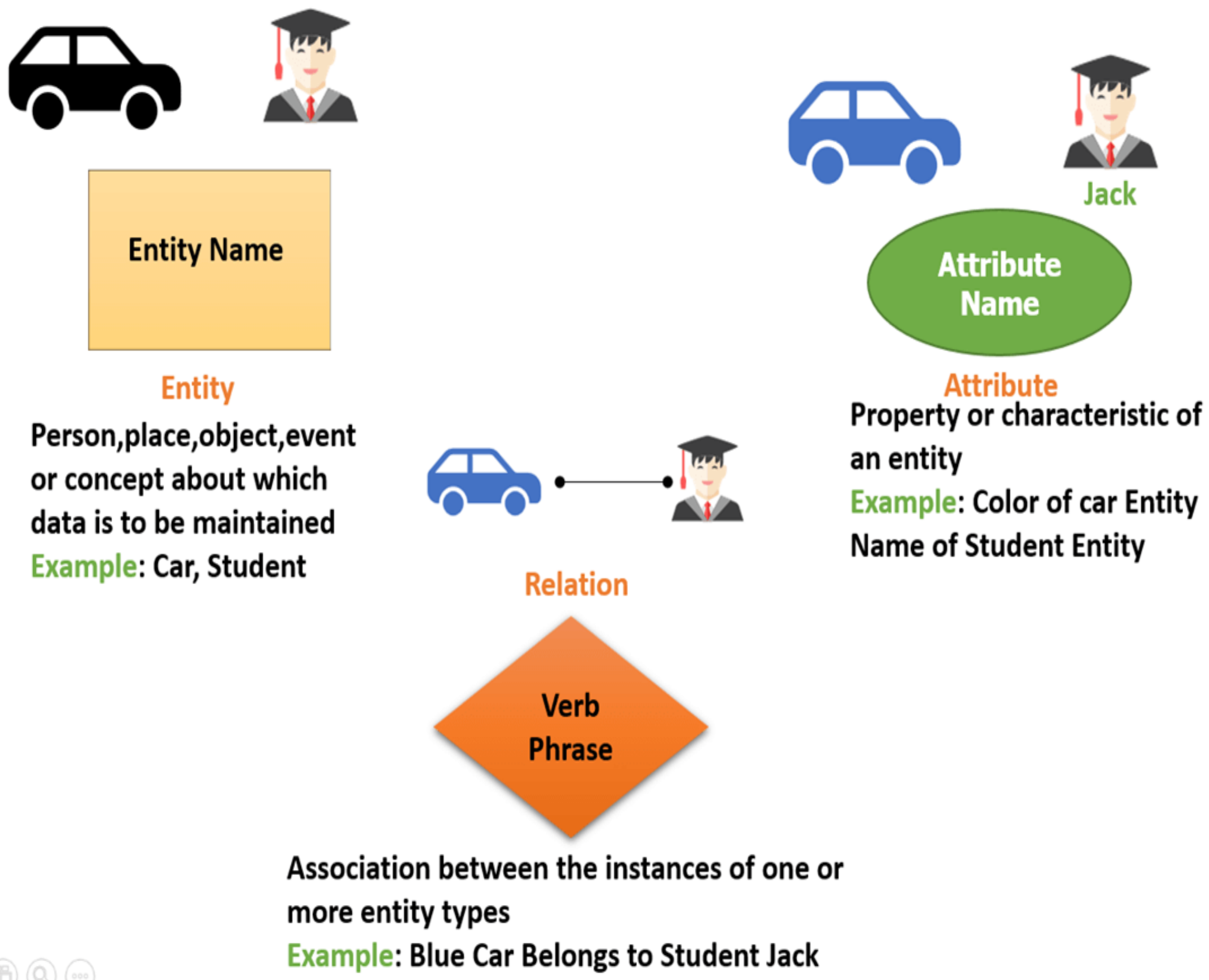
This model is based on three basic concepts:

- Entities

- Attributes
- Relationships

ER Diagram Examples

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.


Entity Name
Entity
Person,place,object,event or concept about which data is to be maintained
Example: Car, Student
Jack
Attribute Name
Attribute
Property or characteristic of an entity
Example: Color of car Entity Name of Student Entity
Relation
Verb Phrase
Association between the instances of one or more entity types
Example: Blue Car Belongs to Student Jack

Components of the ER Diagram

WHAT IS ENTITY?

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Examples of entities:

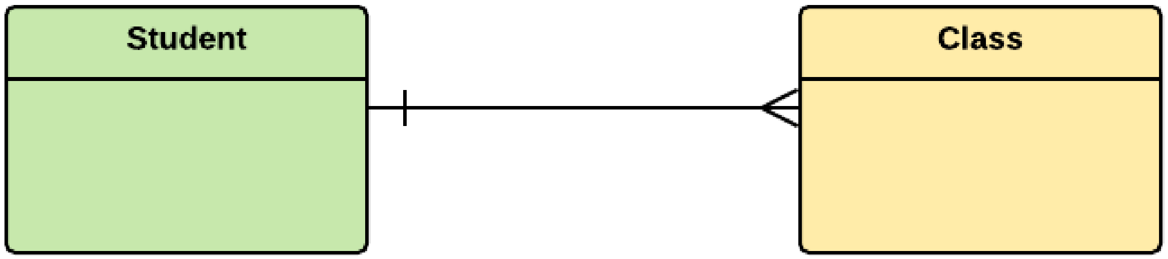
- **Person:** Employee, Student, Patient
- **Place:** Store, Building
- **Object:** Machine, product, and Car
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course, book, machine

Notation of an Entity

Entity set:

Student

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.



