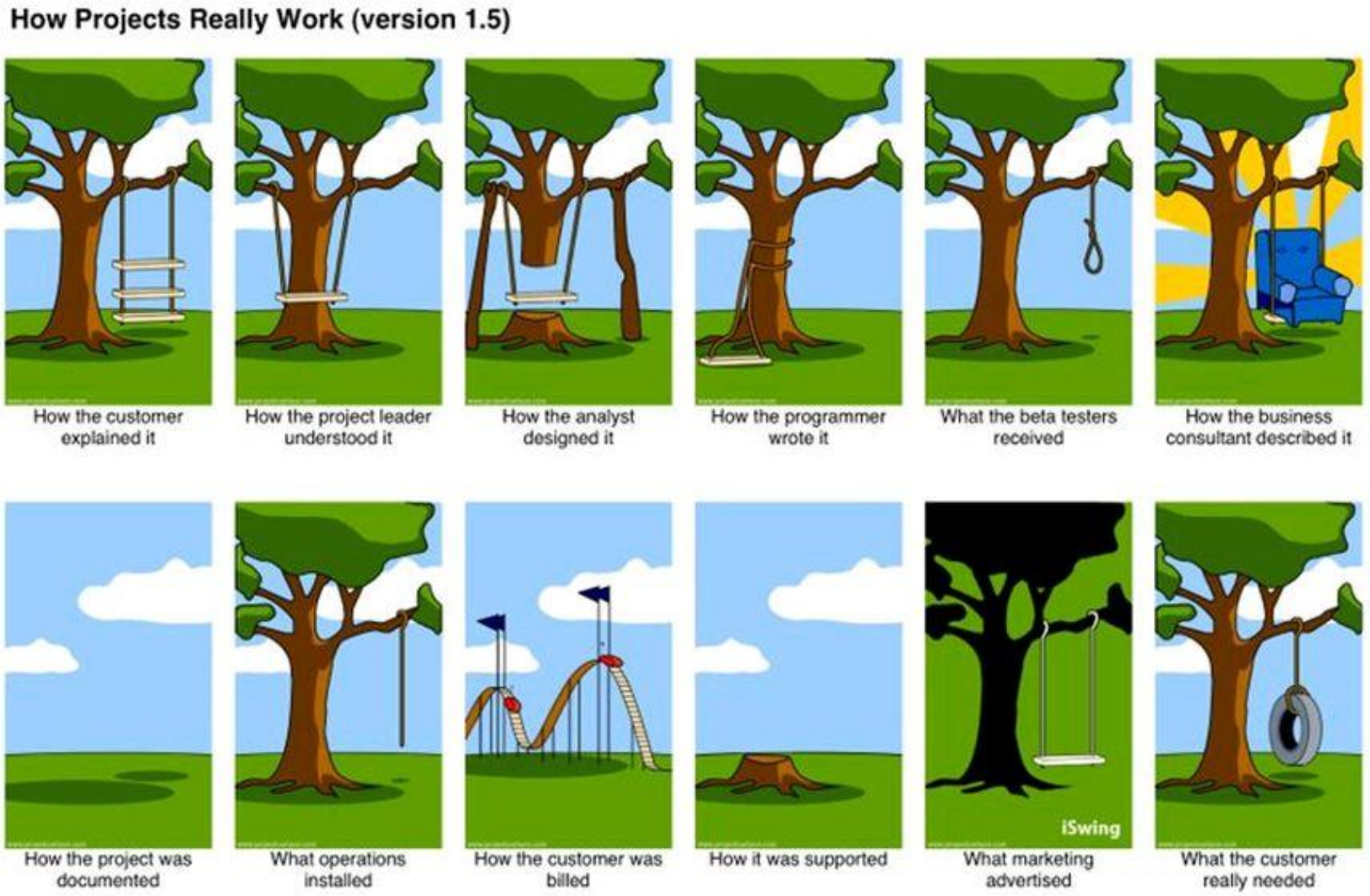
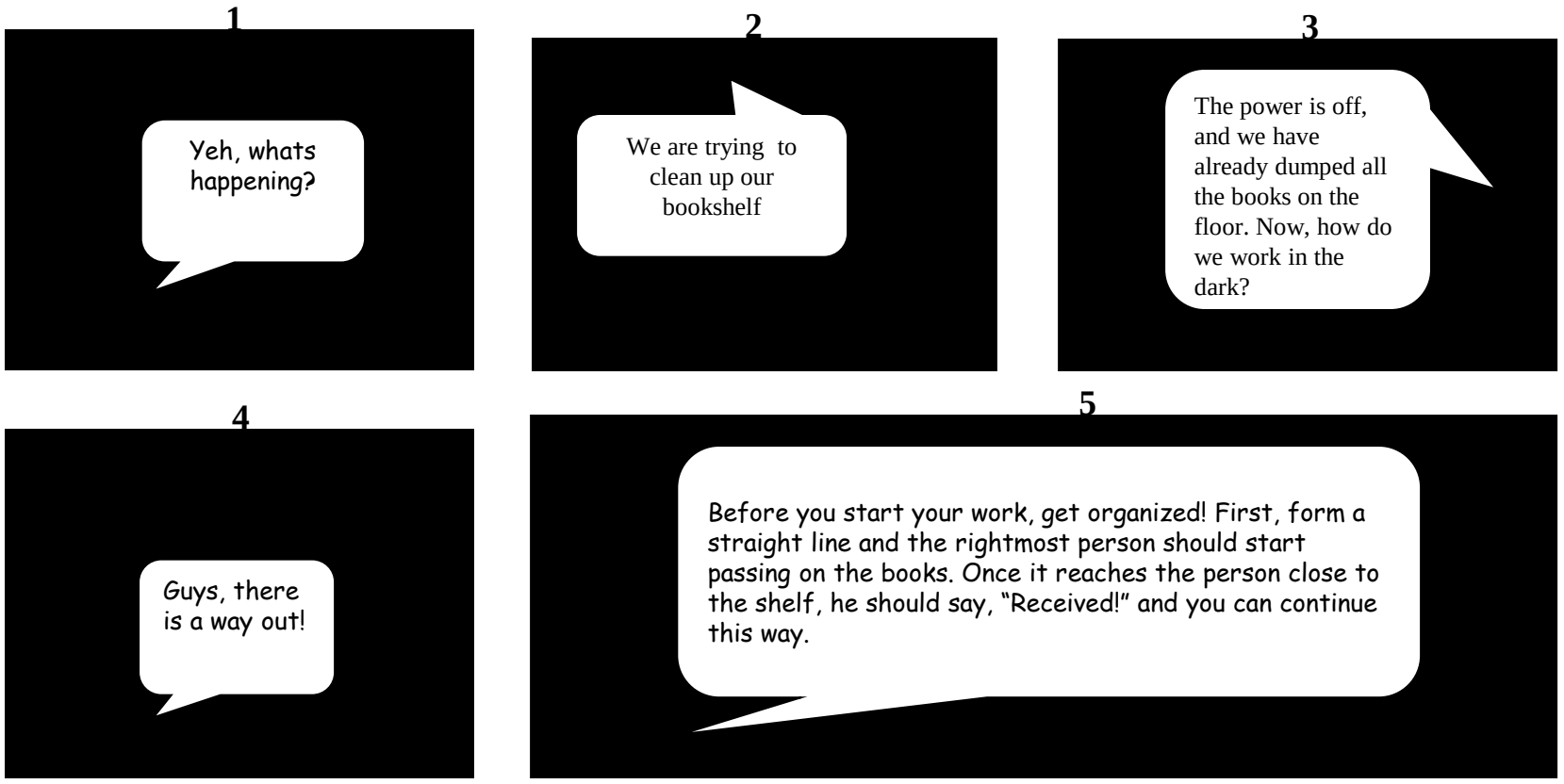
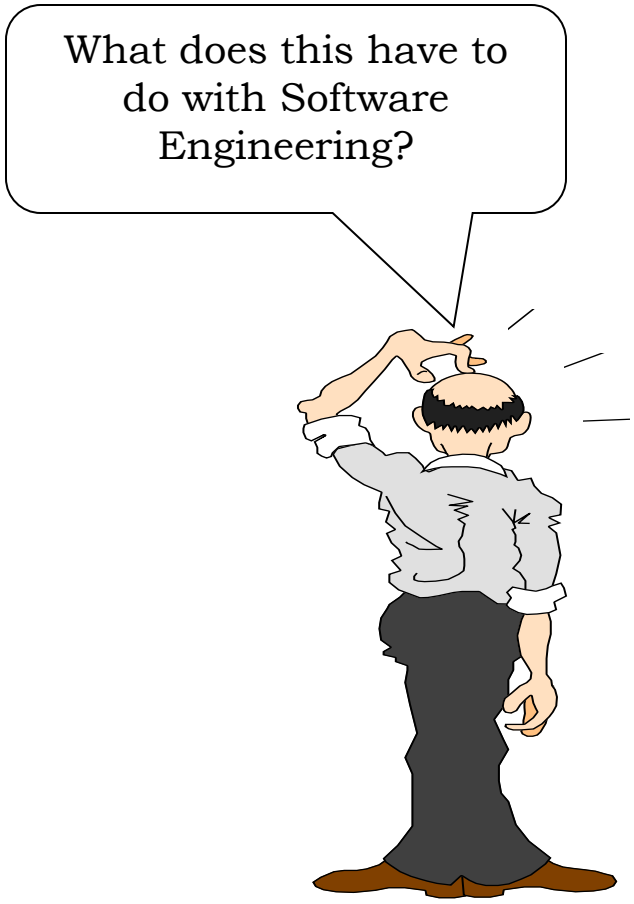
Who, when, how…


How Projects Really Work (version 1.5)
How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
What the beta testers received
How the business consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
iSwing
What marketing advertised
What the customer really needed

What is Software Engineering?



What is Software Engineering?



No matter what!
Organization plays a crucial role in any kind of job.

Software engineering does exactly this to Software Development!

FEASIBILITY STUDY

Whether it is worth carrying on Requirements Engineering and system development process?

All new projects should start with a feasibility study during requirements analysis stage.

1) **Does system contribute to the overall objectives of the organization?**
2) **Can system be implemented in the existing environment and within given cost and schedule constraints?**
3) **Can the proposed system be integrated with the existing systems which are currently active?**

- **How will the organization cope if this system is not implemented?**
- **What are all problems with current processes and how would a new system help to eliminate these problems?**
- **Does system require new technology which has not been previously used in that organization?**
- **Information sources for feasibility study :**
    - **1) Software engineers familiar with system**
    - **2) End Users**
    - **3) Technical Experts**

Feasibility study report may propose changes to scope, budget and schedule of the system.

REQUIREMENTS ENGINEERING

What are Requirements ?

- A condition or capability needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system to satisfy a contract, standard, specification, or other formally imposed document
- A software requirements specification (SRS) is a document containing a complete description of what the software will do without describing how it will do it.

Requirements…

- Software requirements specify what the software must do to meet the business needs.
- For example, a stores manager might state his requirements in terms of efficiency in stores management.
- A bank manager might state his requirements in terms of time to service his customers.
- It is the analyst's job to understand these requirements and provide an appropriate solution.
- To be able to do this, the analyst must understand the client's business domain: who are all the stake holders, how they affect the system, what are the constraints, etc

Requirements…

- The analyst should not blindly assume that only a software solution will solve a client's problem.
- He should have a broader vision.
- Sometimes, re-engineering of the business processes may be required to improve efficiency
- A detailed statement of what the software must do to meet the client's needs should be prepared. This document is called Software Requirements Specification (SRS) document.

SOURCES OF REQUIRMENTS

- Stakeholder wants and needs
- Current organization and systems
- Existing documents
- Suggested types of requirements
- Reusable requirements
- Domain models
- Current situation model

Template for SRS

1. Introduction
    1.1 Purpose of requirements document
    1.2 Scope of the product
    1.3 Definitions, acronyms and abbreviations
    1.4 References
    1.5 Overview of the remainder of the document
2. General description
    2.1 Product perspective
    2.2 Product functions
    2.3 User characteristics
    2.4 General constraints
    2.5 Assumptions and dependencies
3. Specific requirements covering functional, non-functional and interface requirements.
4. Appendices
5. Index

SOFTWARE REQUIREMENTS SPECIFICATIONS

Uses of a Requirements document

- Customers – Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements.
- Managers- Use the document to plan a bid for the system and to plan the system development process
- System engineers – Use the requirements to understand what system is to be developed
- System test engineers –Use the requirements to develop validation tests for the system
- System Maintenance engineers-Use the requirements to help understand the system and the relationships between its parts

Characteristics of a Good SRS

- Correct (No errors)
- Unambiguous
- Complete
- No use of TBDs *(To Be Determined)*
- Verifiable (words like “highly”, “usually” not to be used)
- Consistent (non-conflict)
- Modifiable
- Traceable

TYPES OF REQUIREMENTS

Physical Environment

- Where is the equipment to function ?
- Is there one location or several ?
- Are there any environmental restrictions, such as temperature, humidity, or magnetic interference ?

