Chapter 6
System Engineering

CHAPTER OVERVIEW AND COMMENTS

This intent of this chapter is to provide a brief introduction to the system engineering process. The overall structure of computer-based systems is discussed and a brief overview of the system engineering hierarchy is presented. Business process engineering (BPR) and product engineering are discussed in overview fashion.

Note: Some reviewers of this edition argued that a discussion of system engineering is beyond the scope of a software engineering text.

S/W Eng occurs as a consequence of a process called *System engineering*. System Engineering focuses on analyzing, designing, and organizing those elements into a system that can be a product, a service, or a technology for the transformation of information.

6.1 Computer-Based Systems

This section introduces the systems view of engineering (all complex systems can be viewed as being composed of cooperating subsystems). The elements of computer-based systems are defined as software, hardware, people, database, documentation, and procedures.

A computer-based system makes use of a variety of system elements.

Software: programs, data structures, and related work products.
Hardware: electronic devices that provide computing capabilities.
People: Users and operators of hardware and software.
Database: A large, organized collection of information that is accessed via S/w and persists over time.
Documentation: manuals, on-line help files.
Procedures: the steps that define the specific use of each system element.

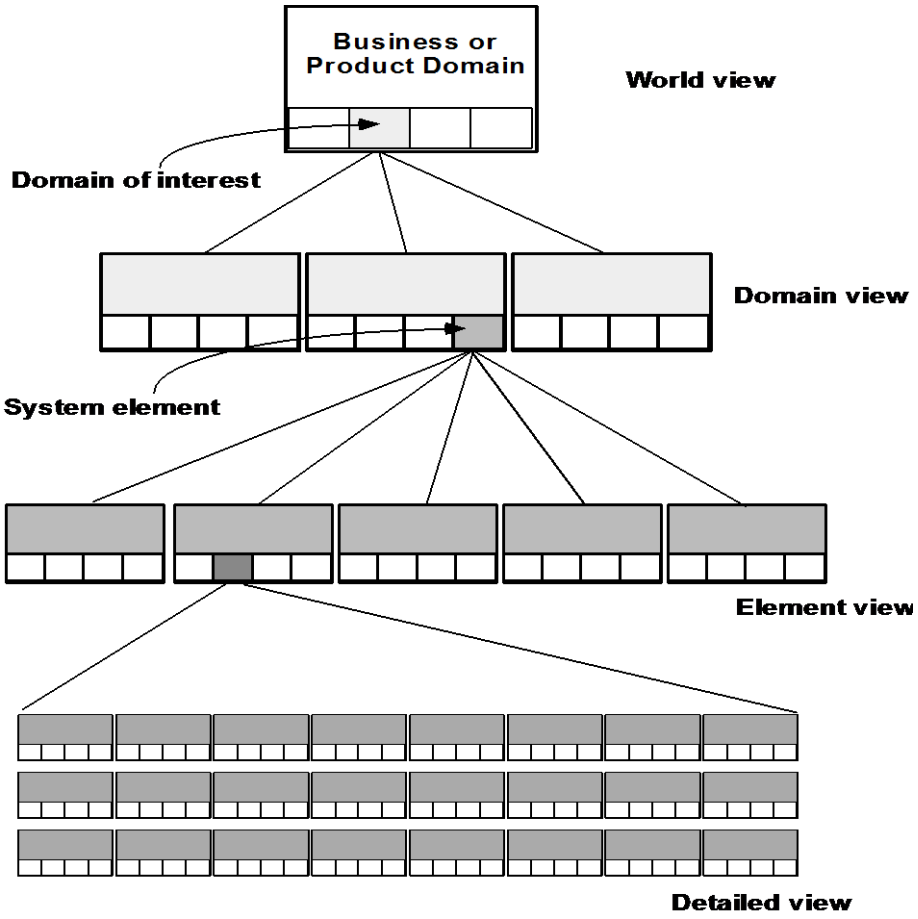
One complicating characteristic of computer-based system is that the elements constituting one system may also represent one macro element of a still large system. The *micro-element* is a computer-based system that is one part of a larger computer based system.

6.2 The System Engineering Hierarchy

The key to system engineering is a clear understanding of context. For software development this means creating a "world view" and progressively narrowing its focus until all technical detail is known.

In software engineering there is rarely one right way of doing something. Instead designers must consider the tradeoffs present in the feasible solutions and select one that seems advantageous for the current problem. This section lists several factors that need to be examined by software engineers when evaluating alternative solutions (assumptions, simplifications, limitations, constraints, and preferences).

Regardless of its domain of focus, system eng. Encompasses a collection of top-down and bottom-up methods to navigate the hierarchy illustrated below:



The system eng. Process usually begins with a "world view." The entire business or product domain is examined to ensure that the proper business or technology context can be established.

The world view is refined to focus more fully on a specific domain of interest.

Within a specific domain, the need for targeted system elements (data, S/W, H/W, and people) is analyzed.

Finally, the analysis, design, and construction of a targeted system element are initiated.

6.2.1 System Modeling

System modeling is an important element of the system eng. Process. The Engineer creates models that:

1. Define the processes that serve the needs of the view under consideration.
2. Represent the behavior of the processes and the assumptions on which the behavior is based.
3. Explicitly define both exogenous and endogenous input to the model.
4. Exogenous inputs link one constituent of a given view with other constituents at the same level of other levels; endogenous input links individual components of a constituent at a particular view.
5. Represent all linkages (including output) that will enable the engineer to better understand the view.

To construct a system model, the engineers should consider a number of restraining factors: "Read examples of each in book page# 127."

1. *Assumptions* that reduce the number of possible permutations and variations, thus enabling a model reflect the problem in a reasonable manner.
2. *Simplifications* that enable the model to be created in a timely manner.
3. *Limitations* that help to bound the system.
4. *Constraints* that will guide the manner in which the model is created and the approach taken when the model is implemented.
5. *Preferences* that indicate the preferred architecture for all data, functions, and technology.

6.3 Business Process Engineering: An Overview

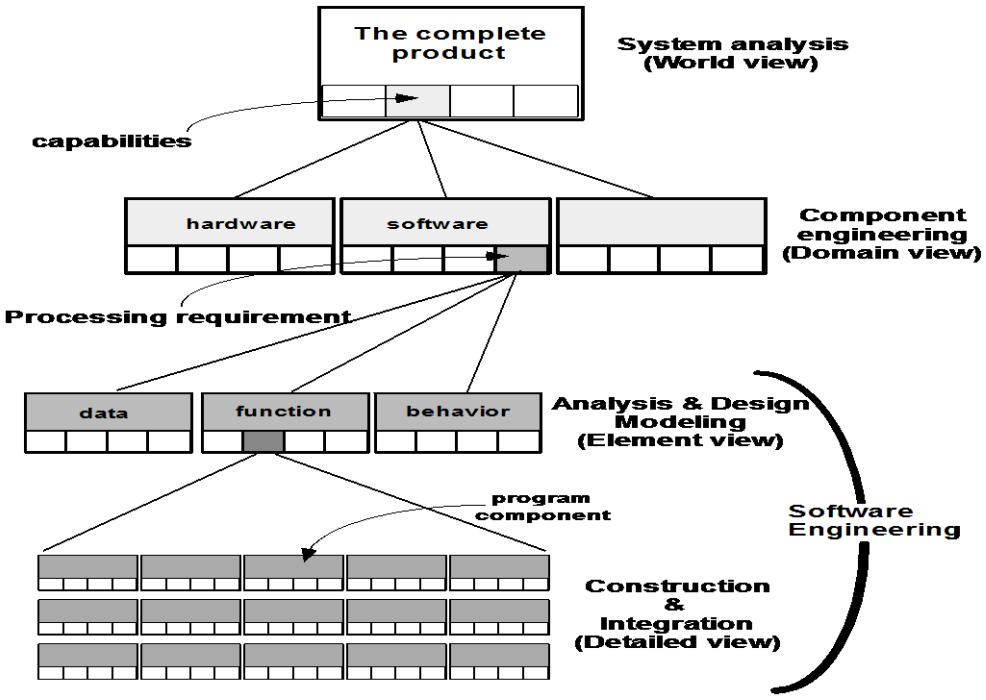
The goal of *Business Process Engineering (BPE)* is to define architectures that will enable a business to use information effectively.
BPE is one process for creating an overall plan for implementing the computing architecture.
Three different architectures must be analyzed and designed within the context of business objectives and goals:

- Data architecture
- Application architecture
- Technology infrastructure

The *data architecture* provides a framework for the information needs of a business. The building blocks of the architecture are the data objects that are used by the business. Once a set of data objects is defined, their relationships are identified. A *relationship* indicates how objects are connected to one another.

The *application architecture* encompasses those elements of a system that transform objects within the data architecture for some business purpose.

The *technology infrastructure* provides the foundation for the data and application architectures. The infrastructure encompasses the h/w and s/w that are used to support the applications and data.

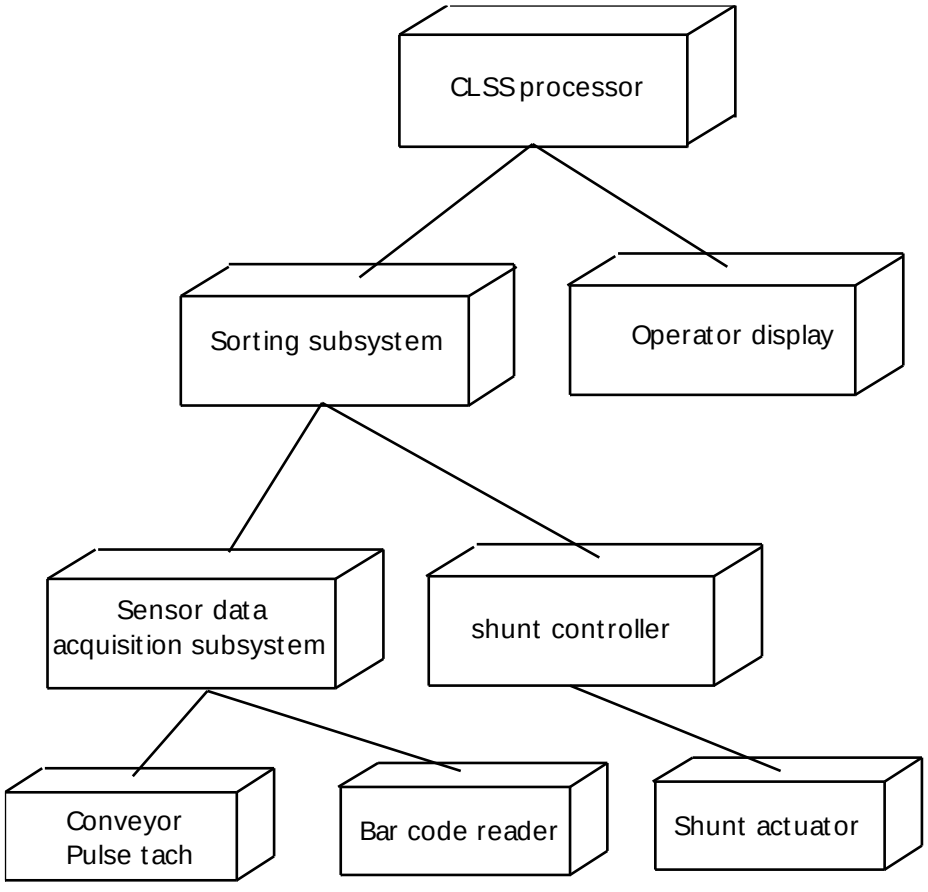


6.4 Product Engineering: An Overview

Emphasize that software engineers participate in all levels of the product engineering process that begins with requirements engineering. The analysis step maps requirements into representations of data, function, and behavior. The design step maps the analysis model into data, architectural, interface, and software component designs.

