

**UNIT-II**

**SOFTWARE  
ENGINEERING**

**Software Engineering**

# Syllabus

## **Unit-II: Software Requirement Specifications (SRS)**

Requirement Engineering Process: Elicitation, Analysis, Documentation, Review and Management of User Needs, Feasibility Study, Information Modeling, Data Flow Diagrams, Entity Relationship Diagrams, Decision Table Software Quality Assurance (SQA): Verification and Validation, SQA Plans, Software Quality Frameworks, ISO 9000 Models, SEI-CMM Model.

# REQUIREMENT ENGINEERING

- Requirements of the customer play a key role in designing a software product.
- A missed or bad requirement can result into a software error which requires lots of rework and also lead to financial and legal implications.
- The key component is the requirement specification document(RSD) or software requirement specification(SRS) and the process of developing this document is called requirement engineering.

# REQUIREMENT ENGINEERING

- A Requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose.
- It can be defined as a discipline, which addresses requirements of objects all along a system- development process.
- **IEEE defines a requirement as:**
  - A **Condition or capability** needed by a user to solve a **problem or achieve an objective**
  - A **Condition or capability** that **must be met by a system** to satisfy a correct, standard, specification document.
  - A **Document representation** of a condition or capability as in definition (a),(b).

# REQUIREMENT ENGINEERING

- Requirements describe the “what” of a system, not the “how.”
- Requirements engineering **produces one large document**, written in a **natural language**, and contains a description of **what the system will do** without describing how it will do it.
- **Requirements engineering is the systematic use of-**
  - ▢ Proven principles.
  - ▢ Techniques, and language tools.
  - ▢ Cost-effective analysis.
  - ▢ Documentation.
  - ▢ On-going evaluation of the user’s needs.
  - ▢ Specifications of the external behavior of a system to satisfy those user needs.

# REQUIREMENT ENGINEERING

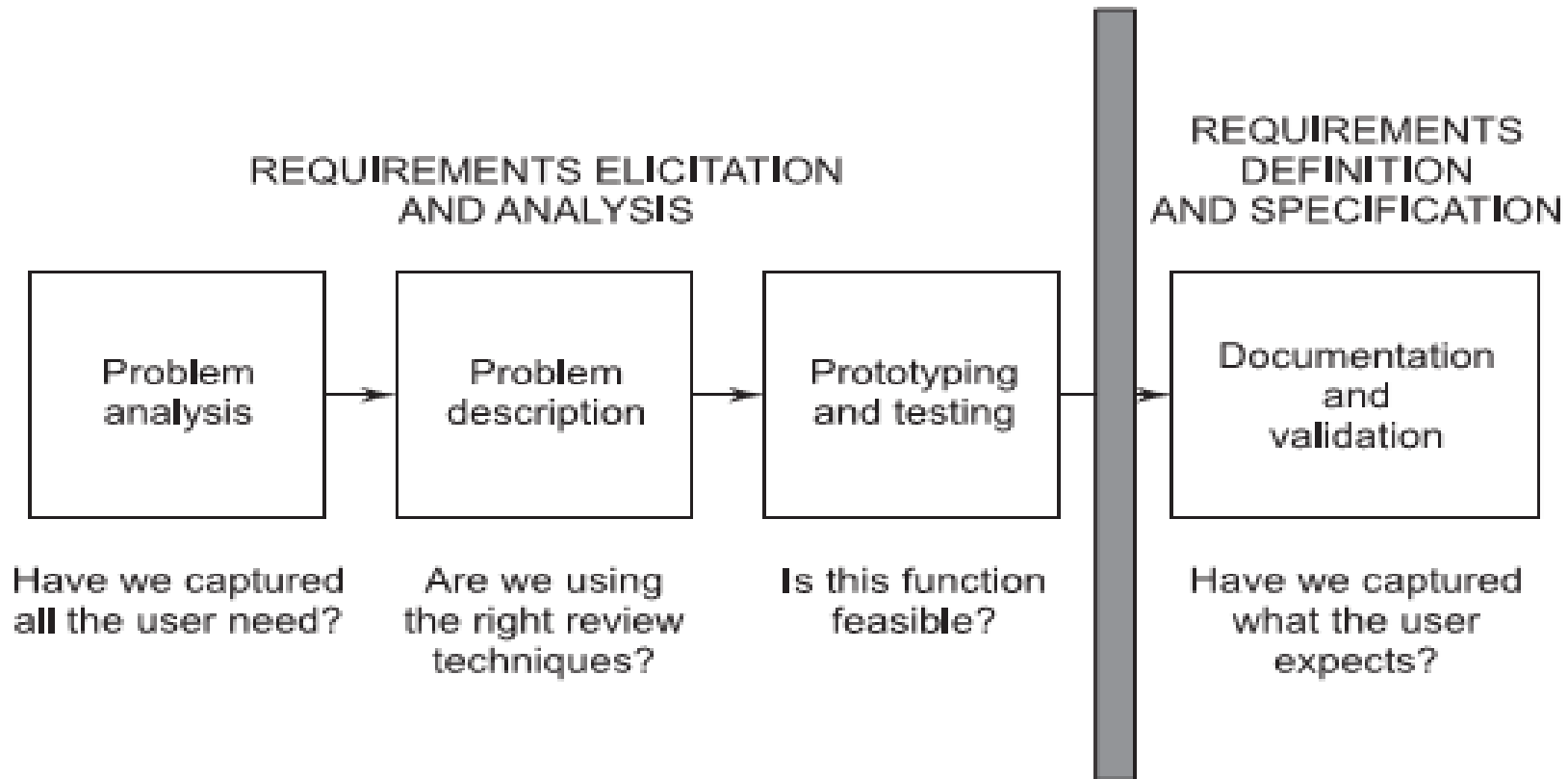


Fig: The Process of Determining Requirements

# REQUIREMENT ENGINEERING

- The **Input** to Requirements Engineering is the **Problem Statement** prepared by the **Customer**.
- The **Output** of the Requirements Engineering (RE) process is a **system Requirements Specification** called the **Requirement Definition and Description (RDD)**.
- The System Requirements Specification forms the basis for designing software solutions.

# REQUIREMENT ENGINEERING TECHNIQUES

- **To solve the requirements problem a wide range of skill sets and knowledge need to applied, few are defined as-**
  - ❑ Psychology and Sociology(**Interviewing, Observing, Video Recording**).
  - ❑ Marketing.
  - ❑ Object Oriented Analysis.
  - ❑ Structured Analysis(**DFD,ERD**).
  - ❑ Participative Design(**Scandinavian Approach**).
  - ❑ Human Computer Interaction.
  - ❑ Quality.



# Types of Requirements

- There are **various categories of the requirements**.
- On the basis of their **priority**, the requirements are classified into the following **three types**-
  - ▣ Those that should be absolutely met.
  - ▣ Those that are highly desirable but not necessary.
  - ▣ Those that are possible but could be eliminated.

# Types of Requirements

- On the **basis of their functionality**, the requirements are classified into the following **two** types:-
  - **Functional requirements:-**
    - ▢ They define factors, such as I/O formats, storage structure, computational capabilities, timing, and synchronization.
  - **Non-functional requirements:-**
    - ▢ They define the **properties or qualities of a product** including usability, efficiency, performance, space, reliability, portability, etc.

# Types of Requirements

## **Functional requirements**

- Statements of services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations

## **Non-functional requirements**

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

### **2.2.1 Functional requirements**

- Describe functionality or system services
- Depend on the type of software, expected users and the type of system where the software is used
- Functional user requirements may be high-level statements of what the system should do; functional system requirements should describe the system services in detail

#### **Examples**

- The user shall be able to search either all of the initial set of databases or select a subset from it
- The system shall provide appropriate viewers for the user to read documents in the document store
- Every order shall be allocated a unique identifier (ORDER\_ID) which the user shall be able to copy to the account's permanent storage area

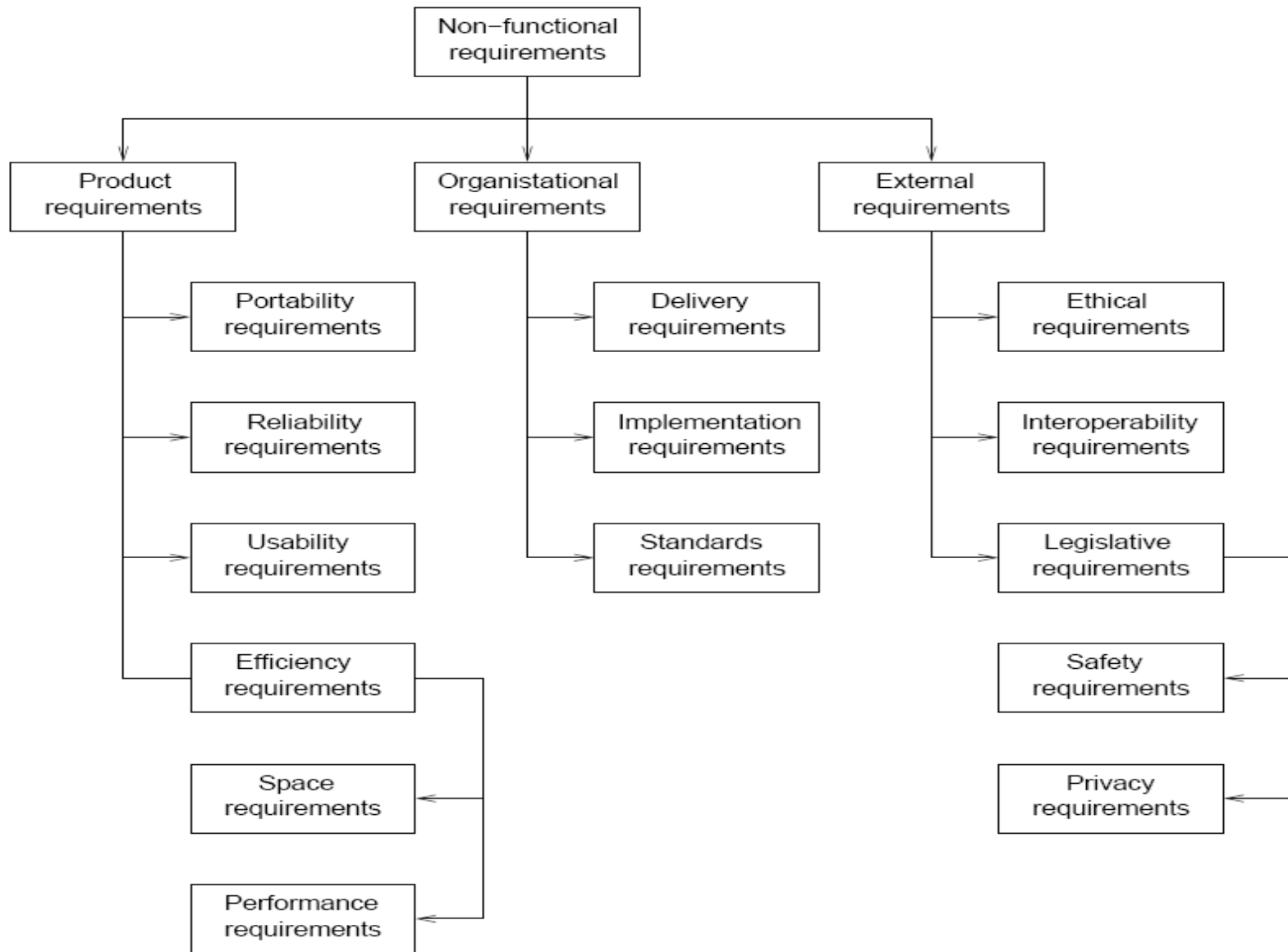


Figure 2.1: Non-functional Requirements

# REQUIREMENTS ENGINEERING PROCESS

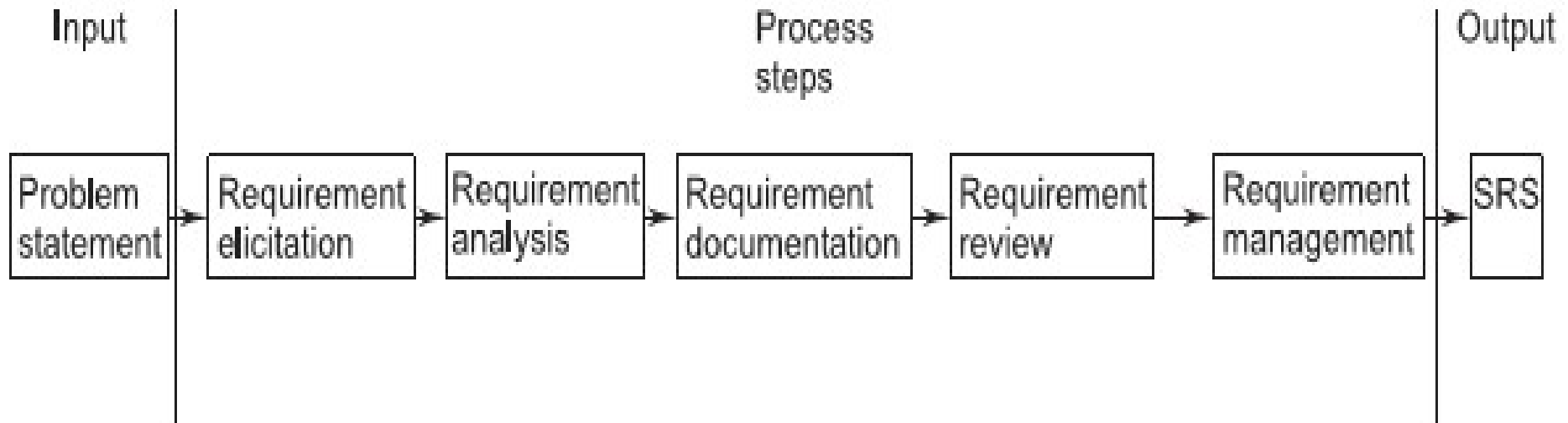
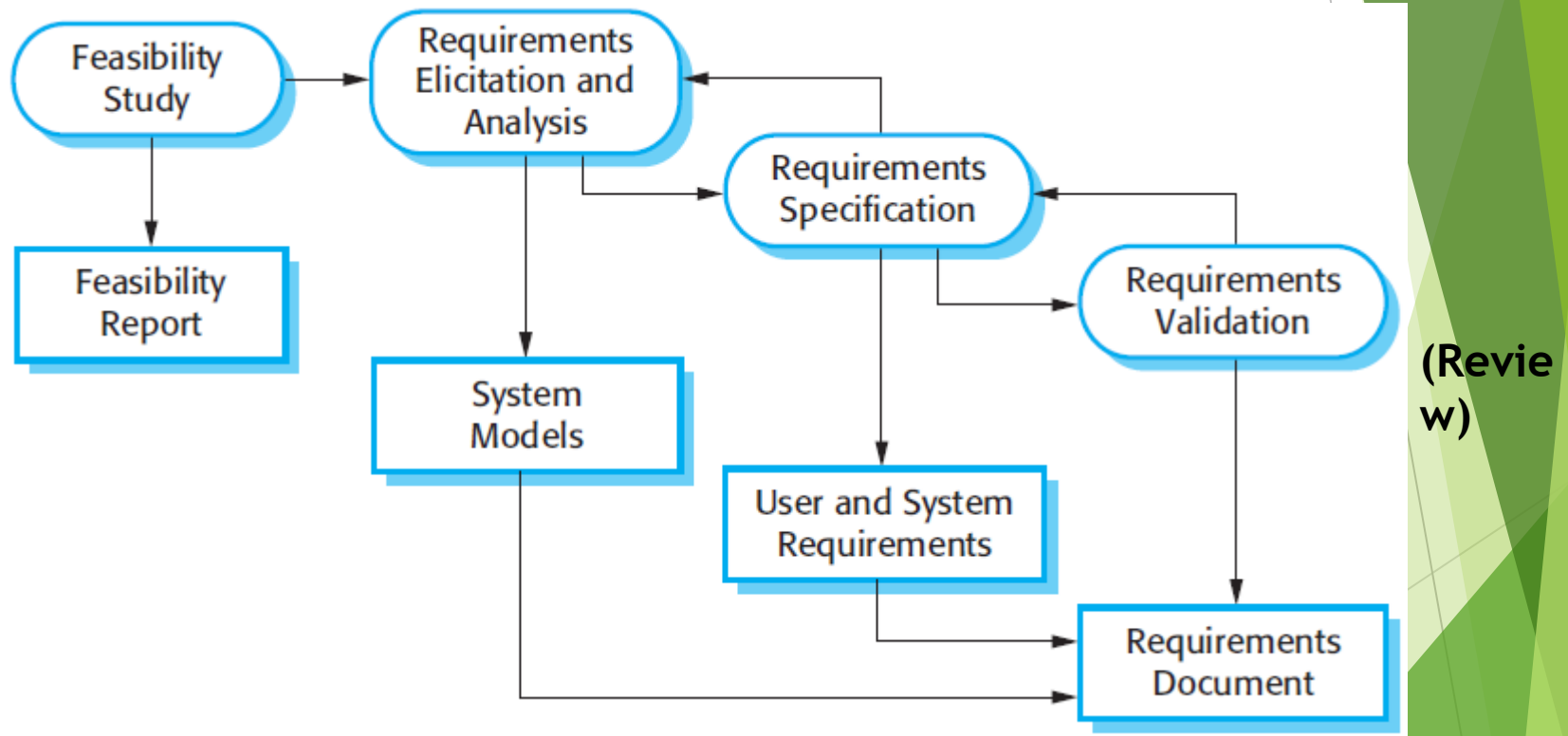


Fig: Process Steps of Requirements Engineering

# REQUIREMENTS ENGINEERING PROCESS



# REQUIREMENTS ENGINEERING PROCESS

- **Requirements engineering consists of the following processes-**
  - A. Feasibility study**
  - B. Requirements Elicitation.**
  - C. Requirements Analysis and Modeling.**
  - D. Requirements Documentation.**
  - E. Requirements Review(Validation).**
  - F. Requirements Management.**



# A. Feasibility Study

## Purpose of feasibility study

“evaluation or analysis of the potential impact of a proposed project or program.”

## Focus of feasibility studies

- Is the product concept viable?
- Will it be possible to develop a product that matches the project's vision statement?
- What are the current estimated cost and schedule for the project?

## Focus of feasibility studies

- How big is the gap between the original cost & schedule targets & current estimates?
- Is the business model for software justified when the current cost & schedule estimate are considered?
- Have the major risks to the project been identified & can they be surmounted?
- Is the specifications complete & stable enough to support remaining development work?
- Have users & developers been able to agree on a detailed user interface prototype? If not, are the requirements really stable?
- Is the software development plan complete & adequate to support further development work?

Feasibility depends upon non technical Issues like:

- Are the project's cost and schedule assumption realistic?
- Does the business model realistic?
- Is there any market for the product?

Feasibility study can be divided into following:

1. Economic feasibility
2. Technical feasibility
3. Legal feasibility
4. Organizational feasibility (man power resources and work culture of organization)

## B-Requirements Elicitation.

- Requirement Elicitation is a **Communication Process** between the parties involved and affected in the problem situation.
- The **Tools in elicitation are-** meetings, interviews, video conferencing, e-mails, and existing documents study and facts findings.
- More than **90% to 95% elicitation** should be complete in the **initiation stage** while the **remaining 5%** is completed **during the development life-cycle**.

# Requirements Elicitation

- The Requirements are gathered from various sources for elicitation.
- **The Sources are-**
  - Customer (Initiator).
  - End Users.
  - Primary Users.
  - Secondary Users.
  - Stakeholders.

# Requirement Elicitation Process

- **General Techniques that used in Requirement Elicitation Process are as follows:**

1. **Interviews.**
2. **Brainstorming.**
3. **Facilitated Application Specification Techniques(FAST).**
4. **Joint Application Design(JAD).**
5. **Task Analysis.**
6. **Quality Function Deployment.**
7. **Form Analysis.**
8. **Delphi Techniques.**
9. **User Scenario**

# 1-Interviews

- Once a Problem statement is received from a customer, then this stage comes, where the meeting with a customer is arranged.
- Interview is a one of the most popular method for understanding the problem domains.
- During interview requirement engineers and customers interact with each other.
- Requirement engineer must be open minded and asked the question freely from customer.



# Interviews

## **The following steps in interviews are-**

- ❑ Create the Questionnaire.
- ❑ Select the Interviewer.
- ❑ Plan Contacts.
- ❑ Conduct the interview(face to face).

## 2-Brainstorming

- It is used many in business application, is a group of techniques to promote creative thinking and used during elicitation process to generate new ideas.
- Promote the creative thinking by group discussions.
- It becomes very popular and it is widely used by many companies.
- There will not be critic for the ideas.

# Brainstorming

- Roles for a Brainstorming Session-
  - ▣ Leader-(facilitator or moderator).
  - ▣ Scribe(penman) (*one who writes; a draughtsman*)
  - ▣ Participant(involved many stakeholders).

### 3-Facilitated Application Specification Techniques(FAST).

- This technique was specifically for collecting requirements **similar to brainstorming.**
- The objective of this technique is to **bridge the expectation gap**- a difference between what developers think they are suppose to build and what customers think they are going to get.
- So for **reducing the expectation gap**, a team oriented approach is developed for requirements gathering and is called the FAST.

# 4-Joint Application Development (JAD)

- ❑ **Joint application design (JAD)** is a process used in the prototyping life cycle area of the Dynamic Systems Development Method (DSDM) to collect business requirements while developing new information systems for a company.
- ❑ The JAD process also includes approaches for enhancing user participation, expediting development, and improving the quality of specifications.
- ❑ It consists of a workshop where knowledge workers and IT specialists meet, sometimes for several days, to define and review the business requirements for the system.
- ❑ The attendees include high level management officials who will ensure the product provides the needed reports and information at the end.

# 4-Joint Application Development (JAD)

## Key participants-

- **Executive Sponsor:** The executive who charters the project, the system owner. They must be high enough in the organization to be able to make decisions and provide the necessary strategy, planning, and direction.
- **Subject Matter Experts:** These are the business users, the IS professionals, and the outside experts that will be needed for a successful workshop. This group is the backbone of the meeting; they will drive the changes.
- **Facilitator/Session Leader:** meeting and directs traffic by keeping the group on the meeting agenda. The facilitator is responsible for identifying those issues that can be solved as part of the meeting and those which need to be assigned at the end of the meeting for follow-up investigation and resolution. The facilitator serves the participants and does not contribute information to the meeting.
- **Scribe/Modeler/Recorder/Documentation Expert:** Records and publish the proceedings of the meeting and does not contribute information to the meeting.
- **Observers:** Generally members of the application development team assigned to the project. They are to sit behind the participants and are to silently observe the proceedings.