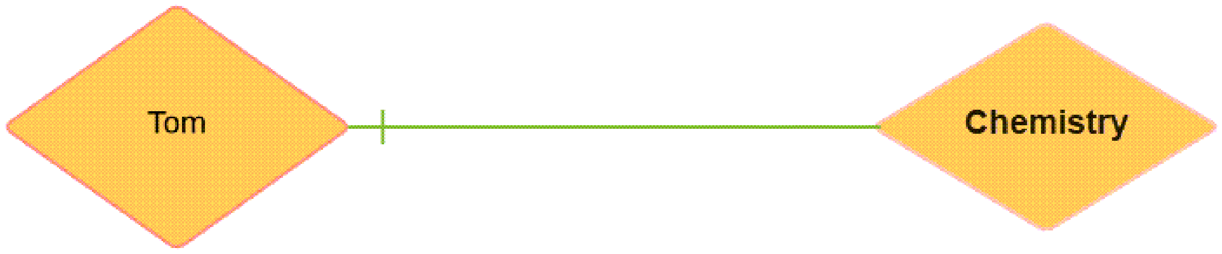
Example of Entities:

A university may have some departments. All these departments employ various lecturers and offer several programs.

Some courses make up each program. Students register in a particular program and enroll in various courses. A lecturer from the specific department takes each course, and each lecturer teaches a various group of students.

Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.



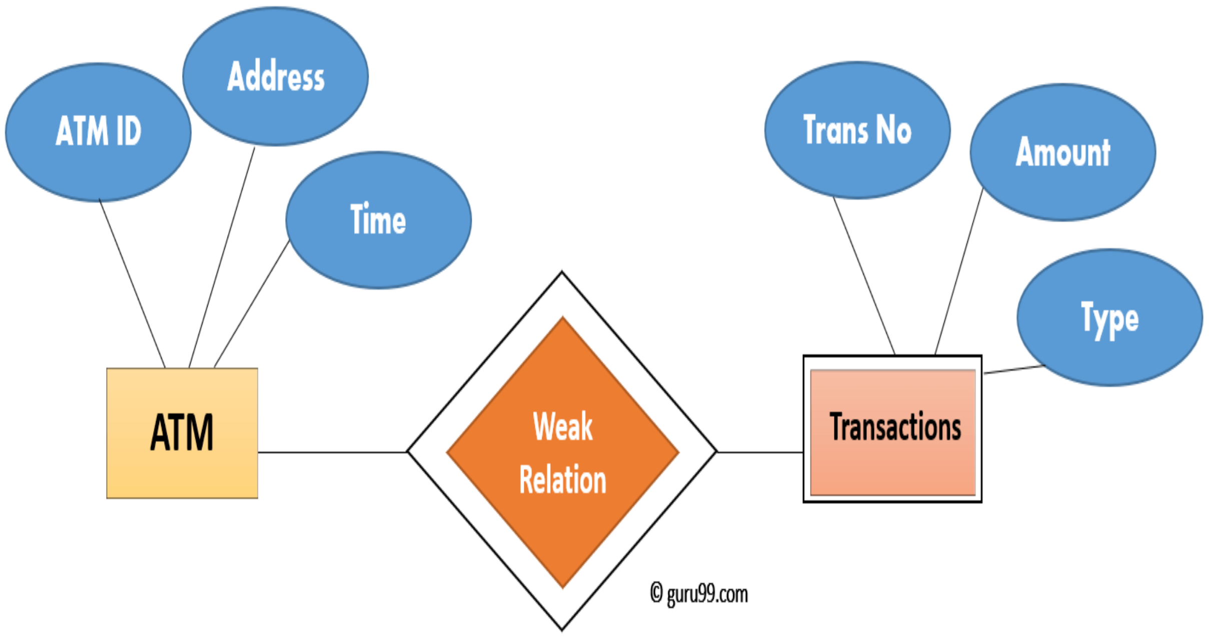
Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

For example:

- You are attending this lecture
- I am giving the lecture
- Just loke entities, we can classify relationships according to relationship-types:
- A student attends a lecture
- A lecturer is giving a lecture.

Weak Entities

A weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.



In above ER Diagram examples, "Trans No" is a discriminator within a group of transactions in an ATM.

Let's learn more about a weak entity by comparing it with a Strong Entity

Strong Entity Set	**Weak Entity Set**
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.

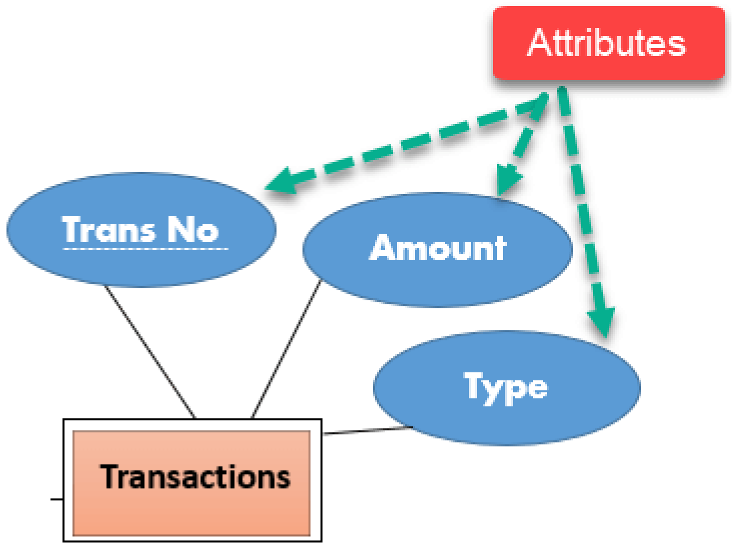
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

Attributes

It is a single-valued property of either an entity-type or a relationship-type.

For example, a lecture might have attributes: time, date, duration, place, etc.