6.5 System Modeling with UML

- In terms of the data that describe the element and the operations that manipulate the data Deployment diagrams
  - Each 3-D box depicts a hardware element that is part of the physical architecture of the system
- Activity diagrams
  - Represent procedural aspects of a system element
- Class diagrams
  - Represent system level elements



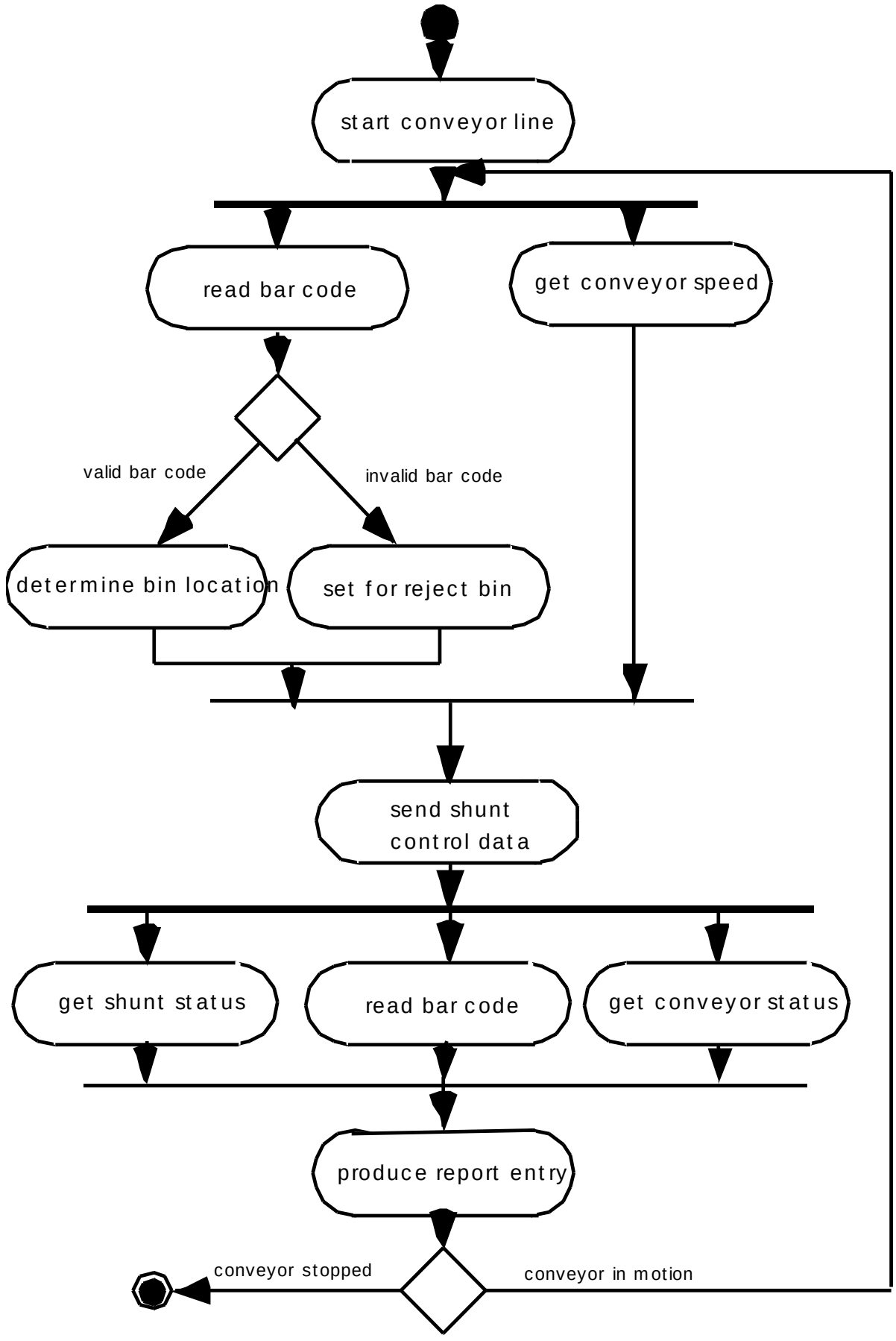
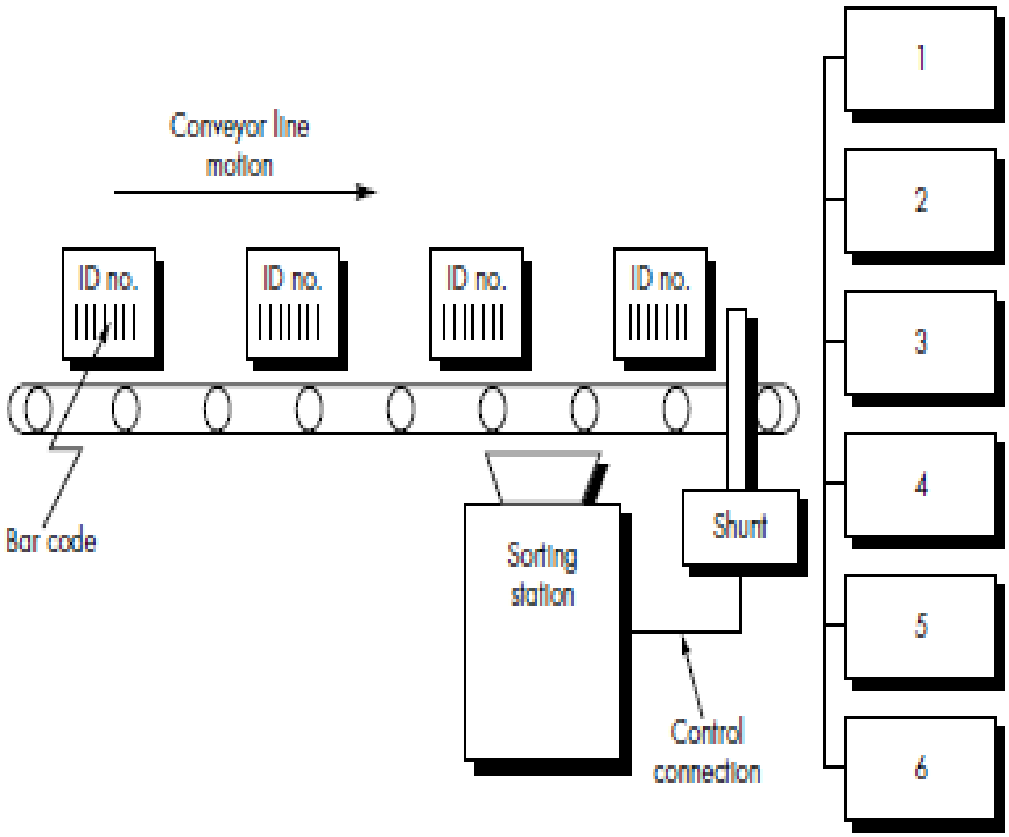
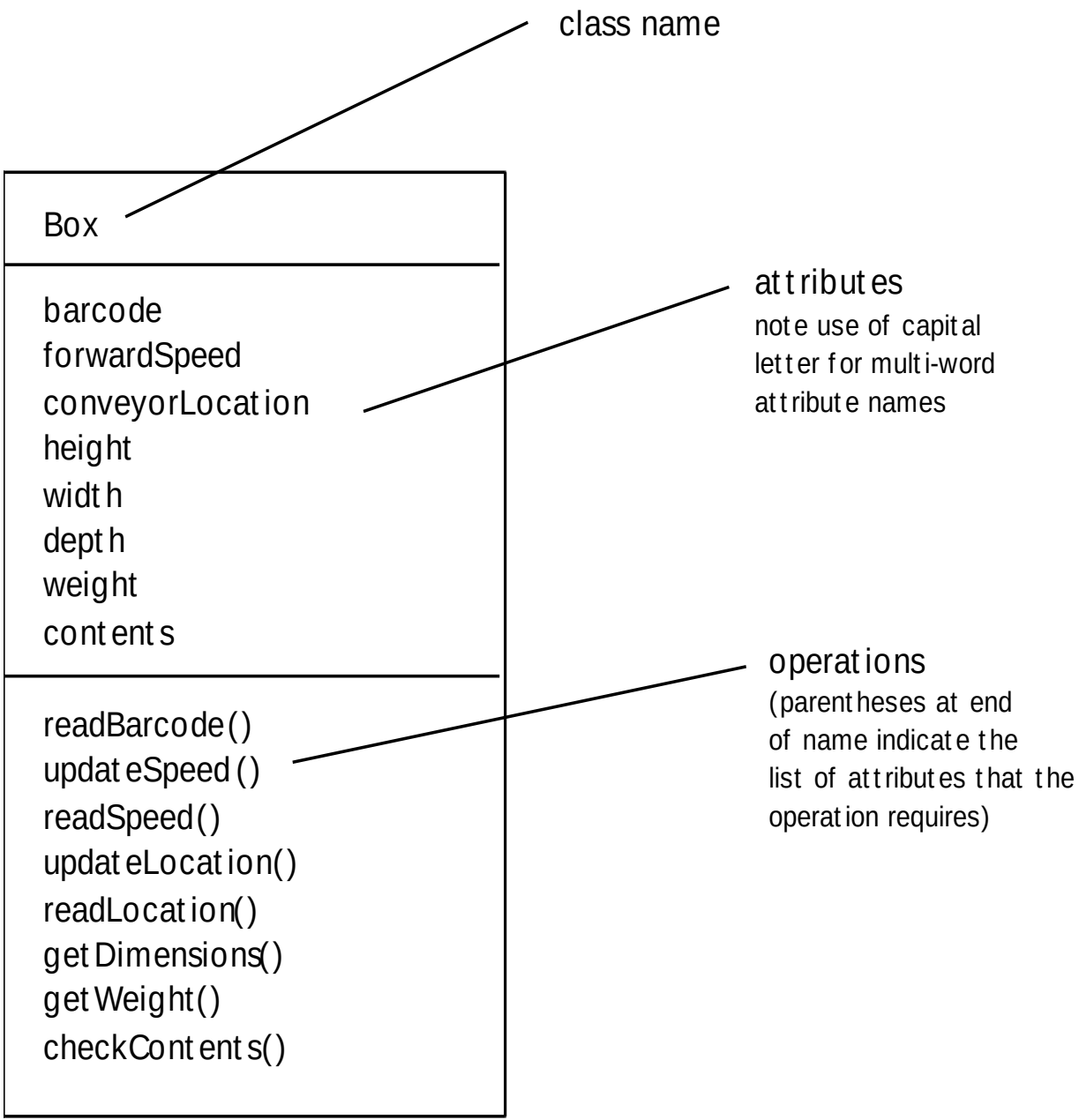

start conveyor line
read bar code
get conveyor speed
valid bar code
invalid bar code
determine bin location
set for reject bin
send shunt control data
get shunt status
read bar code
get conveyor status
produce report entry
conveyor stopped
conveyor in motion

FIGURE 5.1
A conveyor line sorting system




class name
Box
barcode
forwardSpeed
conveyorLocation
height
width
depth
weight
contents
attributes
note use of capital letter for multi-word attribute names
readBarcode()
updateSpeed()
readSpeed()
updateLocation()
readLocation()
getDimensions()
getWeight()
checkContents()
operations
(parentheses at end of name indicate the list of attributes that the operation requires)

4 Software Requirements

4.1 Introduction

In requirement engineering there is a systematic use of principles, technique and tools for cost effective analysis, documentation and user needs. Both the software engineer and customer take an active role in requirement engineering.

In this chapter we will discuss the concept of user and functional requirements. We describe functional and non functional requirements. Finally we will learn how software requirements may be organized in requirements document.

What is requirement engineering ?

Requirement engineering is the process of

- *establishing the services that the customer requires from a system*
- *and the constraints under which it operates and is developed.*

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement?

A requirement can range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

The requirement must be open to interpretation and it must be defined in detail.

Types of requirements

The requirements can be classified as

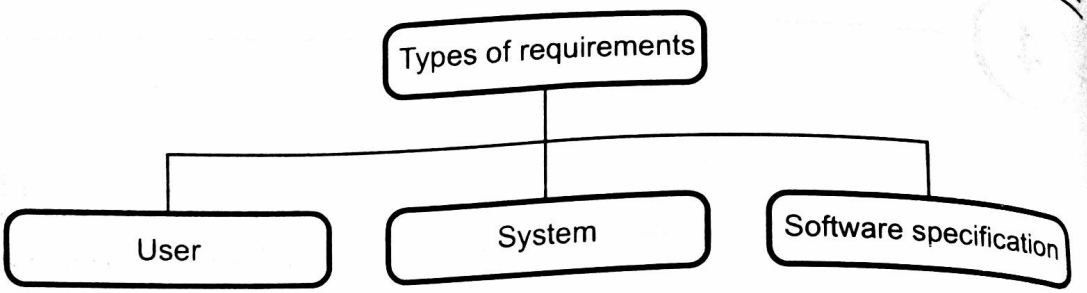


Fig. 4.1 Types of requirements

- **User requirements**

It is a collection of statements in natural language plus description of the services the system provides and its operational constraints. It is written for customers.

- **System requirements**

It is a structured document that gives the detailed description of the system services. It is written as a contract between client and contractor.

- **Software specification**

It is a detailed software description that can serve as a basis for design or implementation. Typically it is written for software developers.