## 5-Task Analysis.

- The problem domain is decomposed into hierarchy of task and subtask.
- The customer approaches to engineer and express his desire of what software should contribute to the organization ,how the software should work, what services should provide and so on.

## -Quality Function Deployment.

- This is a technique of **Quality Management** that helps to incorporate the voice of customers.
- Transform the voice of the customer [VOC] into engineering characteristics for a product.
- technical requirement are documented and result in the software Requirements Specification(SRS) documents.
- And these requirement further translated in to design requirements.
- Customer requirement is prime concerns of this techniques.



## 7-Form Analysis

- Form play an important role in any organization and contain lot of meaningful information about data objects of the domain as shown in the employee registration form.

### **Employee Registration Form**

**Employee Name:**

**Address:**

**Position:**

**Year of**

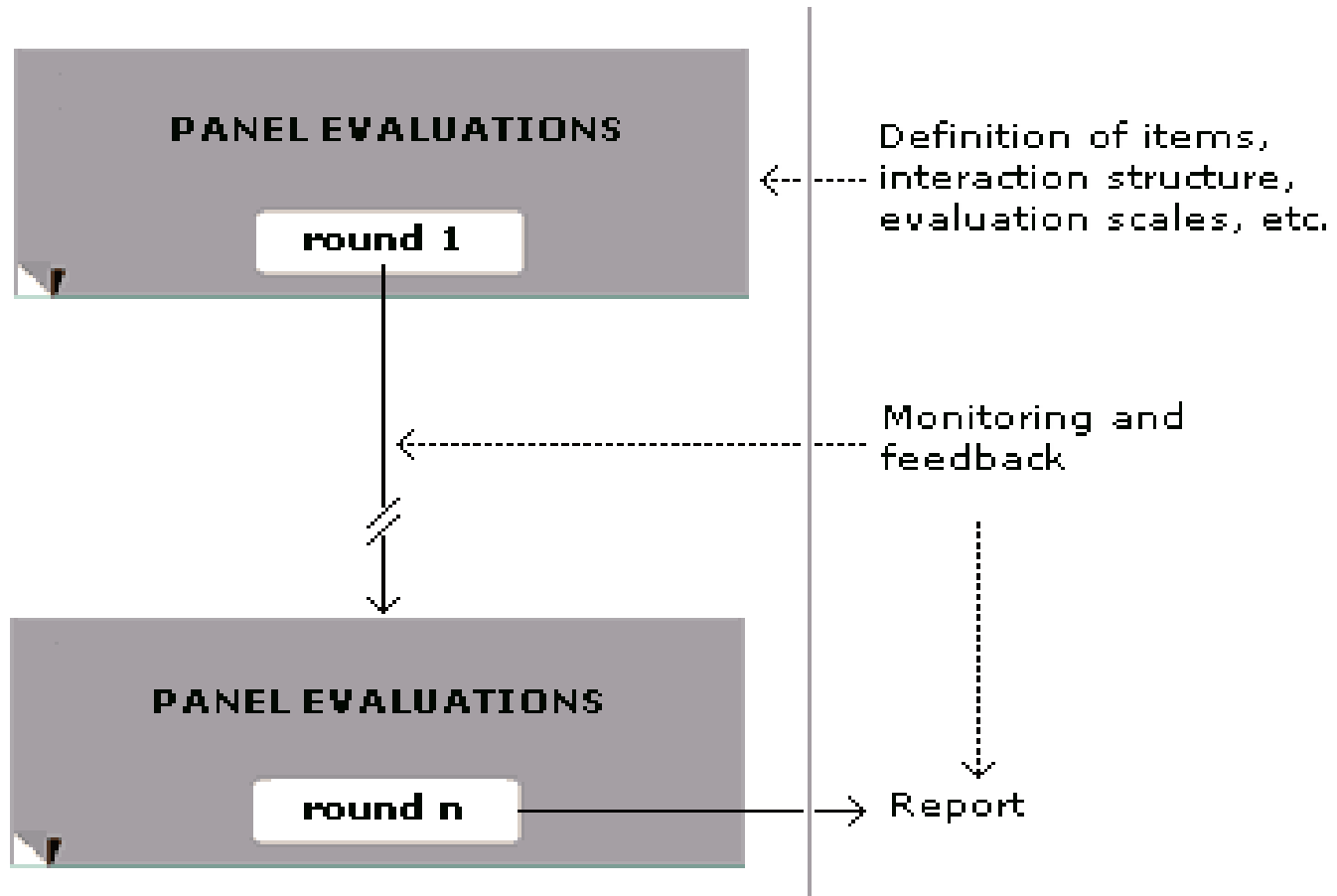
**joining:**

**Salary:**

# 8-Delphi Technique

- The **Delphi method** is a **structured communication technique**, originally developed as a **systematic, interactive forecasting method** which relies on a **panel of experts**.
- The **experts** answer questionnaires in two or more rounds.
- After **each round**, a **facilitator provides** an **anonymous summary of the experts' forecasts** from the **previous round** as well as the **reasons** they provided for their judgments.
- Thus, **experts are encouraged** to revise their **earlier answers** in **light of the replies of other members of their panel**.
- It is believed that during this process the **range of the answers will decrease** and the group will converge towards the "**correct**" answer.
- Finally, the process is stopped after a **pre-defined stop criterion**.
- Delphi is based on the principle that **forecasts (or decisions)** from a structured groups or unstructured groups of individuals.

# 8-Delphi Technique



## 9-User Scenario(With Help of use case diagram)

- This technique use for capturing requirements from
  - ▶ point of view.
- A scenario is a story that illustrates how a perceived
  - ▶ system will satisfy a user needs.

# C-Requirement Analysis

- Requirement analysis is a very important and essential activity after elicitation.
- In this phase, each requirement is analyzed from the point-of-view of validity, consistency, and feasibility for consideration in the RDD and then in the SRS.
- Validity confirms its relevance to goals and objectives and consistently confirms that it does not conflict with other requirements but supports others where necessary.

# Requirement Analysis

- The Second portion of Analysis attempts to find for each requirement, its **functionality, features, and facilities** and the need for these under different conditions and constraints-
  - ▣ **Functionality**-“How to achieve the requirement goal.”
  - ▣ **Features**- Describe the attributes of functionality,
  - ▣ **Facilities**-Provide its delivery, administration, and communication to other systems.

# Requirement Analysis

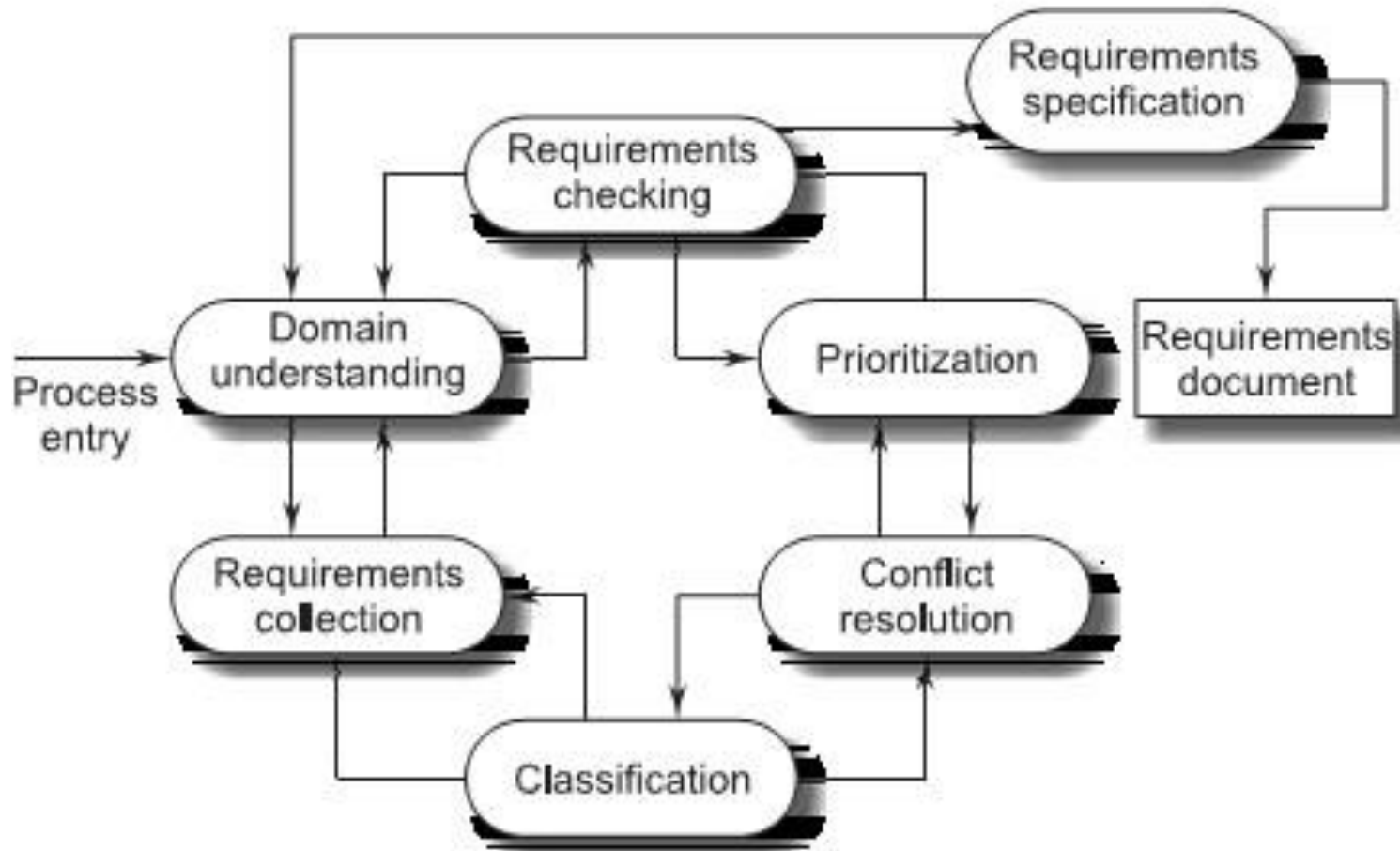
- **Following question should be made understandable-**
  - ❑ What is the problem.
  - ❑ Why it is important to solve the problem?
  - ❑ What are the possible solutions to the problem?
  - ❑ What are the data input the system?
  - ❑ What might complexities that arise while solving the problem?

# Process Model of Elicitation and Analysis

- Each organization will have its own version or representation .
- This general model depending on local factors, such as the expertise of the staff, the type of system being developed, the standards used, etc.
- A generic process model of the elicitation and analysis process is shown as in fig.



# Requirements Elicitation and Analysis Process Model



# D-Requirements Documentation

- Requirements documentation is a very important activity, which is written after the requirements elicitation and analysis.
- This is the way **to represent requirements in a consistent format.**
- The requirements document is called the Software Requirements Specification (SRS).

# Requirements Documentation

- The SRS is a specification for a particular software product, program, or set of programs that perform certain functions in a specific environment.
- It **serves a number of purposes** depending on **who is writing it-**
  - ▣ **First**, the SRS could be written **by the customer** of a system. in this SRS is used to **define the needs and expectations of the users**.
  - ▣ **Second**, the SRS could be written **by a developer** of the system. It serves as a **contract document** between **customer and developer**.

# Requirements Documentation

- Thus, Requirements must be written so they are **meaningful not only to the customers but also to designers on the development team.**
- This Requirements definition document (RDD) describes **what the customer would like to see** in this-
  - **First**, we outline the general purpose of the system.
  - **Next**, we describe the background and objectives of system development.
  - If the customer has a proposed new approach to solving the problem, we outline a description of the approach.
  - Once we record this overview of the problem, we describe the detailed characteristics of the proposed system.
  - Finally, we discuss the environment in which the system will operate.

# E-Requirements Review

- A Requirements review is a **manual process**, which involves **multiple readers from both client and developer side**, checking the requirements document for anomalies and omissions.
- A requirements review can be **Informal or Formal**.
  - ▣ **Informal Review-** informal reviews simply **involve developers** discussing requirements with as many system **stakeholders** as possible. In this review there is **no conformation** that the documented requirements are what the stakeholders really said they wanted.

# Requirements Review

- **Formal Review-** in this development team should „walk“ to the client through the system requirements, explaining the implications of each requirement.
- The review team should check each requirement for consistency and should check the requirements as a whole for completeness. Reviewers may also check for-
  - ❑ **Verifiability:** are the requirements as stated realistically testable?
  - ❑ **Comprehensibility:** is the requirement property understood by the procurers or end-users of the system?
  - ❑ **Traceability:** is the origin of the requirement clearly stated? You may have to go back to the source of the requirement to assess the impact of a change
  - ❑ **Adaptability:** is the requirement adaptable?

# Requirements Review

- Conflicts, contradictions, errors, and omissions in the requirements should be pointed out during the review and formally recorded.
- It is then up to the users, the system developer to negotiate a solution to these identified problems.

# F-Requirements Management

- Requirements define the capability that the software system solution must deliver and the intended results that must result on its application to business problems.
- In order to generate such requirements, a systematic approach is necessary, through a formal management process called Requirements Management.
- Requirements management is defined as a systematic approach to eliciting, organizing, and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and project team on the changing requirements of the system.



# Classes of Requirements Management

- Classes of Management requirements fall into **two classes-**
  - ▣ **Enduring Requirements**-These are relatively **stable requirements** which derive from the core activity of the organization and which relate directly to the domain of the system.
  - ▣ **Volatile Requirements**-These are requirements which are **likely to change during the system development** or after the system has been put into operation.

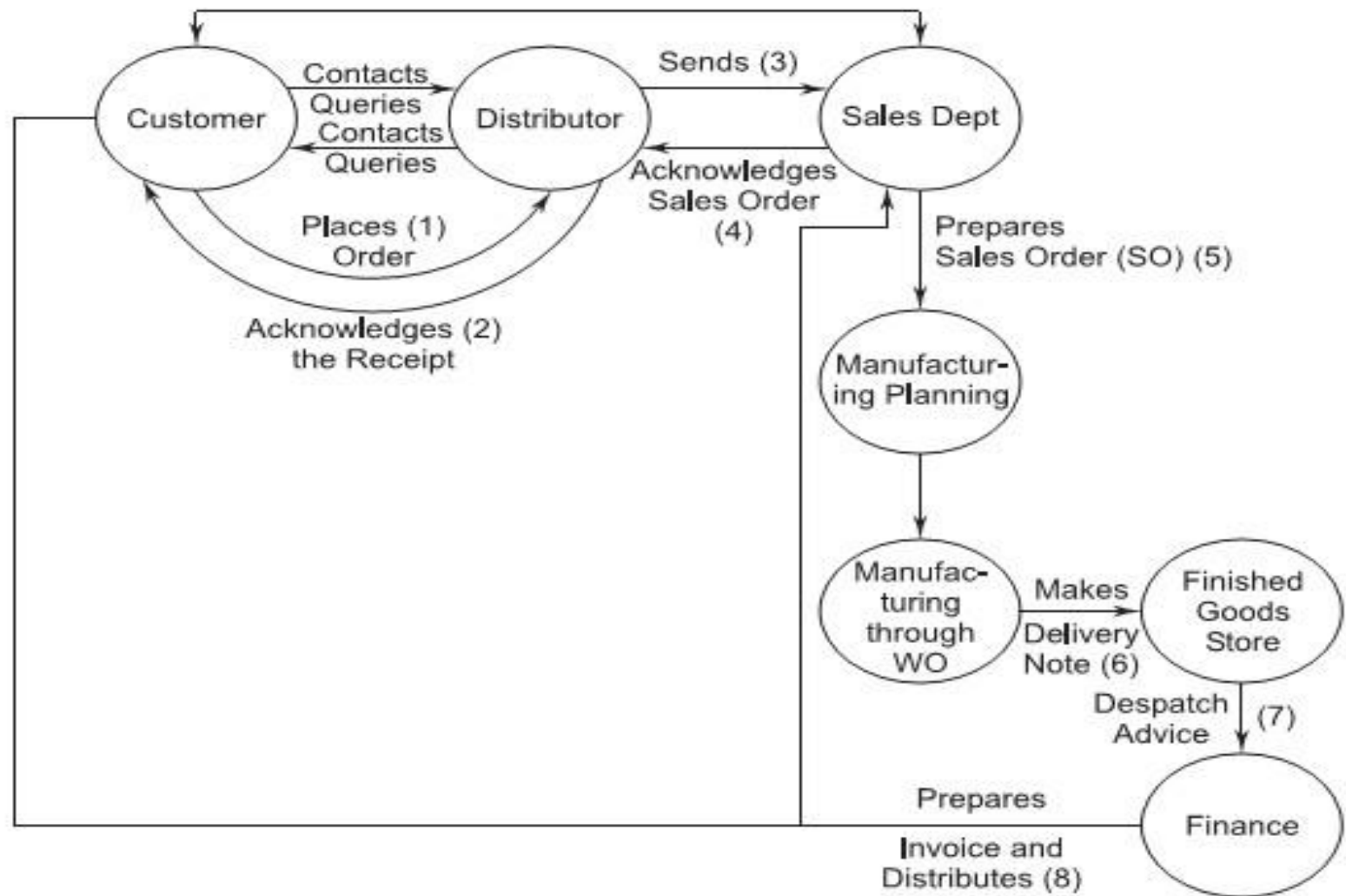
# Requirements Management Planning

- Requirements management is **very expensive** and, for each project, the planning stage establishes the level of requirements management detail required.
- During the **requirements management stage**, you have to decide on-
  - ❑ **Requirements identification.**
  - ❑ **A change management process.**
  - ❑ **Traceability policies.**
  - ❑ **CASE tool support.**

# INFORMATION MODELING

- The Information Flow Model (IFM) is used to understand the **sources and destination of information flow**, which is **required to execute the business process**.
- IFM is generally a **high-level model** showing **main flows, internal flows of information from sources**, such as product catalogs, and manufacturing schedules.

# INFORMATION MODELING



# SA/SD METHODOLOGY

- SA/SD methodology consist of two activities-
  - ▣ Structural Analysis(SA)
  - ▣ Structural Design(SD)
- The aim of SA/SD is to transform the textual description of a problem in to a graphical model.
- During SA/SD the functional decomposition of the system takes place.

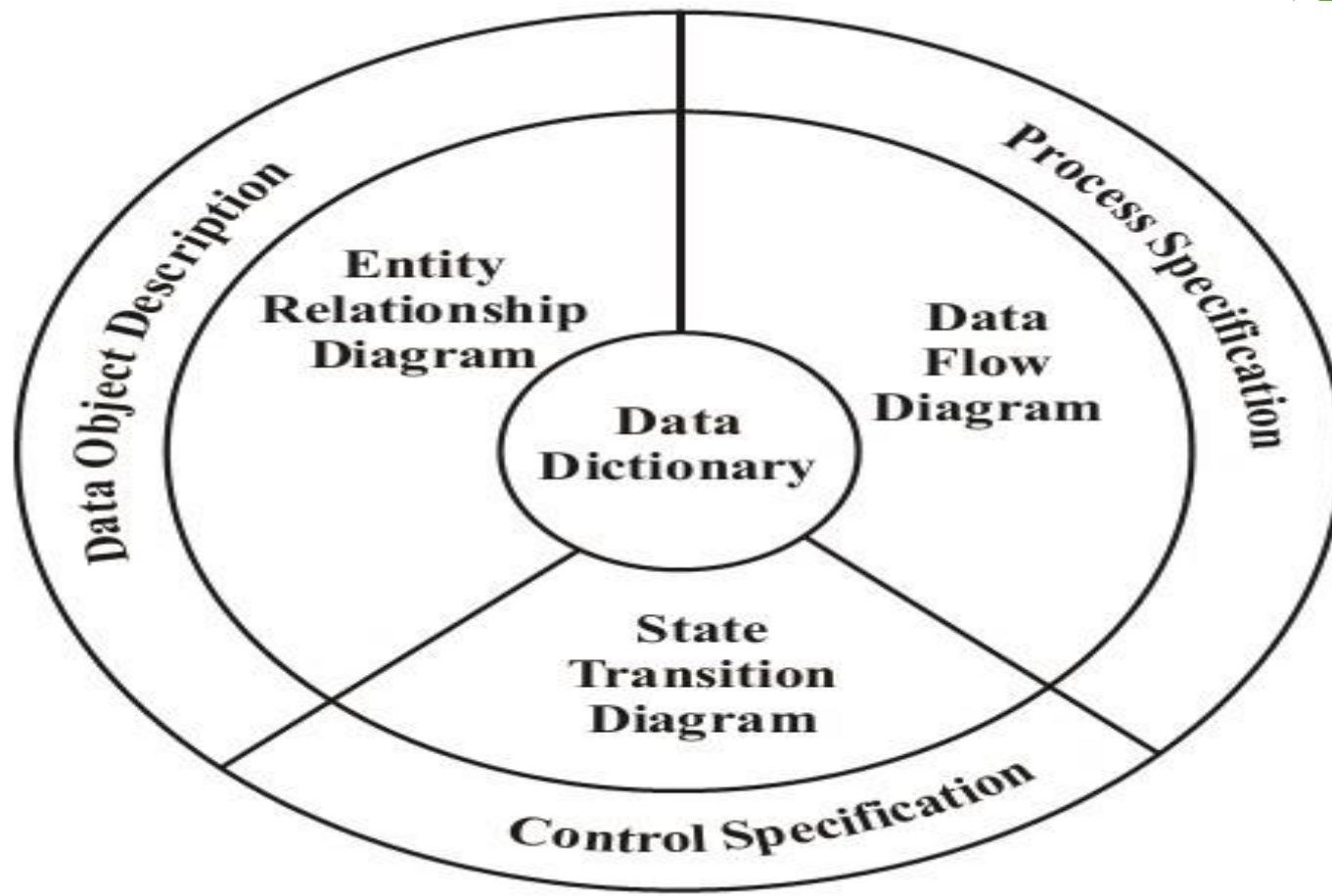
# Tools For SA

- **Data Flow Diagram(DFD)**
- **Event List**
- **Data Dictionary**
- **Process Specification**
- **Entity Relationship Diagram**
- **State Transition Diagram**

# Tools for SD

- **Structure charts**

# The Structure of the Analysis Model



# DATA-FLOW DIAGRAMS

- Data-Flow Diagrams (DFD) are also known as data-flow graphs or bubble charts.
- A DFD serves the purpose of clarifying system requirements and identifying major transformations.
- DFDs show the flow of data through a system.
- It is an important modeling tool that allows us to picture a system as a network of functional processes.
- Data-flow diagrams are well-known and widely used for
  - ▶ specifying the functions of an information system.
- They describe systems as collections of data that are manipulated by functions.