An attribute in ER Diagram examples, is represented by an Ellipse



Types of Attributes	Description

Simple attribute	Simple attributes can't be divided any further. For example, a student's contact number. It is also called an atomic value.
Composite attribute	It is possible to break down composite attribute. For example, a student's full name may be further divided into first name, second name, and last name.
Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age should not be stored directly. Instead, it should be derived from the DOB of that employee.
Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

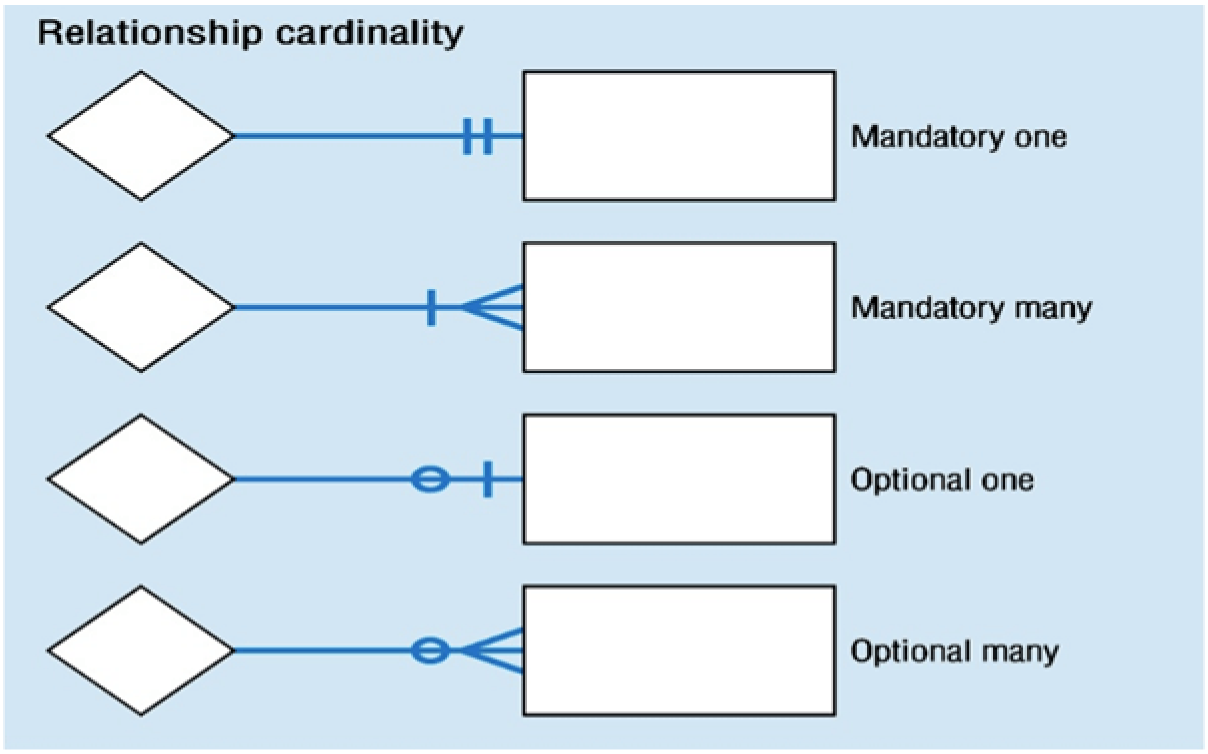
Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets.

Different types of cardinal relationships are:

- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships

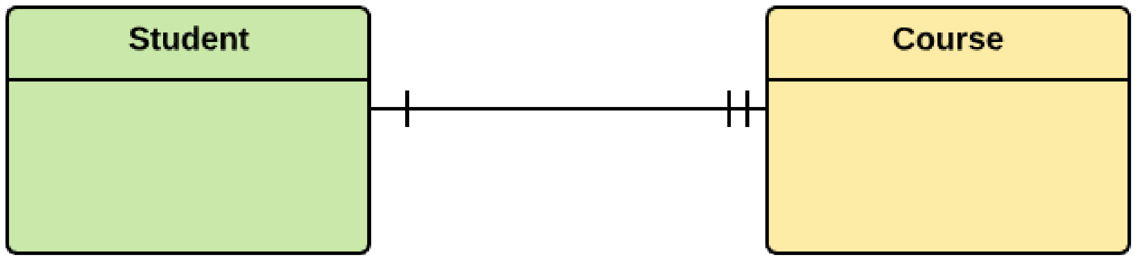
- Many-to-Many Relationships



1.One-to-one:

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

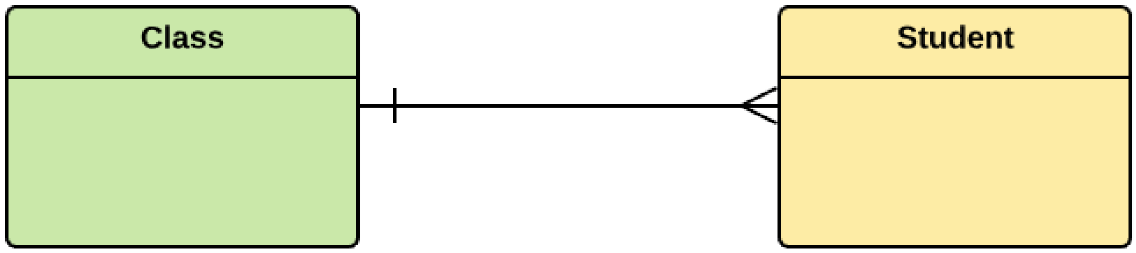
Example: One student can register for numerous courses. However, all those courses have a single line back to that one student.



2.One-to-many:

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

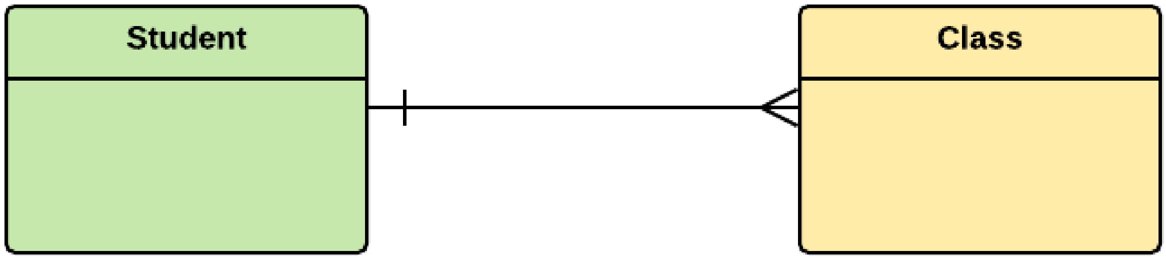
For example, one class is consisting of multiple students.