4.2 Functional and Non Functional Requirements

Software system requirements can be classified as functional and non functional requirements.

4.2.1 Functional Requirements

- Functional requirements should describe all the required functionality or system services.
- The customer should provide statement of service. It should be clear how the system should react to particular inputs and how a particular system should behave in particular situation.
- Functional requirements are heavily dependant upon 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 but functional system requirements should describe the system services in detail.

For example : Consider a library system in which there is a single interface provided to multiple databases. These databases are collection of articles from different libraries. A user can search for, download and print these articles for a personal study.

From this example we can obtain functional Requirements as–

1. The user shall be able to search either all of the initial set of databases or select a subset from it.
2. The system shall provide appropriate viewers for the user to read documents in the document store.
3. A unique identifier (ORDER_ID) should be allocated to every order. This identifier can be copied by the user to the account's permanent storage area.

4.2.1.1 Problems Associated with Requirements

- **Requirements imprecision**

1. Problems arise when requirements are not precisely stated.
2. Ambiguous requirements may be interpreted in different ways by developers and users.
3. Consider meaning of term 'appropriate viewers'

- **User intention** - special purpose viewer for each different document type;
- **Developer interpretation** - Provide a text viewer that shows the contents of the document.
- **Requirements completeness and consistency –**

The requirements should be both complete and consistent. *Complete* means they should include descriptions of all facilities required. *Consistent* means there should be no conflicts or contradictions in the descriptions of the system facilities.

Actually in practice, it is impossible to produce a complete and consistent requirements document.

4.2.2 Non Functional Requirements

- The non functional requirements define system properties and constraints.

  Various properties of a system can be : Reliability, response time, storage requirements. And constraints of the system can be : Input and output device capability, system representations etc.
- Process requirements may also specify programming language or development method.
- Non functional requirements are more critical than functional requirements. If the non functional requirements do not meet then the complete system is of no use.

4.2.2.1 Types of Non Functional Requirements

The classification of non functional requirements is as given below.

Fig. 4.2 Types of non functional requirement

Product requirements

These requirements specify how a delivered product should behave in a particular way. For instance: execution speed, reliability.

Organizational requirements

The requirements which are consequences of organizational policies and procedures come under this category. For instance: process standards used implementation requirements.

External requirements

These requirements arise due to the factors that are external to the system and its development process. For instance : interoperability requirements, legislative requirements.

In short, non functional requirements arise through

i) user needs

ii) because of budget constraints

iii) organizational policies

iv) the need for interoperability with other software or hardware systems

v) because of external factors such as safety regulations.

Metrics used for specifying the non functional requirements

Property	Metric
Speed	Events per response time processed transactions per second.
Size	Kilo bytes.
Reliability	Mean time to failure. Rate of failure. Occurrence availability.
Robustness	Time to restart after failure. Probability of events causing failure.
Portability	Number of target statements.

4.3 User Requirements

- The user requirements should describe functional and non functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams because these are the representations that can be understood by all users.

Various problems that can arise in the requirement specifications when requirements are given in natural language –

Lack of clarity

Sometimes requirements are given in ambiguous manner. It is expected that text should help in clear and precise understanding of the requirements.

Requirements confusion

There may be confusion in functional requirements and non functional requirements, system goals and design information.

Requirements mixture

There may be a chance of specifying several requirements together as a single requirement.

4.3.1 Guidelines for Writing User Requirements

- Prepare a standard format and use it for all requirements.
- Apply consistency in the language. Use
  - *'shall'* for mandatory requirements
  - and *'should'* for desirable requirements.
- The text which is mentioning the key requirements should be highlighted.
- Avoid the use of computer jargon (computer terminologies). It should be written in simple language.

For example

Consider a spell checking and correcting system of a word processor. The user requirements can be given in natural language as

1. The system should posses a traditional **word dictionary and user supplied dictionary**. It shall provide a user-activated facility which checks the spelling of words in the document against spellings in the system dictionary and user-supplied dictionaries.
2. When a word is found in the document which is not given in the dictionary, then the system should suggest 10 alternative words. These alternative words should be based on a match between the word found and corresponding words in the dictionaries.
3. When a word is found in the document which is not in any dictionary, the system should propose following options to user :
   1. Ignore the corresponding instance of the word and go to next sentence.
   2. Ignore all instances of the word

3. Replace the word with a suggested word from the dictionary
4. Edit the word with user-supplied text
5. Ignore this instance and add the word to a specified dictionary

4.4 System Requirements

- System requirements are more detailed specifications of system functions, services and constraints than user requirements.
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.
- The system requirements can be expressed using system models.
- The requirements specify what the system does and design specifies how it does.
- System requirement should simply describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented.
- For a complex software system design it is necessary to give all the requirements in detail.
- Usually, natural language is used to write system requirements specification and user requirements.

Why requirement and design are inseparable?

- A system architecture may be designed to structure the requirements;
- The system may inter-operate with other systems and that may generate design requirements;
- The use of a specific design may be a domain requirement.

4.5 Interface Specification

Sometimes there is already existing system which can be used with the newly created software system. This conjunction of old system with new system is called system interface. In such situation the interfaces of already existing systems must be specified clearly. There are three types of interfaces that can be defined –

1. **Procedural interfaces** : These are popularly known as Application Programming Interfaces (API). Such procedures are intended to offer services that may be used by calling procedures.

2. **Data structures :** Data structures are the descriptors of data. They play an important role in organization of data for given algorithm. The data structures can be passed from one sub-system to another.
3. **Representation of data :** This level of specification is used in certain programming languages like ADA. For real time applications these kind of interfaces are often useful. Sometime to describe this interface diagrams can be used.

The interface should be defined in an unambiguous manner but such interfaces can be understood only after special training. For example

```
Interface Hello {
    void initialize(string S);
    string SayHello();
}
```

This interface will display the message hello on encountering this interface.

4.6 The Software Requirements Document

The software requirements document is the specification of the system.It should include both a definition and a specification of requirements. It is not a design document. As far as possible, it should set of what the system should do rather than how it should do it.

Software Requirements Specification

The software requirements provide a basis for creating the Software Requirements Specifications (SRS).

The SRS is useful in estimating cost, planning team activities, performing tasks, and tracking the team's progress throughout the development activity.

Typically software designers use IEEE STD 830-1998 as the basis for the entire Software Specifications.The standard template for writing SRS is as given below.

Document Title

Author(s)
Affiliation
Address
Date
Document Version

1. Introduction

1.1 Purpose of this document
Describes the purpose of the document.

1.2 Scope of this document
Describes the scope of this requirements definition effort. This section also details any constraints that were placed upon the requirements elicitation process, such as schedules, costs.

1.3 Overview
Provides a brief overview of the product defined as a result of the requirements elicitation process.

2. General Description

- Describes the general functionality of the product such as similar system information, user characteristics, user objective, general constraints placed on design team.
- Describes the features of the user community, including their expected expertise with software systems and the application domain.

3. Functional Requirements

This section lists the functional requirements in ranked order. A functional requirement describes the possible effects of a software system, in other words, *what* the system must accomplish. Each functional requirement should be specified in following manner

- **Short, imperative sentence stating highest ranked functional requirement.**

1. **Description**
   A full description of the requirement.
2. **Criticality**
   Describes how essential this requirement is to the overall system.
3. **Technical issues**
   Describes any design or implementation issues involved in satisfying this requirement.

4. **Cost and schedule**
   Describes the relative or absolute costs of the system.
5. **Risks**
   Describes the circumstances under which this requirement might not able to be satisfied.
6. **Dependencies with other requirements**
   Describes interactions with other requirements.
7. **... any other appropriate**

4. Interface Requirements

This section describes how the software interfaces with other software products or users for input or output. Examples of such interfaces include library routines, token streams, shared memory, data streams, and so forth.

- **4.1 User Interfaces**
  Describes how this product interfaces with the user.
  - **4.1.1 GUI**
    Describes the graphical user interface if present. This section should include a set of screen dumps to illustrate user interface features.
  - **4.1.2 CLI**
    Describes the command-line interface if present. For each command, a description of all arguments and example values and invocations should be provided.
  - **4.1.3 API**
    Describes the application programming interface, if present.
- **4.2 Hardware Interfaces**
  Describes interfaces to hardware devices.
- **4.3 Communications Interfaces**
  Describes network interfaces.
- **4.4 Software Interfaces**
  Describes any remaining software interfaces not included above.

5. Performance Requirements

Specifies speed and memory requirements.

6. Design Constraints

Specifies any constraints for the design team such as software or hardware limitations.

7. Other non-functional attributes

Specifies any other particular non functional attributes required by the system. Such as :

7.1 Security

7.2 Binary Compatibility

7.3 Reliability

7.4 Maintainability

7.5 Portability

7.6 Extensibility

7.7 Reusability

7.8 Application Compatibility

7.9 Resource Utilization

7.10 Serviceability

... others as appropriate

8. Operational Scenarios

This section should describe a set of scenarios that illustrate, from the user's perspective, what will be experienced when utilizing the system under various situations.

9. Preliminary Schedule

This section provides an initial version of the project plan, including the major tasks to be accomplished, their interdependencies, and their tentative start/stop dates.

10. Preliminary Budget

This section provides an initial budget for the project.

11. Appendices

11.1 Definitions, Acronyms, Abbreviations

Provides definitions terms, and acronyms, can be provided.

11.2 References

Provides complete citations to all documents and meetings referenced.

4.6.1 Characteristics of SRS

Various characteristics of SRS are

- Correct – The SRS should be made up to date when appropriate requirements are identified.
- Unambiguous – When the requirements are correctly understood then only it is possible to write an unambiguous SRS.
- Complete – To make the SRS complete, it should be specified what a software designer wants to create a software.
- Consistent – It should be consistent with reference to the functionalities identified.
- Specific – The requirements should be mentioned specifically.
- Traceable – What is the need for mentioned requirement? This should be correctly identified.

4.6.2 Example of SRS

Software Requirements Specification For Attendance Maintenance System

Prepared by Anjali

December 1, 2007

Release 1.0

Version 1.0

Table of Contents

1. Introduction

1.1 Purpose

This document gives detailed functional and non functional requirements for attendance maintenance system. The purpose of this document is that the requirements mentioned in it should be utilized by software developer to implement the system.

1.2 Scope

This system allows the Teacher to maintain attendance record of the classes to which it is teaching. With the help of this system Teacher should be in a position to send e-mail to the students who remain absent for the class. The system provides a cumulative report at every month end for the corresponding class.

1.3 Overview

This system provides an easy solution to the Teacher to keep track of student attendance, and statistics.

2. General Description

This attendance maintenance system replaces the traditional, manual attendance system by which a lot of paper work will be reduced. The Teacher should able to view photograph of a student along with his attendance in his Laptop. This is the

primary feature of this system. Another feature is that Teacher can be allowed to edit particular record at desired time. The system should produce monthly attendance report. And there should be facility to send an e-mail/warning to the student remaining absent in the class.

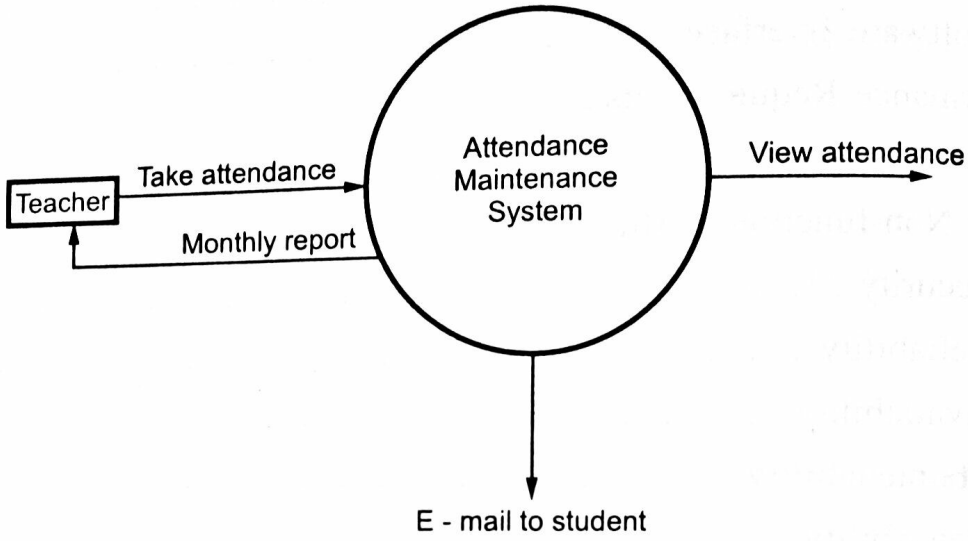


Fig. 4.3

Every Teacher should have Laptop with wireless internet connection. A Teacher may teach to different classes and a separate record for the corresponding classes should be maintained.