Interfaces

* Is the input coming from or more other systems ?

* Is the output to one or more other systems ?

- Is there a prescribed way in which the data must be formatted ?
- Is there a prescribed medium that the data must use ?

Users and Human factors

* Who will use the system ?

- Will there be several types of users ?
- What is the skill level of each type of user ?
- What kind of training will be required for each type of user ?
- How easy will it be for a user to understand and use the system?
- How difficult will it be for a user to misuse the system ?

Functionality

* What will the system do ?

* When will the system do it ?

* Are there several modes of operation ?

* How and when can the system be changed or enhanced ?

* Are there constraints on execution speed, response time ?

Documentation

*How much documentation is required?

*Should it be on-line, in book format or both ?

* To what audience is each type of documentation addressed ?

Data

For both input and output, what should be format of the data be ?

How often will they be received or sent ?

How accurate must they be ?

How much data flow through the system ?

Must any data be retained for any period of time ?

Resources

- What materials, personnel or other resources are required to build, use and maintain the system ?
- What skills must the developers have ?
- What are the requirements for power, heating or air conditioning ?
- Is there a limit on the amount of money to be spent on development or on hardware and software ?

Security

Must access to the system or to information be controlled ?

How often will the system be backed up ?

Should precautions be taken against fire, water damage or theft ?

Quality Assurance

What are the requirements for reliability, availability, maintainability, security and other quality attributes ?

How the characteristics of the system be demonstrated to others ?

Must the system isolate and detect faults ?

What is the MTBF ?

Is there a maximum time allowed for restarting the system after a failure?

Library Problem

- Check out a copy of a book, return a copy of a book
- Add a copy of a book to the Library
- Remove a copy of a book from the library
- Get the list of books by a particular author or in a particular subject area
- Find out the list of books currently checked out by particular borrower
- Find out which borrower last checked out a particular copy of a book
- There are two types of users : Staff Users and Ordinary Borrowers. Transactions 1 and 2 are restricted to staff users

CONSTRAINTS

1. Copies in the library must be available for checkout or be checked out
2. Copy of the book may be both available and checked out at the same time
3. A borrower may not have more than a predefined number of books checked out at one time

Ambiguities

- What is a Library ?
- Who is a user ?
- Confusion regarding *" Book "* and *" Copy "*
- What does *" Available "* mean ?
- What do *" Last Checked Out "* and *" Currently "* mean ?

Incompleteness

- Initialization
- Addition of a new book
- Remove all copies of a book
- Create a library
- Add a staff user / ordinary user
- Error Handling
- Missing Constraints
  - Period of Lending
  - Not more than one copy of the same book can be borrowed

Further Examples for Bad Specification

- The counter value is picked up from the last record.
- Inversion of a square, matrix ‘M’ of size ‘n’ such as LM ML = I_n, where ‘L’ is the inverse matrix and I_n is the identity matrix of size ‘n’.
- The software should be highly user friendly
- The output of the program shall usually be given within 10 seconds.
- The software should be developed on DOS system, but should be portable to other systems.

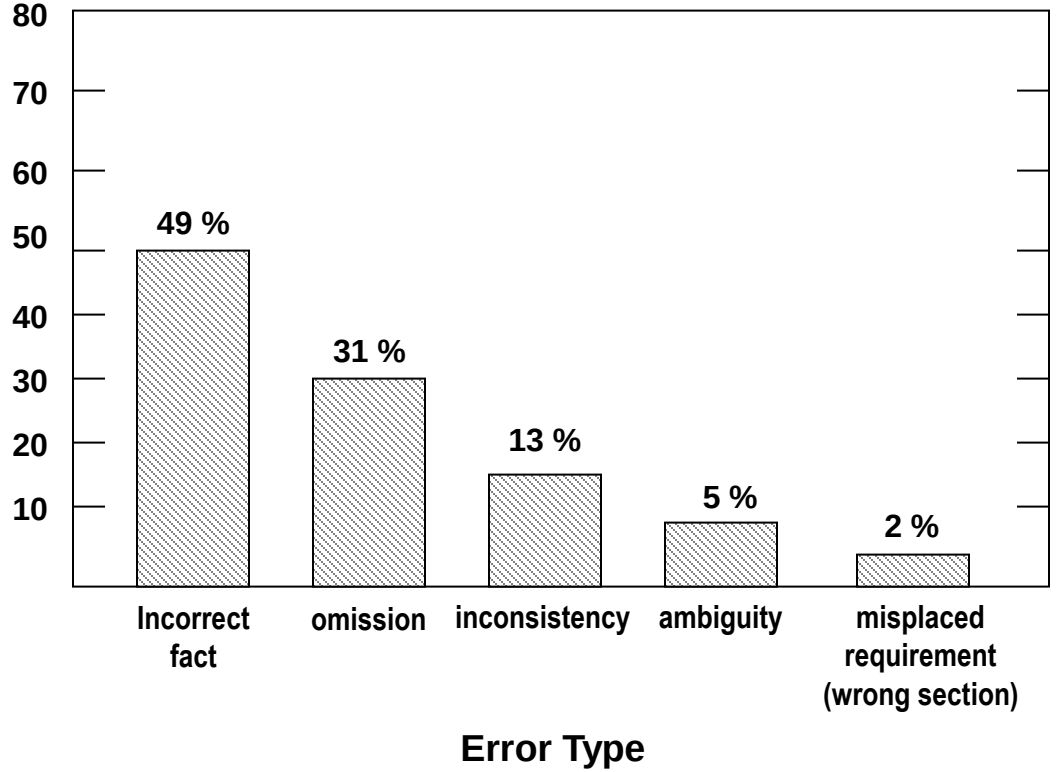
Further Examples for Bad Specification

- In first statement, the word 'last' is ambiguous. It could mean the last accessed record, which could be anywhere in a random access file, or, it could be physically the last record in the file
- Second statement though appears to be complete, is missing on the type of the matrix elements. Are they integers, real numbers, or complex numbers. Depending on the answer to this question, the algorithm will be different.
- How does one determine, whether this requirement is satisfied or not.
- What are the exceptions to the 'usual 10 seconds' requirement?

How Users and Developers view each other ?

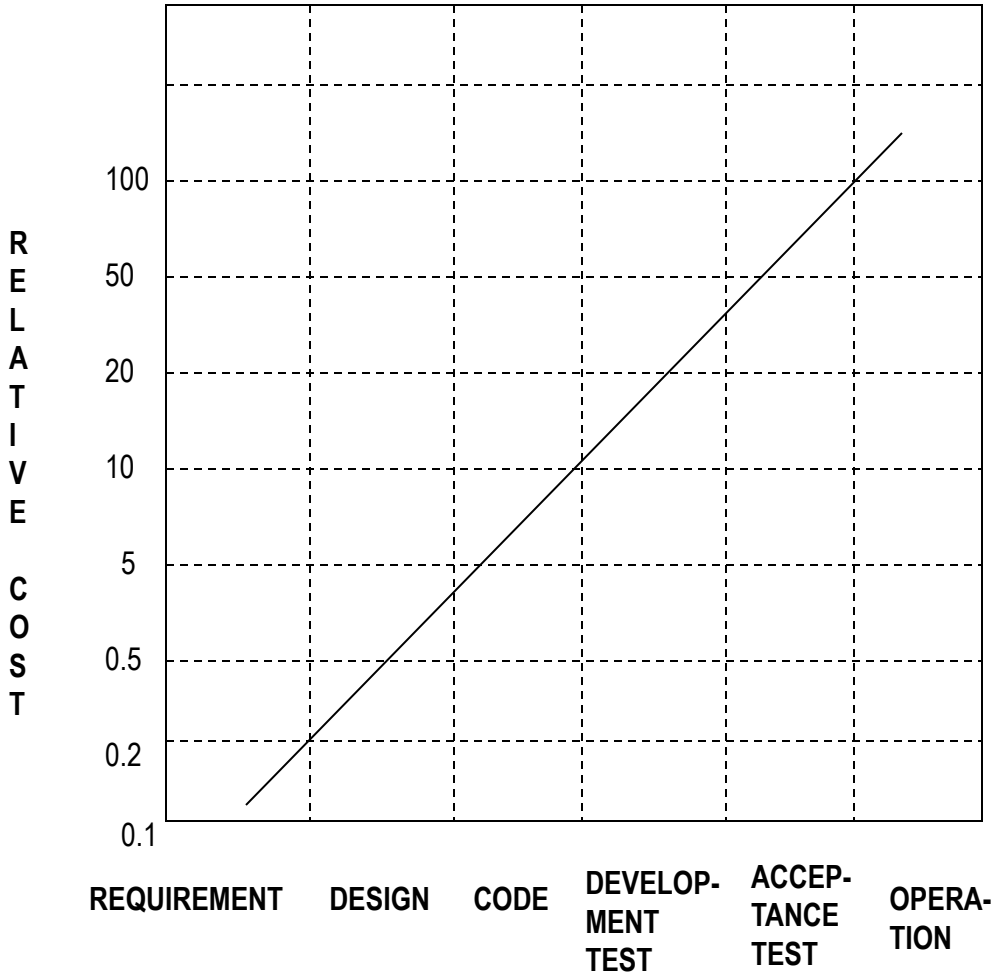
How developer see users	**How users see developers ?**
Users don't know what they want.	Developer's don't understand operational needs
Users want everything right now.	Developer's can't translate clearly stated needs into a successful system
User's can't prioritize needs.	Developers say no all the time.
Users refuse to take responsibility for the system	Developers are always late.
Users are unwilling to compromise	Developers are unable to respond
User's can't remain on schedule	quickly to legitimately changing needs
User's are unable to provide a Usable statement of needs	Developers try to tell us how to do our job
	Developer's can't translate clearly stated needs into a system

Distribution of Errors in Requirements Specification



[NAVY A - 7E AIRCRAFTS OPERATIONAL FLIGHT PROGRAM; 5th INT. CONF. ON SOFTWARE ENGINEERING, 1981]

Relative Cost To Fix An Error



The Requirements Definition and Analysis Phase is Critical to keep the Development and Maintenance Costs to a Minimum

Types Of Requirements

- Functional Requirements
  - what system should do

The requirements that specify the Inputs (Stimuli) to the system, the Outputs (Responses) from the system, and the behavioral relationship between them

- Non-Functional Requirements
  - specify the overall quality attributes the system must satisfy.