3. Many to One

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

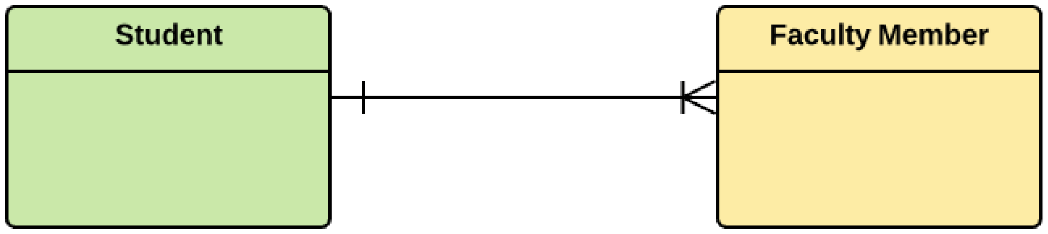
For example, many students belong to the same class.



4. Many to Many:

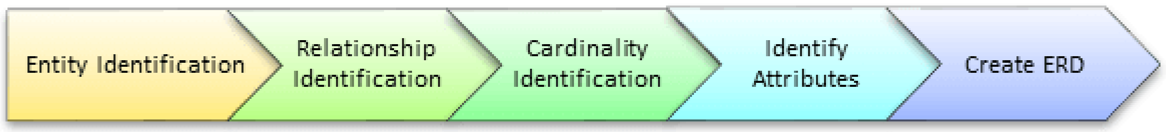
One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.



How to Create an Entity Relationship Diagram (ERD)

Now in this ERD Diagram Tutorial, we will learn how to create an ER Diagram. Following are the steps to create an ER Diagram:



Steps to Create an ER Diagram

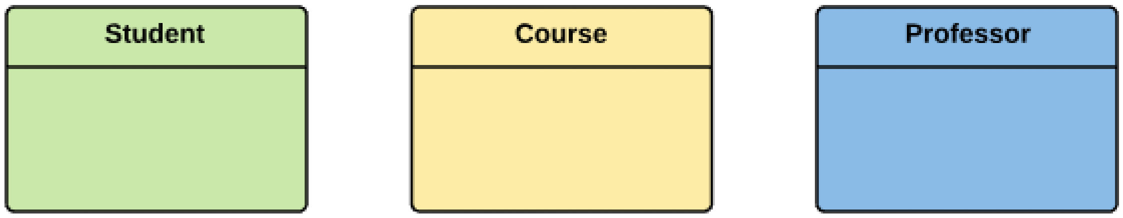
Let's study them with an Entity Relationship Diagram Example:

```
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course
```

Step 1) Entity Identification

We have three entities

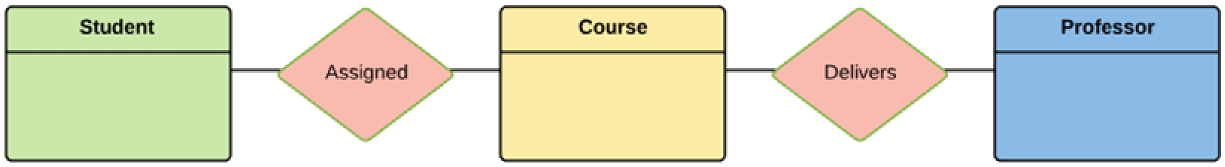
- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

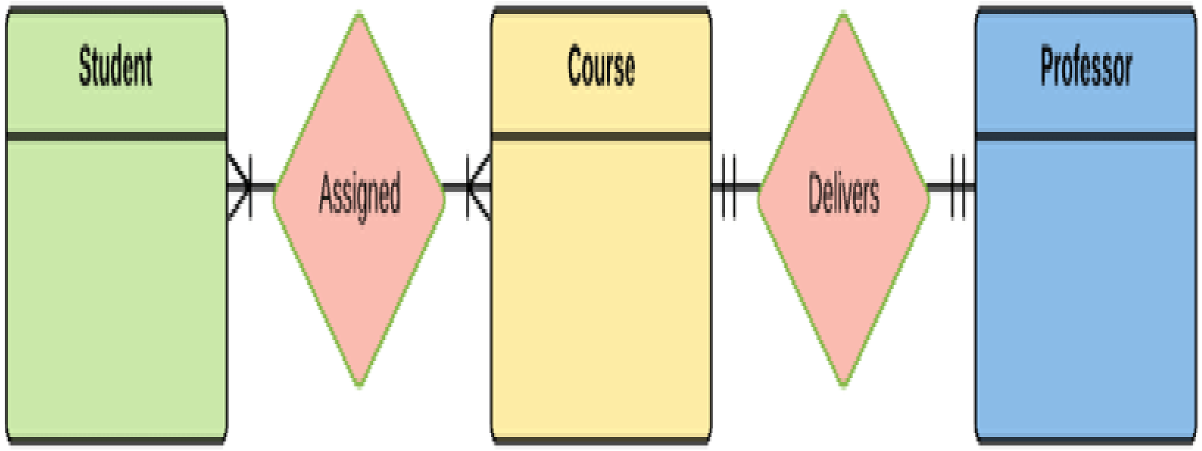
- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course



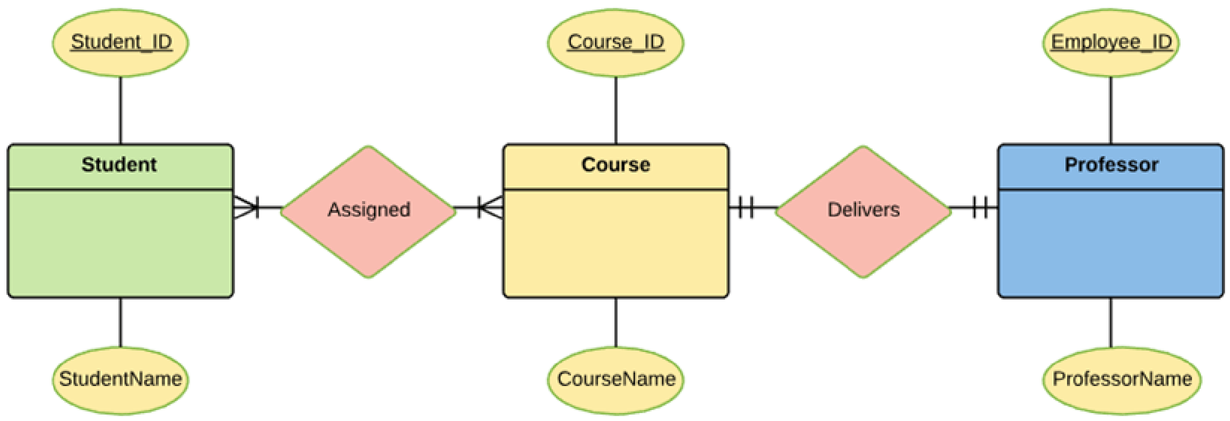
Step 4) Identify Attributes

You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.