2.1 User Manual

The system should provide **Help** option in which how to operate the system should be explained. Also hard copy of this document should be given to the user in a booklet form.

3. Functional Requirements

3.1 Description

The identity of student is verified and then marked present at particular date and Time. The system should display student photograph along with their names for that corresponding class. The student may be marked present or absent depending upon his presence. The system should send e-mails to absent students. A statistical report should display individual's report or a cumulative report whenever required.

3.2 Technical Issues

The system should be implemented in VC++

4. Interface Requirements

4.1 GUI

GUI 1 : Main menu should provide options such as File, Edit, Report, Help

GUI 2 : In File menu one can create a new record file or can open an existing record file. For example : If file *SECOMP_SEMI_07* is opened then we may view attendance record of SE COMPUTER class in year 2007 and a record for semester-I can be viewed.

GUI 3 : The display of record should be

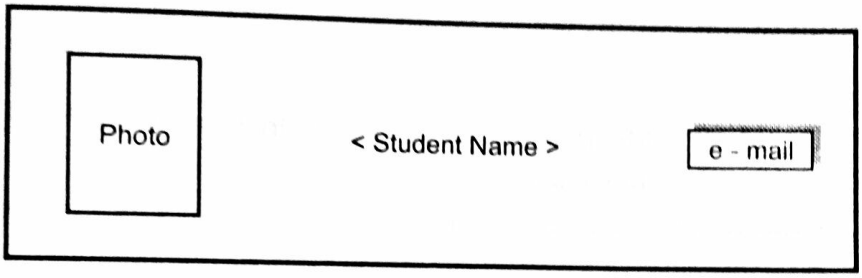


Fig. 4.4

The photo can be clicked to mark student present for particular class. The **e-mail** button can be clicked to send e-mail to the student being absent.

GUI 4 : Report option should display statistical report. It may be for particular student or for the whole class.

GUI 5 : Help option should describe functionality of the system. It should be written in simple HTML.

4.2 Hardware Interface

Hardware Interface 1 : The system should be embedded in the laptops.

Hardware Interface 2 : The laptop should use wireless Ethernet card to send e-mails. These Laptops should be the clients of departmental database server.

4.3 Software Interface

Software Interface 1 : Attendance maintenance system.

Software Interface 2 : The attendance database should be transmitted to departmental database server.

Software Interface 3 : E-mail message generator which generates standard message of absence.

Software Interface 4 : Report generators

5. Performance Requirements

This system should work concurrently on multiple processors between the college hours. The system should support 50 users.

The e-mail should be sent within one hour after the class gets over.

The system should support taking attendance of maximum 100 students per class.

6. Design Constraints

The system should be designed within 6 months.

7. Other Non-functional Attributes

7.1 Security

The teacher should provide password to log on to the system. He/she should be able to see the record of his/her class.

The password can be changed by the Teacher by providing existing password.

7.2 Reliability

Due to wireless connectivity , reliability can not be guaranteed.

7.3 Availability

The system should be available during college hours.

7.4 Maintenability

There should be a facility to add or delete or update Teachers and students for each semester.

7.5 Reusability

The same system will be used in each new semester.

8. Operational Scenarios

There will be student database, Teacher database. The student database will contain students name, class, attendance, e-mail address, address, phone number.

The Teacher database will contain Teacher's name, classes taught, e-mail address, phone number.

9. Preliminary Schedule

The system has to be implemented within 6 months.

4.6.3 Who are using SRS?

The SRS is the most important software document that is needed by various stakeholders. Following Fig. 4.5 illustrates the typical users of SRS.

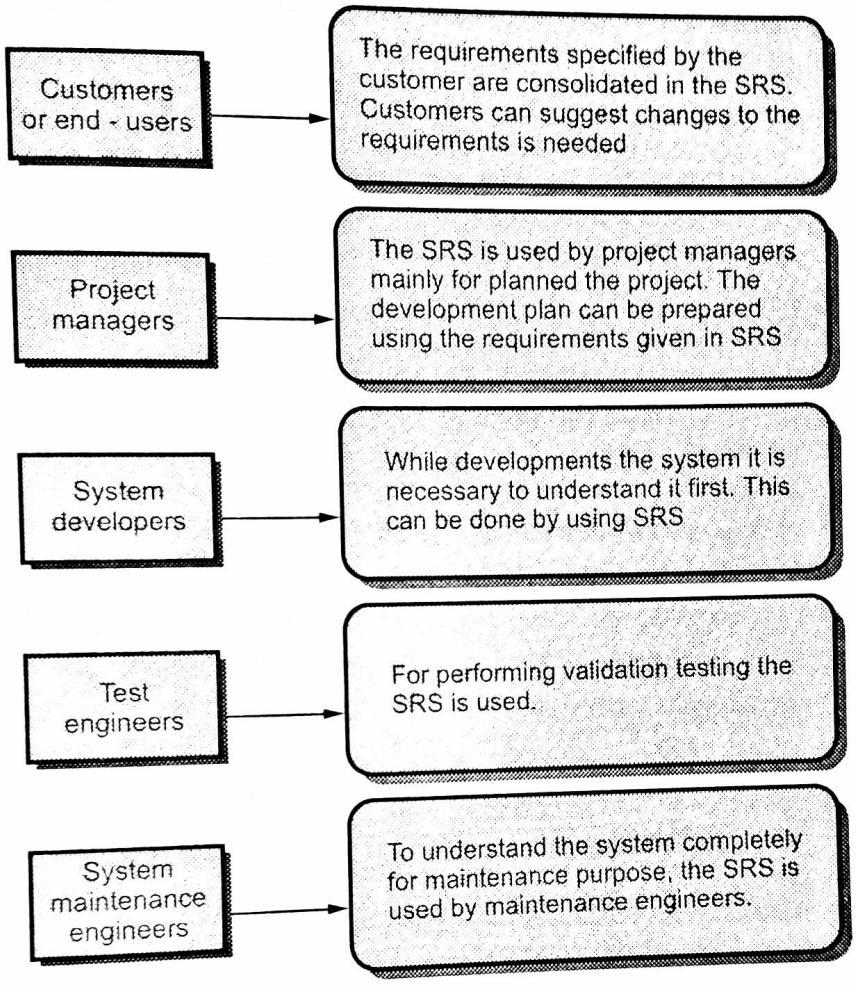


Fig. 4.5 Users of software requirement specification

Thus in this chapter we have discussed about software requirements. In the next chapter we will focus on requirements engineering process.

Review Questions

1. *What is requirement engineering ? Give different types of requirements with suitable example.*
2. *Explain functional requirements with suitable example. Discuss the problems associated with these requirements.*
3. *What do you mean by non function requirements ? Give various types of non functional requirements.*
4. *What are the metrics used for non functional requirements ?*

6 System Models

6.1 Introduction

As per our discussion in previous chapter, the most common way to specify the user requirements is a natural language. It is the simplest method of mentioning the requirements of the system. But at the same time there are many problems associated with the use of natural language. Hence another method has came up to specify the user requirements. That is creation of **System models**. The system model is a graphical representation that is used to describe various processes of the system, the type of input and output of the system. These system models not only specify the user requirements but they also serve as an important element in **analysis and design phase**. We can develop the system model using different perspectives and use of these perspectives gives the categorization of system models into different models –

- Using **External Perspective** *context model* of the system can be developed.
- Using **Behavioural perspective** *behavioural model* of the system can be developed.
- Using **Structural perspective** *data model* of the system can be developed.

Thus representation of the system should depict salient characteristics of the system. In this chapter we will discuss how to use different perspective of the system and how to create different models. And hence context model, behavioural model and data model can be understood with the help of some examples. Along with these models we discuss one more important system model called **object model** which is useful for object oriented systems. Let us start our discussion on system models by the very important concept i.e. analysis and designing.

6.2 Analysis and Modelling

Analysis modelling is a technical representation of the system.

- The software engineer (basically called as analyst) builds the model using the requirements elicited from customer.
- In analysis modeling a combination of text and diagrams are used to represent the software requirements in an understandable manner.
- By building analysis models it becomes easy to uncover requirement inconsistencies and omissions.
- Analysis Model Objectives –
- To describe what the customer requires.
- To establish a basis for the creation of a software design.
- To devise a set of valid requirements after which the software can be built.

Analysis Modeling Approaches –

Analysis Modelling Approach	
Structured Approach	**Object Oriented Approach**
The analysis is made on data and processes in which data is transformed as separate entities.	The analysis is made on the classes and interaction among them in order to meet the customer requirements.
Data objects are modeled in a way in which data attributes and their relationship is defined in structured approach.	Unified modelling language and unified processes are used in object oriented modelling approach.

- Team chooses one approach and makes use of all the representations from it. But the effective technique is to choose best from both the approaches.
- The model representation should be such that the best model of software requirements should be given to stakeholders. And this model should help in building software design.

6.3 Context Model

Context model is a graphical representation of the system in which the system **boundaries** are specified. This is the model which represents the **system environment** in which system is working. While deciding the environmental factors of the system it is very much necessary to take decisions on certain aspects such as overall cost of the system and time required to analyse such system. This decision should be made in the

early stage (preferably at requirement elicitation and analysis phase only). For example –

Consider the ***Inventory Control System*** for which the context model of the system can be created.

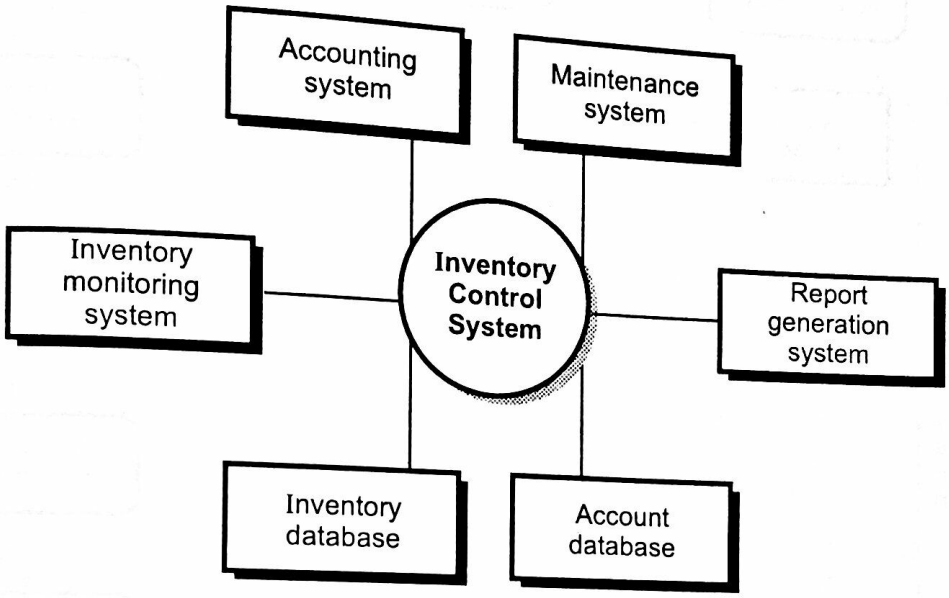


Fig. 6.1 Context model for Inventory Control System

First of all using the requirements of the system the system boundaries are decided and the dependencies of the system are specified in order to define the system environment. In Fig. 6.1, the system boundaries are clearly shown and the environmental factors are defined. The Inventory Control System is connected to various systems such as Inventory monitoring system, Accounting system, Report generation system and Maintenance system. This is an **architectural model** in which an abstract representation of inventory control system is given

Thus context models are the architectural models in which environment of the system is shown. The relationship that exists with other systems is shown but nature of these relationships is not mentioned in the context model. All these defined relationships help in finding the requirements of the system. To detail out such architectural model some **process model** can be used in conjunction with this. The process model is again a graphical representation of system processes. The order of execution of various events can be understood with the help of such process models. For example – For the same Inventory Control System we can draw a process model as shown by Fig. 6.2.