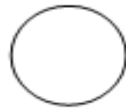


- Data can be organized in several ways:
  - ▢ they can be stored in data repositories, they can flow in data flows, and they can be transferred to or from the external environment.
  - ▢ One of the reasons for the success of DFDs is that they can be expressed by means of an attractive graphical notation that makes them easy to use.

# Symbols Used for Constructing DFDs

- There are different types of symbols used to construct DFDs.
- The meaning of each symbol is explained below:

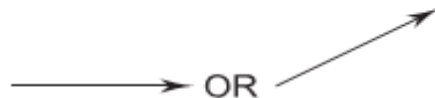
1. **Function symbol.** A function is represented using a circle. This symbol is called a process or a bubble and performs some processing of input data.



2. **External entity.** A square defines a source or destination of system data. External entities represent any entity that supplies or receives information from the system but is not a part of the system.



3. **Data-flow symbol.** A directed arc or arrow is used as a data-flow symbol. A data-flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.



# Symbols Used for Constructing DFDs

4. *Data-store symbol.* A data-store symbol is represented using two parallel lines. A logical file can represent either a data-store symbol, which can represent either a data structure, or a physical file on disk. Each data store is connected

to a process by means of a data-flow symbol. The direction of the data-flow arrow shows whether data is being read from or written into a data store.



5. *Output Symbol.* It is used to represent data acquisition and production during human-computer interaction.

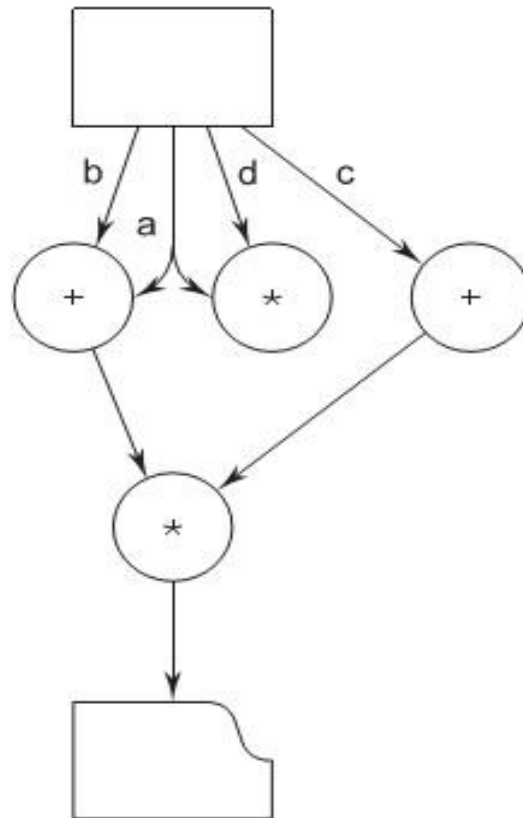


## Example DFD

**Example 3.1.** Figure 3.6 shows how the symbols can be composed to form a DFD. The DFD describes the arithmetic expression

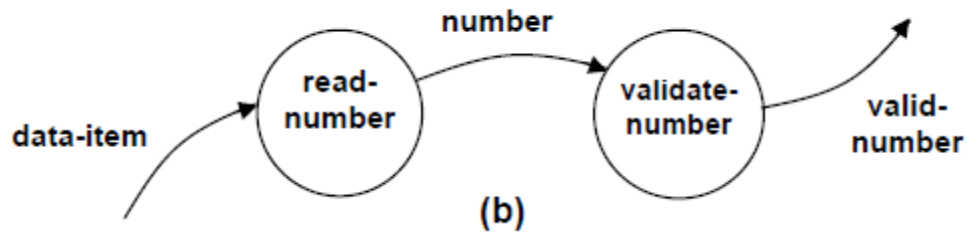
$$(a + b) * (c + a * d)$$

assuming that the data  $a$ ,  $b$ ,  $c$ , and  $d$  are read from a terminal and the result is printed. The figure shows that the arrow can be “forked” to represent the fact that the same datum is used in different places.

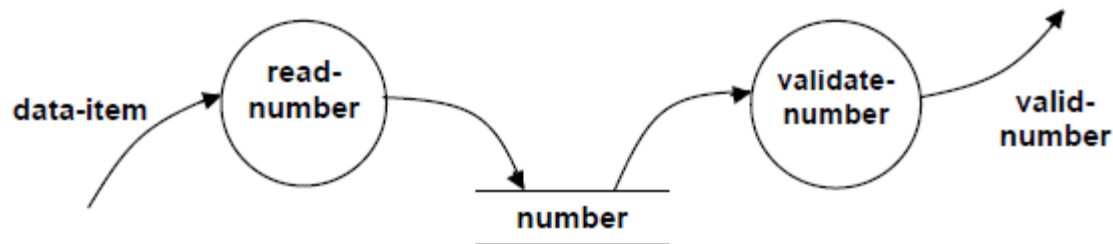


# General Guidelines and Rules for Constructing DFDs

## 1. Synchronous and Asynchronous Operations



**Synchronous(no time delay)**



**Asynchronous(time delay)**

## ► 2. Data Dictionary

- The data dictionary is used to create and store definitions of data, location, format for storage and other characteristics.
- The data dictionary can be used to retrieve the definition of data that has already been used in an application.
- The data dictionary also stores some of the description of data structures, such as entities, attributes and relationships.

- For e.g.

`grossPay = regularPay + overtimePay`

`, a:{integer}`  
etc.

## Symbols

**used:** denotes composition of two data items, e.g. `a+b` represents data `a` and `b`.

`[ , ]`: represents selection, i.e. any one of the data items listed in the brackets can occur. For example, `[a,b]` represents either `a` occurs or `b` occurs.

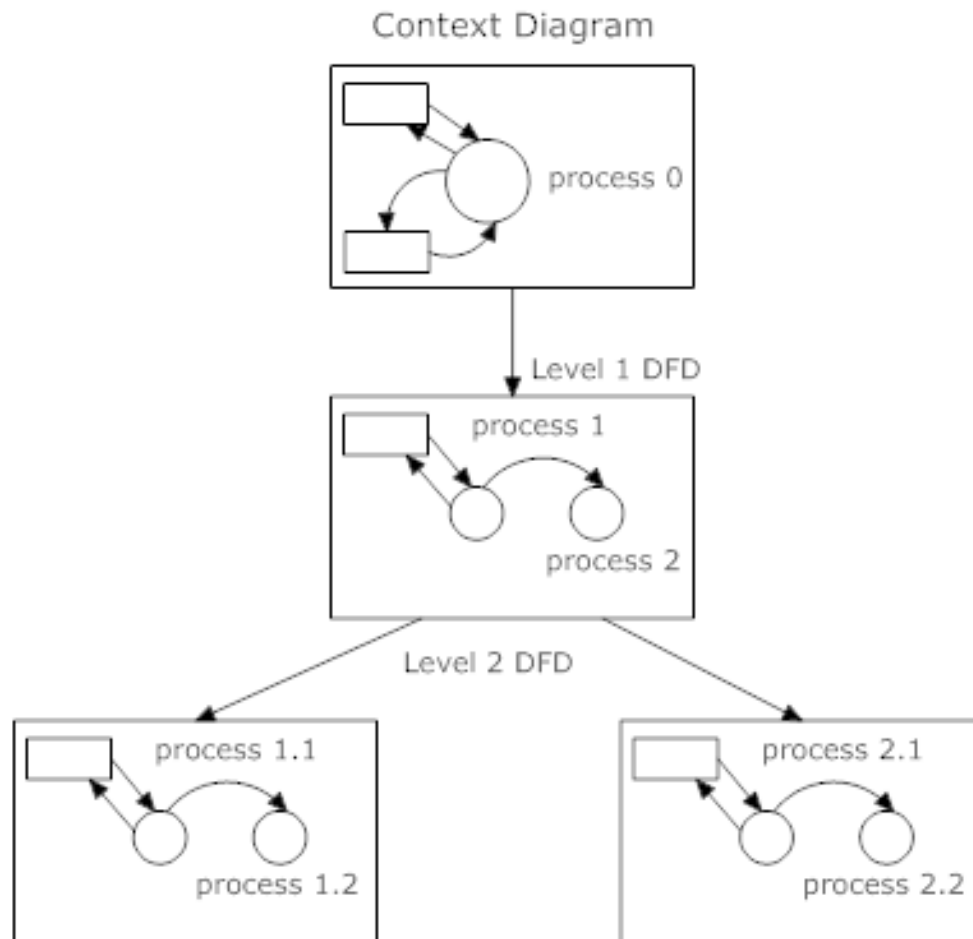
`()`: the contents inside the bracket represent optional data which may or may not appear. e.g. `a+(b)` represents either `a` occurs or `a+b` occurs.

`{ }`: represents iterative data definition, e.g. `{name}5` represents five `name` data. `{name}*` represents zero or more instances of `name` data.

`=`: represents equivalence, e.g. `a=b+c` means that `a` represents `b` and `c`.

`/* */`: Anything appearing within `/*` and `*/` is considered as a comment.

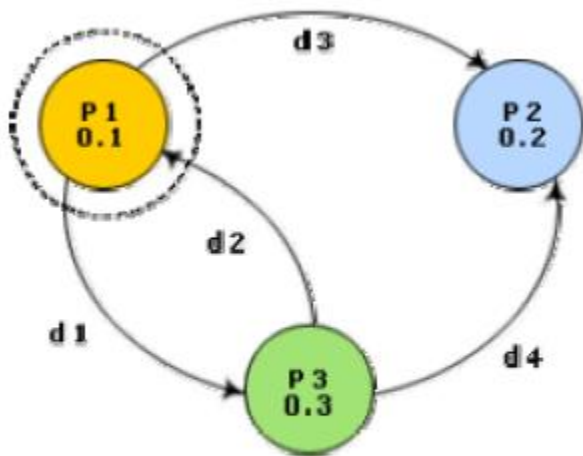
### ► 3. Levels of DFD Model



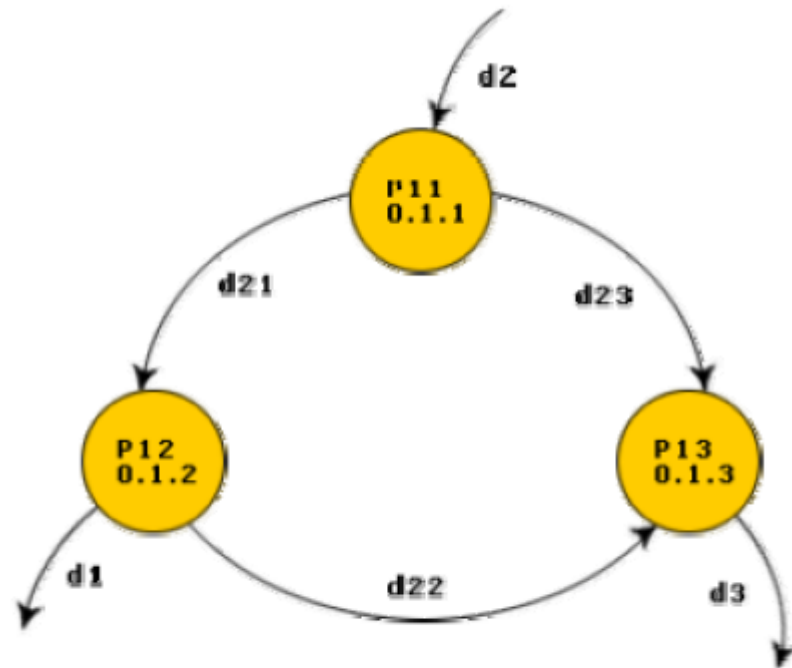


#### ► 4. Balancing of DFD

The data that flow into or out of a bubble must match the data flow at the next level of DFD. This is known as balancing a DFD.



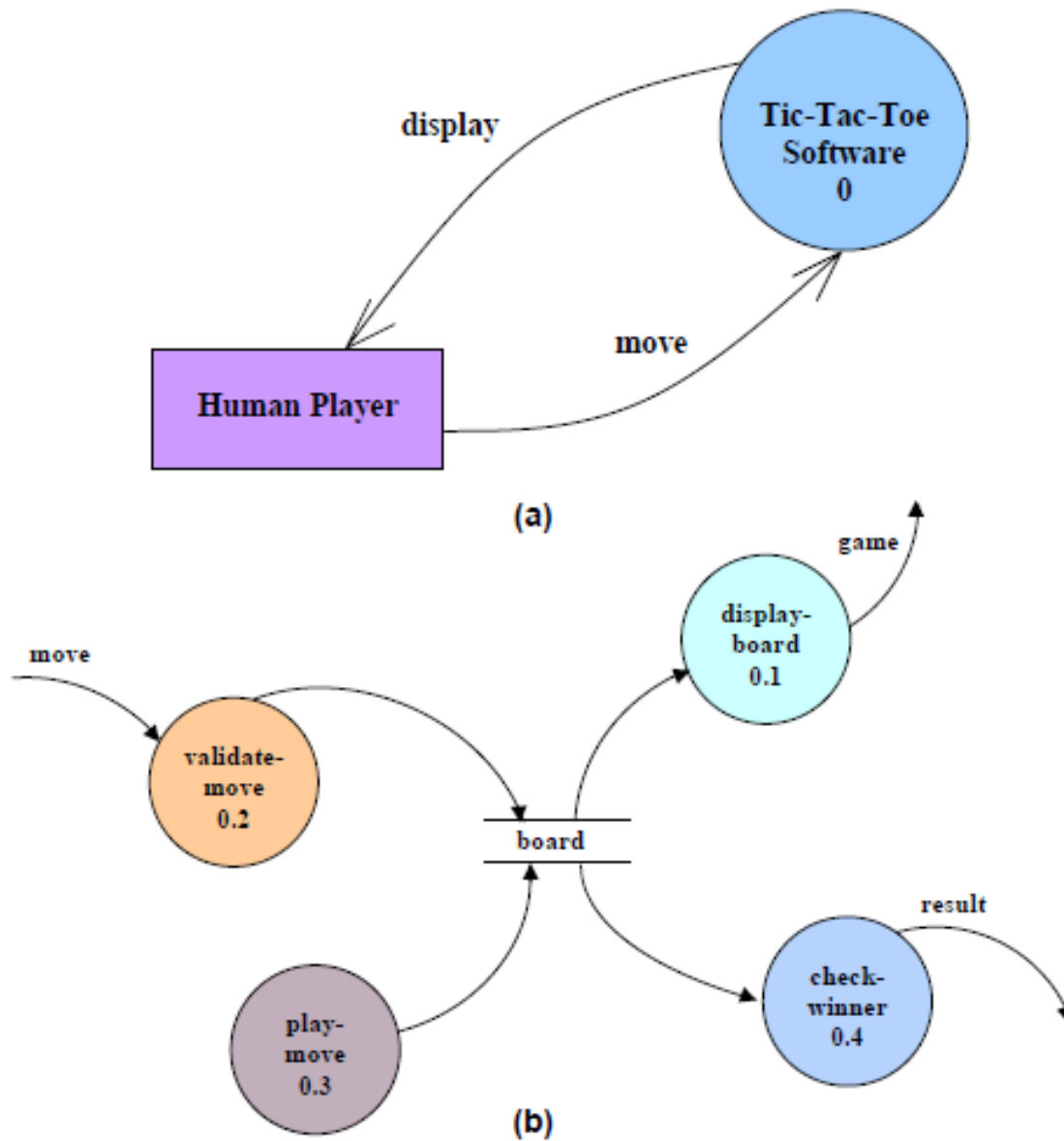
(a) Level 1 DFD



(b) Level 2 DFD

## Example 1: Tic-Tac-Toe Computer Game

Tic-tac-toe is a computer game in which a human player and the computer make alternative moves on a  $3 \times 3$  square. A move consists of marking previously unmarked square. The player who first places three consecutive marks along a straight line on the square (i.e. along a row, column, or diagonal) wins the game. As soon as either the human player or the computer wins, a message congratulating the winner should be displayed. If neither player manages to get three consecutive marks along a straight line, but all the squares on the board are filled up, then the game is drawn. The computer always tries to win a game.



**Fig 5.2 (a), (b)** Level 0 and Level 1 DFD for Tic-Tac-Toe game described in Example 1

## Data dictionary for the DFD model in Example 1

```
move:           integer /*number between 1 and 9 */
display:        game+result
game:           board
board:          {integer}9
result:         ["computer won", "human won" "draw"]
```

# Context Diagram

- A context diagram shows:
  - ▣ Data input to the system,
  - ▣ Output data generated by the system,
  - ▣ External entities.

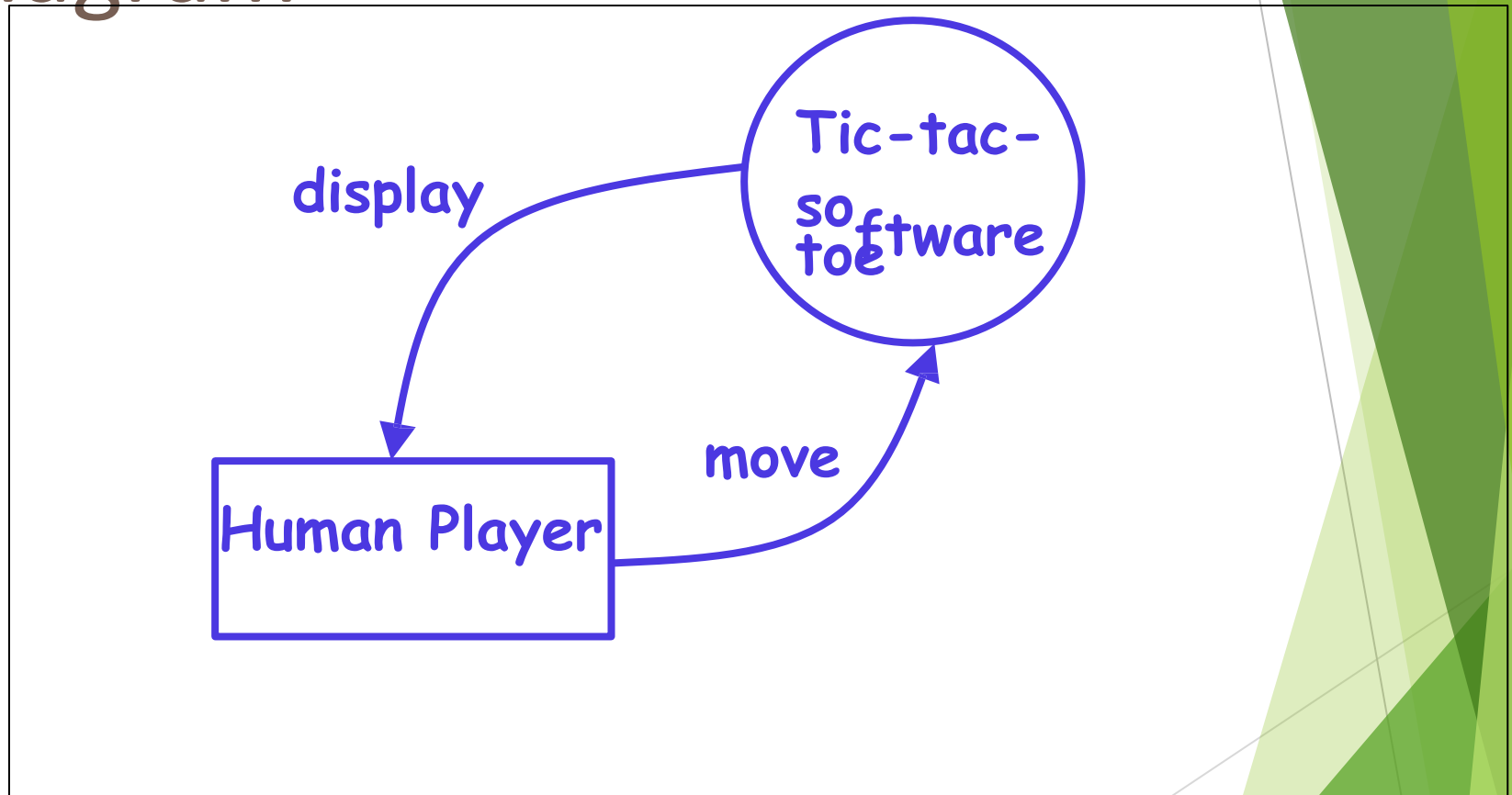
# Context Diagram

- Context diagram captures:
  - ▣ Various entities external to the system and interacting with it.
  - ▣ Data flow occurring between the system and the external entities.
- The context diagram is also called as the level 0 DFD.

# Context Diagram

- Establishes the context of the system, i.e.
  - Represents:
    - Data sources.
    - Data sinks.

# Tic-tac-toe: Context Diagram

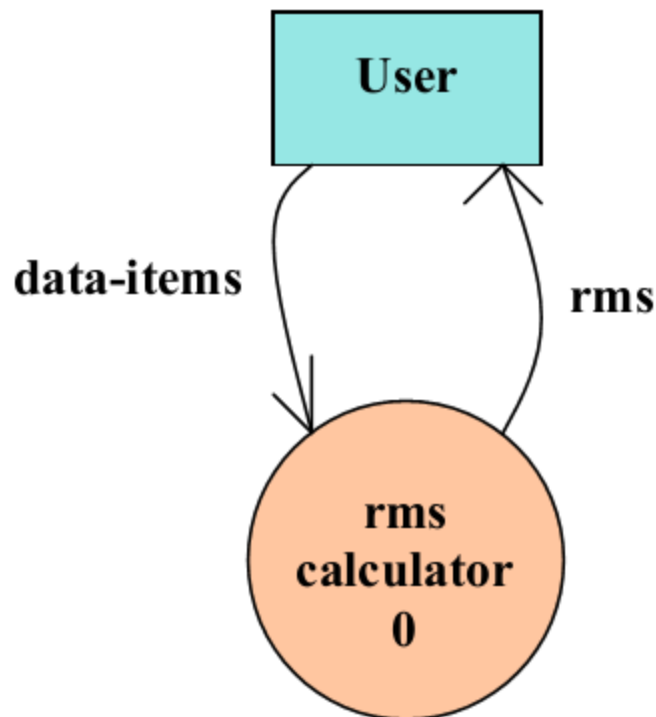




## Example 2: RMS Calculating Software.

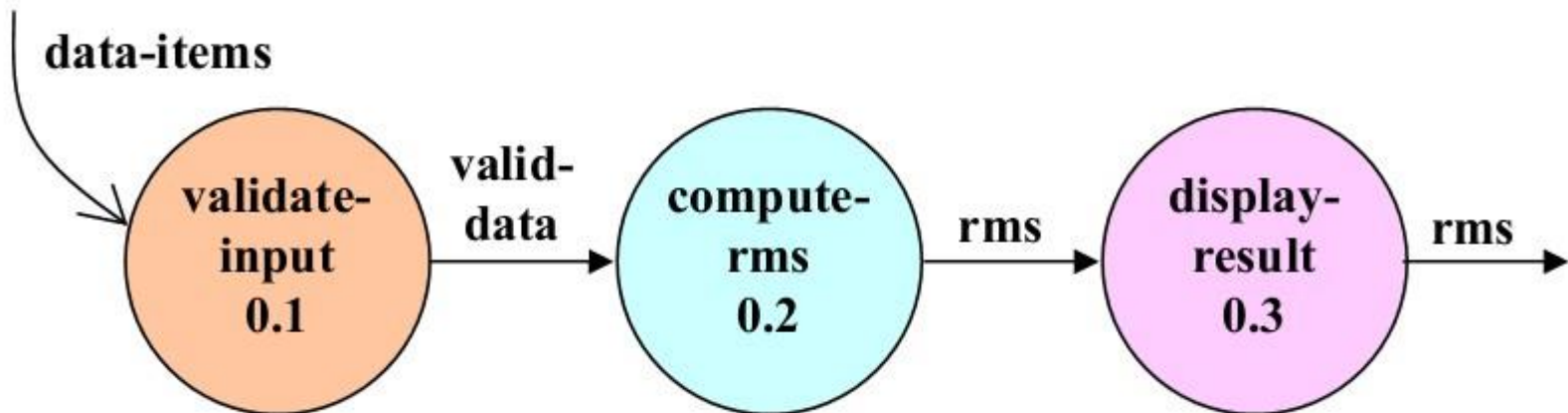
A software system called RMS calculating software would read three integral numbers from the user in the range of -1000 and +1000 and then determine the root mean square (rms) of the three input numbers and display it.