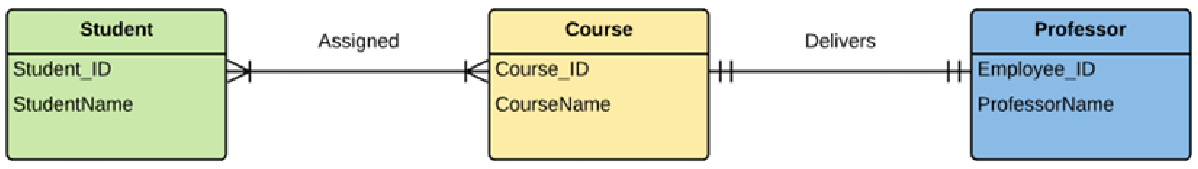
Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName



For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD Diagram

A more modern representation of Entity Relationship Diagram Example



Best Practices for Developing Effective ER Diagrams

Here are some best practice or example for Developing Effective ER Diagrams.

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other
- You should use colors to highlight important portions of the ER diagram

Data Flow Diagram

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

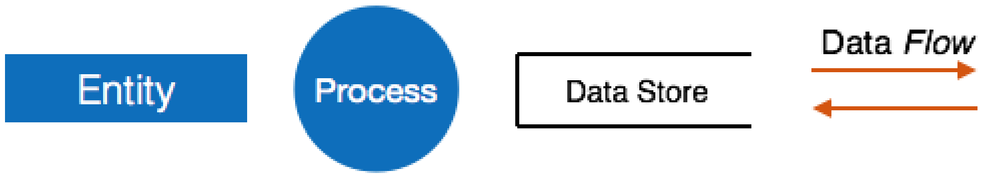
Data Flow Diagrams are either Logical or Physical.

Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system.For example in a Banking software system, how data is moved between different entities.

Physical DFD - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

5.1 DFD Components

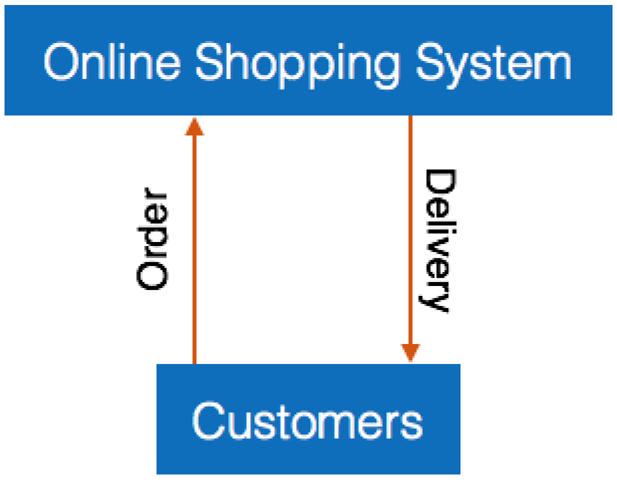
DFD can represent Source, destination, storage and flow of data using the following set of components -



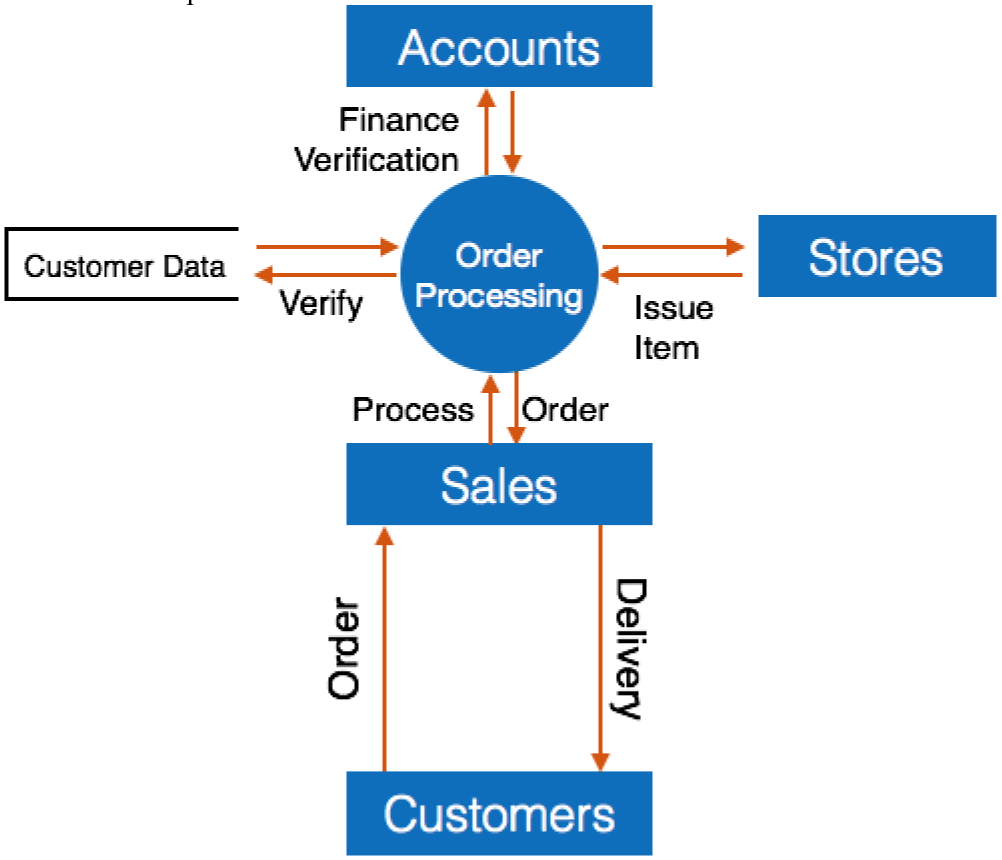
- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