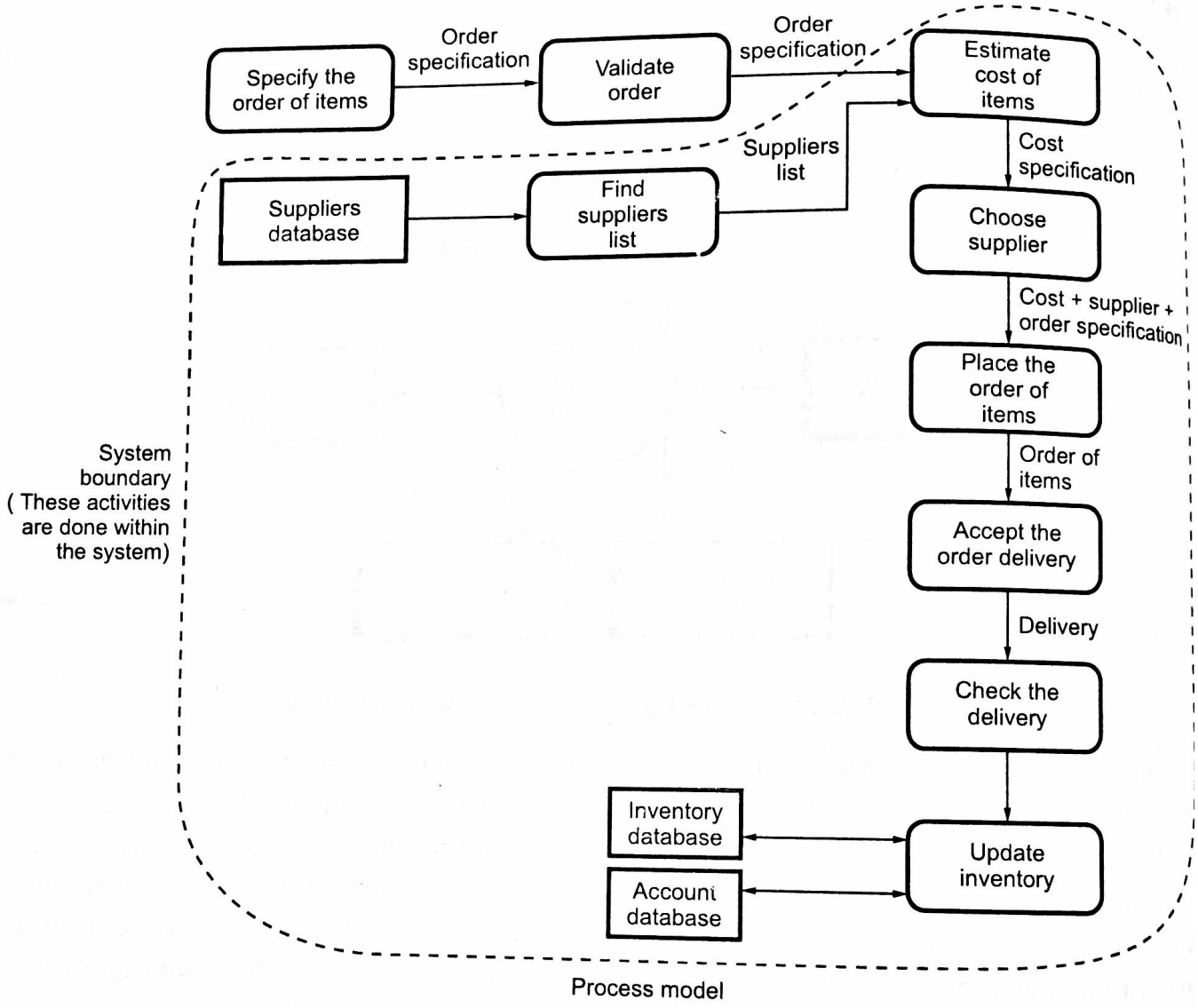
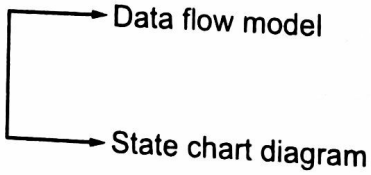


Fig. 6.2 Process model

6.4 Behavioural Models

- Behavioural models are used to describe the overall behaviour of a system. There are two types of models that depict the behaviour of the system



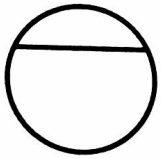
- The data flow model represents the flow of data and state chart diagram represent the states that are occurring in the system.

- These models can be used separately on together depending upon nature of application.

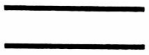
Let us discuss these models in detail –

6.4.1 Data Flow Diagram

- The data flow diagrams depict the information flow and the transforms that are applied on the data as it moves from input to output.
- The symbols that are used in data flow diagrams are –



Process



Data store

Flow of data (may be input data of output data)

External entity

- The data flow diagrams are used to represent the system at any level of abstraction.
- The DFD can be partitioned into levels that represent increase in information flow and detailed functionality.
- A level 0 DFD is called as 'fundamental system model' or 'context model'. In the context model the entire software system is represented by a bubble with input and output indicated by incoming and outgoing arrows.
- Each process shown in level 1 represents the sub functions of overall system.
- The number of levels in DFD can be increased until every process represents the basic functionality.
- As the number of levels gets increased in the DFD, each bubble gets refined. The Fig. 6.3 shows the leveling in DFD. Note that the information flow continuity must be maintained.

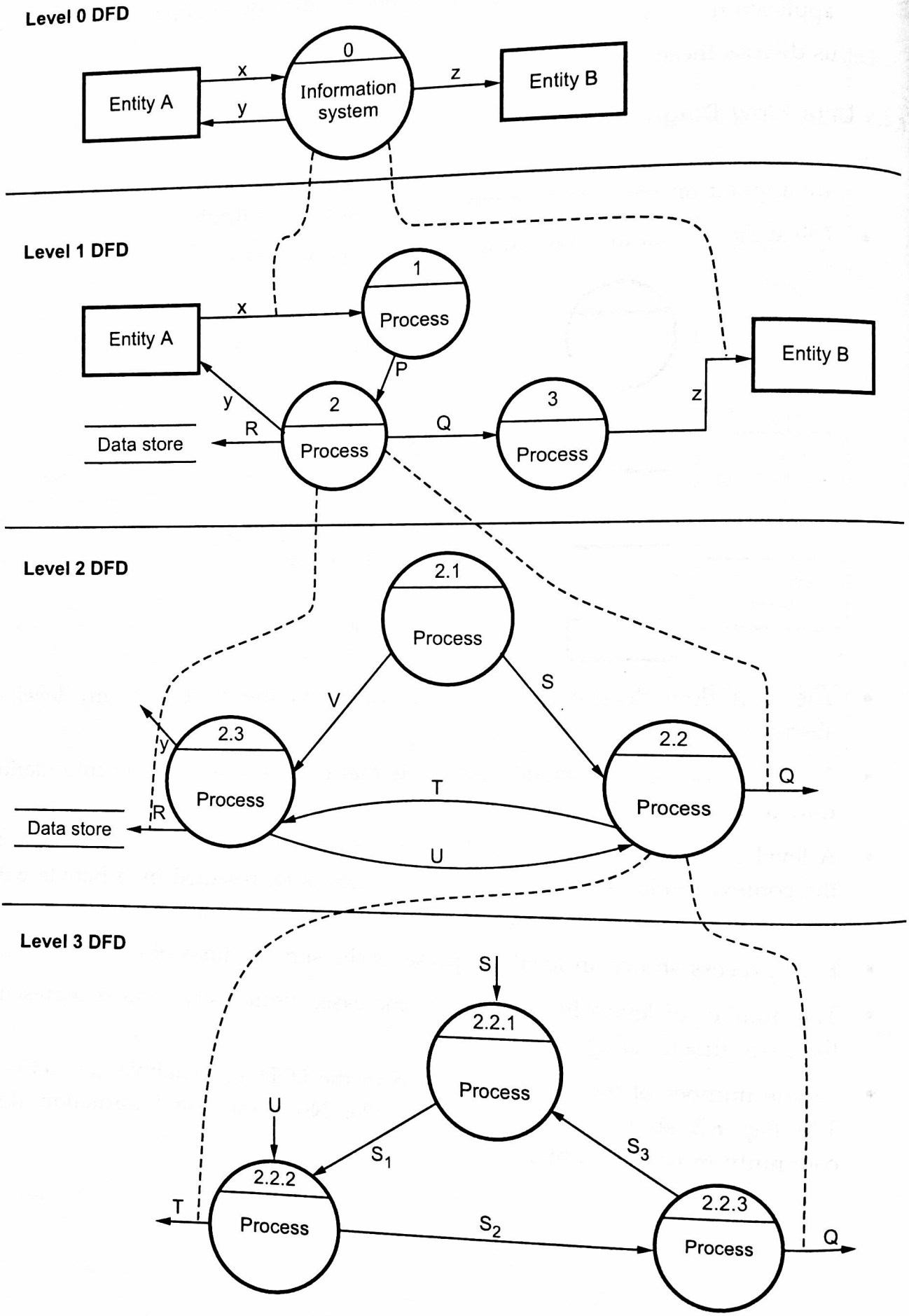


Fig. 6.3 Levels of DFD

6.4.1.1 Control Flow Diagram

- The control flow diagrams show the same processes as in data flow diagrams but rather than showing data flow they show control flows
- The control flow diagrams show how events flow among processes. It also shows how external events activate the processes.
- The dashed arrow is used to represent the control flow or event.
- A solid bar is used to represent the window. This window is used to control the processes used in the DFD based on the event that is passed through the window.
- Instead of representing control processes directly in the model the specifications are used to represent how the processes are controlled.
- There are two commonly used representations of specifications : Control Specification (CSPEC) and Process Specification (PSPEC).
- The CSPEC is used to indicate –
  1. How the software behaves when an event or signal is sensed.
  2. Which processes are invoked as a consequence of the occurrence of event?
- The PSPEC is used to describe the inner workings of the process represented in a flow diagram.
- When a data input is given to the process a **data condition** should occur to get the control output. For Example

See Fig. 6.4 on next page

6.4.2 State Chart Diagram

To understand the design of state chart diagram consider following example –

Consider an elevator for n floors has n buttons one for each floor. The working of such elevator can be given as

1. There is a set of buttons called 'elevator buttons'. If we want to visit a particular floor then the elevator button for corresponding floor is pressed. It causes an illumination and elevator starts moving to visit the desired floor. The illumination is cancelled on reaching to destination.
2. There is another set of buttons called 'floor button' .When a person on particular floor want to visit another floor then the floor button has to be pressed. This makes an illumination at floor button and the elevator starts moving towards the floor where on the person is. And illumination is cancelled when the elevator reaches on the desired floor.

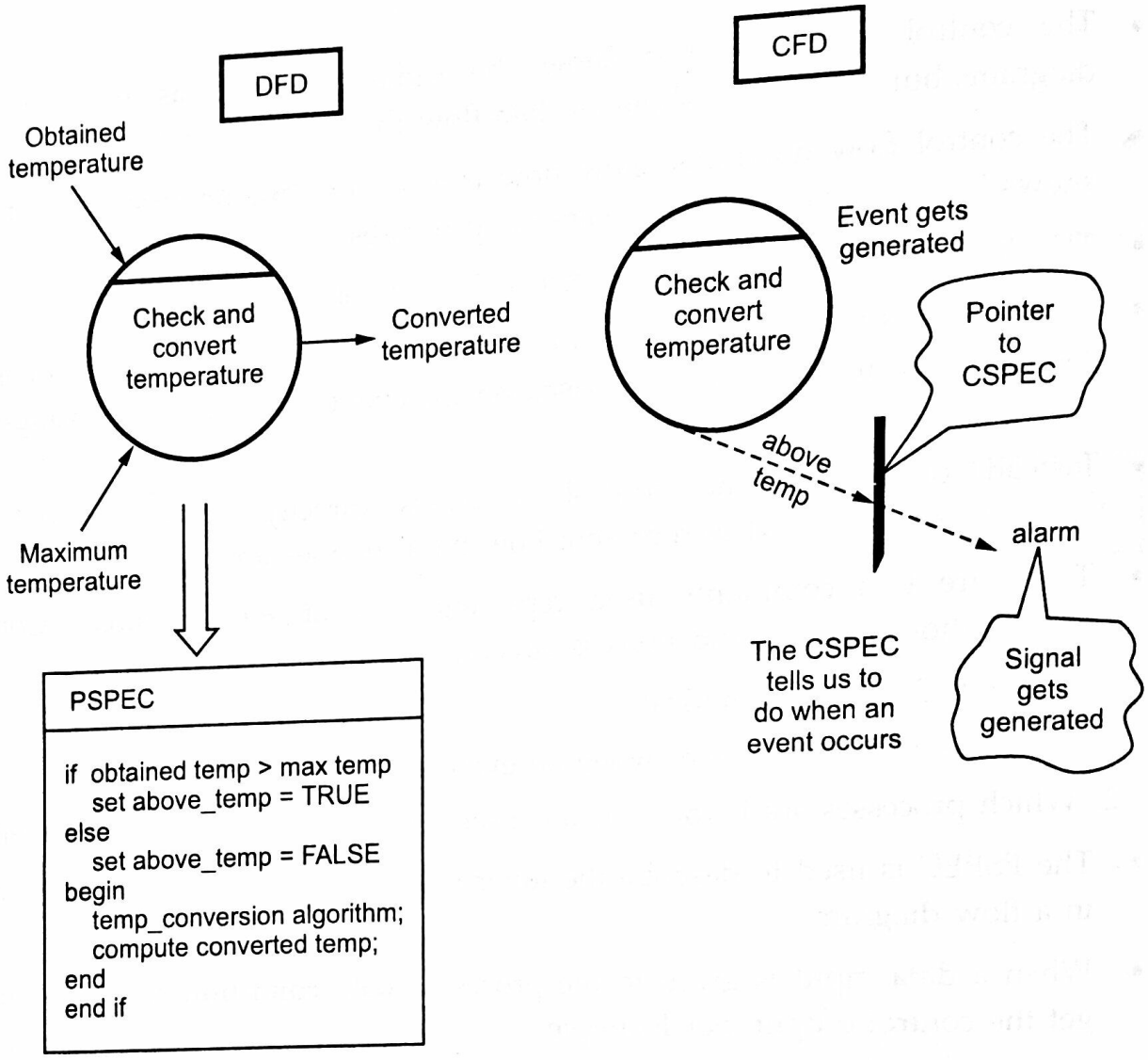


Fig. 6.4 Occurrence of data condition

3. When an elevator has no request it remains at its current floor with its door is closed.

The state chart diagram is as shown in Fig. 6.5.