Functional Requirements (Examples)

- To calculate the compound interest @ 8% per annum on a Fixed Deposit for a period of three years
- To calculate the Tax @ 30% on an annual income equal to and above Rs.2,00,000 but less than Rs.3,00,000
- To invert a Square Matrix (Maximum size 100 X 100) of Real Numbers

Non-Functional Requirements

- The requirements that describe the overall attributes of the system :
  - Portability
  - Reliability
  - Performance
  - Testability
  - Modifiability
  - Security
  - Presentation
  - Reusability
  - Understandability
  - Acceptance Criteria

Non-Functional Requirements (Examples)

Performance

- Number of significant digits to which accuracy should be maintained in a numerical solution
- Maximum response time in a transaction processing system
- Delays and task completion time limits in a real-time system

Environment

- To run the software under a given operating system or use a specific programming language

Security

- The addition and deletion of the book in the system can be carried out by the Library-Chief only

Types of Satisfiability

- Normal Requirements
  * User responses to specific questions
  * Satisfy / dissatisfy in proportion to their presence / absence in system

- Expected Requirements

- Exciting Requirements

Expected Requirements

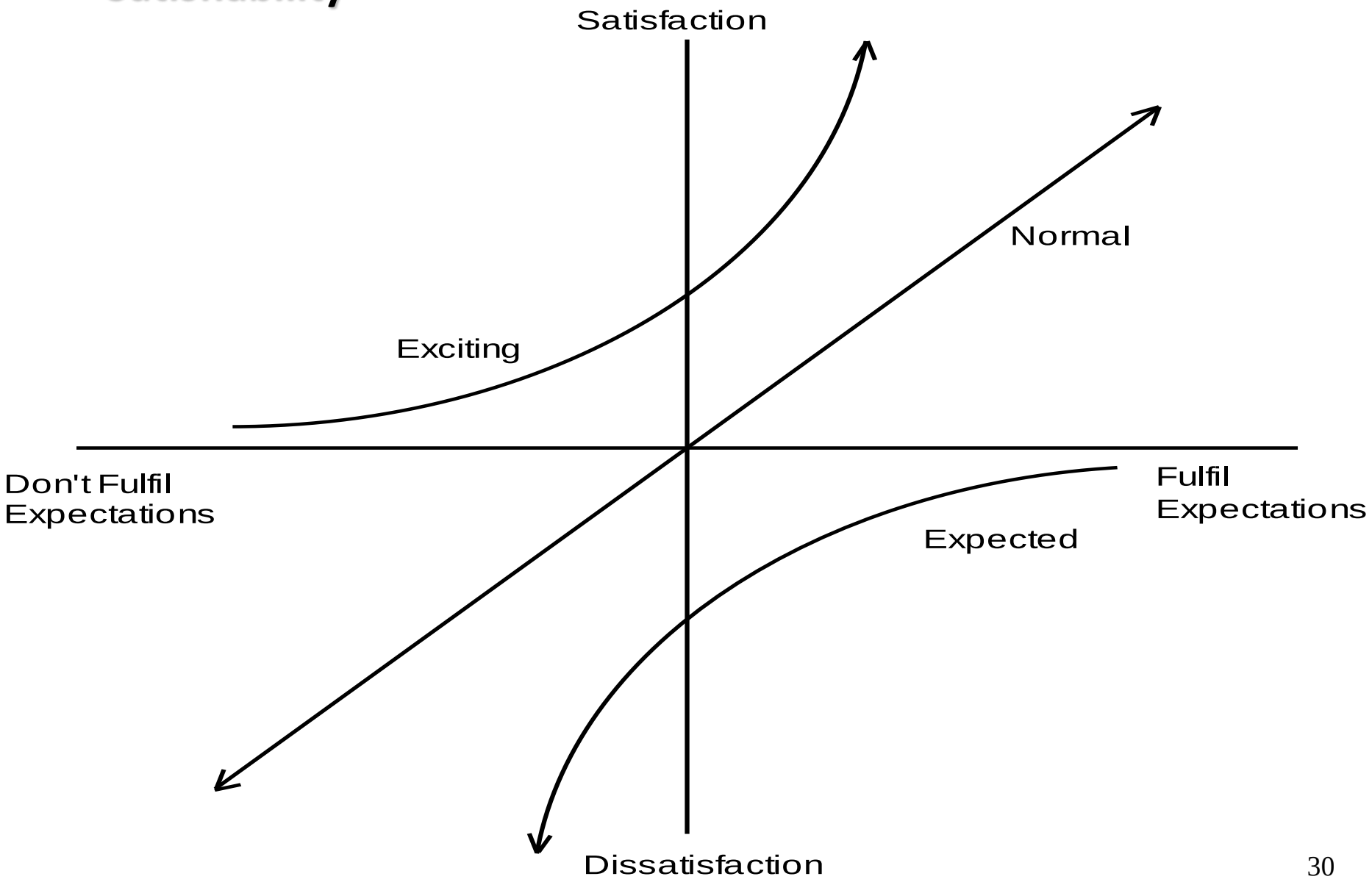
Expected requirements may not be stated by the users, but the developer is expected to meet them. **S**o basic that users may neglect to mention them

- Their presence in solution may meet expectation but not enhance satisfaction
- Absence is very dissatisfying
- "Most Important" for analyst

Exciting Requirements

- Beyond the users' expectations; cannot be elicited from users
- Highly satisfying when met
- Their absence doesn't dissatisfy because they are not expected

Satisfiability



Normal, Expected and Exciting Requirements

The Trend over the Years

- Exciting requirements often become normal requirements

- Some normal requirements become expected requirements
- For example, on-line help feature was first introduced in the UNIX system in the form of *man* pages. At that time, it was an exciting feature. Later, other users started demanding it as part of their systems. Now a days, users do not ask for it, but the developer is expected to provide it.

Modeling Requirements

- Every software system has the following essential characteristics:
- It has a boundary. The boundary separates what is with in system scope and what is outside
- It takes inputs from external agents and generates outputs
- It has processes which collaborate with each other to generate the outputs
- These processes operate on data by creating, modifying, destroying, and querying it
- The system may also use data stores to store data which has a life beyond the system

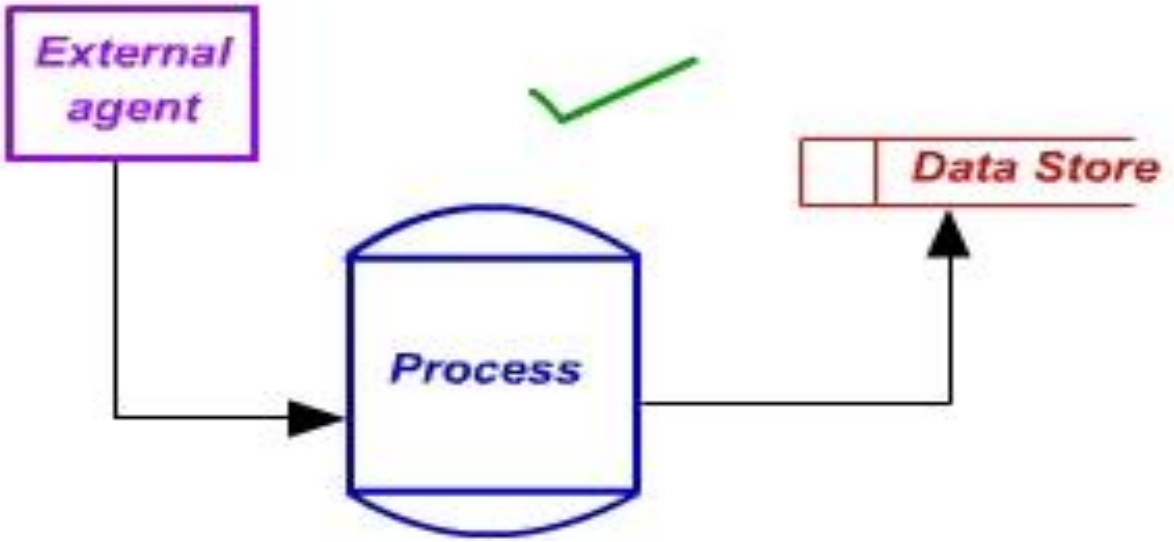
SSADM tools

- **Data Flow Diagram (DFD) for modeling processes and their interactions.**
- **Entity Relationship Diagram (ERD) for modeling data and their relationships.**
- **Decision Tables and Decision Trees to model complex decisions.**
- **State Transition Diagram to model state changes of the system.**
- **Data Dictionary is a repository of various data flows defined in a DFD.**

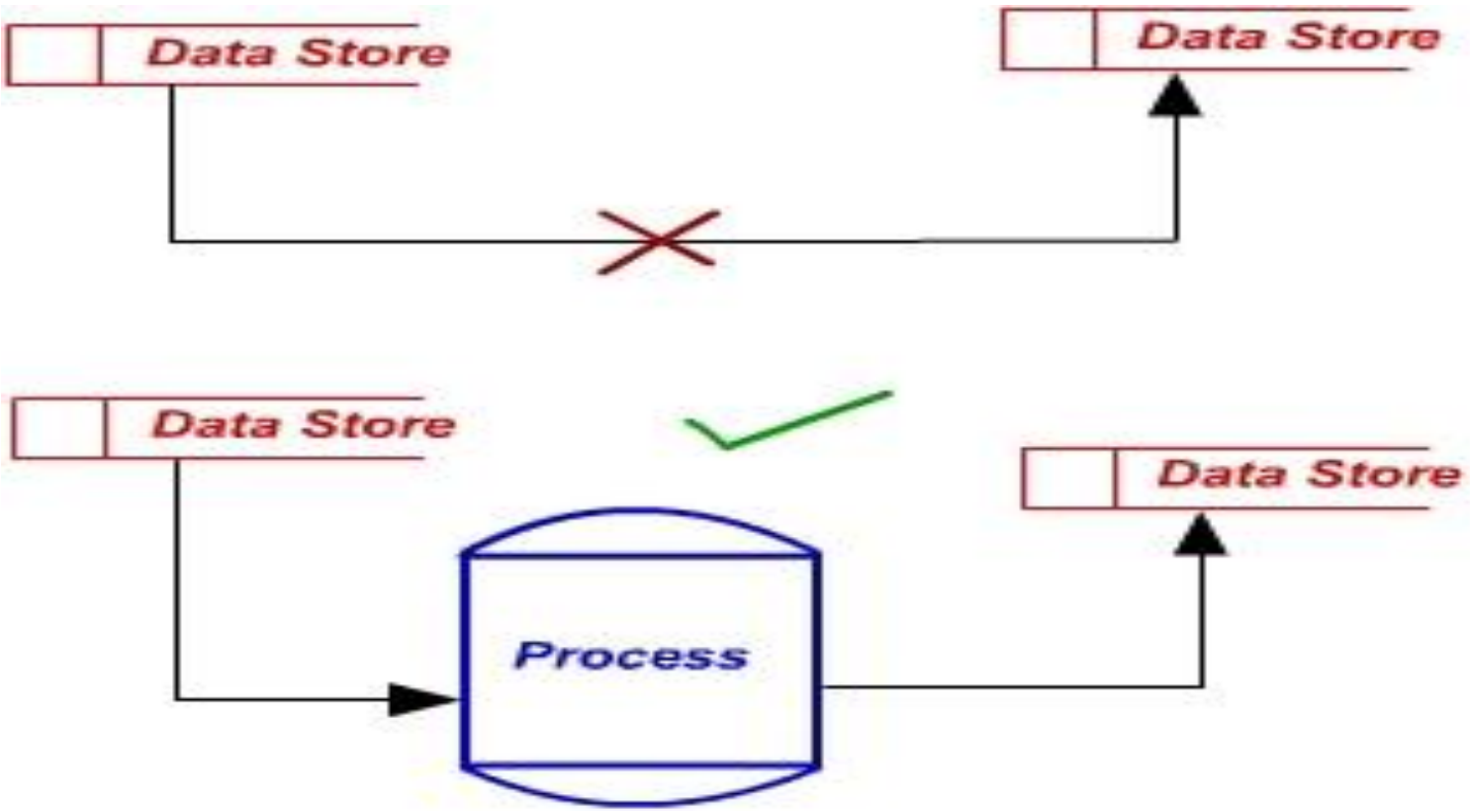
Points to remember

- 1) Remember to name every external agent, every process, every data store, and every dataflow.
- 2) Do not show how things begin and end.
- 3) Do not show loops, and decisions.
- 4) Do not show data flows between external agents. They are outside the scope of the system.
- 5) Do not show dataflow between an external agent and a data store. There should be a process in between.