5.2 Levels of DFD

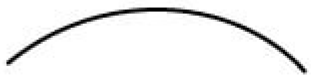
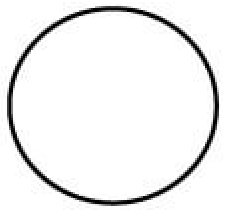
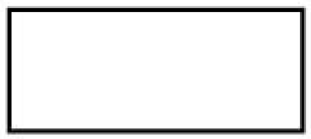
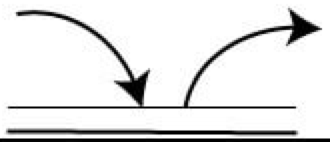
- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



- **Level 1 -** The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



- **Level 2 -** At this level, DFD shows how data flows inside the modules mentioned in Level 1. Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Symbol	Name	Function
	Data flow	Used to Connect Processes to each , other , to sources or Sinks; te arrow head indicates direction of data flow.
	Process	Perfroms Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

Symbols for Data Flow Diagrams

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Data Dictionary

A data dictionary is a collection of descriptions of the data objects or items in a data model for the benefit of programmers and others who need to refer to them. ... The data dictionary would describe each of the data items in its data model for consumer banking (for example, "Account holder" and ""Available credit").

Functions of Data Dictionary

- It defines the **data** objects of each user in the database.

- By this way, it helps various users to know all the objects which exist in the database and who can access it. One cannot remember all the tables, views constraints etc in a huge database ...
- A **data dictionary** is a collection of **data** about **data**.
- It maintains information about the definition, structure, and use of each **data** element that an organization uses.
- It is also used in DBMSs to generate reports

UNIT IV- Design Concepts & Principles.

Topic 1: Explain Software Design Process

- ✓ Software design is an iterative process through which requirements are translated into a "**blueprint**" for constructing the software.
- ✓ Initially, the **blueprint** depicts a holistic view of software, i.e. the design is represented at a high-level of abstraction.
- ✓ Throughout the design process, the quality of the evolving design is assessed with a series of formal technique reviews or design walkthroughs.

Three characteristics serve as a guide for the evaluation of a good design:

- ✓ The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- ✓ The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- ✓ The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality Guidelines

In order to evaluate the quality of a design representation, we must establish technical criteria for good design.

1. A design should exhibit an **architecture** that:
   (1) Has been created using recognizable architectural styles or patterns,
   (2) Is composed of components that exhibit good design characteristics, and
2. A **design should be modular**; that is, the software should be logically partitioned into elements or subsystems
3. A design should **contain distinct representations** of data, architecture, interfaces, and components.
4. A design should lead to **data structures** that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
5. A design should lead to **components** that exhibit independent functional characteristics.
6. A design should lead to **interfaces** that reduce the complexity of connections between components and with the external environment.
7. A design should be represented using a **notation** that effectively communicates its meaning.

Quality Attributes

Hewlett-Packard developed a set of software quality attributes that has been given the acronym FURPS. The FURPS quality attributes represent a target for all software design:

- ✓ *Functionality*: is assessed by evaluating the features set and capabilities of the program.
- ✓ *Usability*: is assessed by considering human factors, consistency, and documentation.

- ✓ *Reliability*: is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure, the ability to recover from failure, and the predictability of the program.
- ✓ *Performance*: is measured by processing speed, response time, resource consumption, throughput, and efficiency.
- ✓ *Supportability*: combines the ability to extend the program extensibility, adaptability, serviceability ➔ maintainability. In addition, testability, compatibility, configurability, etc.

Topic 2: Explain about Design Principles

Basic design principles enable the software engineer to navigate the design process.

- **The design process should not suffer from “tunnel vision.”:** A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job, and the design concepts
- **The design should not reinvent the wheel:** Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention.
- **The design should be traceable to the analysis model:** Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.
- **The design should “minimize the intellectual distance” between the software and the problem as it exists in the real world:** That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain.
- **The design should be structured to accommodate change.**
- **The design should exhibit uniformity and integration:** A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins.
- **The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered** Well- designed software should never “bomb.” It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.
- **Design is not coding, coding is not design.** Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.
- **The design should be assessed for quality as it is being created, not after the fact**. A variety of design concepts and design measures are available to assist the designer in assessing quality.
- **The design should be reviewed to minimize conceptual (semantic) errors.** A design team should ensure that major conceptual elements of the design (omissions, ambiguity, and inconsistency) have been addressed before worrying about the syntax of the design model.

Topic 3: Explain about DESIGN CONCEPTS.

Important software design concepts are

Abstraction

- Abstraction is one of the fundamental ways that humans cope with complexity.
- At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more detailed description of the solution is provided.
- A procedural abstraction refers to a sequence of instructions that have a specific and limited function. The name of a procedural abstraction implies these functions, but specific details are suppressed.
- A data abstraction denotes the essential characteristics of an object that distinguishes it from all kinds of objects.
- Control abstraction implies a program control mechanism without specifying internal details.

Refinement

- Stepwise refinement is a top-down design strategy.
- Refinement is actually a process of elaboration. Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement (elaboration) occurs.
- A program is developed by successively refining levels of procedural detail.
- A hierarchy is developed by decomposing a macroscopic statement of function (a procedural abstraction) in a stepwise fashion until programming language statements are reached.