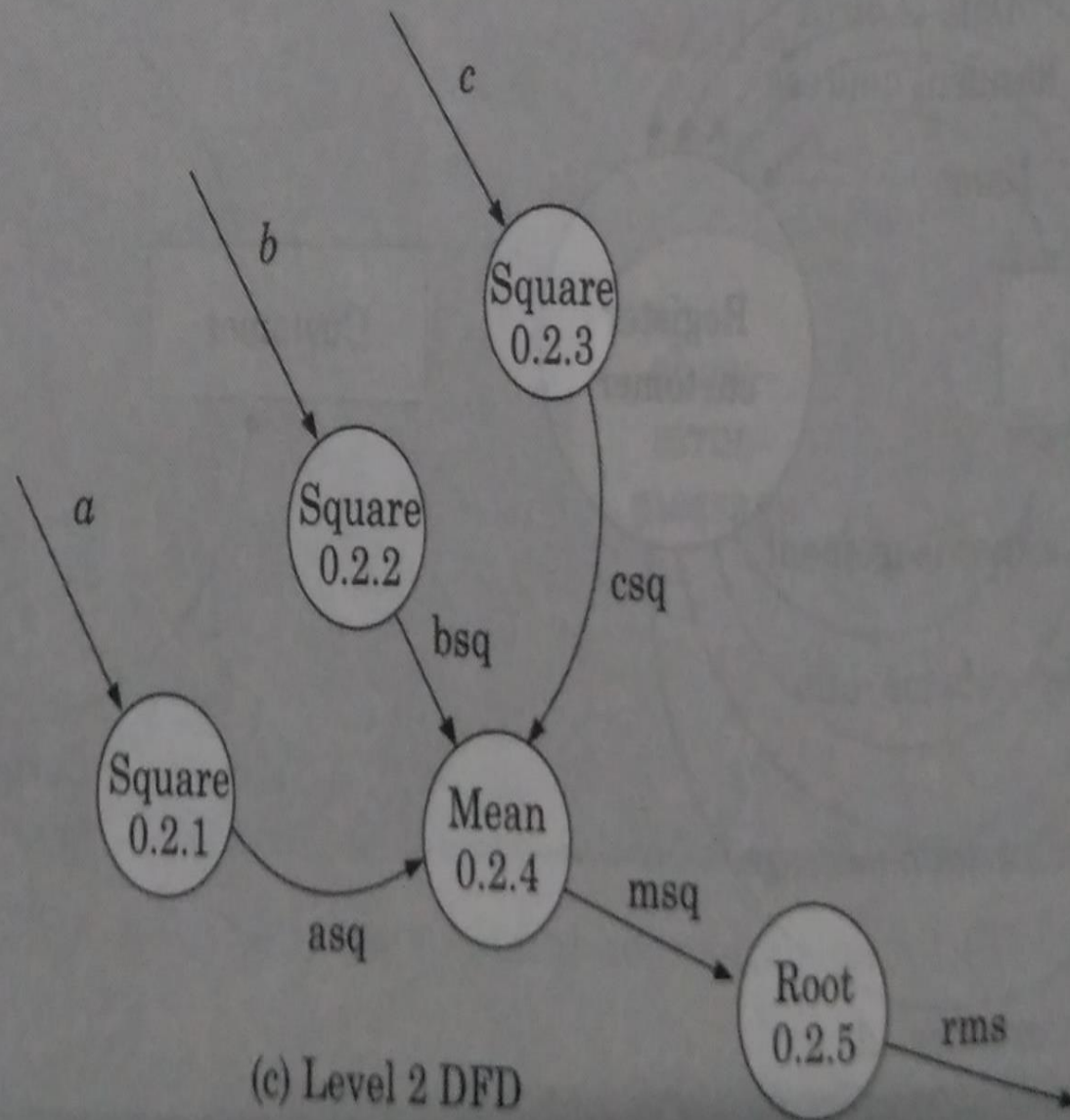
The software reads the numbers from the user and returns the result to him.



Level 0 / Context diagram

## Level 1





(c) Level 2 DFD

## Data dictionary for the DFD model of Example 6.1

data-items: {integer}3

rms: float

valid-data: data-items

a: integer

b: integer

c: integer

asq: integer

bsq: integer

csq: integer

msq: integer

# Example 3: Trading-House Automation System (TAS)

- A large trading house wants us to develop a software:
  - ▣ To automate book keeping activities associated with its business.
- It has many regular customers:
  - ▣ Who place orders for various kinds of commodities.
- The trading house maintains names and addresses of its regular customers.
- Each customer is assigned a unique customer identification number (CIN).
- As per current practice when a customer places order:
  - ▣ The accounts department first checks the credit-worthiness of the customer.

# Example: Trading-House Automation System (TAS)

- The credit worthiness of a customer is determined:
  - By analysing the history of his payments to the bills sent to him in the past.
- If a customer is not credit-worthy:
  - His orders are not processed any further
  - An appropriate order rejection message is generated for the customer.

# Example: Trading-House Automation System (TAS)

- If a customer is credit-worthy:
  - ▣ Items he/she has ordered are checked against the list of items the trading house deals with.
- The items that the trading house does not deal with:
  - ▣ Are not processed any further
  - ▣ An appropriate message for the customer for these items is generated.

# Example: Trading-House Automation System (TAS)

- The items in a customer's order that the trading house deals with:
  - ▣ Are checked for availability in inventory.
- If the items are available in the inventory in desired quantities:
  - ▣ A bill with the forwarding address of the customer is printed.
  - ▣ A material issue slip is printed.



# Example: Trading-House Automation System (TAS)

- The customer can produce the material issue slip at the store house:
  - ▣ Take delivery of the items.
  - ▣ Inventory data adjusted to reflect the sale to the customer.

# Example: Trading-House Automation System (TAS)

- If an ordered item is not available in the inventory in sufficient quantity:
  - To be able to fulfill pending orders store details in a "pending-order" file :
    - out-of-stock items along with quantity ordered.
    - customer identification number

# Example: Trading-House Automation System (TAS)

- The purchase department:
  - ▣ would periodically issue commands to generate indents.
- When **generate indents** command is issued:
  - ▣ The system should examine the "pending-order" file
  - ▣ Determine the orders that are pending
  - ▣ Total quantity required for each of the items.

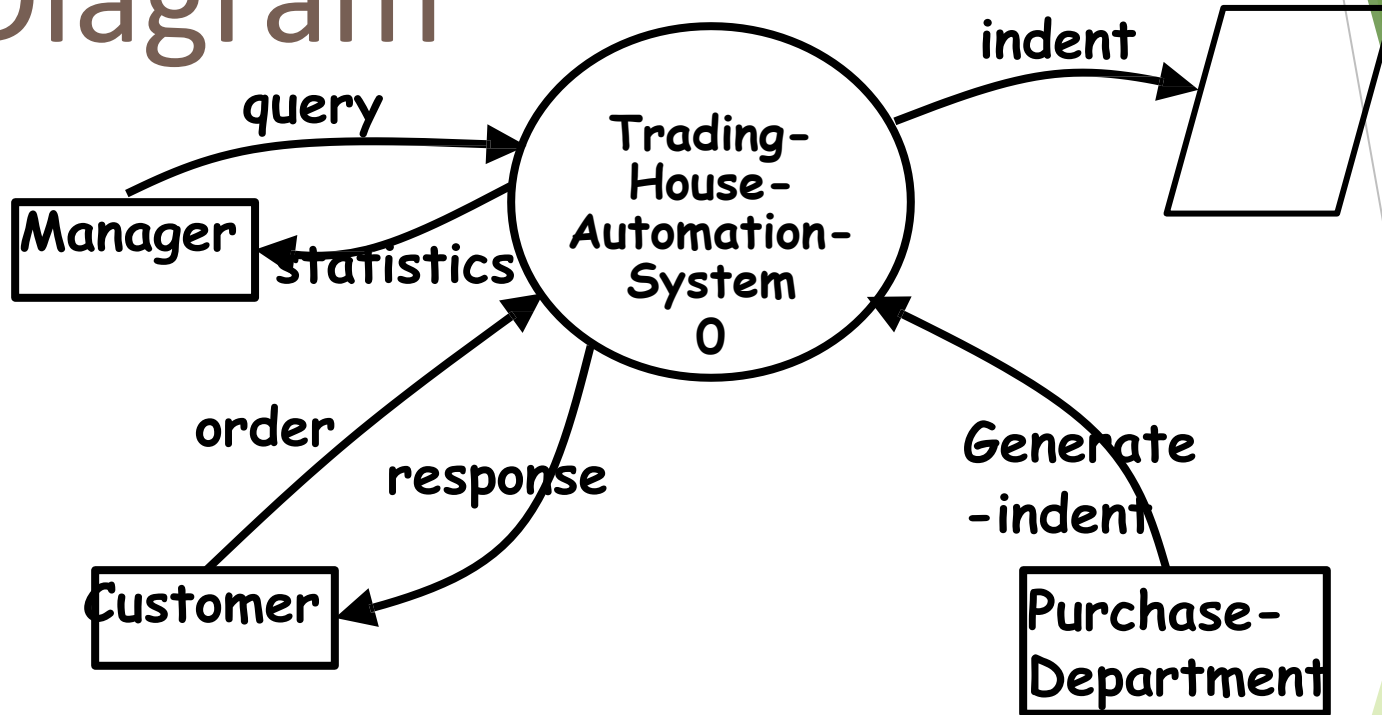
# Example: Trading-House Automation System (TAS)

- TAS should find out the addresses of the vendors who supply the required items:
  - ▢ Examine the file containing vendor details (their address, items they supply etc.)
  - ▢ Print out indents to those vendors.

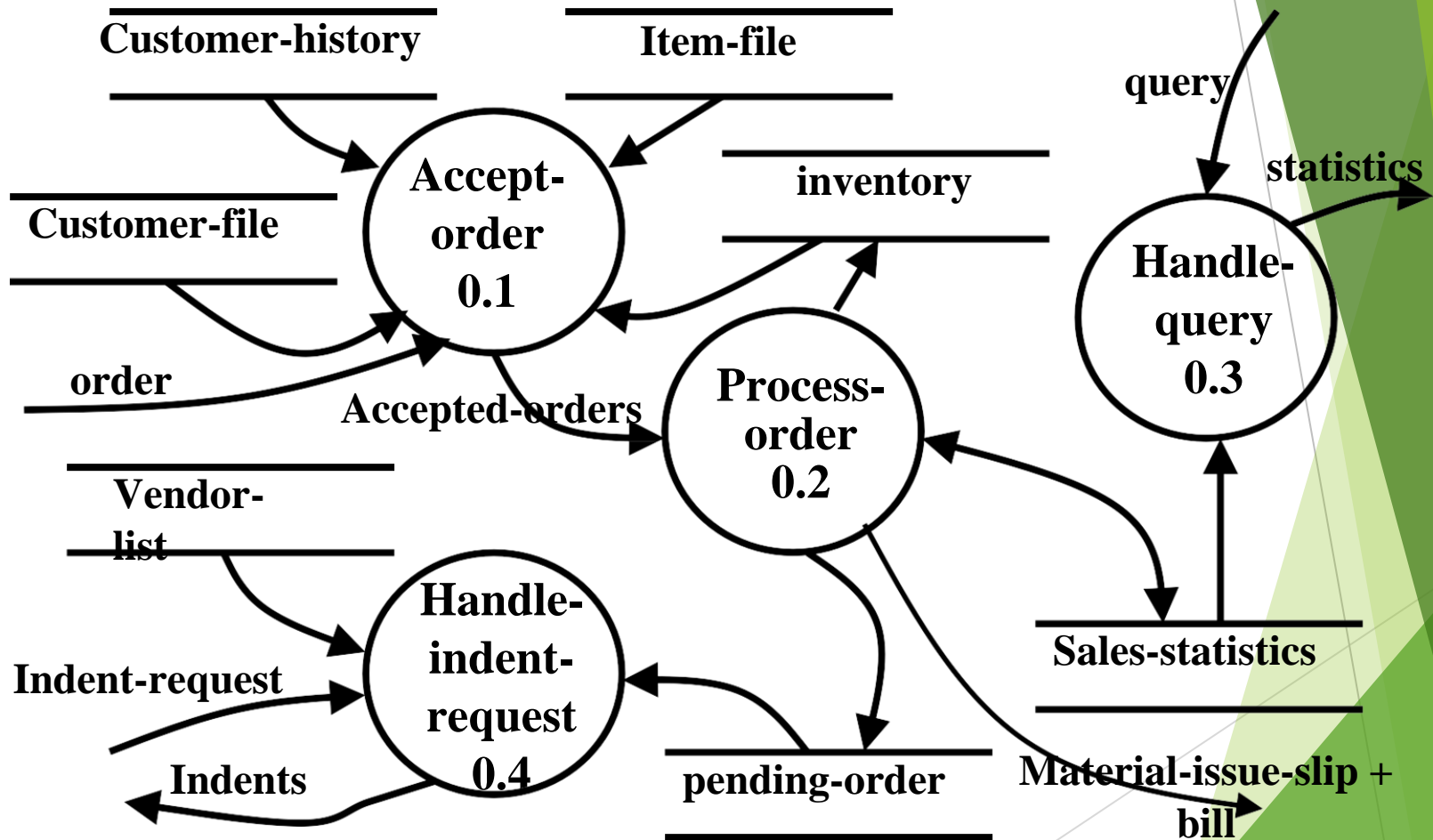
# Example: Trading-House Automation System (TAS)

- TAS should also answers managerial queries:
  - ▣ Statistics of different items sold over any given period of time
  - ▣ Corresponding quantity sold and the price realized.

# Context Diagram



# Level 1 DFD



# Example: Data Dictionary

- response: [bill + material-issue-slip, reject-message]
- query: period /\* query from manager regarding sales statistics\*/
- period: [date+date,month,year,day]
- date: year + month + day
- year: integer
- month: integer
- day: integer
- order: customer-id + {items + quantity}\*
- accepted-order:           order   /\* ordered items available in inventory \*/
- reject-message:           order + message /\* rejection message \*/
- pending-orders:           customer-id + {items+quantity}\*
- customer-address: name+house#+street#+city+pin



# Example: Data Dictionary

- item-name: string
- house#: string
- street#: string
- city: string
- pin: integer
- customer-id: integer
- bill: {item + quantity + price}\* + total-amount + customer-address
- material-issue-slip: message + item + quantity + customer-address
- message: string
- statistics: {item + quantity + price }\*
- sales-statistics: {statistics}\*
- quantity: integer

# How is Structured Analysis Performed?

- Initially represent the software at the most abstract level:
  - ▣ Called the context diagram.
  - ▣ The entire system is represented as a single bubble,
  - ▣ This bubble is labelled according to the main function of the system.

# Level 1 DFD

- Examine the SRS document:
  - ▣ Represent each high-level function as a bubble.
  - ▣ Represent data input to every high-level function.
  - ▣ Represent data output from every high-level function.

# Higher Level DFDs

- Each high-level function is separately decomposed into sub-functions:
  - ▣ Identify the sub-functions of the function
  - ▣ Identify the data input to each sub-function
  - ▣ Identify the data output from each sub-function
- These are represented as DFDs.

# Decomposition

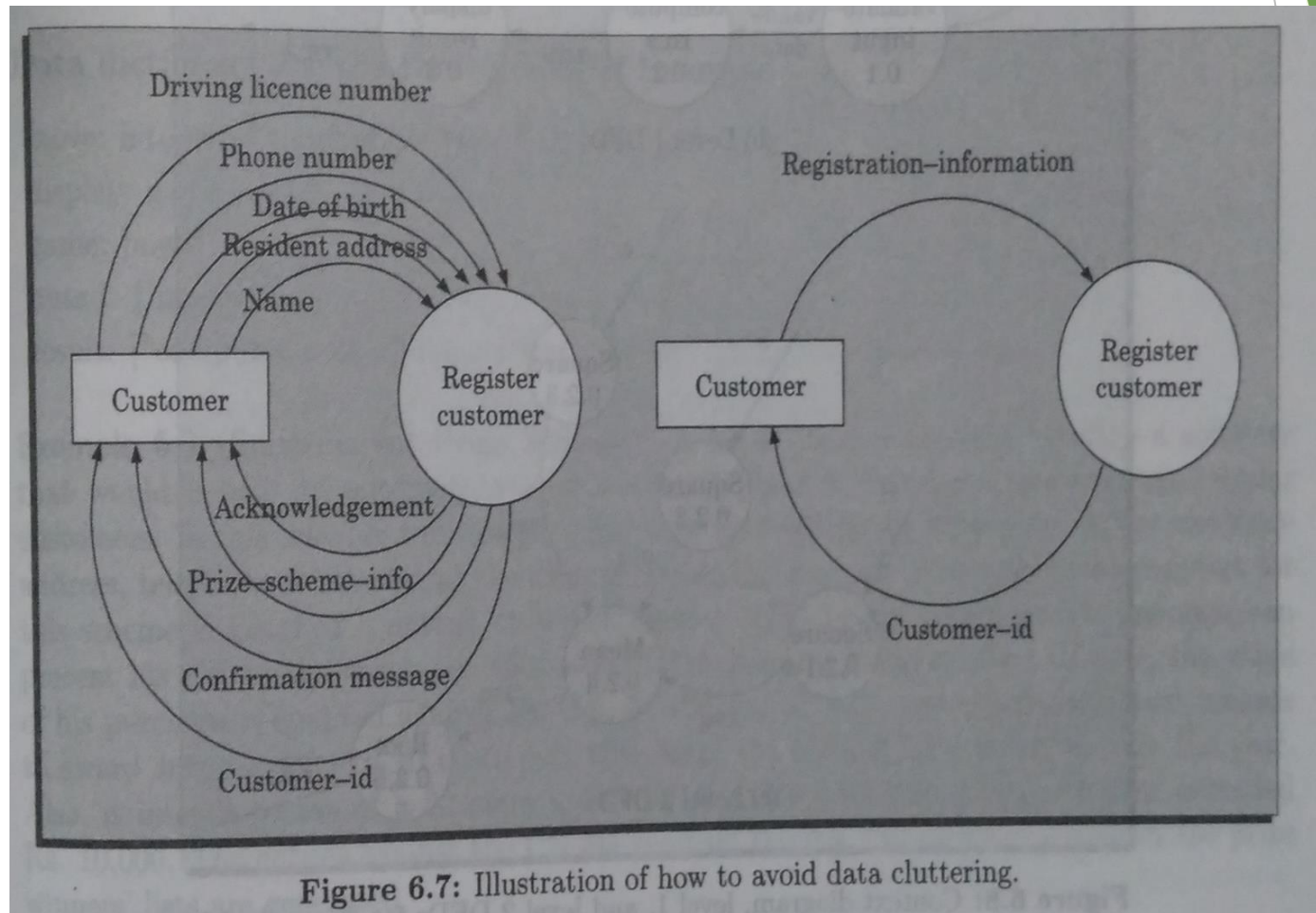
- Decomposition of a bubble:
  - ▢ Also called **factoring** or **exploding**.
- Each bubble is decomposed to
  - ▢ Between 3 to 7 bubbles.
- Too few bubbles make decomposition superfluous:
  - ▢ If a bubble is decomposed to just one or two bubbles:
    - Then this decomposition is redundant.
- Too many bubbles:
  - ▢ More than 7 bubbles at any level of a DFD.
  - ▢ Make the DFD model hard to understand.

# Decompose How Long?

- Decomposition of a bubble should be carried on until:
  - A level at which the function of the bubble can be described using a simple algorithm.

# Precautions while making DFD's

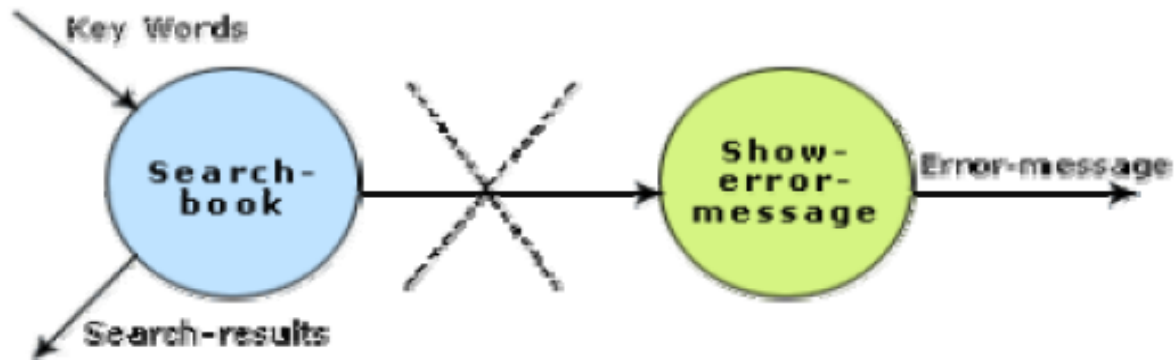
## ► 1. Avoid Data cluttering



- Many beginners commit the mistake of drawing more than one bubble in the context diagram. A context diagram should depict the system as a single bubble.
- Many beginners have external entities appearing at all levels of DFDs. All external entities interacting with the system should be represented only in the context diagram. The external entities should not appear at other levels of the DFD.
- It is a common oversight to have either too less or too many bubbles in a DFD. Only 3 to 7 bubbles per diagram should be allowed, i.e. each bubble should be decomposed to between 3 and 7 bubbles.
- Many beginners leave different levels of DFD unbalanced.
- A common mistake committed by many beginners while developing a DFD model is attempting to represent control information in a DFD. It is important to realize that a DFD is the data flow representation of a system, and it does not represent control information. For an example mistake of this kind:



- Consider the following example. A book can be searched in the library catalog by inputting its name. If the book is available in the library, then the details of the book are displayed. If the book is not listed in the catalog, then an error message is generated. While generating the DFD model for this simple problem, many beginners commit the mistake of drawing an arrow (as shown in fig. 5.10) to indicate the error function is invoked after the search book. But, this is a control information and should not be shown on the DFD.