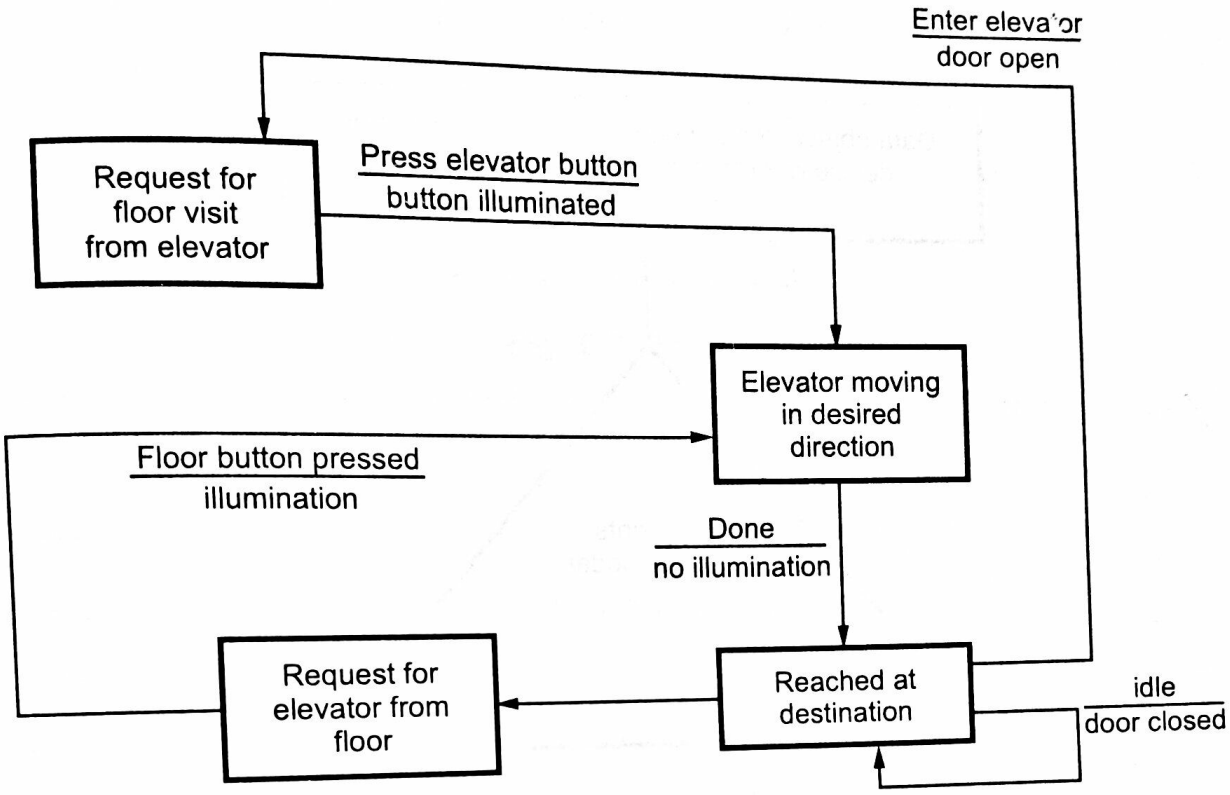


Fig. 6.5 State chart diagram

6.5 Data Model

- Data modelling is the basic step in the analysis modelling. In data modelling the data objects are examined independently of processing.
- The data domain is focused. And a model is created at the customer's level of abstraction.
- The data model represents how data objects are related with one another.

6.5.1 Data Objects, Attributes and Relationships

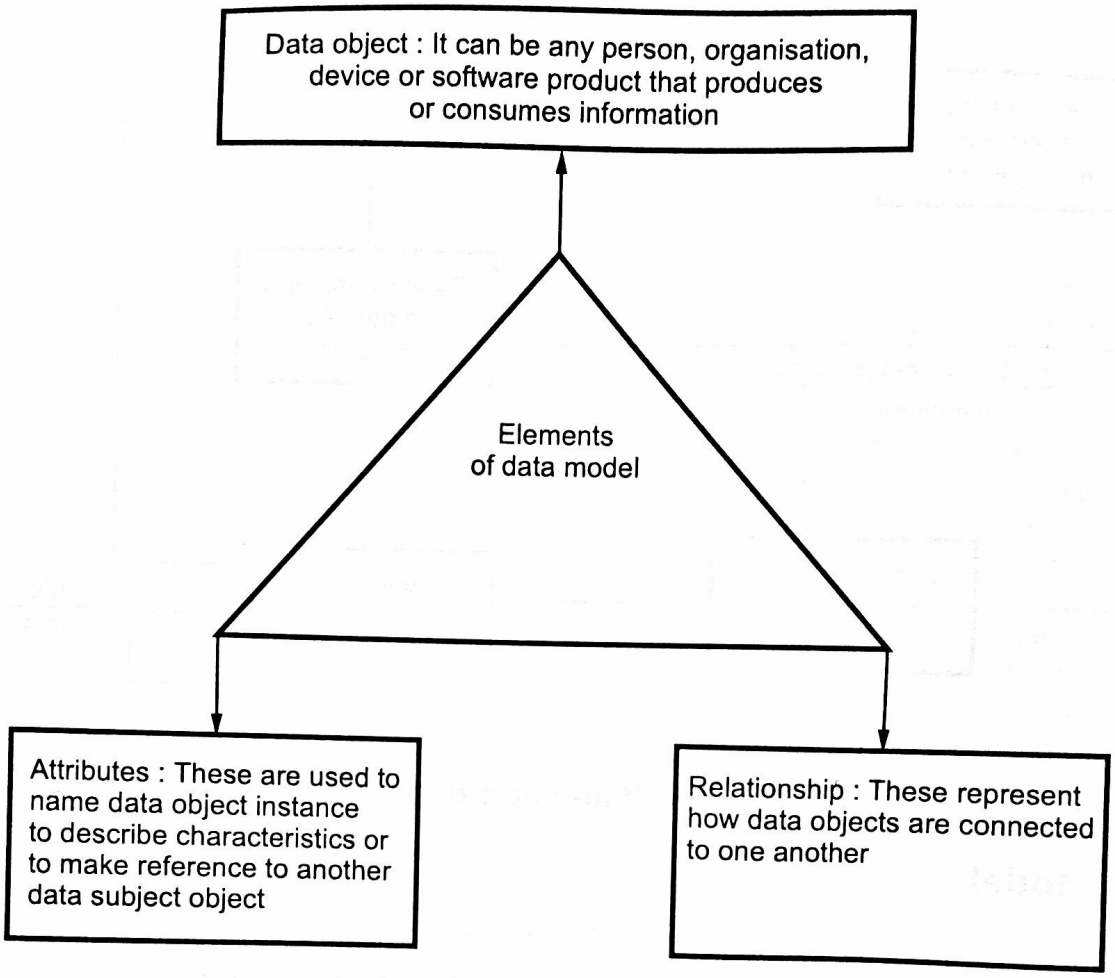


Fig. 6.6 Elements of data model

What is data object?

- Data object is a set of attributes (data items) that will be manipulated within the software(system).
- Each instance of data object can be identified with the help of unique identifier. For example: A student can be identified by using his roll number.
- The system can not perform without accessing to the instances of object.

Each data object is described by the attributes which themselves are data items.

Data object is a collection of attributes that act as an aspect, characteristic, quality, or descriptor of the object.

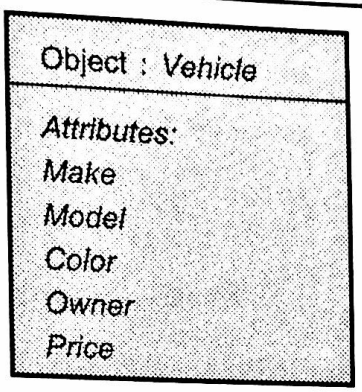


Fig. 6.7 Object

The vehicle is a data object which can be defined or viewed with the help of set of attributes.

Typical data objects are

- External entities such as printer, user, speakers
- Things such as reports, displays, signals
- Occurrences or events such as interrupts, alarm, telephone call
- Roles such as manager, engineer, customer
- Organizational units such as division, departments
- Places such as manufacturing floor, workshops
- Structures such as student records, accounts, file

What are attributes?

Attributes define properties of data object

Typically there are three types of attributes –

1. **Naming attributes** – These attributes are used to name an instance of data object. For example : In a *vehicle* data object *make* and *model* are naming attributes.
2. **Descriptive attributes** – These attributes are used to describe the characteristics or features of the data object. For example : In a *vehicle* data object *color* is a descriptive attribute.
3. **Referential attribute** – These are the attributes that are used in making the reference to another instance in another table. For example : In a *vehicle* data object *owner* is a referential attribute.

What is relationship?

Relationship represents the connection between the data objects. For example The relationship between a shopkeeper and a toy is as shown below

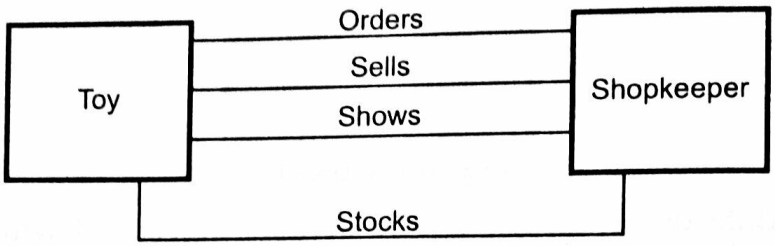


Fig. 6.8 Relationship

Here the toy and shopkeeper are two objects that share following relationships-

- Shopkeeper orders toys
- Shopkeeper sells toys
- Shopkeeper shows toys.
- Shopkeeper stocks toys.

6.5.2 Cardinality and Modality

Cardinality in data modelling, cardinality specifies how the number of occurrences of one object is related to the number of occurrences of another object.

- One to one (1:1)- one object can relate to only one other object.
- One to many(1:N)- one object can relate to many objects.
- Many to many (M:N) - some number of occurrences of an object can relate to some other number of occurrences of another object.

Modality indicates whether or not a particular data object must participate in the relationship.

Modality of a relationship is 0 (zero) if there is no explicit need for the relationship to occur or the relationship is optional. The modality is 1 (one) if an occurrence of the relationship is mandatory.