Modularity

- The software is divided into separately named and addressable components, often called modules that are integrated to satisfy problem requirements.
- "Modularity is the single attribute of software that allows a program to be intellectually manageable"
- Five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system are Modular Decomposability, Modular Composability, Modular Understandability, Modular Continuity and Modular Protection

Software Architecture

- Architecture is the structure or organization of program components (modules), the manner in which these components interact, and the structure of data that are used by the components.
- This aspect of the architectural design representation defines the components of a system (e.g., modules, objects, filters) and the manner in which those components are packaged and interact with one another.
- Software architecture represents “the overall structure of the software and the ways in which that structure provides conceptual integrity for a system”

Control Hierarchy

- Control hierarchy, also called program structure, represents the organization of program components (modules) and implies a hierarchy of control.
- Depth and Width provide an indication of the number of levels of control and overall span of control, respectively.
- Fan-out is a measure of the number of modules that are directly controlled by another module. Fan-in specifies how many modules directly control a given a module
- A module that controls another module is said to be superordinate to it, and conversely, a module controlled by another is said to be subordinate to the controller

Structural Partitioning

- If the architectural style of a system is hierarchical, the program structure can be partitioned both horizontally and vertically.
- Horizontal partitioning defines separate branches of the modular hierarchy for each major program function. Control modules, represented in a darker shade are used to coordinate communication between and execution of the functions.
- The simplest approach to horizontal partitioning defines three partitions—input, data transformation (often called processing) and output.
- Vertical partitioning, often called factoring, suggests that control (decision making) and work should be distributed top-down in the program structure. Top- level modules should perform control functions and do little actual processing work. Modules that reside low in the structure should be the workers, performing all input, computation, and output tasks.

Data Structure

- Data structure is a representation of the logical relationship among individual elements of data. Data structure dictates the organization, methods of access, degree of associativity, and processing alternatives for information.
- The organization and complexity of a data structure are limited only by the skill of the designer.
- Limited number of classic data structures that form the building blocks for more sophisticated structures.
- Scalar Items, vectors, and spaces may be organized in a variety of formats

Program Structure

- Program structure defines control hierarchy without regard to the sequence of processing and decisions.
- Software procedure focuses on the processing details of each module individually. Procedure must provide a precise specification of processing, including sequence of events, exact decision points, repetitive operations, and even data organization and structure.
- The processing indicated for each module must include a reference to all modules subordinate to the module being described. That is, a procedural representation of software is layered

Information Hiding

- The principle of information hiding suggests that modules be characterized by design decisions that (each) hides from all others.
- Modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information.
- Hiding implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function.
- Hiding defines and enforces access constraints to both procedural detail within a module and any local data structure used by the module.

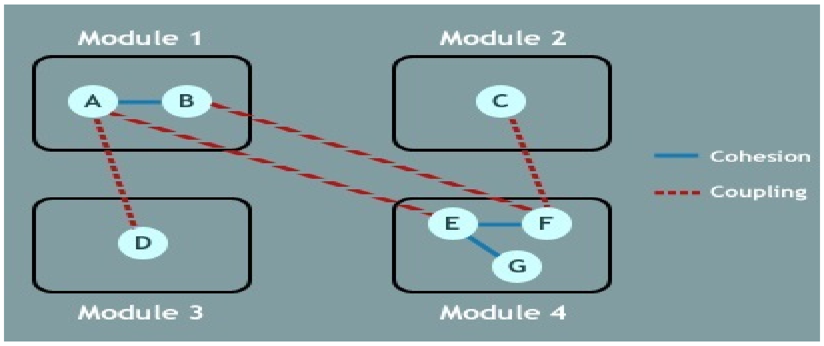
Topic 4: Explain in detail about EFFECTIVE MODULAR DESIGN.

A modular design reduces complexity, facilitates change, and results in easier implementation by encouraging parallel development of different parts of a system.

Functional Independence

- The concept of functional independence is a direct outgrowth of modularity and the concepts of abstraction and information hiding.
- Independent modules are easier to maintain (and test) because secondary effects caused by design or code modification are limited, error propagation is reduced, and reusable modules are possible.
- Independence is measured using two qualitative criteria: cohesion and coupling.

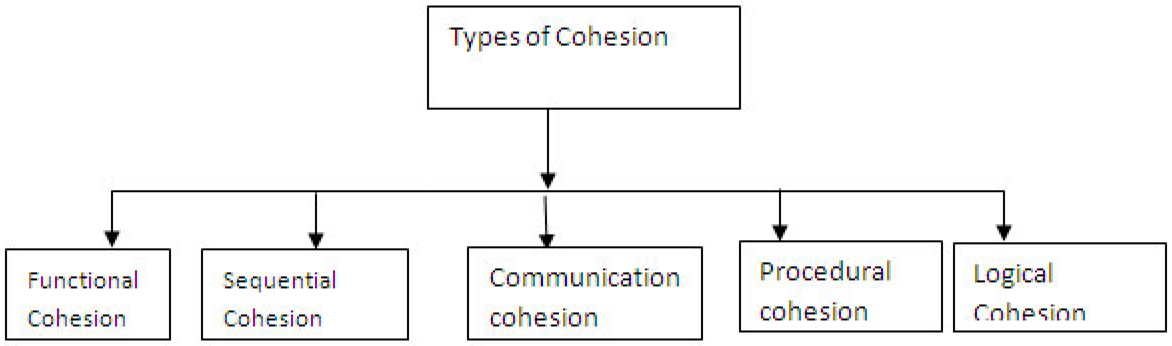
Cohesion and coupling:



Cohesion:

- Cohesion is a measure of the relative functional strength of a module.
- Cohesion can be defined as the degree of the closeness of the relationship between its components. In general, it measures the relationship strength between the pieces of functionality within a given module in the software programming.
- It is described as low cohesion or high cohesion.

Types of Cohesion: There are many different types of cohesion in the software engineering. Some of them are worst, while some of them are best. We have defined them below:



Functional Cohesion: It is best type of cohesion, in which parts of the module are grouped because they all contribute to the module's single well defined task.

2. **Sequential Cohesion:** When the parts of modules grouped due to the output from the one part is the input to the other, then it is known as sequential cohesion.
3. **Communication Cohesion:** In Communication Cohesion, parts of the module are grouped because they operate on the same data. For e.g. a module operating on same information records.
4. **Procedural Cohesion:** In Procedural Cohesion, the parts of the module are grouped because a certain sequence of execution is followed by them.
5. **Logical Cohesion:** When the module's parts are grouped because they are categorized logically to do the same work, even though they are all have different nature, it is known as Logical Cohesion. It is one of the worst type of the cohesion in the software engineering.