**Fig. 5.10:** Showing control information on a DFD - incorrect

- Another error is trying to represent when or in what order different functions (processes) are invoked and not representing the conditions under which different functions are invoked.
- If a bubble A invokes either the bubble B or the bubble C depending upon some conditions, we need only to represent the data that flows between bubbles A and B or bubbles A and C and not the conditions depending on which the two modules are invoked.

- A data store should be connected only to bubbles through data arrows. A data store cannot be connected to another data store or to an external entity.
- All the functionalities of the system must be captured by the DFD model. No function of the system specified in its SRS document should be overlooked.
- Only those functions of the system specified in the SRS document should be represented, i.e. the designer should not assume functionality of the system not specified by the SRS document and then try to represent them in the DFD.
- Improper or unsatisfactory data dictionary.
- The data and function names must be intuitive. Some students and even practicing engineers use symbolic data names such a, b, c, etc. Such names hinder understanding the DFD model.

# Decision Tree

## Decision tree

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

### **Example: -**

Consider Library Membership Automation Software (LMS) where it should support the following three options:

- **New member**
- **Renewal**
- **Cancel membership**

### **New member option-**

**Decision:** When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

**Action:** If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

### **Renewal option-**

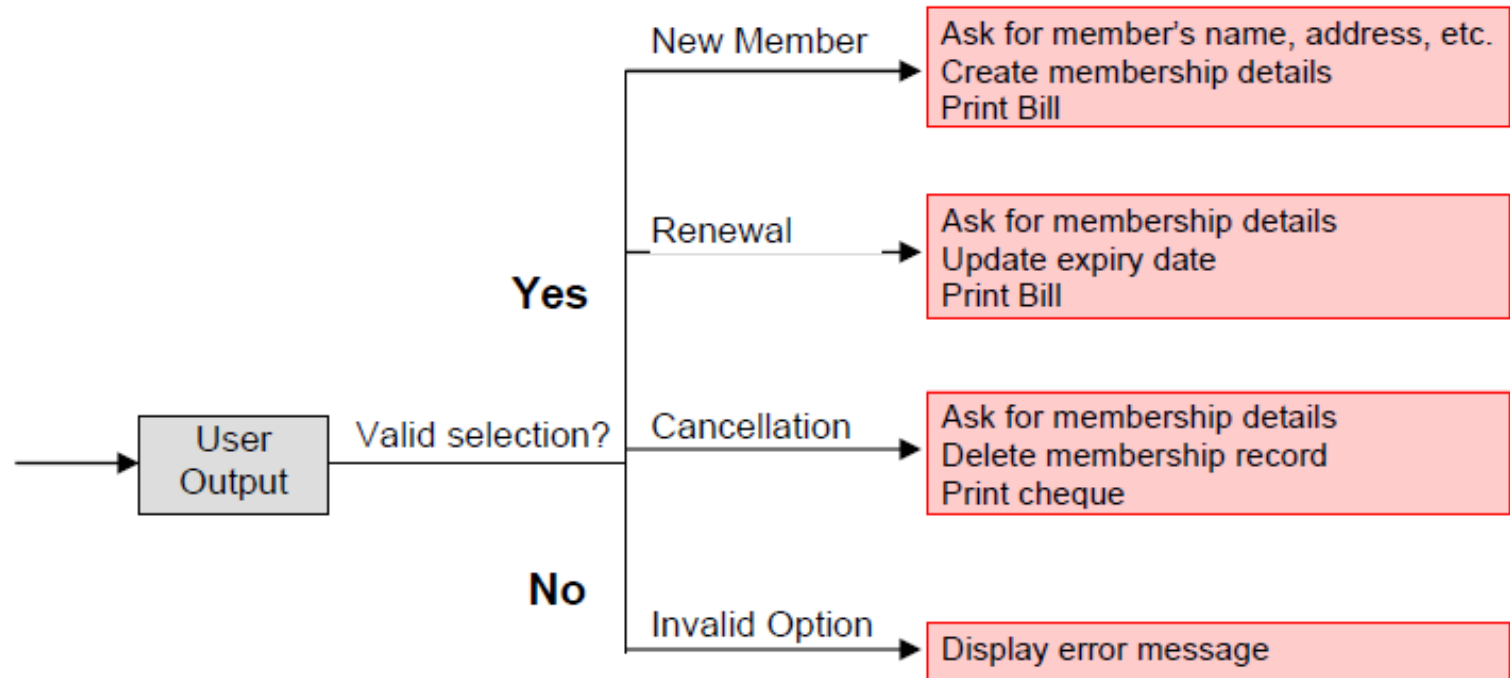
**Decision:** If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

**Action:** If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

### **Cancel membership option-**

**Decision:** If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

**Action:** The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.



**Fig. 3.4:** Decision tree for LMS



# Decision Table

## Decision table

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a *rule*. A rule implies that if a condition is true, then the corresponding action is to be executed.

### Example: -

Consider the previously discussed LMS example. The following decision table (fig. 3.5) shows how to represent the LMS problem in a tabular form. Here the table is divided into two parts, the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

### Conditions

Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
<b>Actions</b>				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

**Fig. 3.5: Decision table for LMS**



# Data Dictionary

- A DFD is always accompanied by a data dictionary.
- A data dictionary lists all data items appearing in a DFD:
  - ▣ Definition of all composite data items in terms of their component data items.
  - ▣ All data names along with the purpose of the data items.
- For example, a data dictionary entry may be:
  - ▣  $\text{grossPay} = \text{regularPay} + \text{overtimePay}$

# Importance of Data Dictionary

- Provides all engineers in a project with standard terminology for all data:
  - ▣ A consistent vocabulary for data is very important
  - ▣ Different engineers tend to use different terms to refer to the same data,
    - Causes unnecessary confusion.

# Importance of Data Dictionary

- Data dictionary provides the definition of different data:
  - ▣ In terms of their component elements.
- For large systems,
  - ▣ The data dictionary grows rapidly in size and complexity.
  - ▣ Typical projects can have thousands of data dictionary entries.
  - ▣ It is extremely difficult to maintain such a dictionary manually.

# Data Dictionary

- CASE (Computer Aided Software Engineering) tools come handy:
  - ▣ CASE tools capture the data items appearing in a DFD automatically to generate the data dictionary.

# Data Dictionary

- CASE tools support queries:
  - About definition and usage of data items.
- For example, queries may be made to find:
  - Which data item affects which processes,
  - A process affects which data items,
  - The definition and usage of specific data items, etc.
- Query handling is facilitated:
  - If data dictionary is stored in a relational database management system (RDBMS).

# Data Definition

- Composite data are defined in terms of primitive data items using following operators:
  - $+$ : denotes composition of data items, e.g.
    - $a+b$  represents data  $a$  and  $b$ .
  - $[,,,]$ : represents selection,
    - i.e. any one of the data items listed inside the square bracket can occur.
    - For example,  $[a,b]$  represents either  $a$  occurs or  $b$  occurs.

# Data Definition

- ( ): contents inside the bracket represent optional data
  - which may or may not appear.
  - $a+(b)$  represents either  $a$  or  $a+b$  occurs.
- { }: represents iterative data definition,
  - e.g. {name}5 represents five name data.

# Data Definition

- $\{ \text{name} \}^*$  represents
  - zero or more instances of name data.
- $=$  represents equivalence,
  - e.g.  $a=b+c$  means that a represents b and c.
- $* \text{ } *$ : Anything appearing within  $* \text{ } *$  is considered as comment.



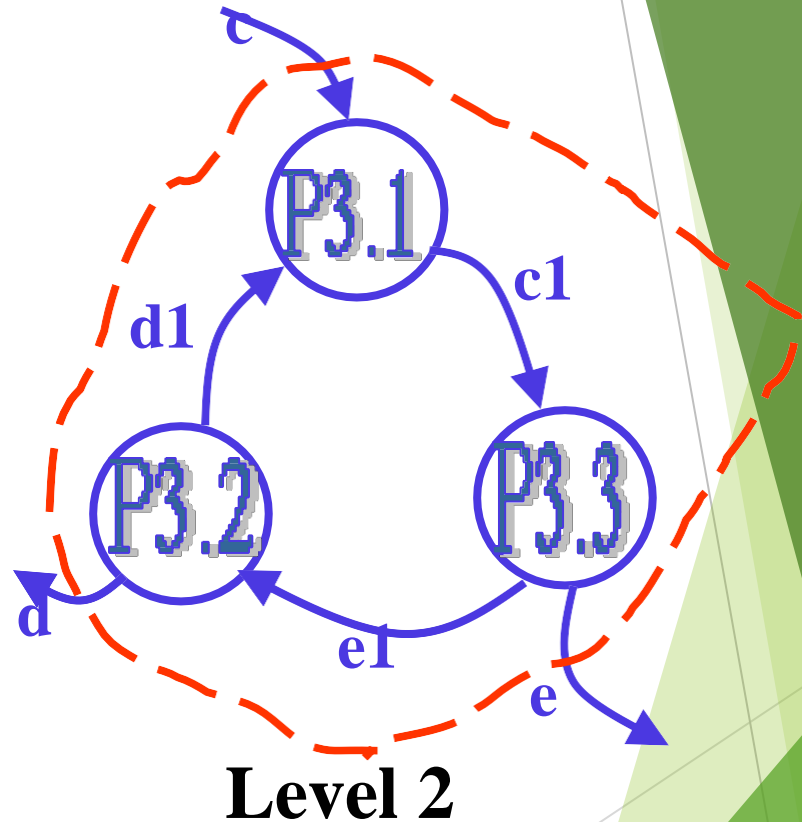
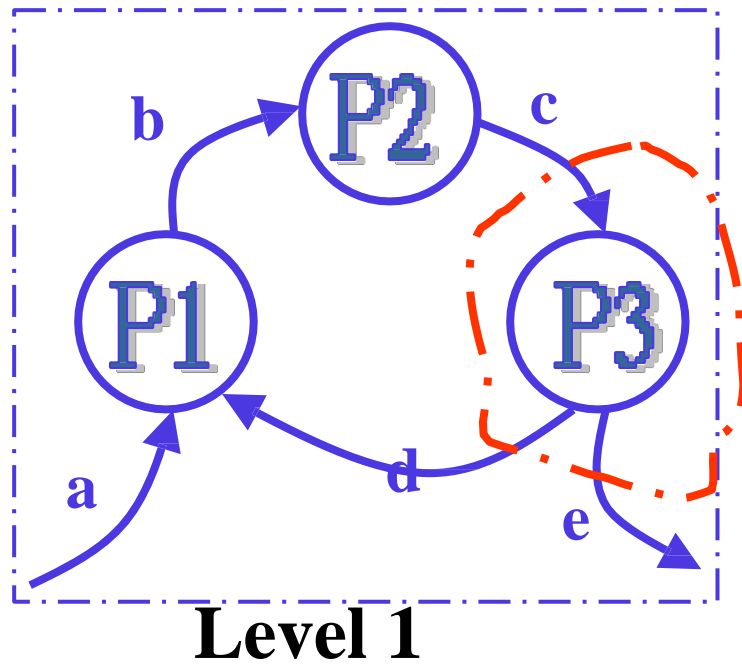
# Data Dictionary for RMS Software

- numbers=valid-numbers=a+b+c
- a:integer               \* input number \*
- b:integer               \* input number \*
- c:integer               \* input number \*
- asq:integer
- bsq:integer
- csq:integer
- squared-sum: integer
- Result=[RMS,error]
- RMS: integer           \* root mean square value\*
- error:string           \* error message\*

# Balancing a DFD

- Data flowing into or out of a bubble:
  - ▣ Must match the data flows at the next level of DFD.
- In the level 1 of the DFD,
  - ▣ Data item c flows into the bubble P3 and the data item d and e flow out.
- In the next level, bubble P3 is decomposed.
  - ▣ The decomposition is balanced as data item c flows into the level 2 diagram and d and e flow out.

# Balancing a DFD



# Numbering of Bubbles

- Number the bubbles in a DFD:
  - ▢ Numbers help in uniquely identifying any bubble from its bubble number.
- The bubble at context level:
  - ▢ Assigned number 0.
- Bubbles at level 1:
  - ▢ Numbered 0.1, 0.2, 0.3, etc
- When a bubble numbered x is decomposed,
  - ▢ Its children bubble are numbered x.1, x.2, x.3, etc.

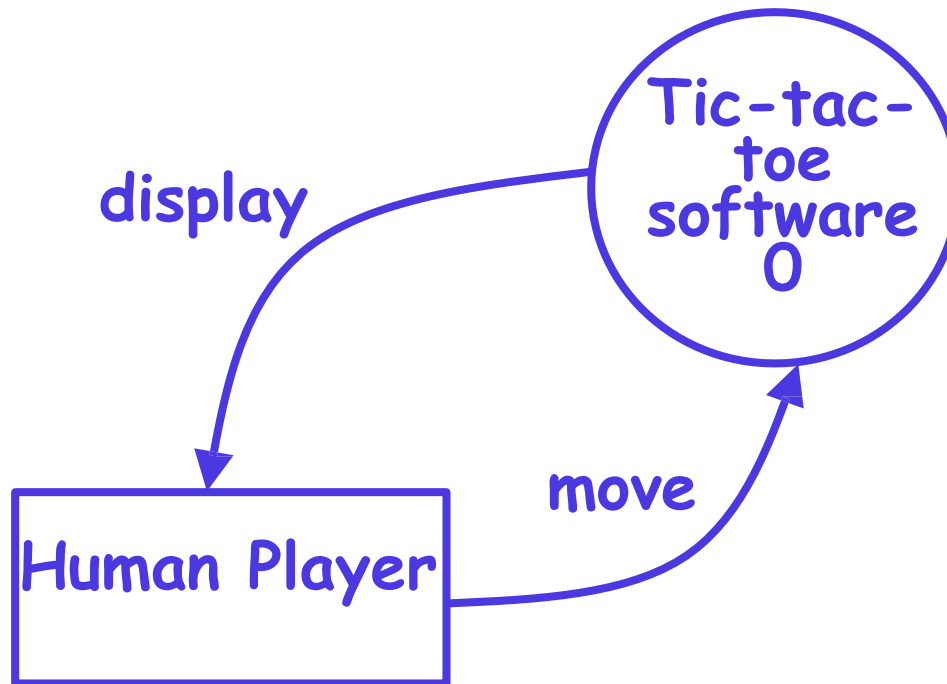
# Example 2: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3 X 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.

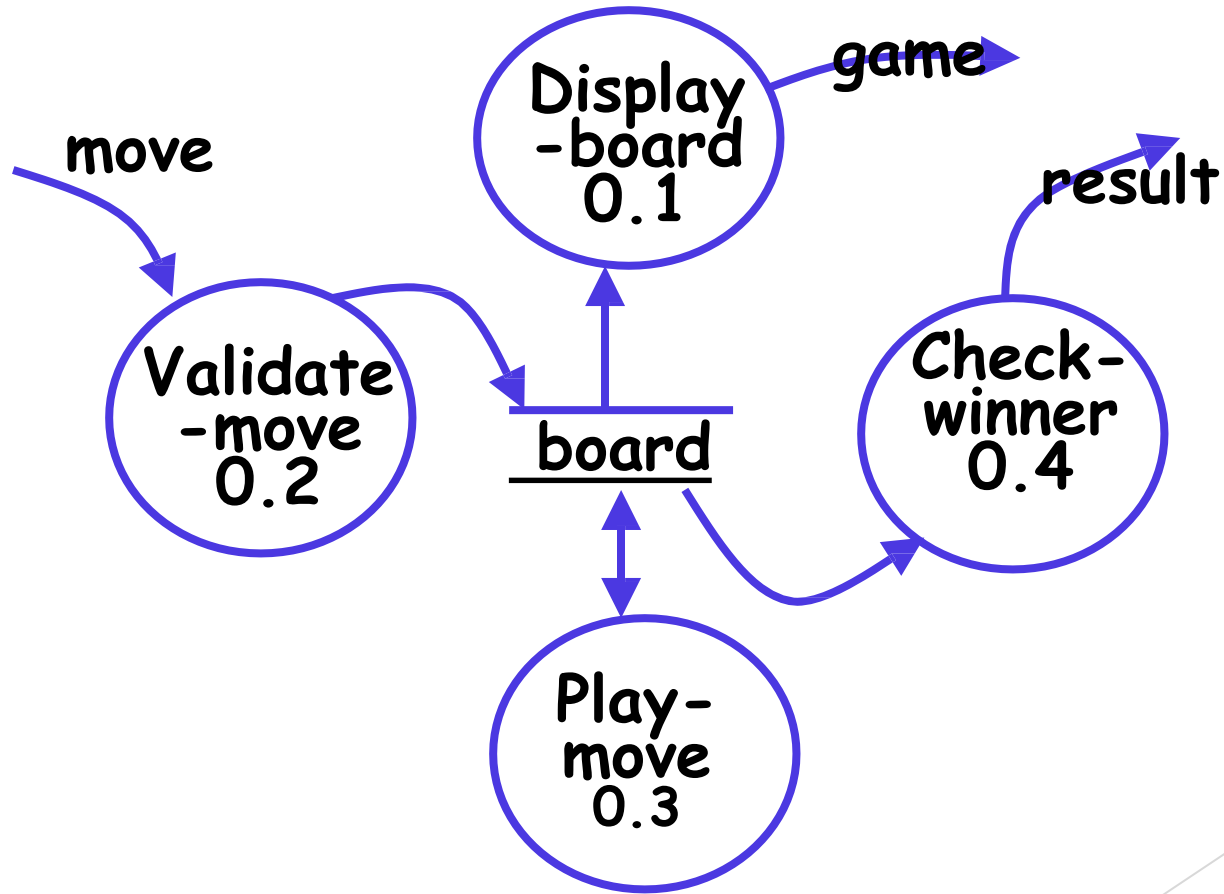
# Example: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
  - ▣ A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
  - ▣ And all the squares on the board are filled up,
  - ▣ Then the game is drawn.
- The computer always tries to win a game.