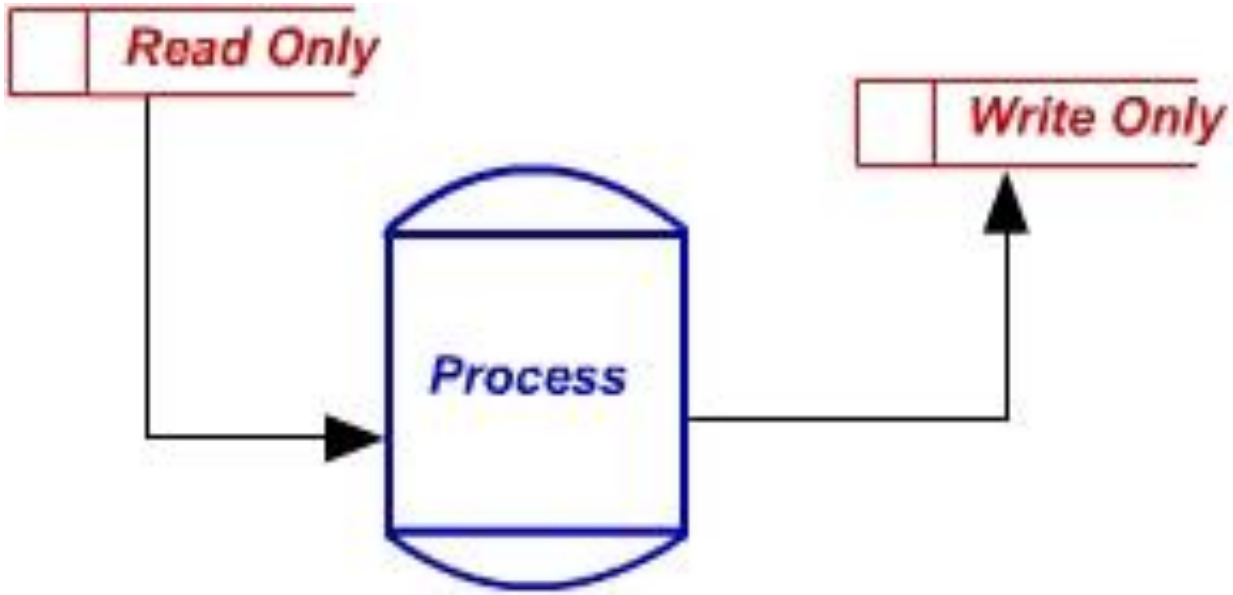
Points to remember



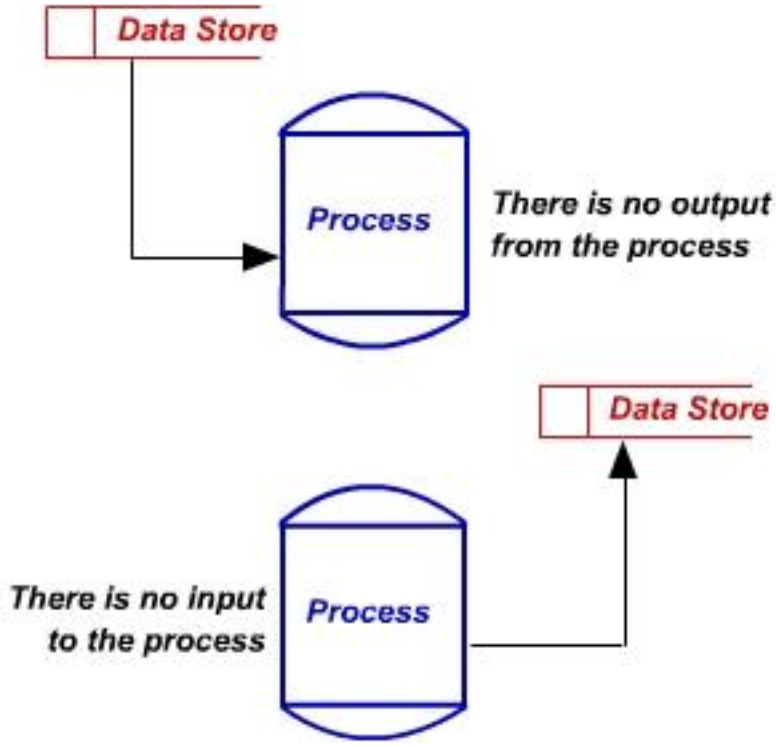
- **6) Do not show dataflow between two data stores. There should be a process in between.**



- **7) There should not be any unconnected external agent, process, or data store.**
- **8) Beware of read-only or write-only data stores**



- **9) Beware of processes which take inputs without generating any outputs. Also, beware of processes which generate outputs spontaneously without taking any inputs**



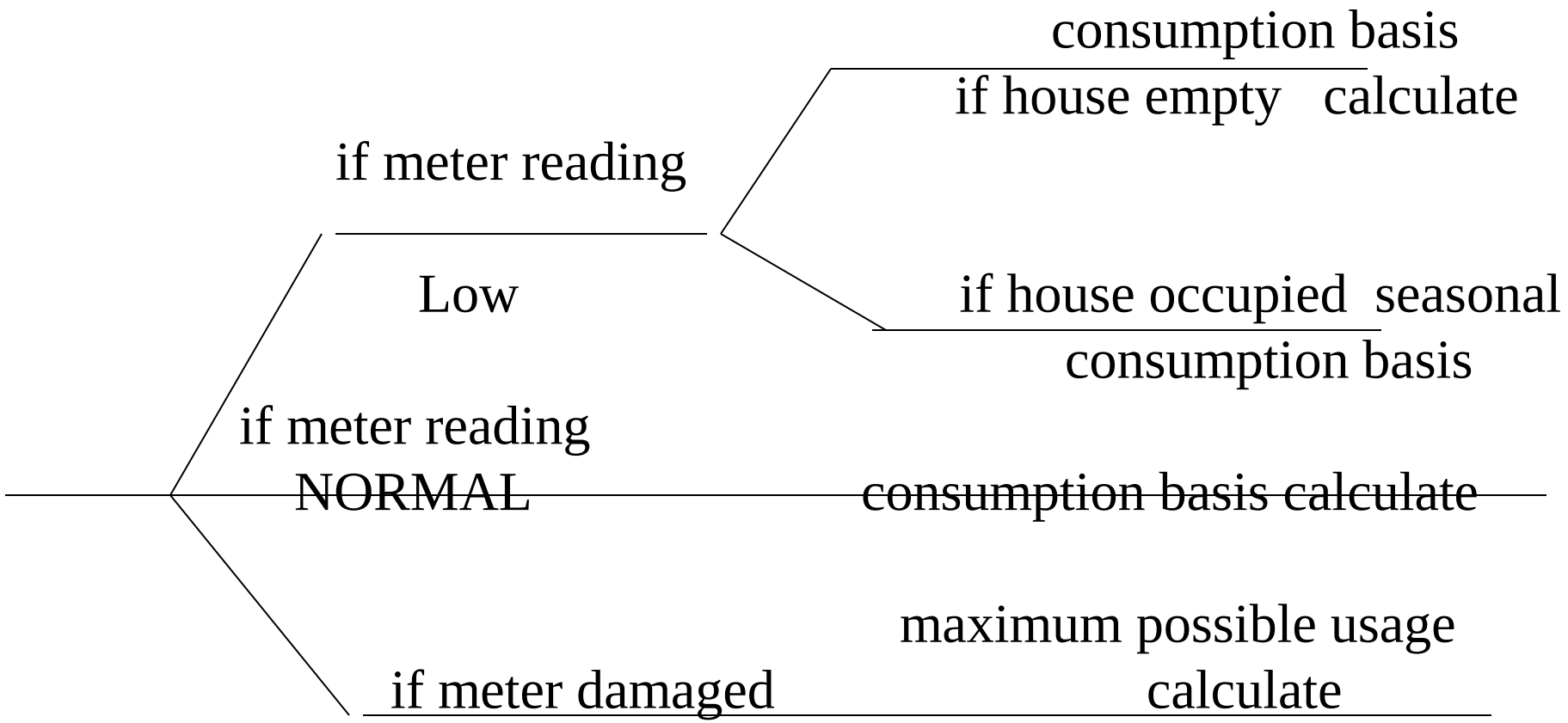
- **10) Ensure that the data flowing in to a process exactly matches the data flowing in to the exploded view of that process. Similarly for the data flowing out of the process.**
- **11) Ensure that the data flowing out of a data store matches data that has been stored in it before.**

DECISION TREE

- **A decision tree represents complex decisions in the form of a tree. Though visually it is appealing, it can soon get out of hand when the number and complexity of decisions increase. An example is given below.**
  First the textual statement is given and then the corresponding decision tree is given:

  Rules for electricity billing are as below:
- **If the meter reading is "OK", calculate on consumption basis(i.e. meter reading)**
  If the meter reading appears "LOW", then check if the house is occupied
  If the house is occupied, calculate on seasonal consumption basis otherwise calculate on consumption basis
  If the meter is damaged, calculate based on maximum possible electricity usage

DECISION TREE



DECISION TABLE

- **There are two types of decision tables, binary-valued(yes or no) and multi-valued. An example follows:**

  ELECTRICITY BILL CALCULATION BASED ON CUSTOMER CLASS

  If a customer uses electricity for domestic purposes and if the consumption is less than 300 units per month then bill with minimum monthly charges.
  Domestic customers with a consumption of 300 units or more per month are billed at special rate.
  Non-domestic users are charged double that of domestic users (minimum and special rates are double).

BINARY VALUED DECISION TABLE

Domestic Customer	**Y**	**Y**	**N**	**N**
Consumption < 300 units/month	**Y**	**N**	**Y**	**N**
Minimum Rate	**Y**	**N**	**N**	**N**
Special Rate	**N**	**Y**	**N**	**N**
Double Minimum Rate	**N**	**N**	**Y**	**N**
Double Special Rate	**N**	**N**	**N**	**Y**

MULTIVALUED DECISION TABLE

CUSTOMER	D	D	N	N
CONSUMPTION	>=300	<300	>=300	<300
RATE	S	M	2S	2M

- **Like decision trees, binary-value decision tables can grow large if the number of rules increase. Multi-valued decision tables have an edge. In the above example, if we add a new class of customers, called Academic, with the rules:**

  If the consumption is less than 300 units per month then bill with concessional rates. Otherwise bill with twice the concessional rates. then new tables will look like the following:

BINARY VALUED DECISION TABLE

ACADEMIC	N	N	N	N	Y	Y
DOMESTIC CUSTOMER	Y	Y	N	N	N	N
CONSUMPTION < 300UPM	Y	N	Y	N	Y	N
MINIMUM RATE	Y	N	N	N	N	N
SPECIAL RATE	N	Y	N	N	N	N
TWICE MINUMUM RATE	N	N	Y	N	N	N
TWICE SPECIAL RATE	N	N	N	Y	N	N
CONCESSION RATE	N	N	N	N	Y	N
TWICE CONCESSION RATE	N	N	N	N	N	Y

MULTIVALUED DECISION TABLE

CUSTOMER	DOMESTIC	DOMESTIC	NONDOMESTIC	NONDOMESTIC	ACADEMIC	ACADEMIC
CONSUMPTION	>=300	<300	>=300	<300	>=300	<300
RATE	SPL	MIN	TWICE SPL	TWICE MIN	TWICE CONC	CONCESSIONAL

Requirements and design

- In principle, **requirements should state** what the system should **do and the design should describe** how it does this
- In practice, **requirements and design are inseparable**

What Case Tools Can Do ?

- Graphic Tool
  - *Data Flow Diagram*
  - *Flowchart*
  - *Entity Relationship Diagram*
  - *Structure Charts*
  - *State-Transition Diagrams*
- Dictionary Tools
  - *Contents of Files*
  - *Inputs and Outputs*
  - *Properties of Data Elements*
  - *Logic Rules for Processes*

What Case Tools Can Do ?

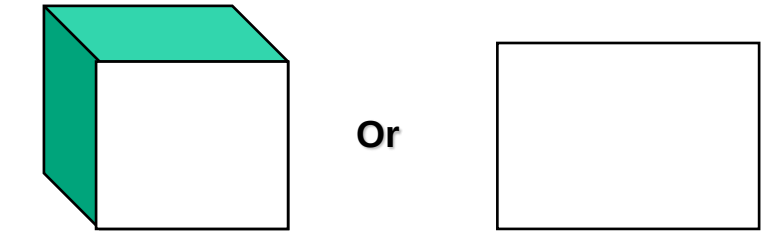
- Code Generators
- Cost-Benefit Analysis Tools
- Project Management Tools
- Configuration Management Tools
- Documentation Assemblers
  - Technical and Non-Technical
  - With Interfaces to popular Word Processors
- Test Data Generators
- Path Coverage Analyzers

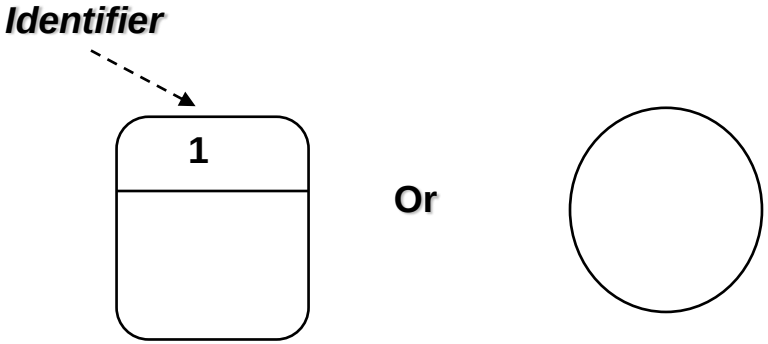
Data-Flow diagrams

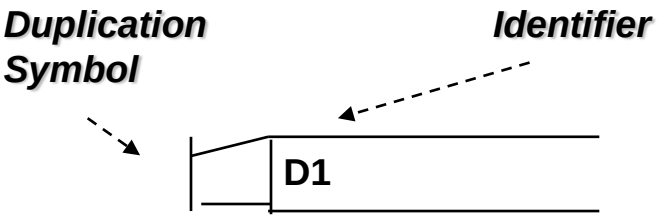
- **A graphical representation of the system**
- **Used to study how information enters a system, and how it is transformed as it flows through the system**
- **Uses symbols to represent**
  - External entities
  - Processes
  - Data flow
  - Data store