Example

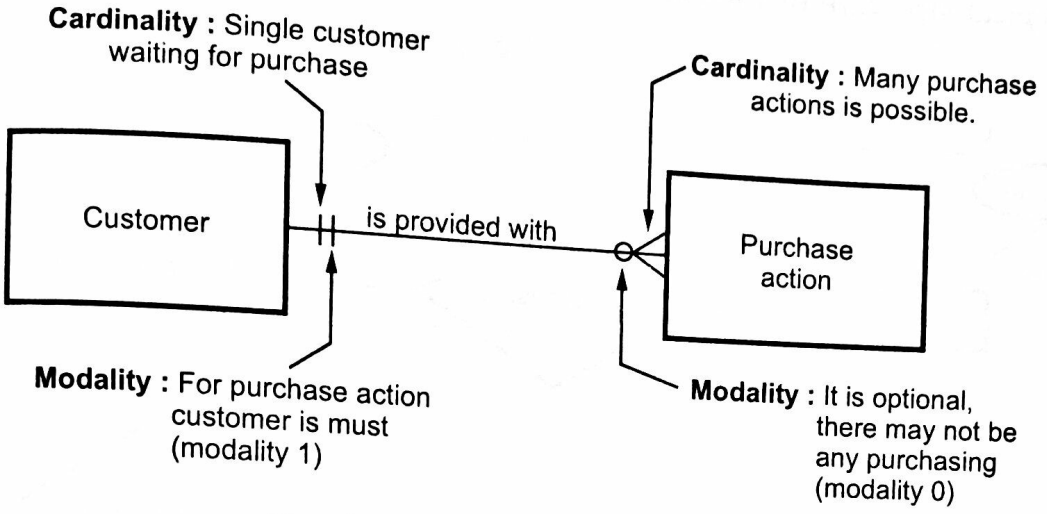


Fig. 6.9 Cardinality and modality

6.5.3 Entity Relationship Diagram

- The object relationship pair can be graphically represented by a diagram called Entity Relationship Diagram(ERD).
- The ERD is mainly used in database applications but now it is more commonly used in data design.
- The ERD was originally proposed by Peter Chen for design of relational database systems.
- The primary purpose of ERD is to represent the relationship between data objects.
- Various components of ERD are –

Entity

- Drawn as a rectangle.
- An entity is an object that exists and is distinguishable.
- Similar to a record in a programming language with attributes.

Relationship

- Drawn as a diamond.
- An association among several entities.
- Relationships may have attributes.
- Relationships have cardinality (e.g., one-to-many)

Attribute

- Drawn as ellipses.
- Similar to record fields in a programming language.
- Each attribute has a set of permitted values, called the domain.
- Primary key attributes may be underlined.

The typical structure of ER-diagram is illustrated as follows.

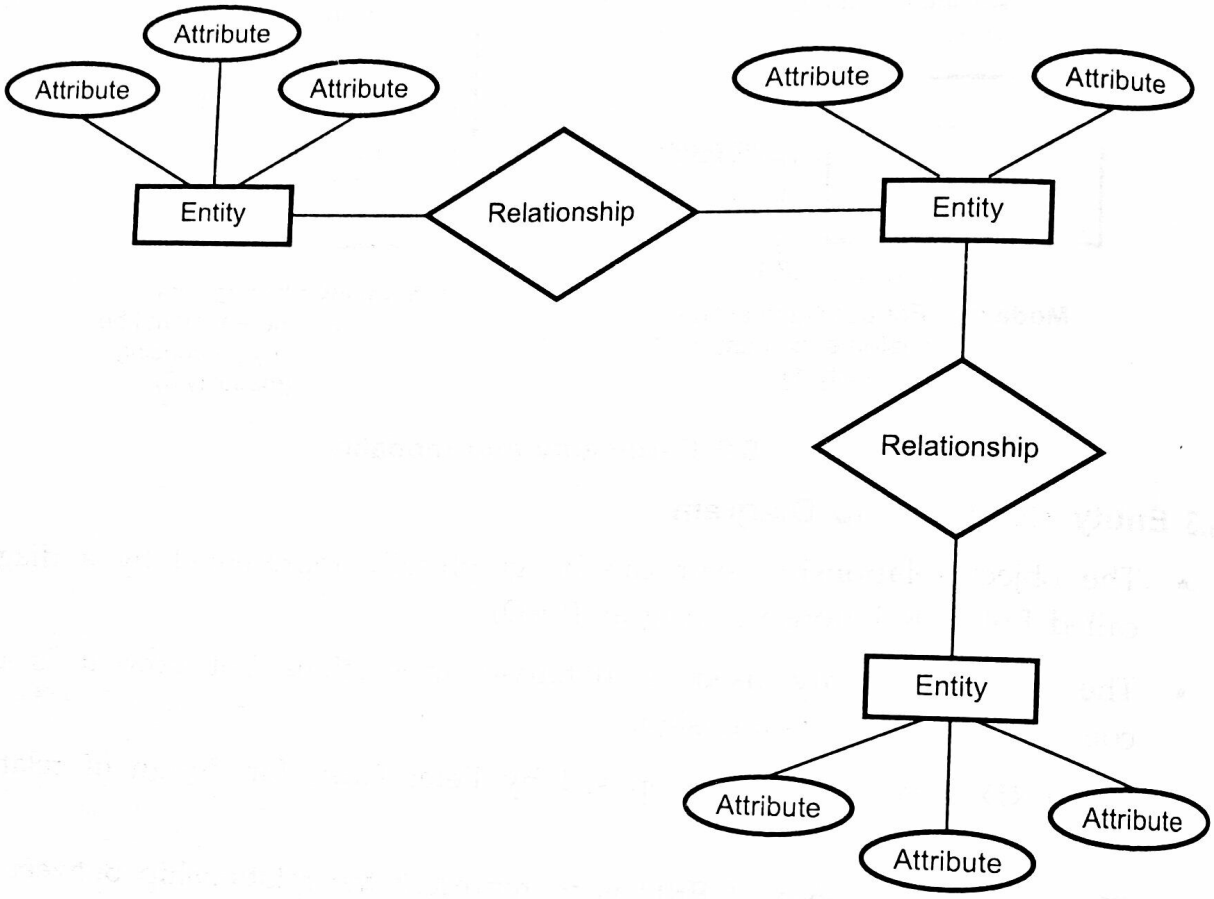
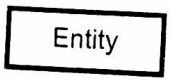


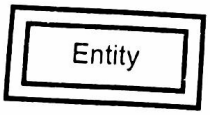
Fig. 6.10 ER diagram

Notations used in ER diagram



Entity

It is an object and is distinguishable it is similar to record.



Weak Entity

When this entity is dependant upon some another entity then it is called weak entity.

Attribute

The attributes are properties or characteristics of an entity.

Derived attribute

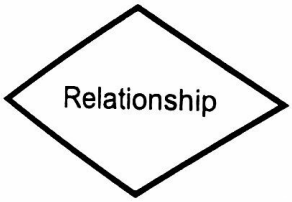
It is a kind of attribute which is based on another attribute.

Key attribute

A key attribute is an unique attribute representing distinguishing characteristic of entity. Typically primary key of record is a key attribute.

Multivalued attribute

A multivalued attribute have more than one value.



Relationship

When two entities share some information then it is denoted by relationship.

Notations to show cardinality

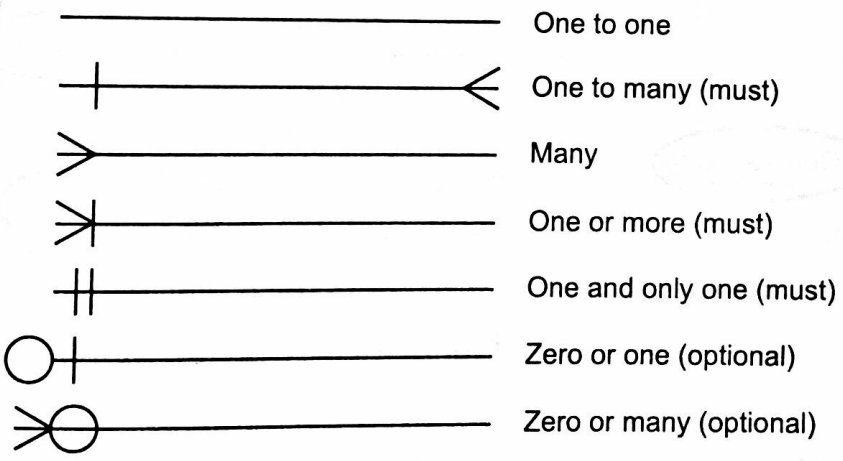


Fig. 6.11

Example 1

Draw an ER diagram for the realtiohship of Teacher and courses. Also specify the association, cardinality and modality.

Ans. :

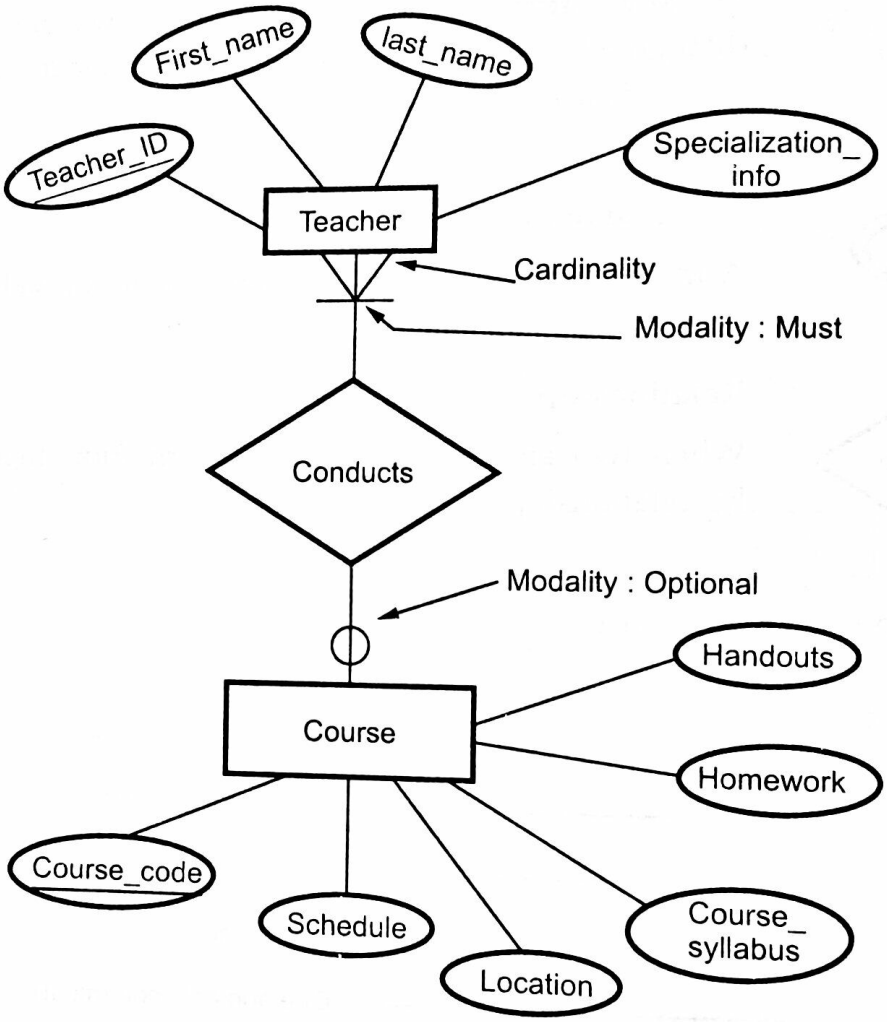


Fig. 6.12

Association

In the above ER diagram, a relatiohship **conducts** is introduced. "Teacher" is associated with "course" by conducting it.

Cardinality

Many Teachers can conduct the single course.

Modality

For conduction of a course, there must be a Teacher.

There may a situation that a Teacher is not conducting any course.

Example 2

Following ERD represents the relationship between customer and banking system.