# Context Diagram for Example



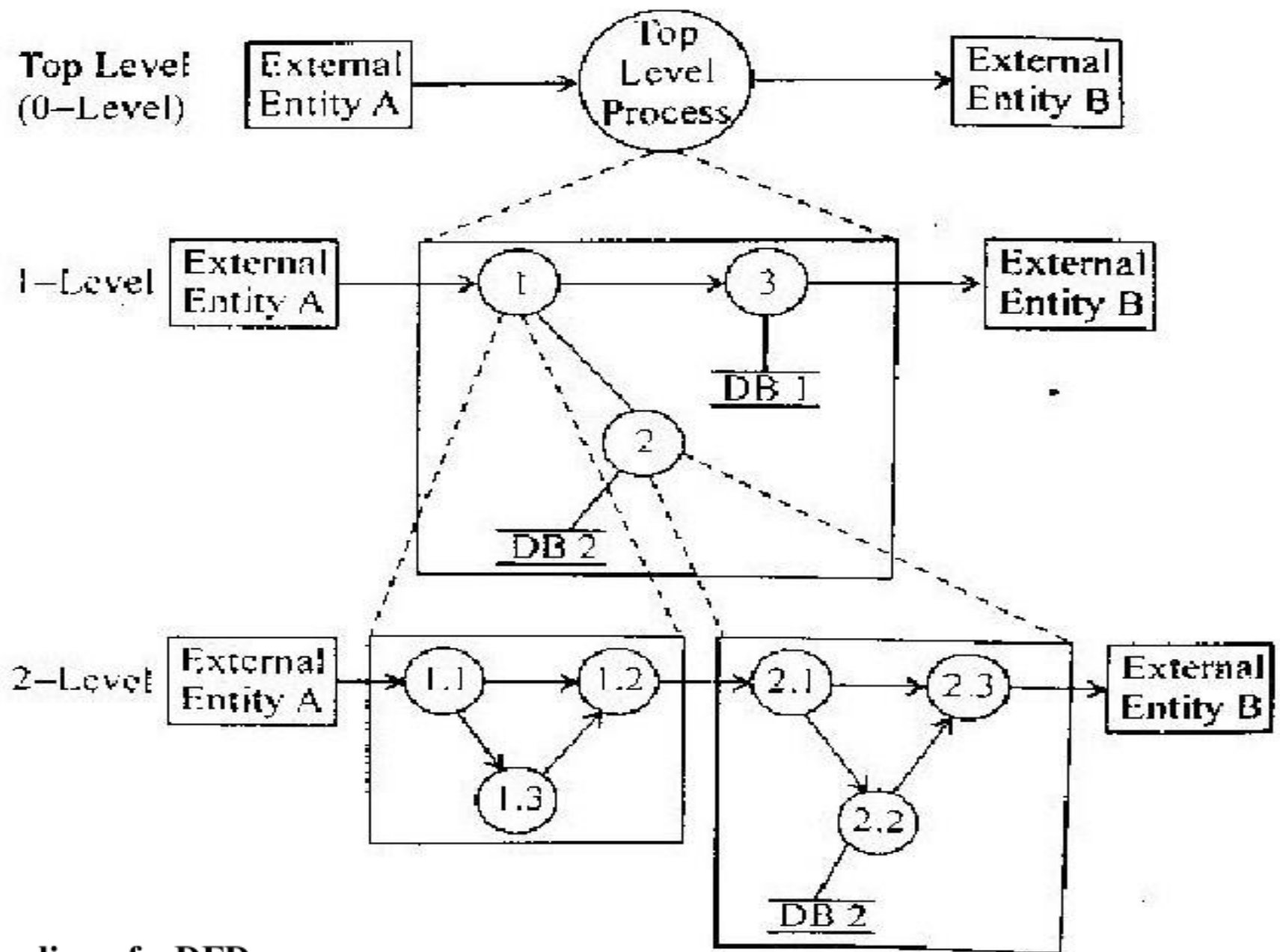
# Level 1 DFD







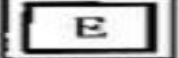


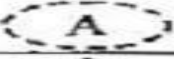

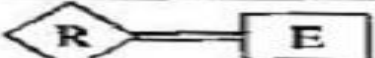

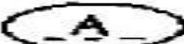



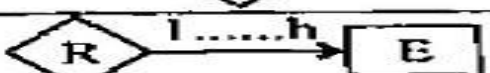



# Data Dictionary

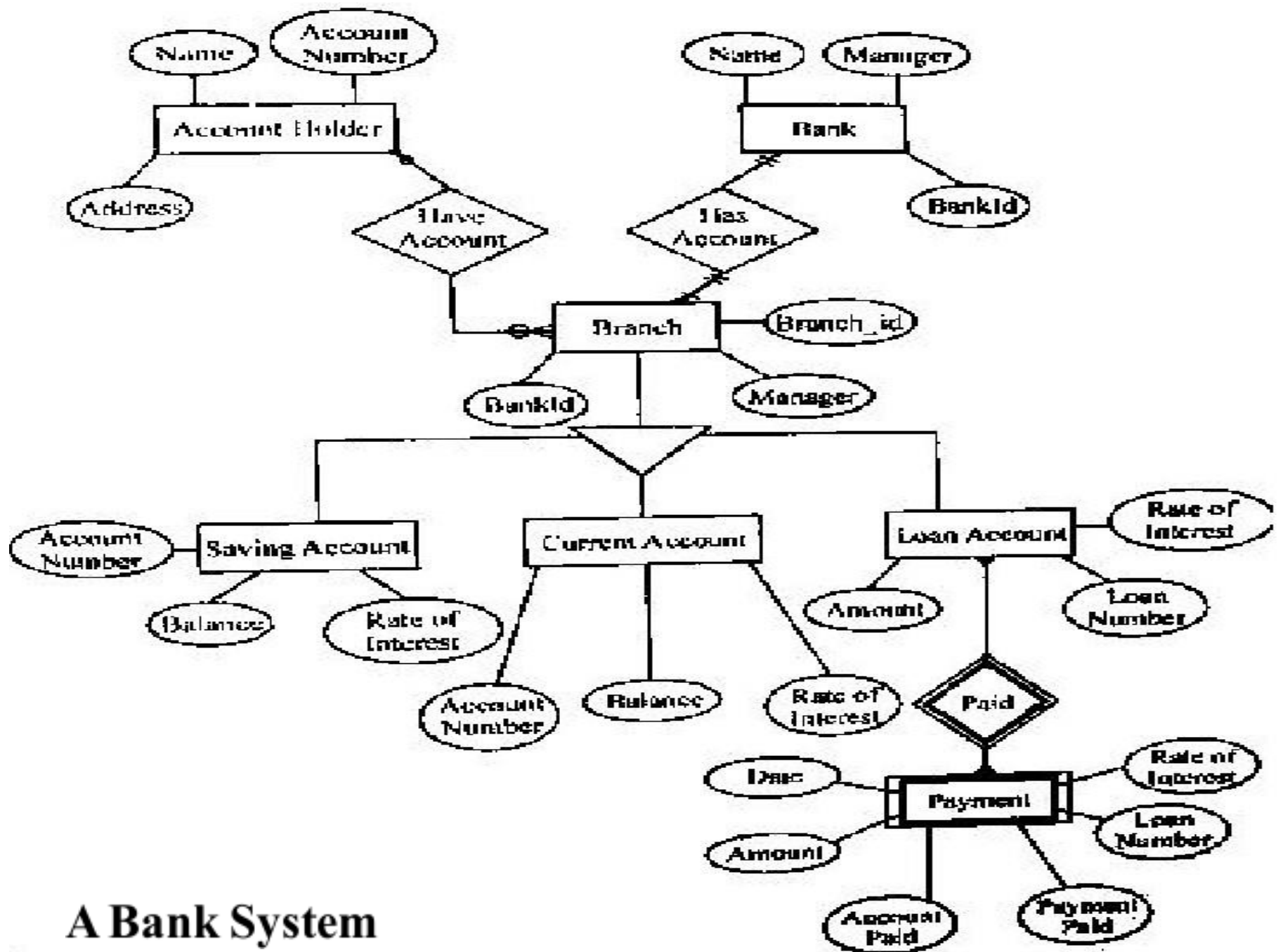
- Display=game + result
- move = integer
- board = {integer}9
- game = {integer}9
- result=string



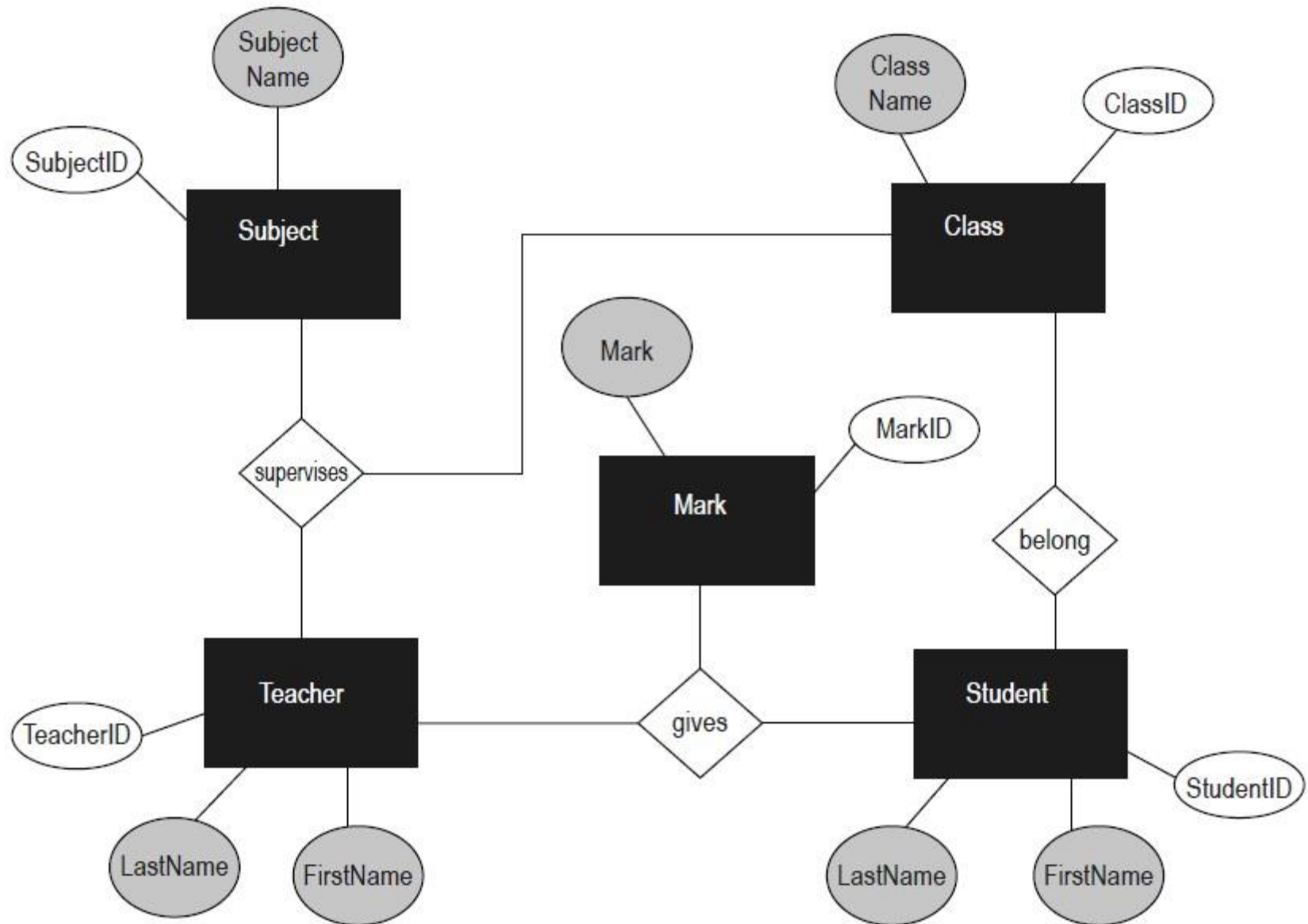
**Leveling of a DFD**

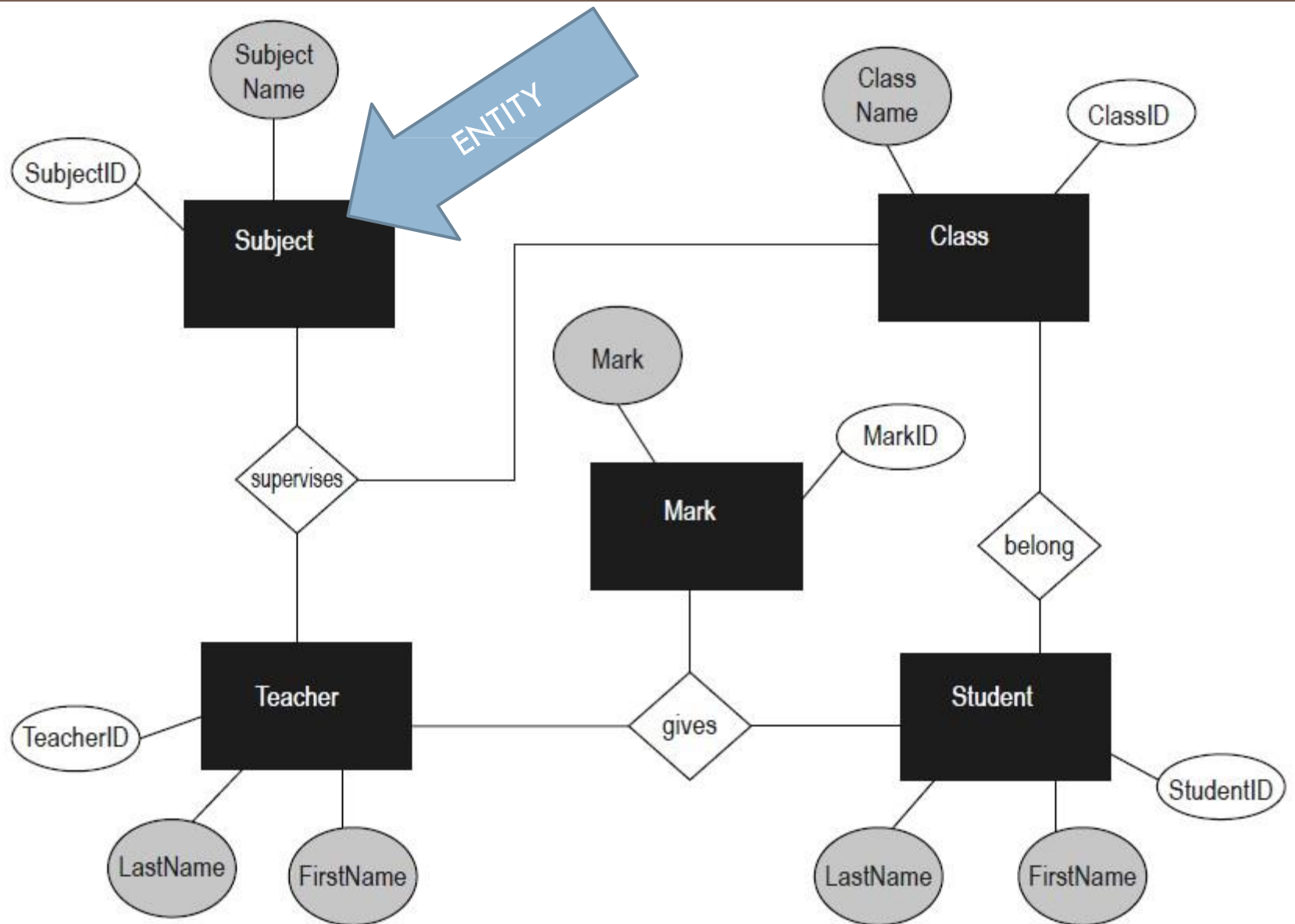
1. **Data Dictionary** - *Data Dictionary* forms the core of the model and contains the description of all the data objects that have been defined or concerned with the software. Data dictionary is surrounded by three different kinds of diagrams.
2. **Entity Relationship Diagram (ERD)** - An *Entity Relationship Diagrams (ERD)* describes how the various data objects are related to each other. ERD is used to conduct the data modeling activity and is concerned with defining the various attributes or properties of data objects.
3. **Data Flow Diagram (DFD)** - *Data Flow Diagrams (DFD)* describes both the functions, that are used to transform a data (input) into some other form (may be output), and the way the data is transformed from input to output. It serves as a basis for functional modeling.
4. **State Transition Diagram (STD)** - *State Transition Diagram* describes how the system behaves to external events. It can be a decision making step. It represents the various modes of the behaviour (called states) and the way in which system transits from one state to another.
5. **Process and Control Specification** - *Process Specification* contains the description of every functional module, defined in the DFD. *Control Specification* holds the additional information regarding the control aspects of the software, responsible for various state transition.
6. **Data Object Description** - *Data Object* description provides the complete knowledge about a data object.

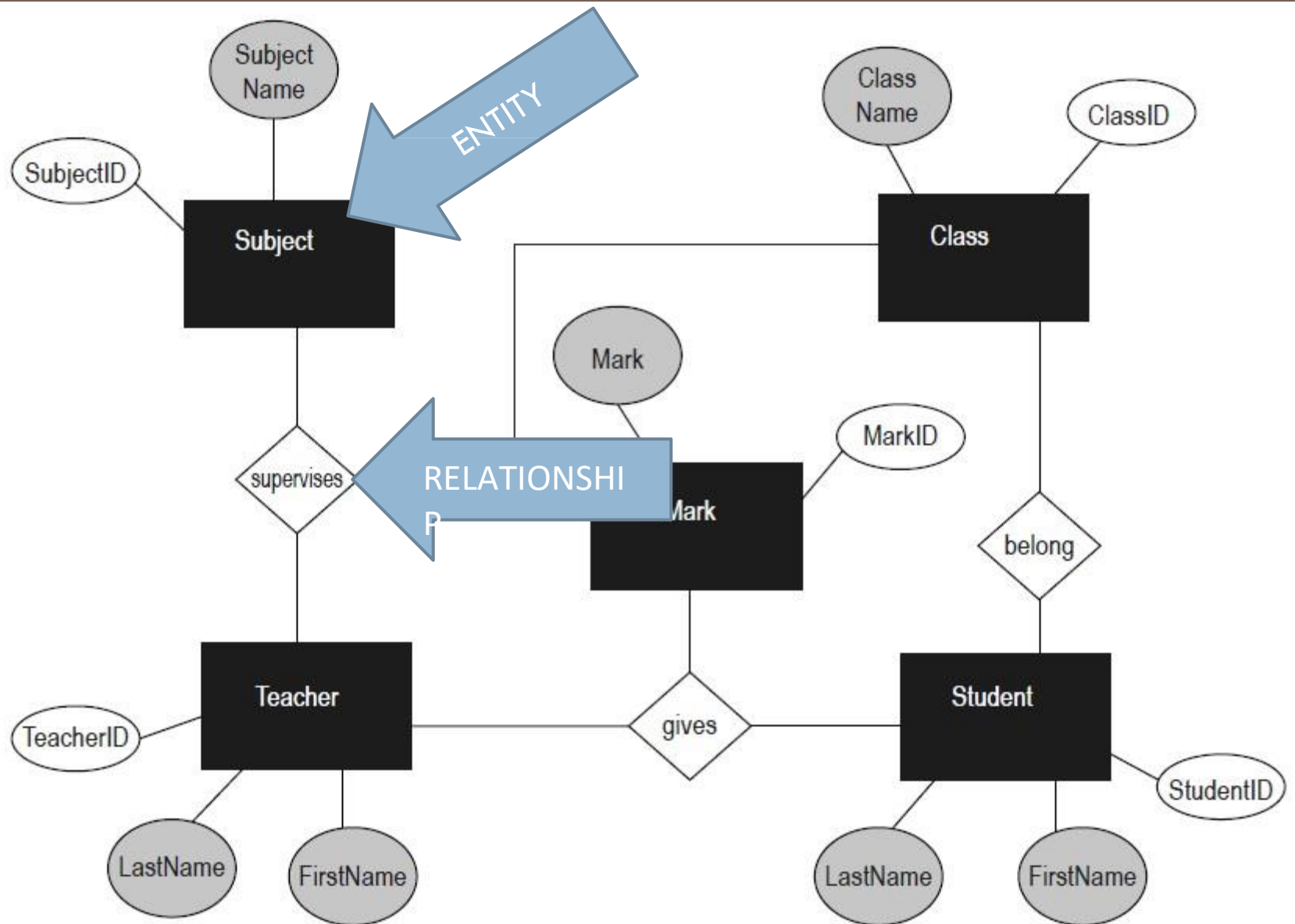
S.No.	Symbol	Names
1		Entity Set
2		Attribute
3		Weak Entity Set
4		Multivalued Attribute
5		Relationship Set
6		Derived Attribute
7		Relationship Set for Weak Entity Set
8		Total Participation of Entity Set
9		Primary Key
10		Discriminating Attribute of Weak Entity Set
11		Many-to-Many Relationship
12		Many-to-One Relationship
13		One-to-One relationship
14		Cardinality Limits
15		Role Indicator
16		ISA (Specialization or Generalization
17		Total Generalization