Data-Flow diagrams

- **DFD Notation**







External Entity

(A source of data to the system or a destination from the system)

Process

(This represents one or more activities which modify data)

Data Flow

(Represents passage of data between processes)

Data Store

(Represents a place where data is kept within the system)

Data-Flow diagrams

- **Level of DFDs**
  - Level 0 - Overview showing scope
    (5-10 subsystems/processes)
  - Level 1 - Working model, whole system
    All “normal” processing
    (20 - 50 processes)
  - Level 2 - Explosion of 1 process on DFD/1
    Shows normal, error and exception processing
    (5-15 processes)

Data-Flow diagrams

- **DFD using Structured Analysis**

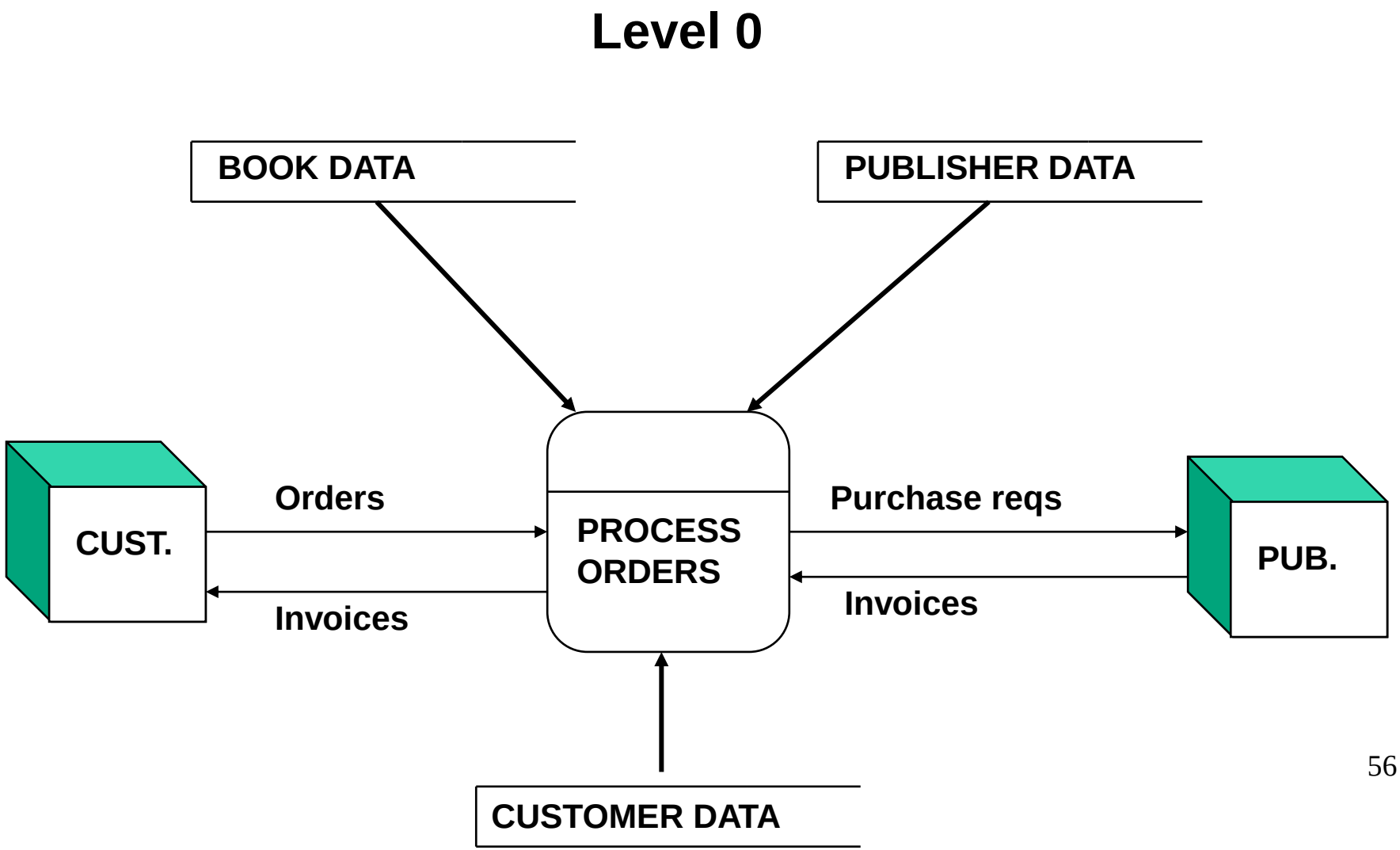
- **The following guidelines are to be followed while constructing DFDs using the structured analysis method:**
  - Study the physical environment of the existing system
  - Construct a DFD for the current non-automated system
  - Construct a logical DFD by using the physical DFD as the basis
  - Construct the logical DFD of the proposed system to show how data will flow in the new system
  - Establish a man machine boundary by specifying what needs to be automated and what will remain manual for the new system in the DFD
  - Present the evaluated specifications

Data-Flow diagrams

- **Graphical Guidelines for DFDs**
  - Identify external entities
  - Identify inputs and outputs (to and from external entities)
  - Produce first draft of DFD
  - Compare first draft with the first two points mentioned above
  - Check each data store: Is it created and maintained ?
  - Produce second draft
  - Conduct walkthroughs
  - Produce explosions
  - Produce final DFD

Data-Flow diagrams

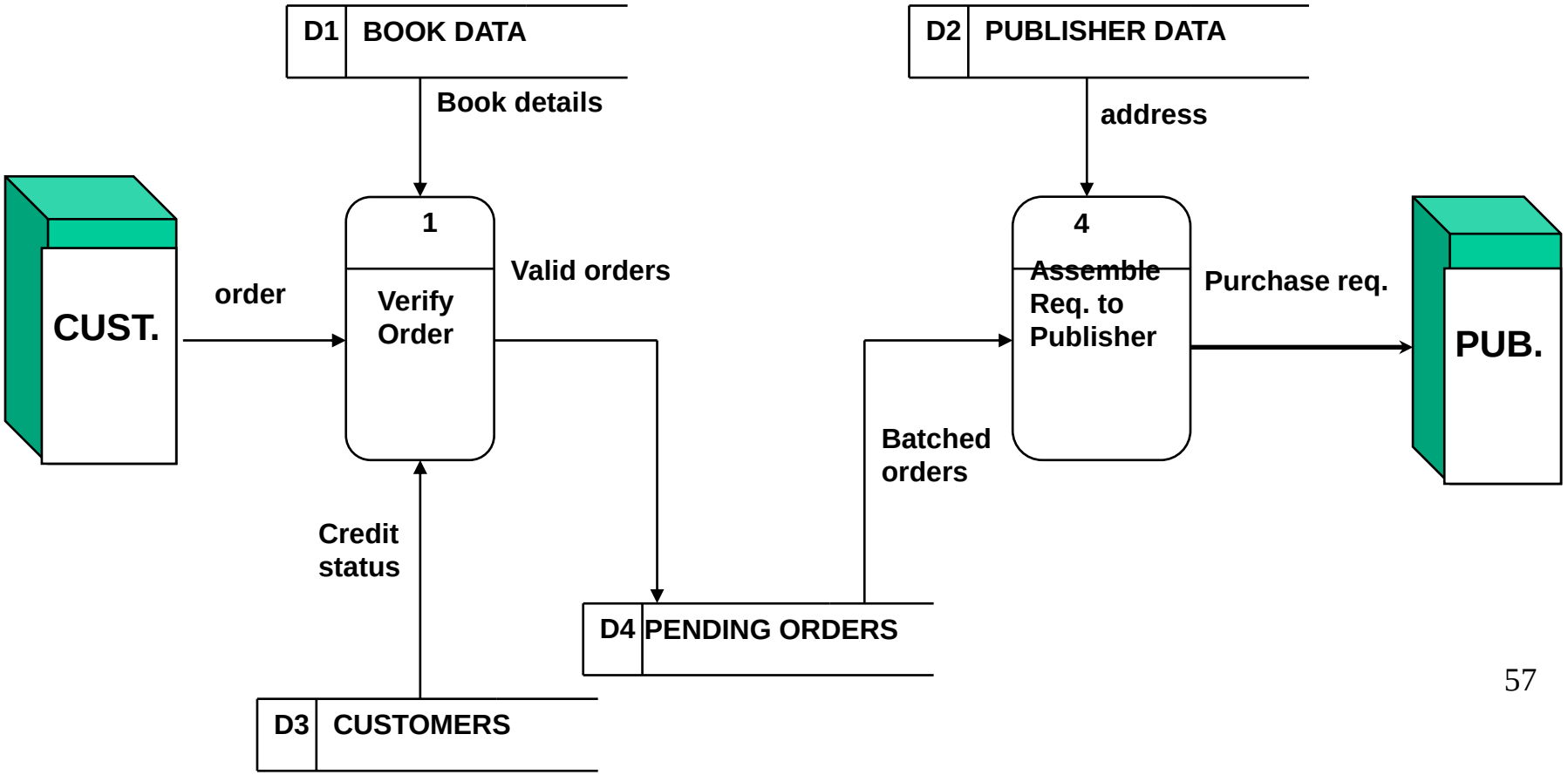
- **Sample Level 0 DFD - Computer books by Mail**



Data-Flow diagrams

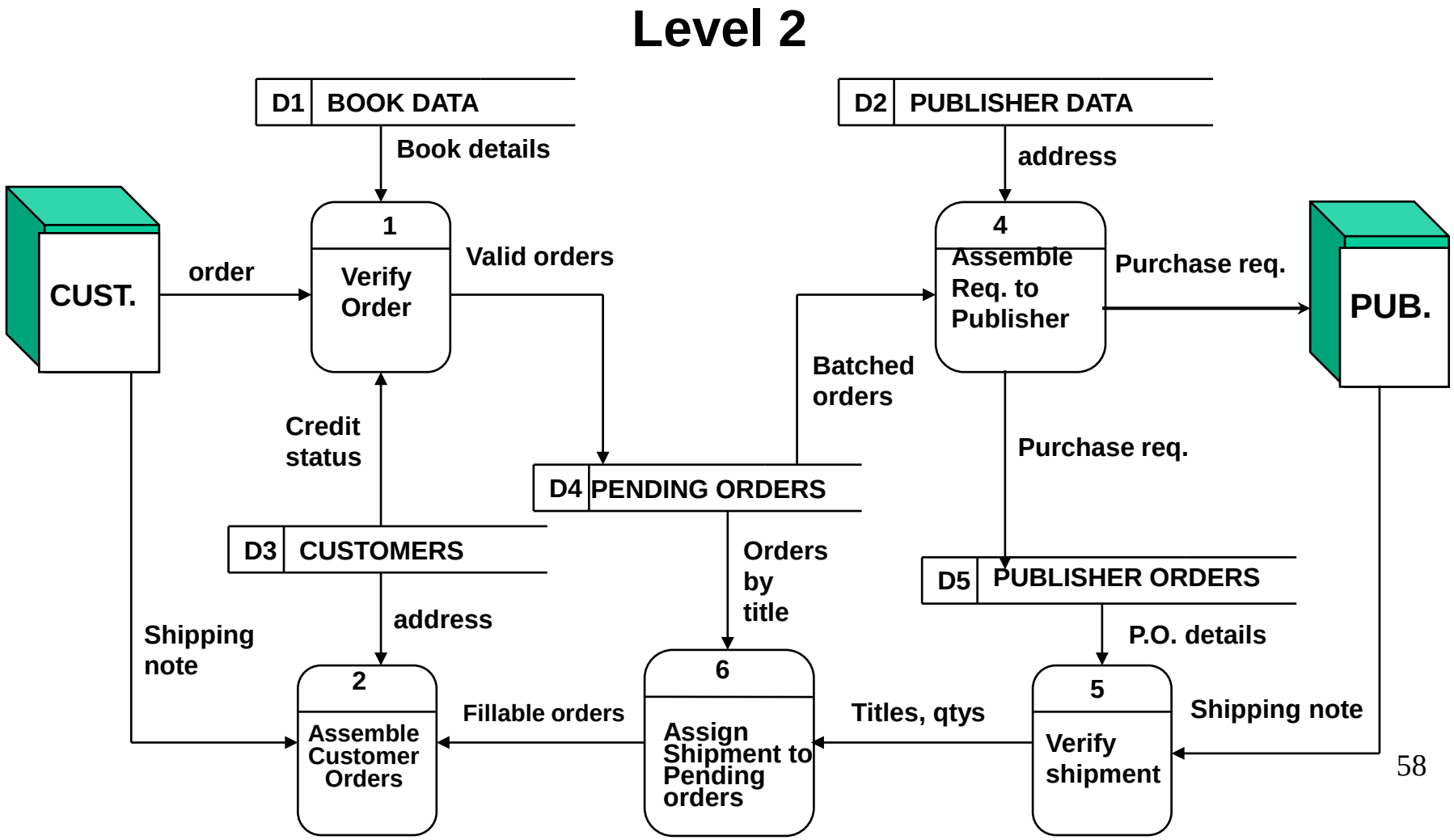
- **Sample Level 1 DFD - Process Orders**

Level 1



Data-Flow diagrams

- **Sample Level 2 DFD – Handling Shipments from Publishers**



Data-Flow diagrams

Sample Case Study - 1

Problem Statement:

Mr. Roy Chowdry, a restaurant owner feels that is business can do better when the existing operations in the system are automated.