Coupling

- Coupling is a measure of interconnection among modules in a software structure.
- In software engineering, the coupling can be defined as the measurement to which the components of the software depend upon each other.
- Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Types of Coupling:

- **Data Coupling:**
  If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent to each other and communicating through data.
- **Stamp Coupling:**
  In stamp coupling, the complete data structure is passed from one module to another module.
- **Control Coupling:**
  If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:**
  In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.

- **Common Coupling:** The modules have shared data such as global data structures.
- **Content Coupling:**
  In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Topic 5: Explain in detail User Interface (UI) and Design Process

- ✓ User interface is the front-end application view to which user interacts in order to use the software.
- ✓ User can manipulate and control the software as well as hardware by means of user interface.
- ✓ UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Command Line Interface (CLI)

- ✓ CLI provides a command prompt, the place where the user types the command and feeds to the system.
- ✓ The user needs to remember the syntax of command and its use.
- ✓ A command is a text-based reference to set of instructions, which are expected to be executed by the system.

CLI Elements

A text-based command line interface can have the following elements:

- **Command Prompt** - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
- **Cursor** - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
- **Command** - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

Graphical User Interface

Graphical User Interface provides the user graphical means to interact with the system.

GUI Elements

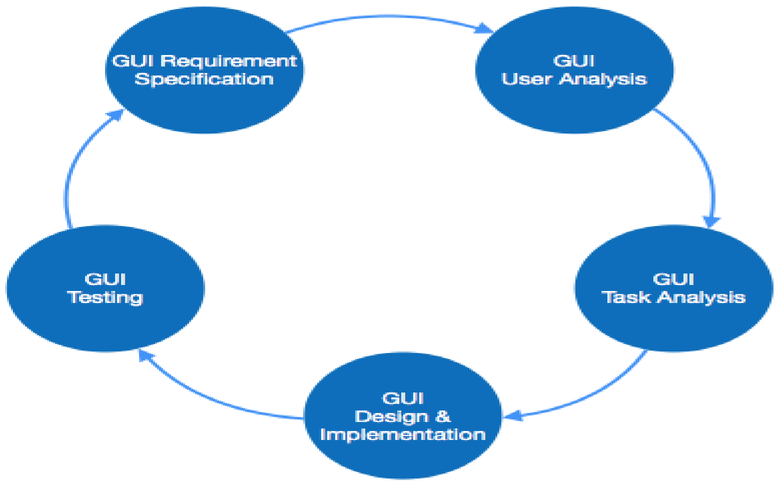
- GUI provides a set of components to interact with software or hardware.
- Every graphical component provides a way to work with the system.

Window - An area where contents of application are displayed.
Tabs - If an application allows executing multiple instances of itself, they appear on the screen as separate windows.
Menu - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window.
Icon - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
Cursor - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.
Application Window - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.
Dialogue Box - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.
Text-Box - Provides an area for user to type and enter text-based data.
Buttons - They imitate real life buttons and are used to submit inputs to the software.
Radio-button - Displays available options for selection. Only one can be selected among all offered.
Check-box - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
List-box - Provides list of available items for selection. More than one item can be selected.
Other elements like Sliders, Combo-box, Data-grid, Drop-down list.

Topic 6: Explain Interface Analysis OR User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.
- **GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
- **Testing** - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

GUI Implementation Tools

There are different segments of GUI tools according to their different use and platform.

Example

Mobile GUI, Computer GUI, Touch-Screen GUI etc. Here is a list of few tools which come handy to build GUI:

- FLUID
- AppInventor (Android)
- LucidChart
- Wavemaker
- Visual Studio
- wordpress

Topic 7: Explain The Golden Rules

The first is to place the user in control (which means have the computer interface support the user's understanding of a task and do not force the user to follow the computer's way of doing things). The second (reduce the user's memory load) means place all necessary information on the screen at the same time. The third is consistency of form and behavior.

The three "golden rules" are:

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent

Place the User in Control

Mandel defines a number of design principles that allow the user to maintain control:

- **Define interaction modes in a way that does not force a user into unnecessary or undesired actions**. The user should always be able to enter and exit the mode with little or no effort.
- **Provide for flexible interaction**. Because different users have different interaction preferences, choices should be provided by using keyboard commands, mouse movements, digitizer pen or voice recognition commands.
- **Allow user interaction to be interruptible and undoable**. A user should be able to interrupt a sequence of actions to do something else without losing the work that has been done. The user should always be able to "undo" any action.
- **Streamline interaction as skill levels advance and allow the interaction to be customized**. Allow to design a macro if the user is to perform the same sequence of actions repeatedly.
- **Hide technical internals from the casual user**. The user interface should move the user into the virtual world of the application. A user should never be required to type O/S commands from within application software.
- **Design for direct interaction with objects that appear on the screen**. The user feels a sense of control when able to manipulate the objects that are necessary to perform a task in a manner similar to what would occur if the object were a physical thing.

Reduce the User's Memory Load

Whenever possible, the system should "remember" pertinent information and assist the user with an interaction scenario that assists recall.

- **Reduce demand on short-term memory**. Provide visual cues that enable a user to recognize past actions, rather than having to recall them.
- **Establish meaningful defaults**. A user should be able to specify individual preferences; however, a reset option should be available to enable the redefinition of original default values.
- **Define shortcuts that are intuitive**. "Example: Alt-P to print. Using easy to remember mnemonics."
- **The visual layout of the interface should be based on a real world metaphor**. Enable the user to rely on well-understood visual cues, rather than remembering an arcane interaction sequence. For a bill payment system use a check book and check register metaphor to guide the user through the process.
- **Disclose information in a progressive fashion**. The interface should be organized hierarchically. The information should be presented at a high level of abstraction.

Make the Interface Consistent

The interface should present and acquire information in a consistent manner:

1. All visual information is organized according to a design standard that is maintained throughout all screen displays,
2. Input mechanisms are constrained to a limited set that is used consistently throughout the application,
3. Mechanisms for navigating from task to task are consistently defined and implemented.

A set of design principles that help make the interface consistent:

Allow the user to put the current task into a meaningful context. The user should be able to determine where he has come from and what alternatives exist for a transition to a new task.
Maintain consistency across a family of applications. "MS Office Suite"
If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Once a particular interactive sequence has become a de facto standard (Alt-S ➔ save file), the user expects this in every application she encounters.