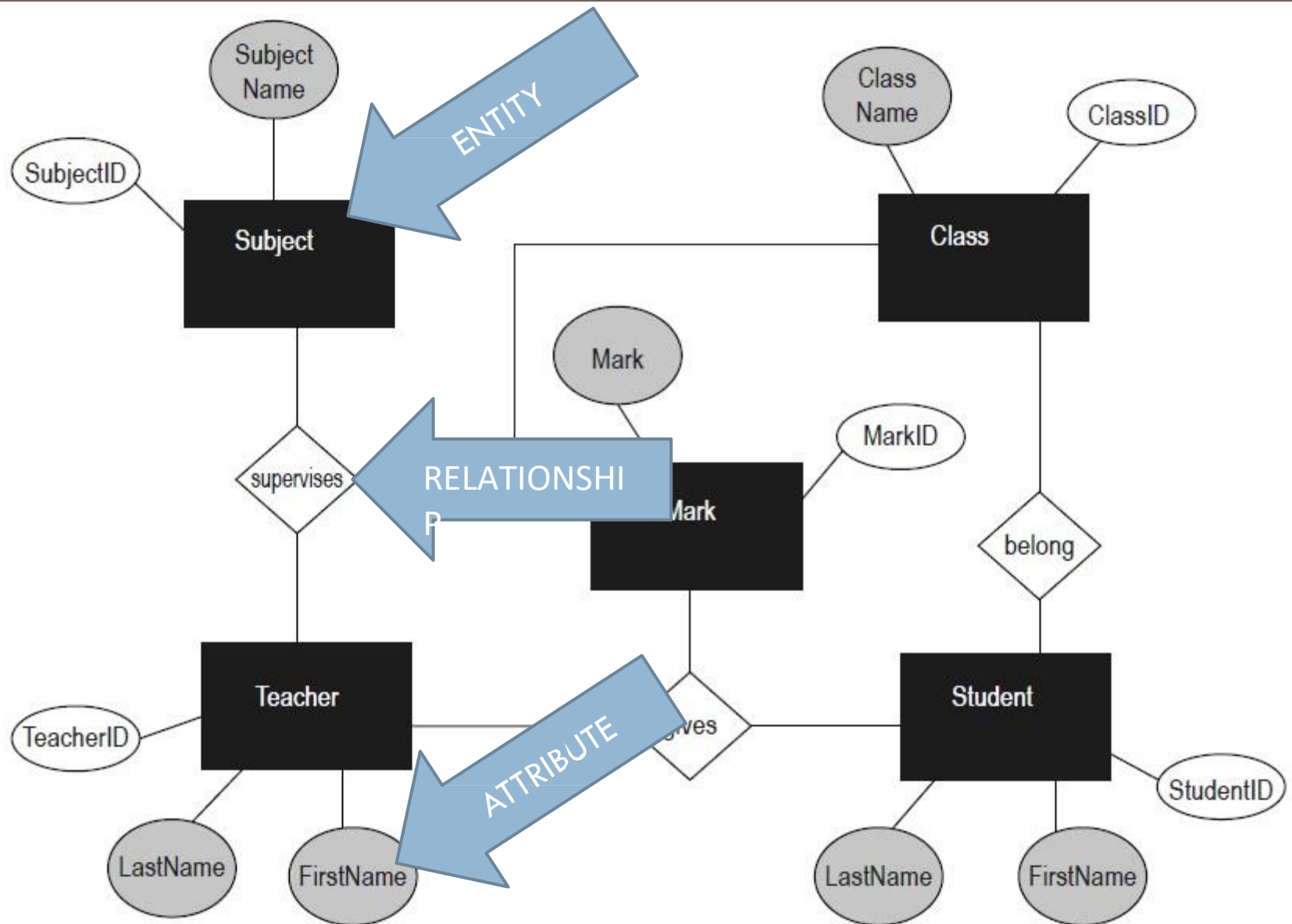
A Bank System

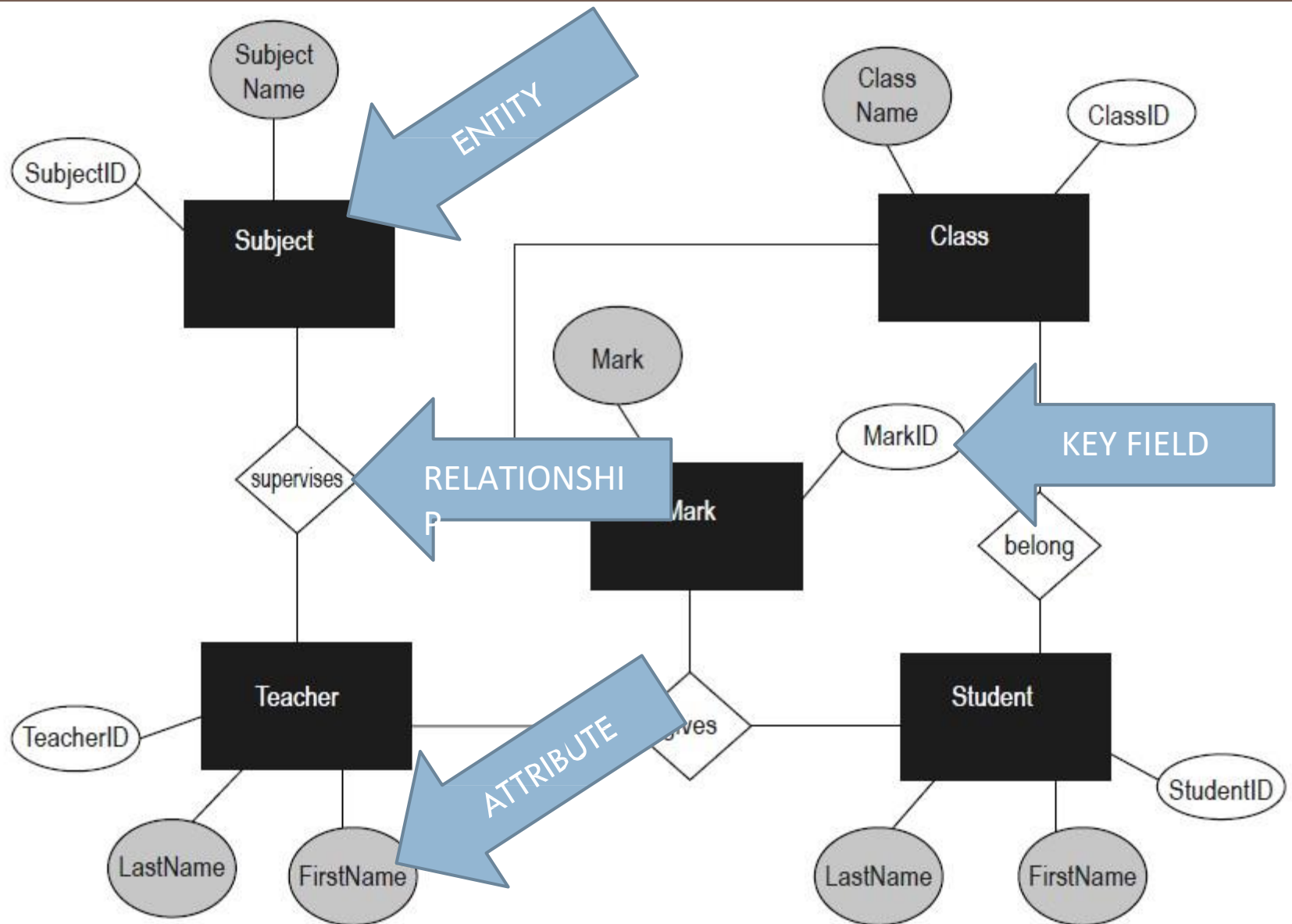


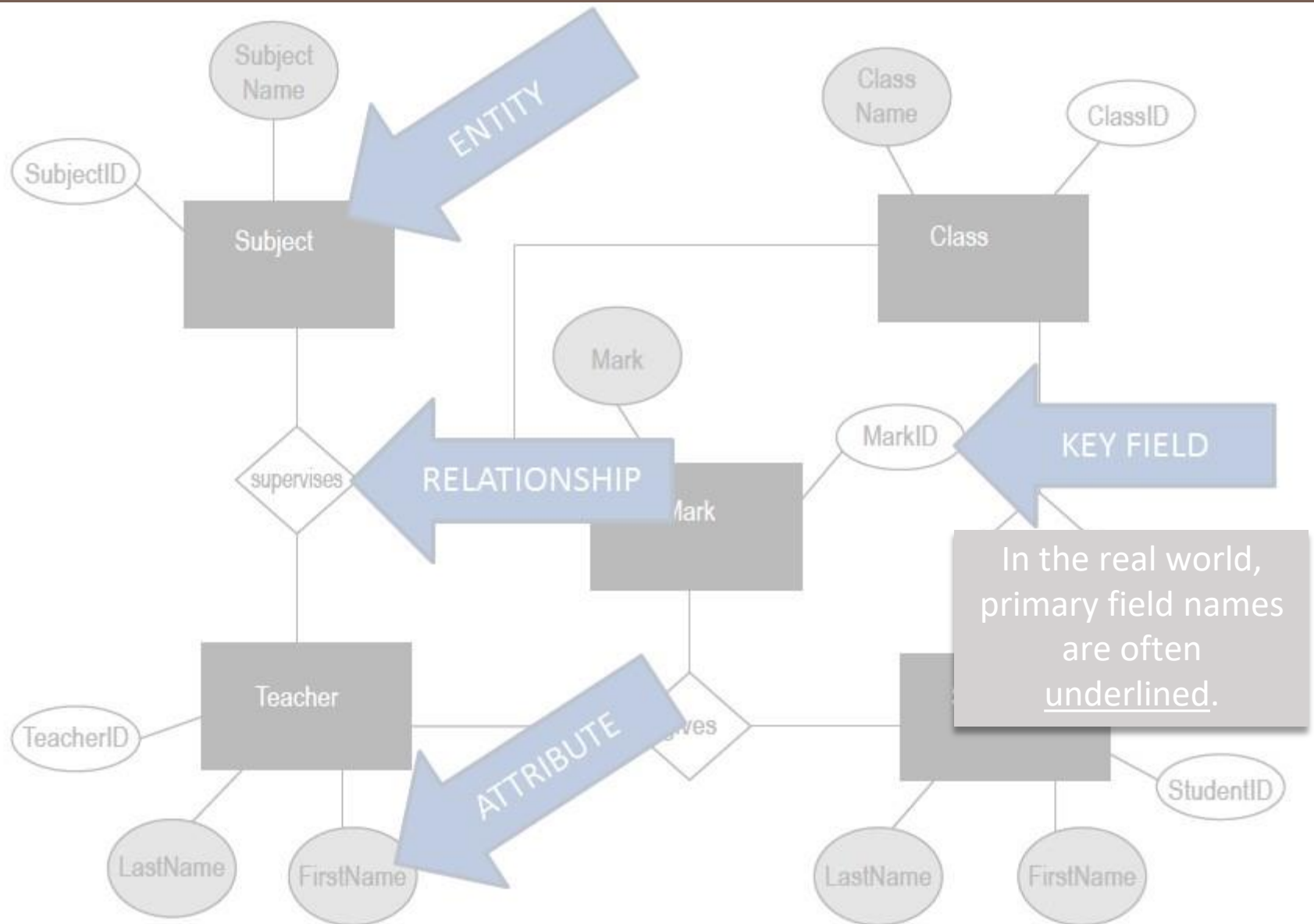




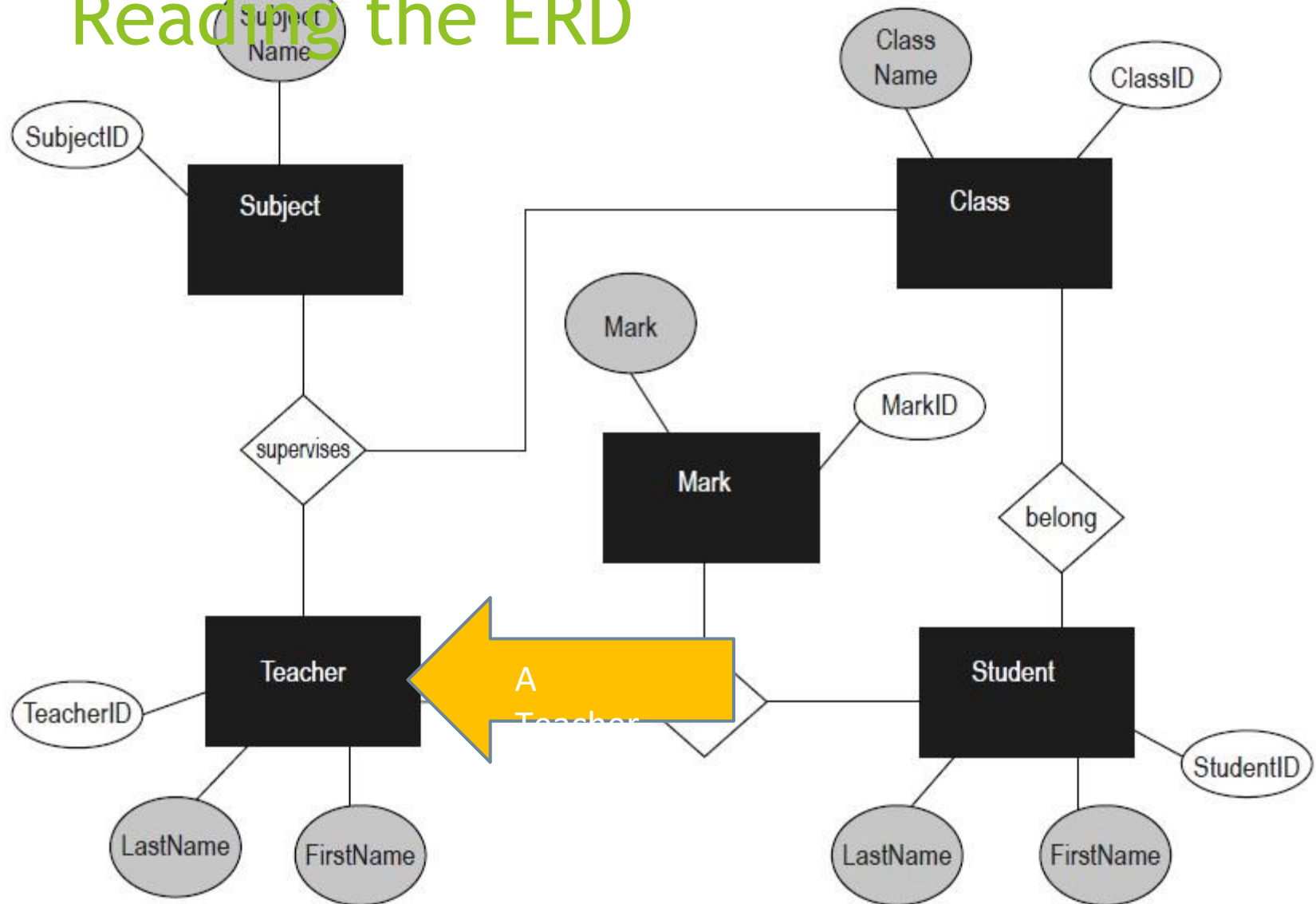




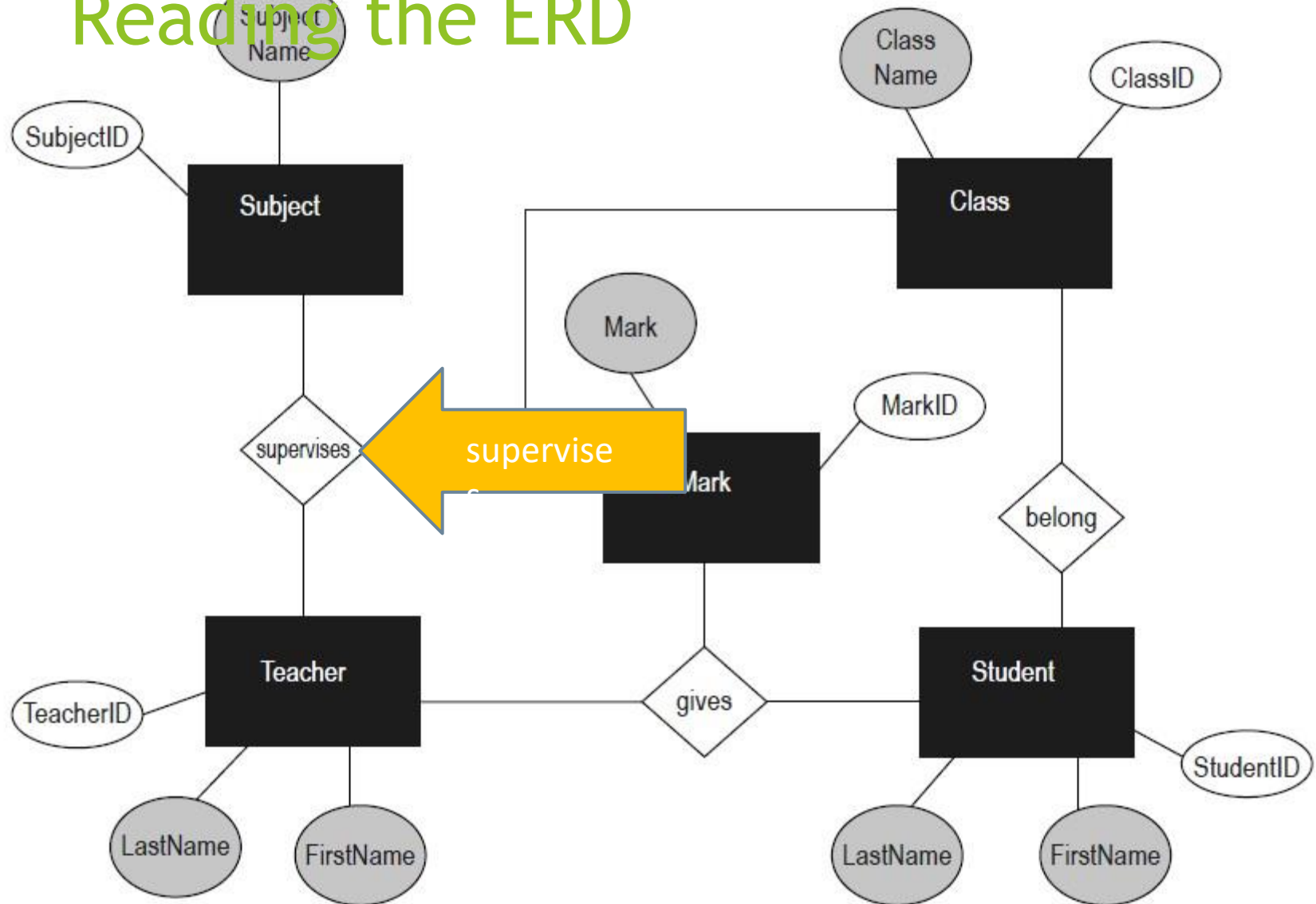




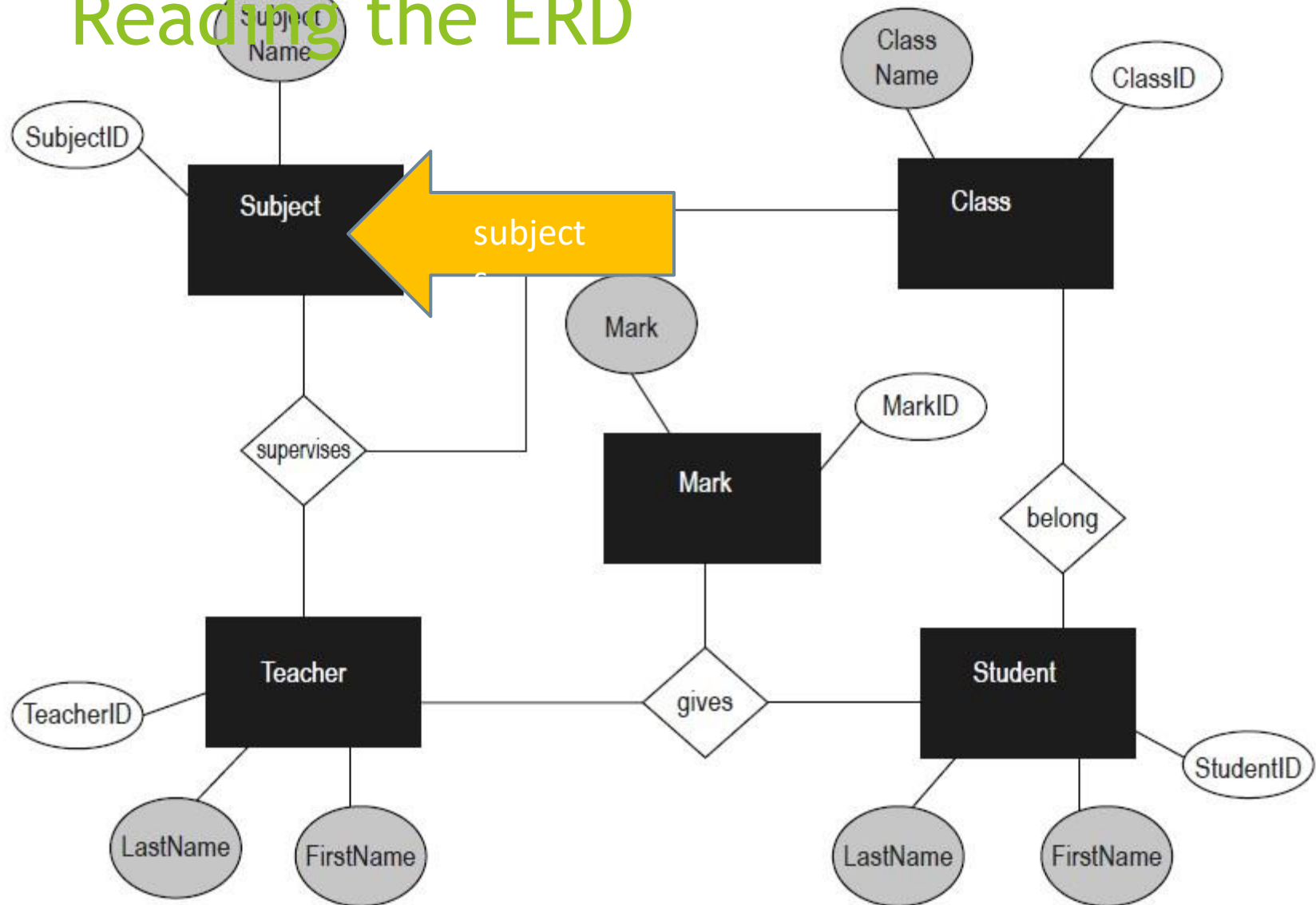
# Reading the ERD



# Reading the ERD

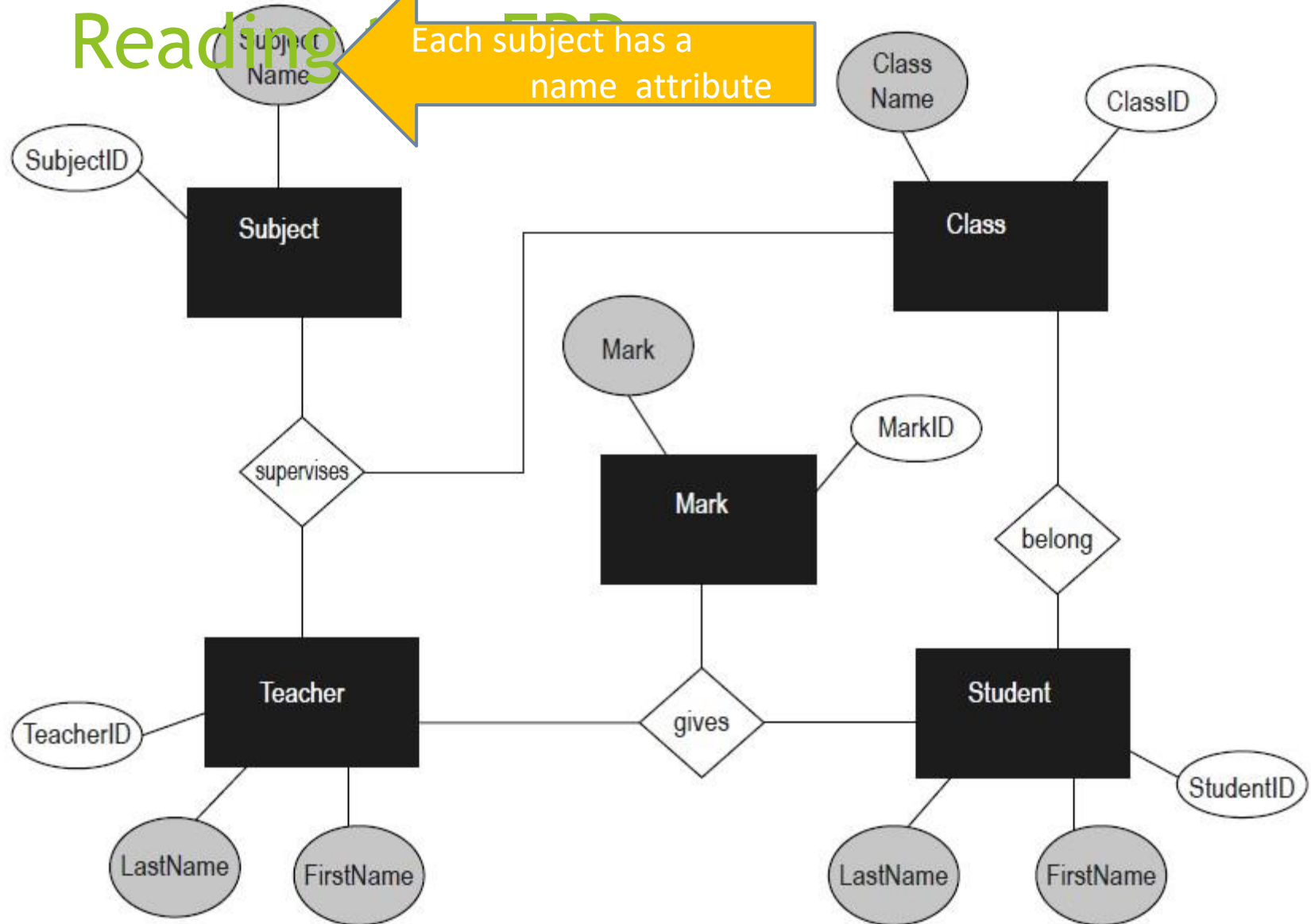


# Reading the ERD

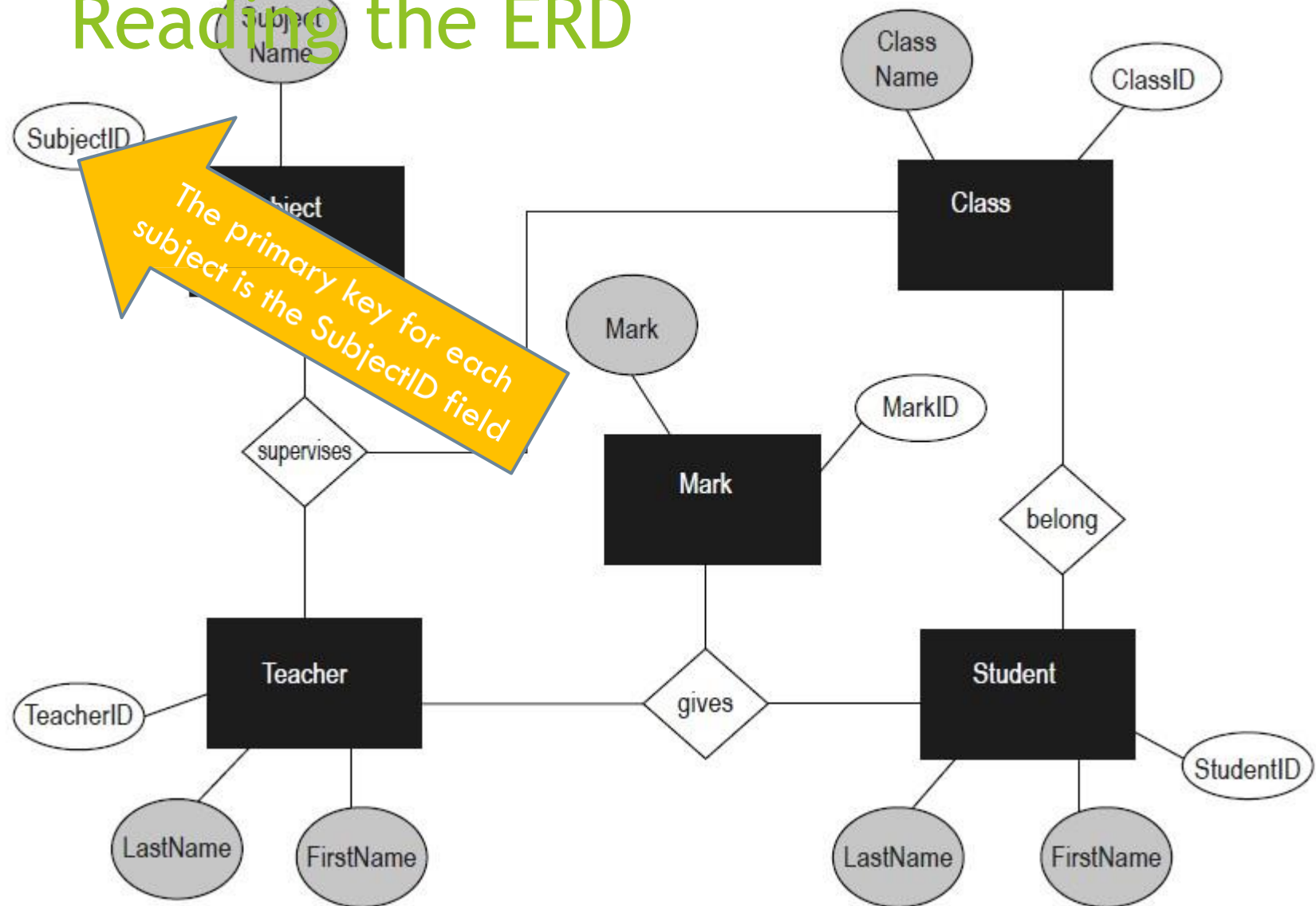


# Reading

Each subject has a name attribute

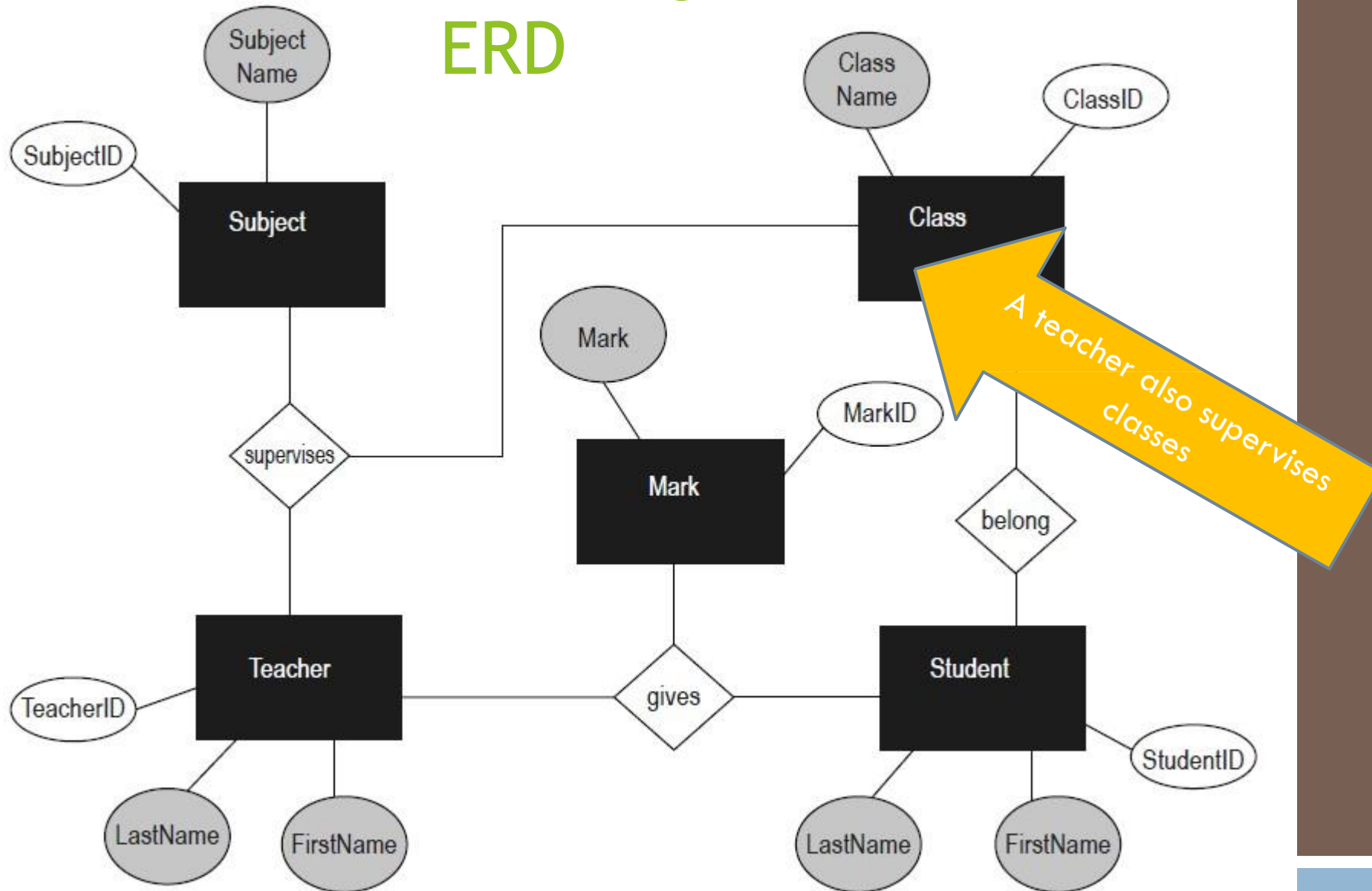


# Reading the ERD

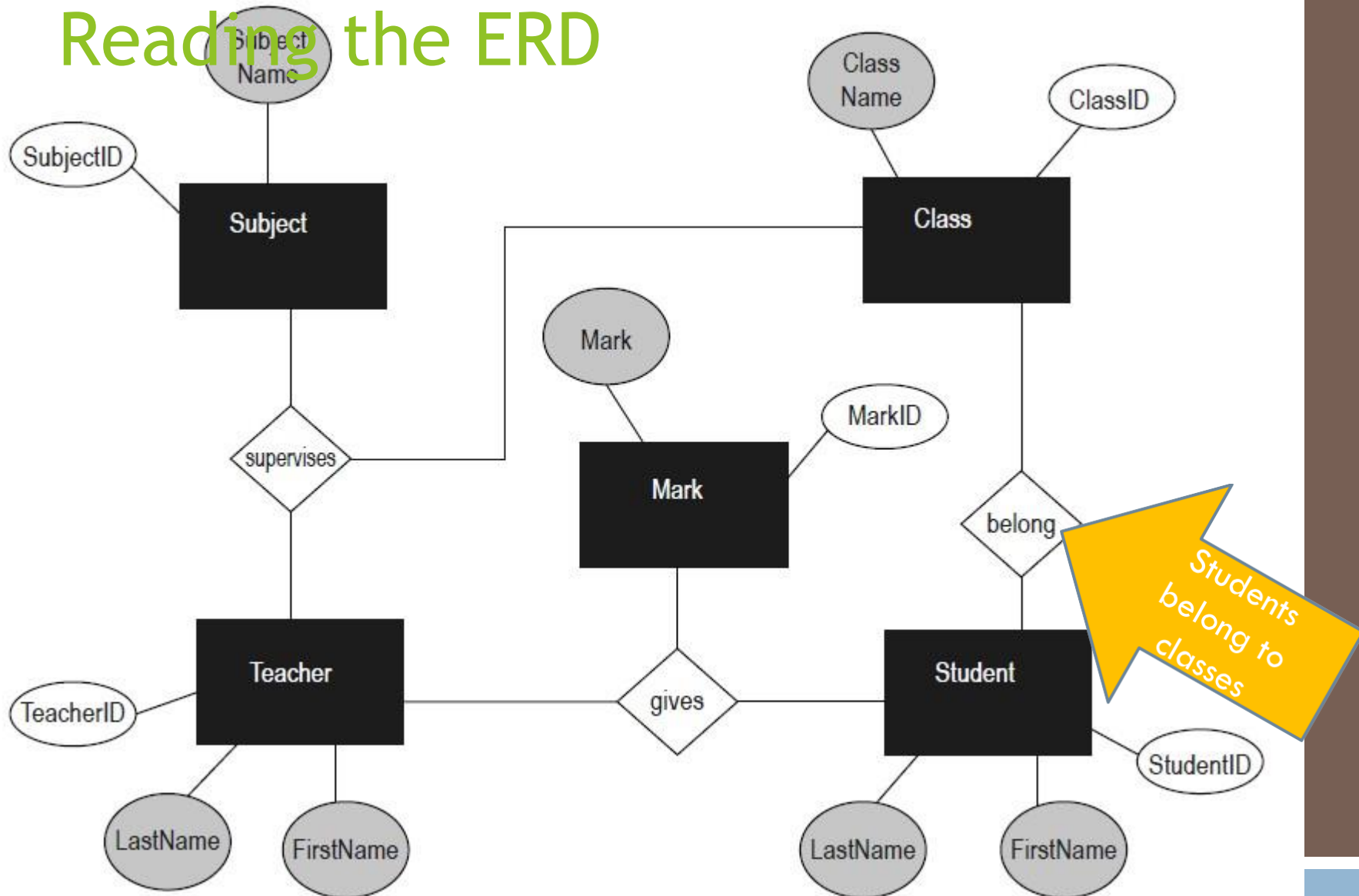




# Reading the ERD



# Reading the ERD



# Decision Tables

- Decision tables are a precise yet compact way to model complicated logic.
- Decision tables, if-then-else and switch-case like statements, associate conditions with actions
- But, unlike the control structures found in traditional programming languages, decision tables perform several actions in an efficient way.

# Decision Tables - Usage

- Decision tables make it easy to observe that all possible conditions are accounted for.
- Decision tables can be used for-
  - ▢ Specifying complex program logic.
  - ▢ Generating test cases (Also known as logic-based testing).

# Decision Tables - Structure

	Decision rules			
	Rule 1	Rule 2	Rule 3	Rule 4
(Condition stub)		(Condition entries)		
(Action stub)		(Action entries)		

# Decision Tables

- Each condition corresponds to a variable, relation or predicate.
- Possible values for conditions are listed among the condition.
- Alternatives-
  - Boolean values (True / False) – Limited Entry Decision Tables.
  - Several values – Extended Entry Decision Tables.
  - Don't care value.
- Each Action is a procedure or operation to perform.
- The entries specify whether (or in what order) the action is to be performed.

# Decision Tables

		Condition entry			
		Rule1	Rule2	Rule3	Rule4
Condition stub	Condition1	Yes	Yes	No	No
	Condition2	Yes	X	No	X
	Condition3	No	Yes	No	X
	Condition4	No	Yes	No	Yes
Action stub	Action1	Yes	Yes	No	No
	Action2	No	No	Yes	No
	Action3	No	No	No	Yes
		Action Entry			

# Decision Tables

	Rule5	Rule6	Rule7	Rule8
Condition1	X	No	Yes	Yes
Condition2	X	Yes	X	No
Condition3	Yes	X	No	No
Condition4	No	No	Yes	X
<b>Default action</b>	Yes	Yes	Yes	Yes



# Decision Table – Example(Printer Troubleshooting)

Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Heck the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

# Advantages of Decision Tables

The various advantages of decision tables include-

- ❑ Decision rules are clearly structured.
- ❑ Managers can be relieved from decision-making.
- ❑ Consistency in decision-making.
- ❑ **Communication is easier between managers and analysts.**
- ❑ Documentation is easily prepared, changed, or updated.
- ❑ Easy to use.
- ❑ Easier to draw or modify compared to flowcharts.
- ❑ Facilitate more compact documentation.
- ❑ Easier to follow a particular path down one column than through complex and lengthy flowcharts.

# Disadvantages of Decision Tables

- The various disadvantages of decision tables include-
  - ❑ Impose an additional burden.
  - ❑ Do not depict the flow.
  - ❑ Not easy to translate.
  - ❑ Cannot list all the alternatives.

# Software Quality

Our objective of software engineering is to produce good quality maintainable software in time and within budget.

Here, quality is very important.

Different people understand different meaning of quality like:

- Conformance to requirements
- Fitness for the purpose
- Level of satisfaction

When user uses the product, and finds the product fit for its purpose, he/she feels that product is of good quality.

# Software Quality Assurance

**Software quality assurance (SQA)** consists of a means of monitoring the software engineering processes and methods used to ensure quality.

Every software developers will agree that high-quality software is an important goal. Once said, "Every program does something right, it just may not be the thing that we want it to do."

# Software Quality Assurance

The definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. **Lack of conformance to requirements** is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. **If the criteria are not followed**, lack of quality will almost surely result.
3. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). **If software conforms to its explicit requirements but fails to meet implicit requirements**, software quality is suspect.

# Verification and Validation

It is the name given to the **checking** and **analysis** process.

The **purpose** is to ensure that the software conforms to its specifications and meets the need of the customer.

**Verification** represents the set of activities that are carried out to confirm that the software correctly implements the specific functionality.

**Validation** represents set of activities that ensure that the software that has built is satisfying the customer requirements.

# Verification and Validation

Verification	Validation
Are we building the product right?	Are we building the right product?
Ensure that the software system meets all the functionality	Ensure that the functionalities meet the intended behavior.
Verifications take place first and include the checking for documentation, code etc	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers	Done by testers
Have static activities as it includes the reviews, walk-throughs and inspections to verify that software is correct or not	Have dynamic activities as it includes executing the software against the requirements.



# Verification and Validation

In the verification and validation, two techniques of system checking and analysis may be used:-

## **1. Software Inspection**

### **1. System testing**

The testing can be carried out using following tests:-

- i. Unit testing
- ii. Module testing
- iii. System testing
- iv. Acceptance testing

# SQA Plans

The SQA Plan provides a **road map** for instituting software quality assurance. Developed by the SQA group, the plan serves as a template for SQA activities that are instituted for each software project.