Steps involved in constructing a DFD

1. Identify the client and the potential users of the system
2. Observe the client practices and design a context diagram for the same
3. Identify and establish the goals of the proposed system
4. Identify the major files in the system
5. Identify the major processes in the system
6. Construct a DFD

Data-Flow diagrams

- **Constructing a DFD – Step 1**

Identifying the different users of the system:

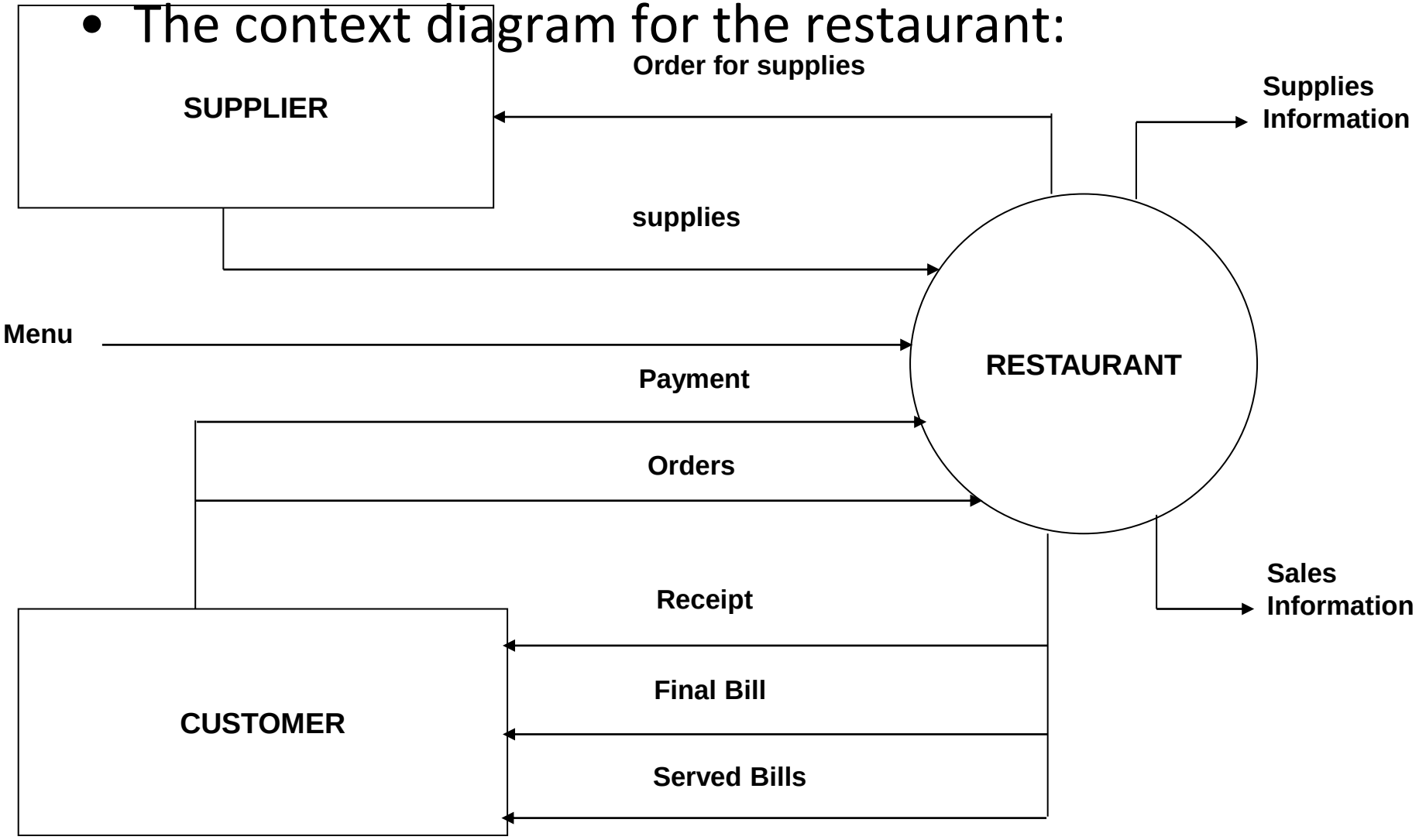
- **Client:**
  - The Restaurant owner

- **Potential Users:**
  - Waiters, cash register operator

Data-Flow diagrams

Constructing a DFD – Step 2

- The context diagram for the restaurant:



Data-Flow diagrams

- **Constructing a DFD – Step 3**

Goals to be achieved:

- Automate much of the order processing and billing
- Automate accounting
- Make supply ordering accurate so as to ensure that:
  - The leftovers at the end of the day are minimized
  - The rejected orders due to non availability are also minimized
- Statistical data and reports on food supplies and sales of different items on a periodic basis

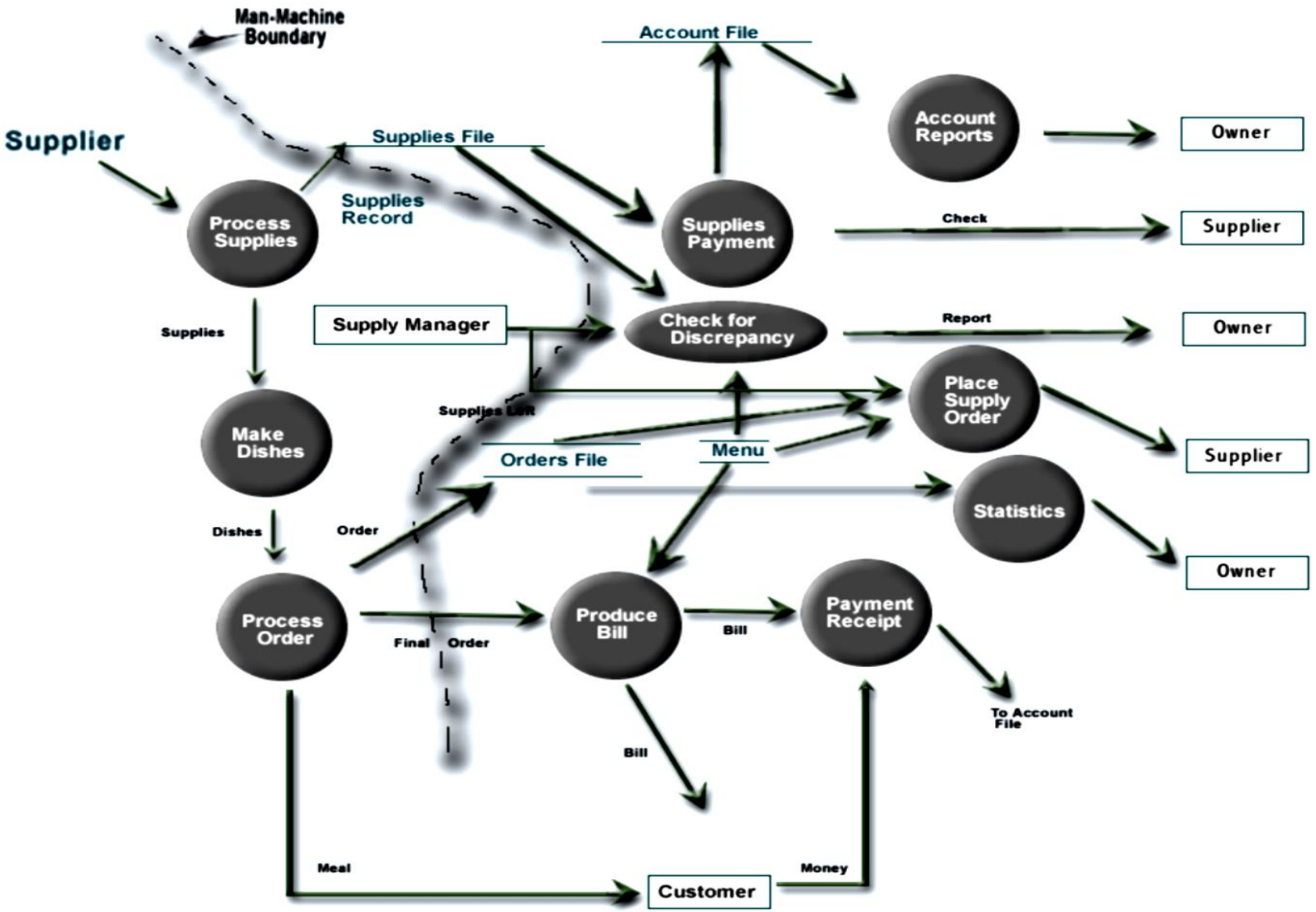
Data-Flow diagrams

- **Constructing a DFD – Step 4**
  - **Major files in the system**
    - Supplies file
    - Accounting file
    - Orders file
    - Menu file

Data-Flow diagrams

- **Constructing a DFD – Step 5**
- Major Processes in the system
  - Process Supplies
  - Make Dishes
  - Process Order
  - Produce Bill
  - Payment Receipt
  - Place Supply Order
  - Check for discrepancy
  - Supplies Payment
  - Check for discrepancy
  - Accounting reports
  - Statistics

Data-Flow diagram: The Final Step - DFD



Data dictionary

Components in the structure of data flow and structure of files are also shown in Data Dictionary.

Various notations used in Data Dictionary are + , | and *

+ indicates sequence or composition

| is used for selection (one OR the other)

* Is used to represent repetition (one or more occurrences)

Data Dictionary for Restaurant

Supplies_file = [date + [item_no + quantity + cost]*]*
Orders_file = [date + [menu_item_no + quantity + status]*]*
Status = satisfied | unsatisfied
Order = [menu_item_no + quantity]*
Menu = [menu_item_no + name + price + supplies_used]*
supplies_used = [supply_item_no + quantity]*
Bill = [name + quantity + price]* + total_price + sales_tax
+ service_charge + grand_total
Discrepancy report =[supply_item_no + amt_ordered +
amt_left + amt_consumed + descr]*

Data-Flow diagrams

- **Sample Case study - 2**
- Generating a DFD for an Air Traffic Controller System

Requirements of the ATCS

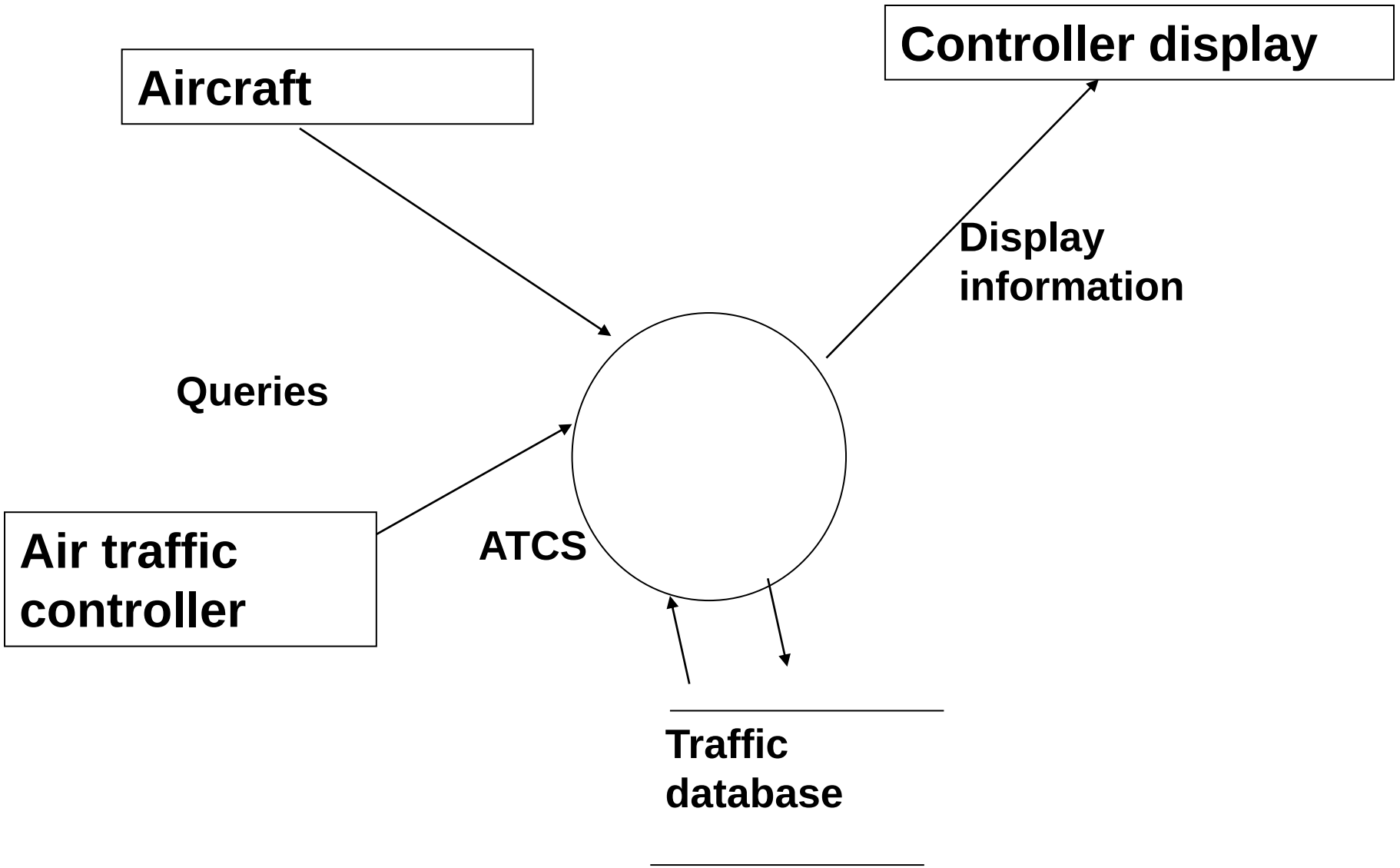
- Receive information from the aircraft about its co-ordinates and velocity
- Process the information at specified intervals
- Update the traffic data base
- Provide information to the air traffic controller when queried

Design constraints

- Data acquisition at specified intervals
- Analysis within specified execution time
- Updating of traffic database at defined intervals
- Controller interaction must not slow down other system functions

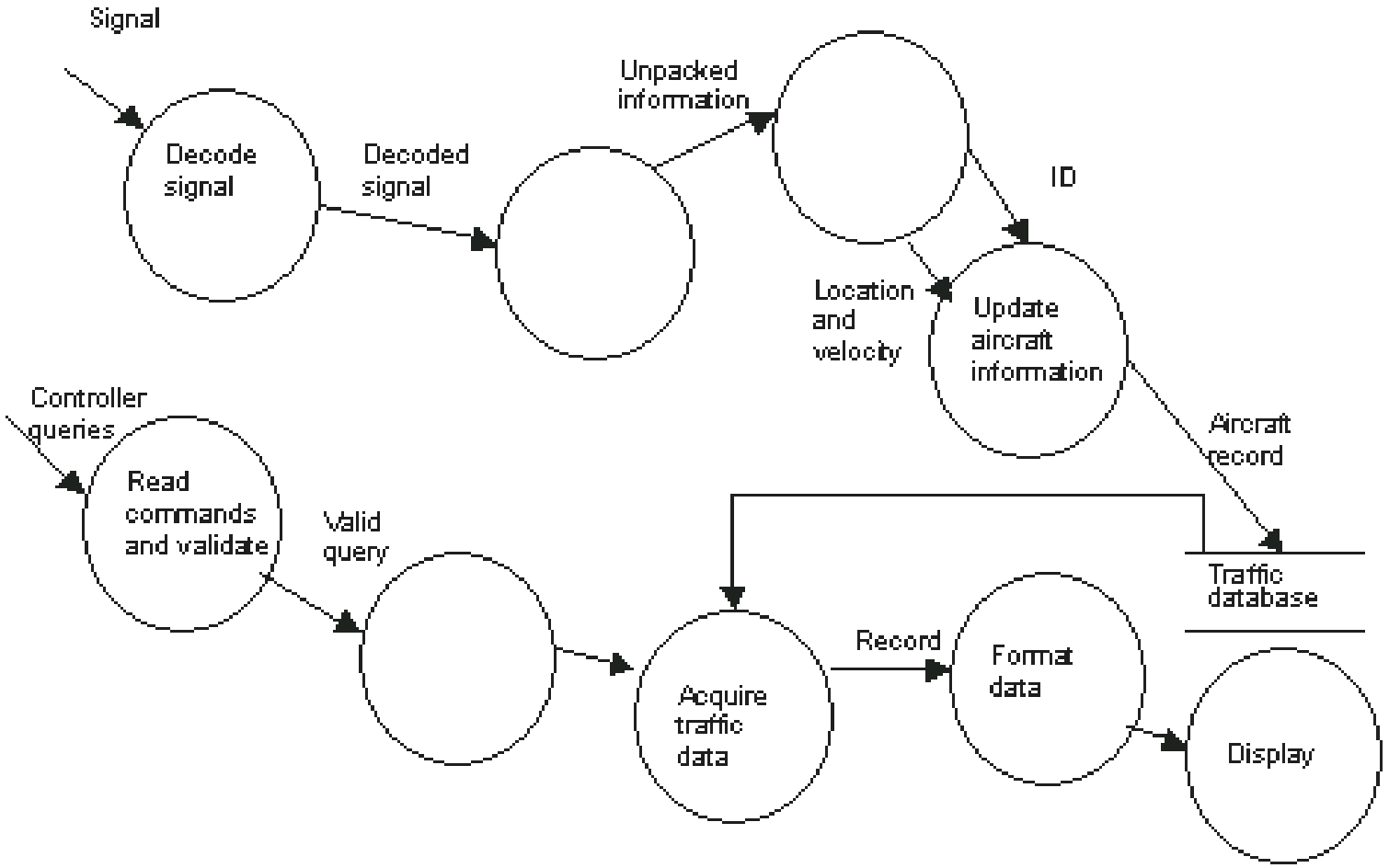
Data-Flow diagrams

- **DFD Level 0 for ATCS**



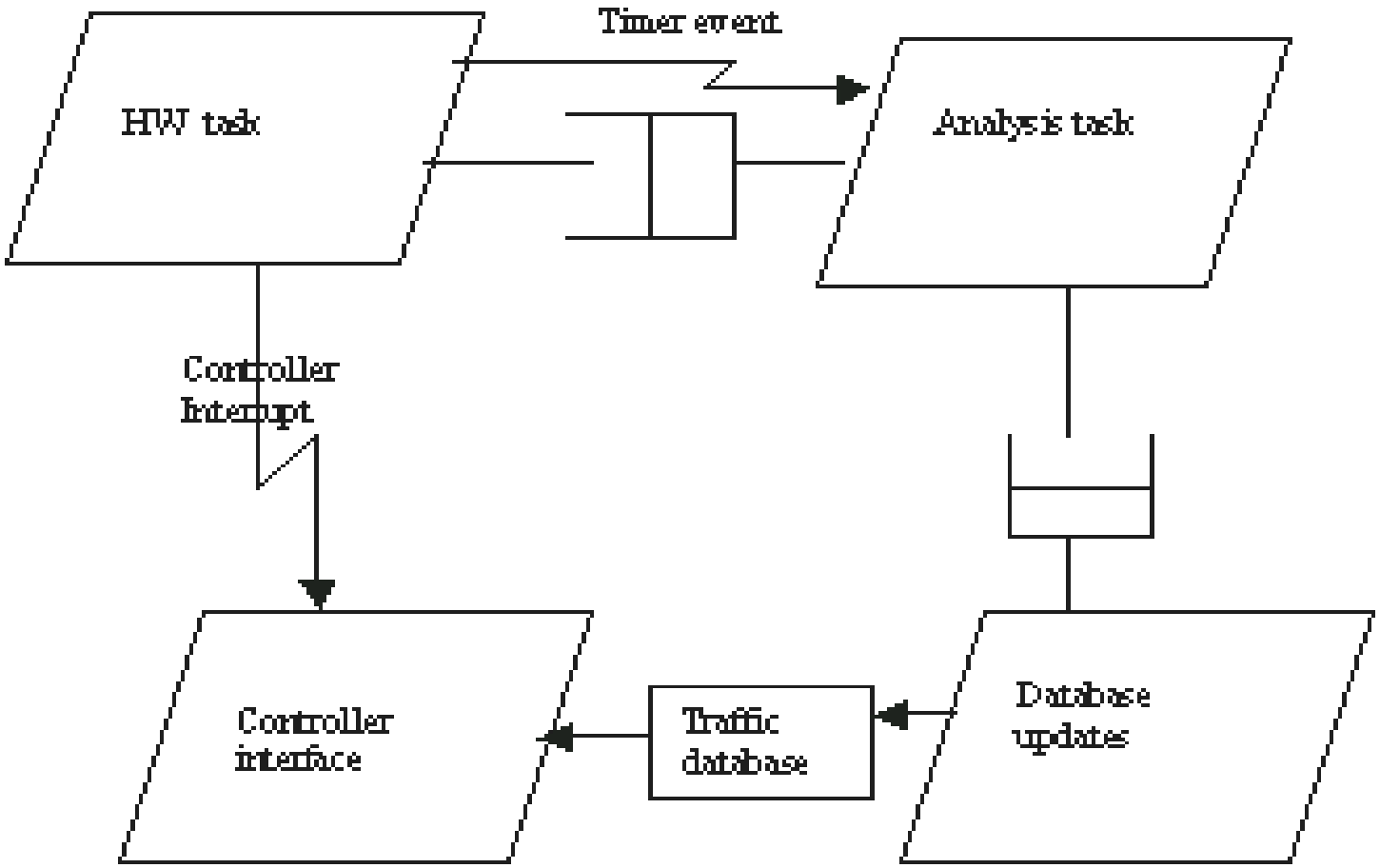
Data-Flow diagrams

DFD - Level 1 for ATCS



Data-Flow diagrams

- **Task architecture**



Data-Flow diagrams

- **Mapping of functions**

- **HW task does the following**
  - Generating timer events
  - Handling signal from aircraft
  - Generating interrupt on controller interactions
- **Analysis task does the following**
  - Processes the signal information to obtain the aircraft information.
  - Queues the information for updating database.
- **Database updates task does the following**
  - Reads the aircraft information
  - Updates the traffic database
- **Controller interface task does the following**
  - Reads the queries from controller and validates them
  - Fetches information from the traffic database
  - Displays the response for the controller

Entity-Relationship Diagrams

- **A pictorial representation of** entities**, their** attributes **and the relationships between the entities.**

What are entities in ERD?