The documentation section describes (by reference) each of the work products produced as part of the software process. These include –

- project documents (e.g., project plan)
- models (e.g., ERDs, class hierarchies)
- technical documents (e.g., specifications, test plans)
- user documents (e.g., help files)

# Methods for SQA

The methods by which quality assurance is accomplished are many and varied, out of which, two are mentioned as follows:

1. ISO 9000

1. Capability Maturity Model

# ISO 9000

- ▶ “ISO” in greek means “equal”, so the association wanted to convey the idea of equality.
- ▶ It is an attempt to improve software quality based on ISO 9000 series standards.
- ▶ It has been adopted by over 130 countries including India and Japan.
- ▶ One of the **problem** with ISO-9000 series standard is that it is not industry specific.
- ▶ It can be interpreted by the developers to diverse projects such as hair dryers, automobiles, televisions as well as software.
- ▶ ISO-9000 applies to all types of organizations.
- ▶ After adopting the standards, a country typically permits only ISO registered companies to supply goods and services to government agencies and public utilities.
- ▶ ISO-9000 series is not just software standard, but are applicable to a wide variety of industrial activities including design/development, production, installation and servicing.

# ISO 9000 Certification

This process consists of following stages:-

1. Application
2. Pre-assessment
3. Document review and adequacy of audit
4. Compliance audit
5. Registration
6. Continued surveillance

# ISO 9000 Series

The types of software industries to which the different ISO standards apply are as follows:

**ISO 9001** : This standard applies to the organization engaged in **design, development, production & servicing of goods**. This is the standard that is applicable to most **software development organization**.

**ISO 9002** : This standard applies to the organization which **do not design products but only involved in production**. E.g, Car manufacturing industries.

**ISO 9003** : This standard applies to the organization **involved only in installation and testing** of the products.

# Benefits of ISO 9000 Certification

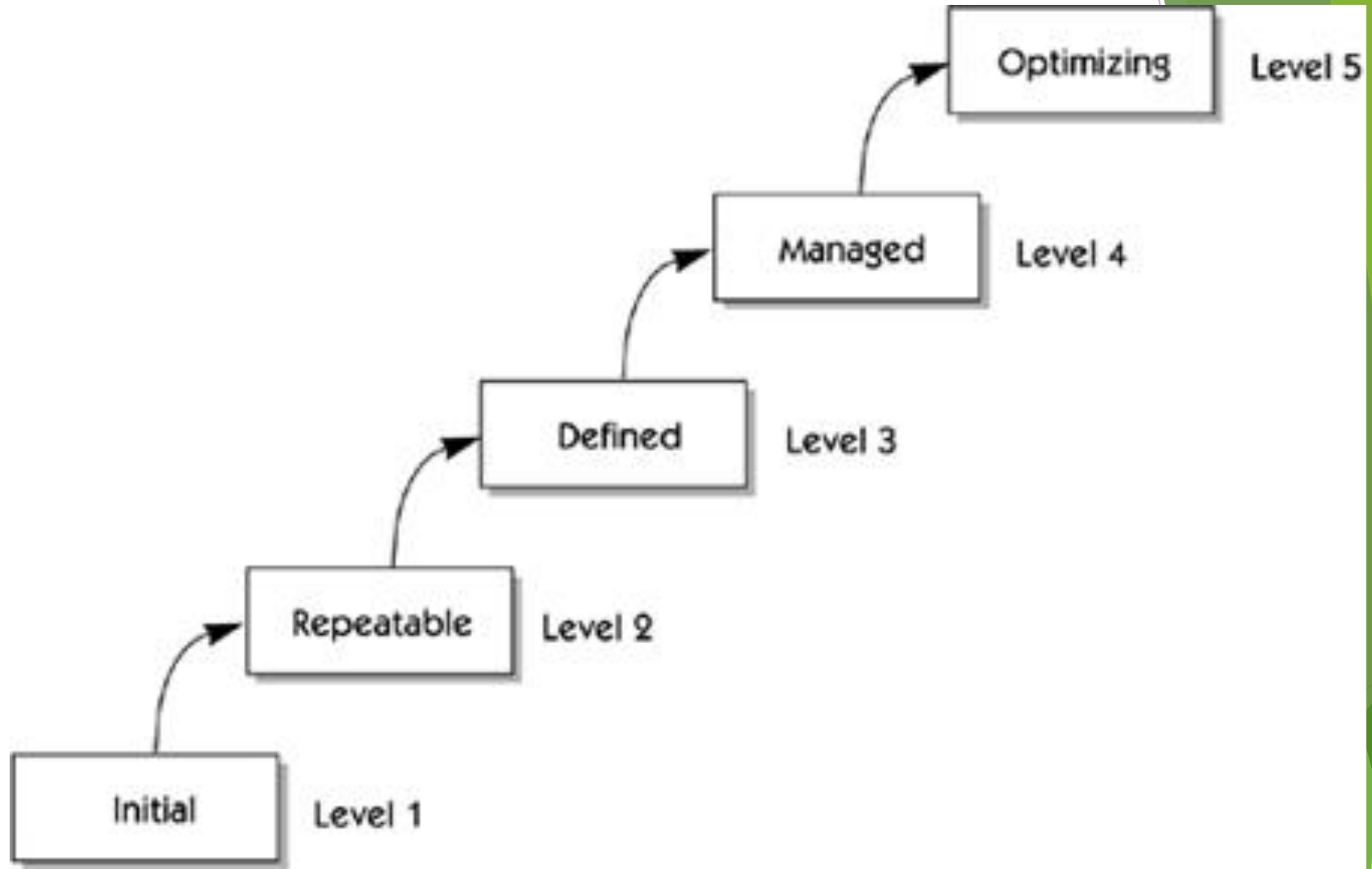
1. Continuous Improvement
2. Improved Customer Satisfaction
3. Eliminate Variations
4. Better product and Services
5. Improved Profit levels
6. Improved Communication
7. Reduced Cost

# Capability Maturity Model

- ▶ It was developed by Software Engineering Institute (SEI) of **Carnegie-Mellon University** in 1986.
- ▶ It specifies an increasing series of levels of a **software development organization**. The higher the level, the better the software development process.
- ▶ It can be used in two ways: **Capability evaluation** and **Software process assessment**.



# CMM Levels



# CMM Levels

- ▶ **Level One :Initial** - The software process is characterized as inconsistent, and occasionally even chaotic. Defined processes and standard practices that exist are abandoned during a crisis. Success of the organization majorly depends on an individual effort, talent, and heroics. The heroes eventually move on to other organizations taking their wealth of knowledge or lessons learnt with them.
- ▶ **Level Two: Repeatable** - This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule, and functionality. The process is in place to repeat the earlier successes on projects with similar applications. Program management is a key characteristic of a level two organization.
- ▶ **Level Three: Defined** - The software process for both management and engineering activities are documented, standardized, and integrated into a standard software process for the entire organization and all projects across the organization use an approved, tailored version of the organization's standard software process for developing, testing and maintaining the application.

# CMM Levels

- ▶ **Level Four: Managed** - Management can effectively control the software development effort using precise measurements. At this level, organization set a quantitative quality goal for both software process and software maintenance. At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable.
- ▶ **Level Five: Optimizing** - The Key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements. At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

# Key Process Areas of each Level

CMM Level	Focus	Key Process Areas
Initial	Competent people	-
Repeatable	Project Management	Software Project Planning Software Configuration Management
Defined	Definition of Processes	Process Definition Training Program Peer Reviews
Managed	Product and process quality	Quantitative Process Metrics Software Quality Management
Optimizing	Continuous Process Improvement	Defect Prevention Process Change Management Technology Change Management

# Comparison of ISO-9000 Certification & SEI CMM

ISO focus on the “customer-supplier relationship”, whereas CMM focus on software supplier to improve its processes to achieve a higher quality product for the benefit of the customer.

ISO 9000 standard is written for a wide range of industries whereas CMM framework is specifies for the software industry.

CMM is a five level framework for measuring software engineering practices, and ISO 9000 defines a minimum level of attributes for a quality management program.

The ISO 9000's concept is to follow a set of standards to make **success repeatable**. The CMM emphasizes a process of **continuous improvement**.

Once an organization has met the criteria to be ISO certified by an independent audit, the next step is only to maintain that level of certification. CMM is an on-going process of evaluation and improvement, moving from one level of achievement to the next. Even at the highest level of maturity, the focus is on continuous improvement.