- A real world object within the scope of the system about which information has to be stored
- **Example**
  - In a banking system, **Customer** and **Transaction** are some of the entities

Attributes in ERD

Refer to the properties of an entity

- Each entity has an associated set of attributes
- **Example:**
  - For the entity *Customer*, the attributes could be **Account_Number**, **Customer_name**, **Customer_address**, and **Account_type**

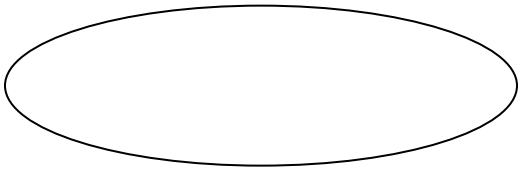
Entity-Relationship Diagrams

- **Some important ERD notations**

Entity →

Dependant/Sub Entity →

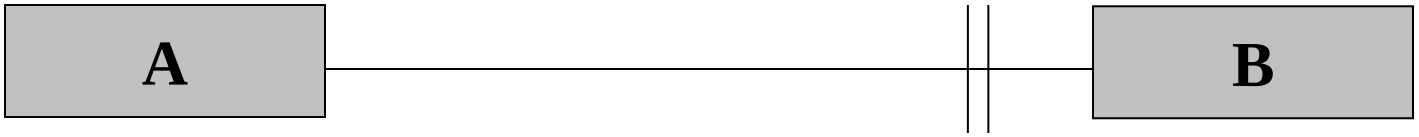
Attribute →



Entity-Relationship Diagrams

- **Symbols used in ERD**

- There can be different types of relationships between entities
- Relationships are depicted using the notations given below:
- Entity A is associated with exactly one instance of entity B (Also, called a one-one relationship)



Entity-Relationship Diagrams

- **Symbols used in ERD**

- Entity A is associated with either zero or one instance of entity B

Entity-Relationship Diagrams

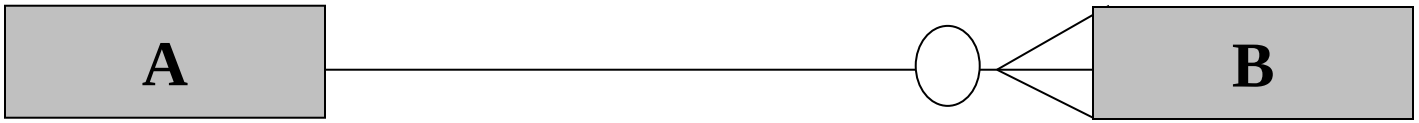
- **Symbols used in ERD**

- Entity A is associated with either one or more instances of entity B

Entity-Relationship Diagrams

- **Symbols used in ERD**

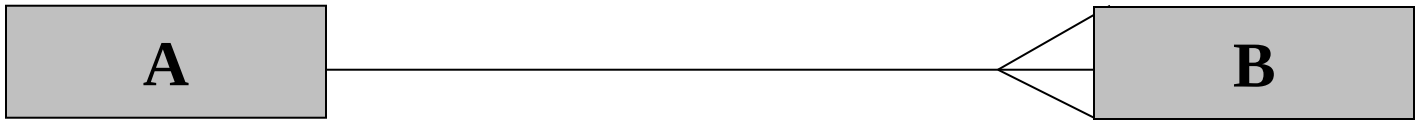
- Entity A is associated with either zero, one or more instances of entity B



Entity-Relationship Diagrams

- **Symbols used in ERD**

  - Entity A is associated with more than one instance of entity B



Entity-Relationship Diagrams

- **Sample Case Study**

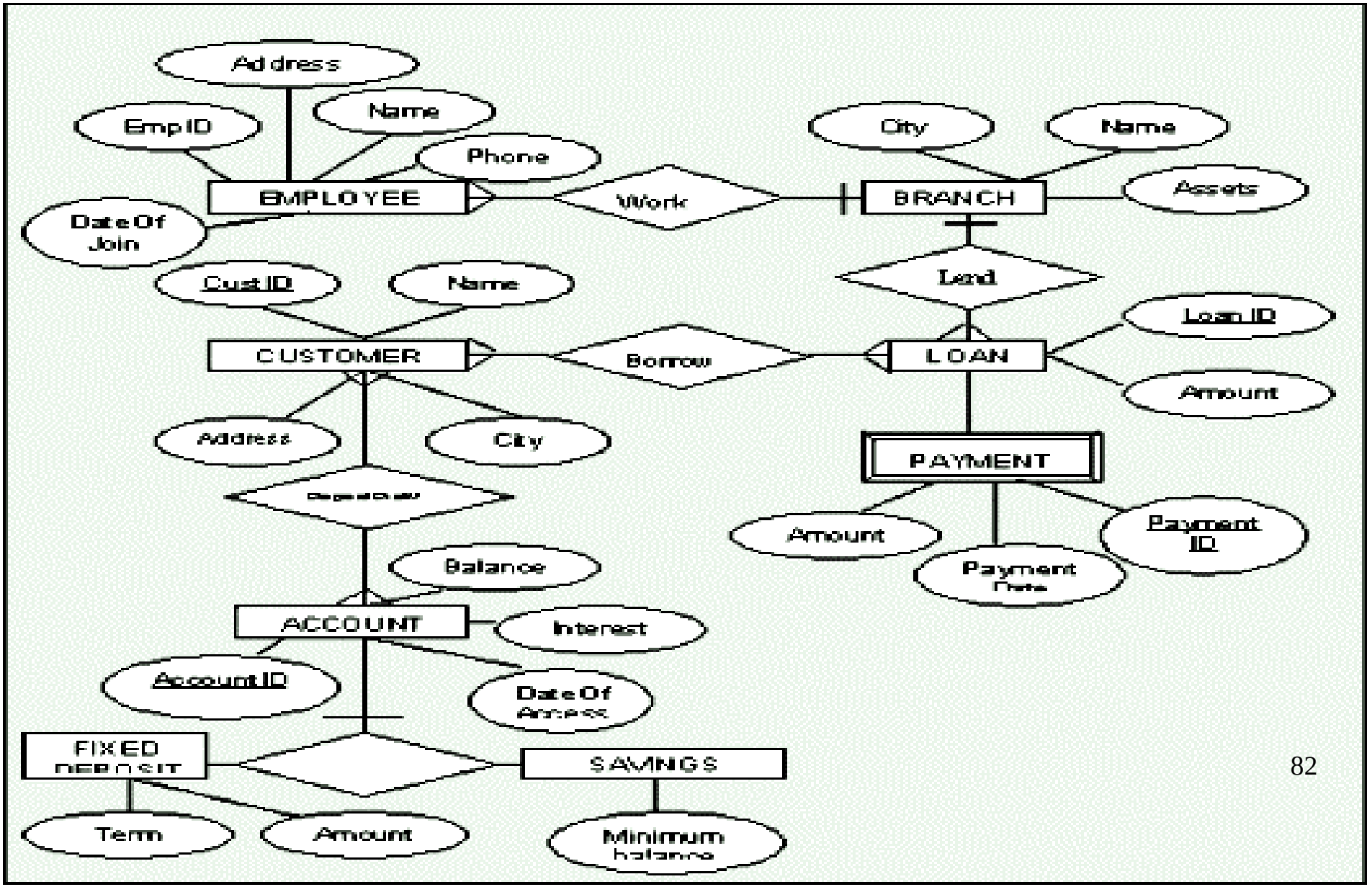
- **AMZ Bank has decided to computerize its operations, as it finds it difficult to manage its wide customer and employee network**
- **The major requirements of the bank are given below:**
  - Bank has multiple branches across the city. Each branch is identified by a unique branch name.
  - Each customer of the bank is identified by a unique customer ID
  - Bank employees are identified by their employee IDs

Entity-Relationship Diagrams

Sample Case Study …*contd.*

- The bank offers two types of accounts – Savings account and Fixed Deposit account. An account can be held by more than one customer, and a customer can have more than one account
- The branch decides on the loans to be issued to one or more customers. A loan is identified by a unique Loan Number.
- For each loan, the bank keeps track of the loan amount. Apart from this, the branch also maintains the payment details of the loan like Payment Number for each loan, the date and amount for each payment.
- The bank also maintains the personal as well as the transactional details of its customers and employees
- ***Note:*** *A payment can exist only if there is a loan.*

Entity-Relationship Diagrams: **ERD for the Banking System**



Question 1 : Suppose you are given the following requirements for a simple database for the National Hockey League (NHL):

- the NHL has many teams,
- each team has a name, a city, a coach, a captain, and a set of players,
- each player belongs to only one team,
- each player has a name, a position (such as left wing or goalie),
- a skill level, and a set of injury records,
- a team captain is also a player,
- a game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 1999) and a score (such as 4 to 2).

Construct a clean and concise ER diagram for the NHL database.

Question 2: A university registrar's office maintains data about the following entities:

1.courses, including number, title, credits, syllabus, and prerequisites;

2.course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom;

3.students, including student-id, name, and program;

4.instructors, including identification number, name, department, and title.

Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled. Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

Question 3 : Construct an ER Diagram for Company having following details :

- Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations.
- A department controls a number of PROJECT. Projects have a unique name, number and a single location.
- Company's EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several projects (not necessarily controlled by her dept). Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee.
- Employee's DEPENDENT are tracked for health insurance purposes (dependent name, birthdate, relationship to employee).

State Transition Diagrams

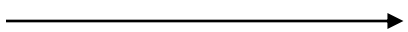
- Used to indicate how the proposed system will behave in response to certain events
- Represent various states of the system and their transitions

State Transition Diagrams

- **Symbols used in State Transition Diagrams**

State

Transition



Condition

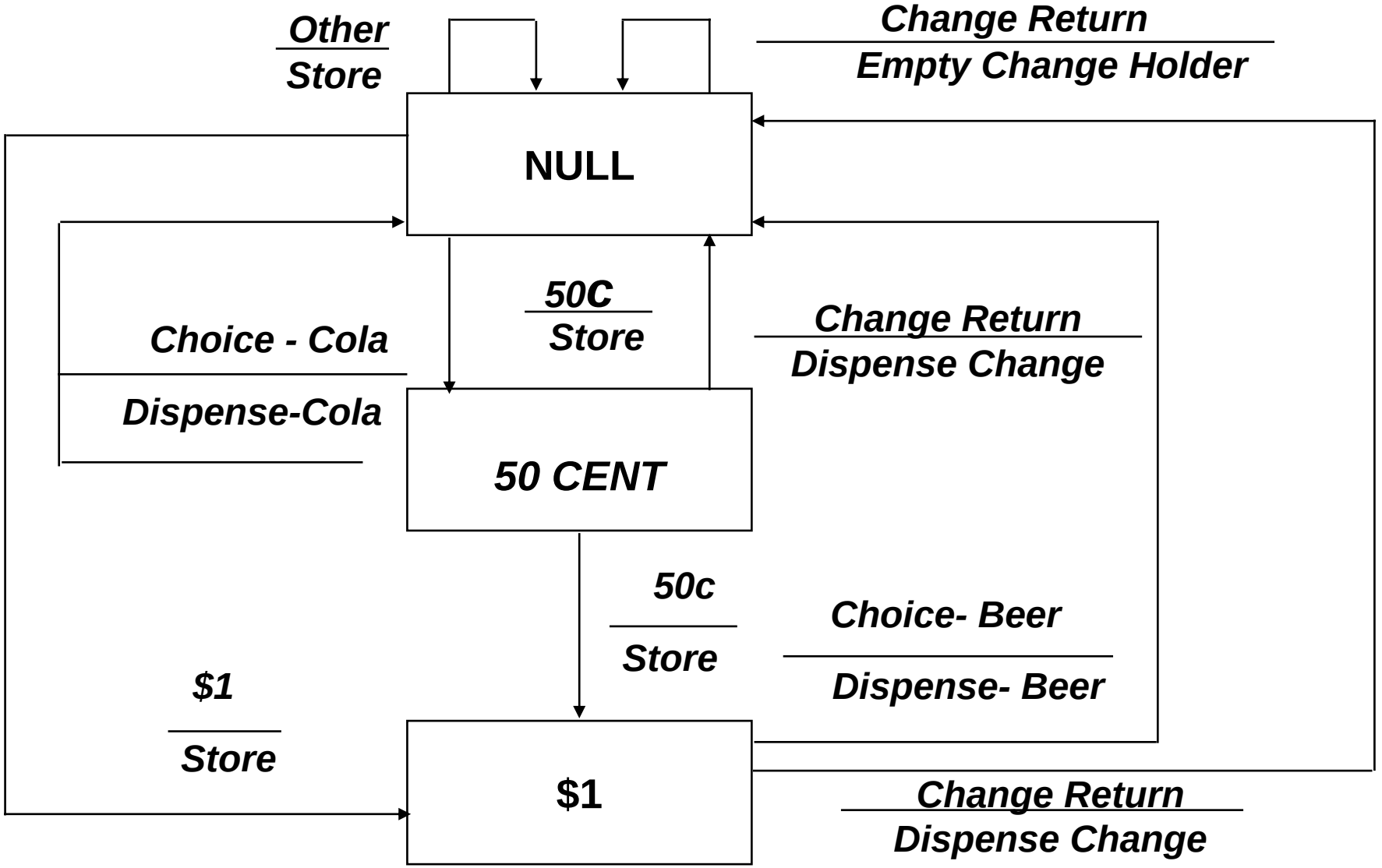
condition

Action

action

State Transition Diagrams

- **A Sample State Transition Diagram**



- **Construct a State Transition Diagram**

- **A coin change machine helps customers with change for $1 bills and $5 bills. When a customer inserts a bill, the machine checks if it is $1 or $5. If it is one of the two, it either returns four quarters or twenty quarters, respectively. If it is not one of the two, it rejects it. If there is not enough coins left to provide change for a $1, the machine illuminates alight and does not accept any bills. If the machine does not have enough coins for $5, it illuminates another light and does not accept any $5 bills. Identify the actors and use cases. For each use case, write down the scenario. Draw an use case diagram showing how the actors are related to the use cases. Also show the use cases that are extends and the ones that are uses.**