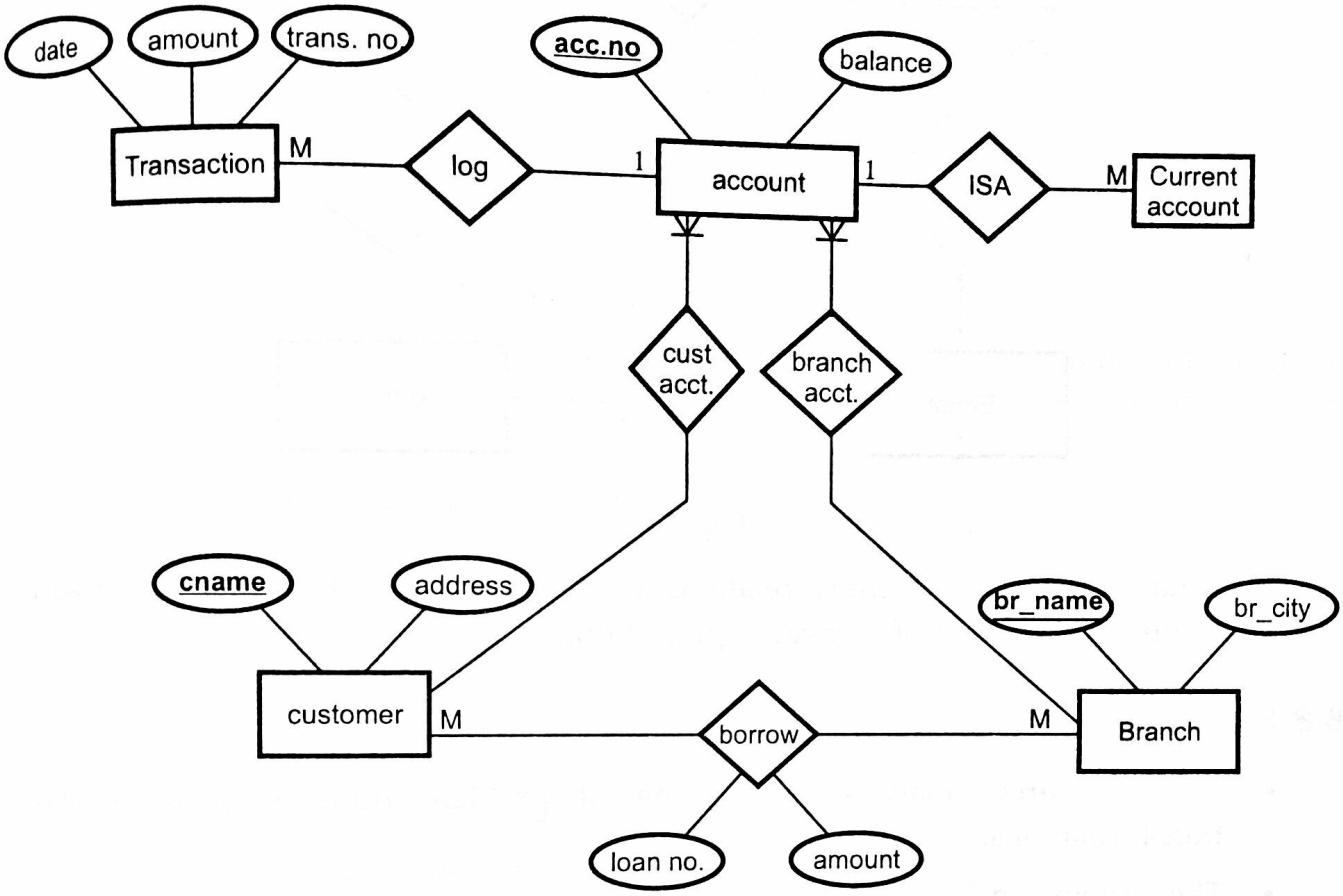


Fig. 6.13 ERD

Example 3

Technical Publication (A well known publishing company) publishes Engineering books on various subjects. The books are written by authors who specialize in his/her subject. The publication employs editors who take sole responsibility of editing one or more manuscripts. While writing a prarticular book, each author interacts with editor. But the author is supposed to submit his work to publisher always. In order improve competitiveness, the publication tries to employ a variety of authors who are specialist in more than one subject.

Draw the E-R diagram for above described scenario.

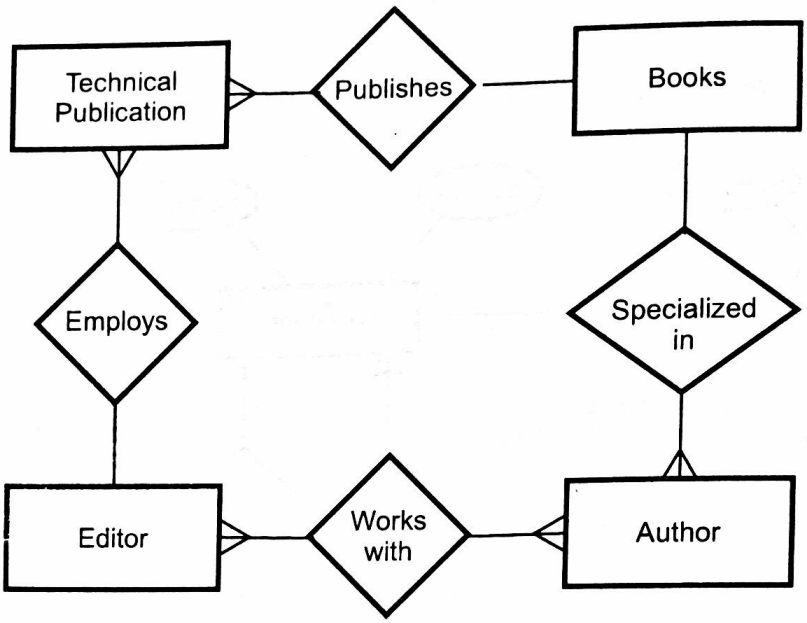


Fig. 6.14

The data modelling and entity relationship diagram helps the analyst to observe the data within the context of software application..

6.6 Structured Analysis

- The structured analysis is mapping of problem domain to flows and transformations.
- The system can be modeled using :
  - o Entity Relationship diagram are used to represent the data model.
  - o Data flow diagram and Control flow diagrams are used to represent the functional model.
- Along with system modeling the specification can be written for the system using

1. Process Specification 2. Control Specification.

6.6.1 Designing Entity Relationship Diagrams

The entity relationship diagram is used to represent the data objects their attributes and the relationship among these data objects. The ERD is constructed in iterative manner. Following guideline is used while drawing the ERD.

1. During the requirement elicitation process, the requirements should be collected in such a way that we can evolve input, output data objects and external entities for system modeling.
2. The analysis and customer should be in a position to define the relationship between the data objects.

3. When ever a connection between data objects is identified the object relationship pair must be established. Thus iteratively relationship between all the objects must be established.
4. For each object relationship pair the cardinality and modality is set.
5. The attributes of each entity must be defined.
6. The entity relationship diagram is formalized and reviewed.
7. All the above steps are repeated until data modeling is complete.

6.6.2 Designing Data Flow Diagrams

- The data flow diagrams are used to model the information and function domain. Refinement of DFD into greater levels helps the analyst to perform functional decomposition.
- The guideline for creating a DFD is as given below –

1. Level 0 DFD i.e. Context level DFD should depict the system as a single bubble.
2. Primary input and primary output should be carefully identified.
3. While doing the refinement isolate processes,data objects and data stores to represent the next level.
4. All the bubbles (processes) and arrows should be appropriately named.
5. One bubble at a time should be refined.
6. Information flow continuity must be maintained from level to level.

- A simple and effective approach to expand the level 0 DFD to level 1 is to perform "grammatical parse" on the problem description. Identify nouns and verbs from it. Typically nouns represent the external entities, data objects or data stores and verbs represent the processes. Although grammatical parsing is not a foolproof but we can gather much useful information to create the data flow diagrams.

Example 1 DFD for library information system.

A student comes to a library for borrowing book. The student makes the book request by giving book title and author name. The student has to submit his library card to the library. Sometimes student may simply give topic and demand for a book (For example "just give me book on data structure"). The library information system maintains list of authors, list of titles, list of topics. This system also keeps record of topics on which books are available with the system. This system maintains information about shelf number on which books are located. Finally the list of demanded book should be displayed, on the console for ease of selection.

The first level of DFD can be

Solution :

In this DFD the whole system is represented with the help of input, processing and output. The input can be -

i) Student requests for a book hence **Book request**.

ii) To show identity of the student he/she has to submit his/her Library card, hence **Library card**. The processing unit can be globally given as

Library information system

The system will produce following outputs-

i) The demanded book will be given to student. Hence **Book** will be the output.

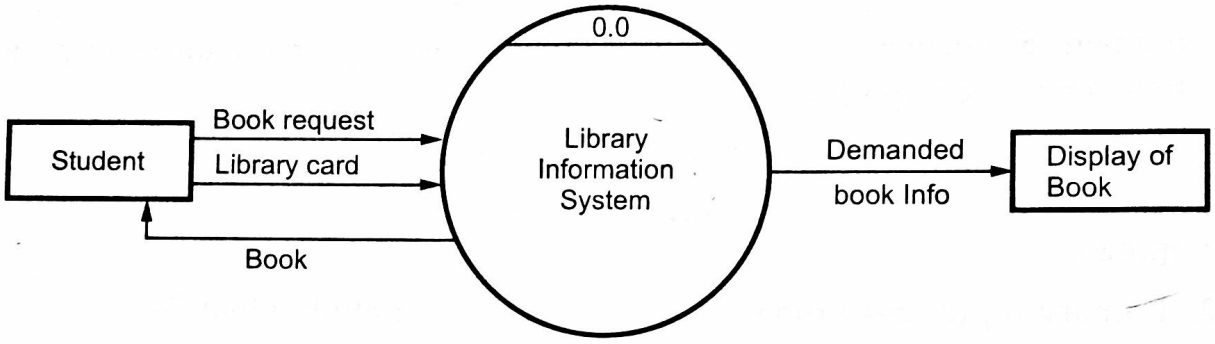


Fig. 6.15 Level 0 DFD (Context level DFD)

ii) The library information system should display **demanded book information** which can be used by customer while selecting the book.

Level 1 DFD

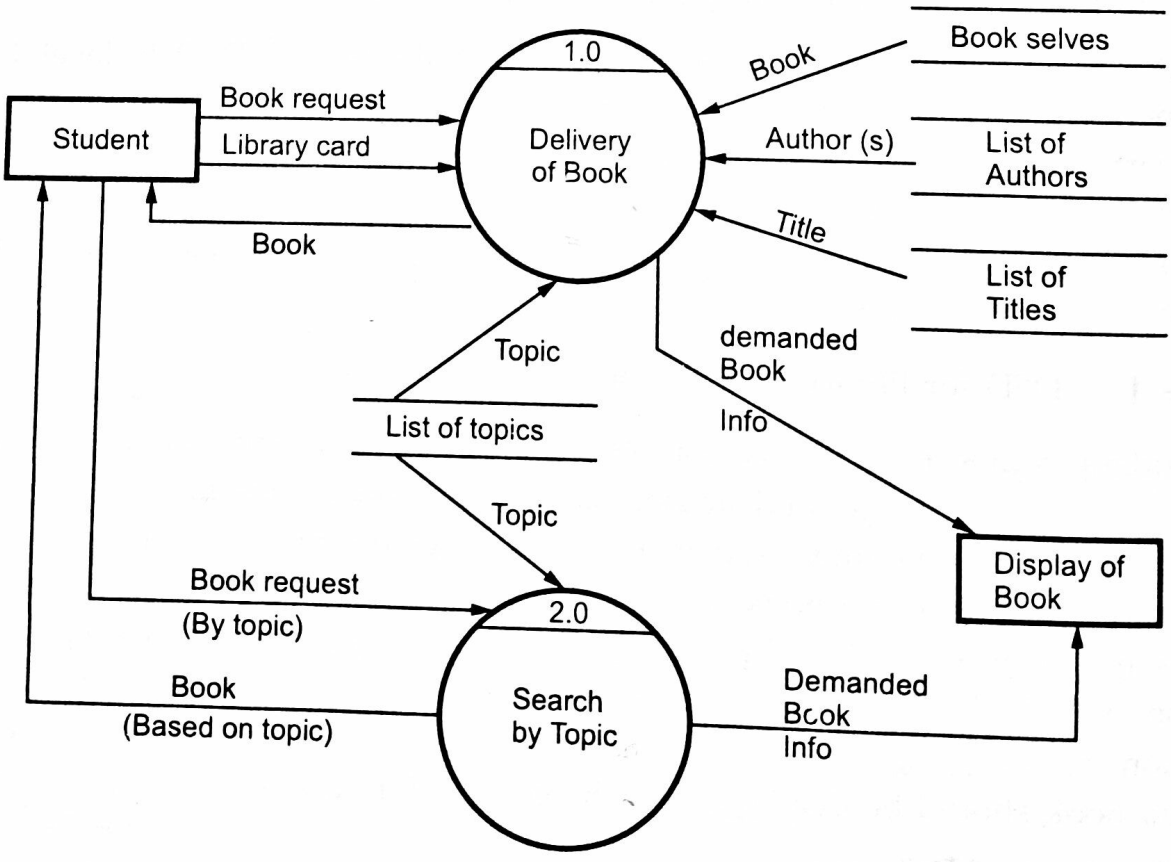


Fig. 6.16 Level 1 DFD

In this level, the system is exposed with more processing details. The processes that need to carried out are -

i) Delivery of Book.

ii) Search by Topic.

These processes require certain information such as List of Authors, List of Titles, List of Topics, the book selves from which books can be located. This kind of information is represented by **data store.**

Level 2 DFD

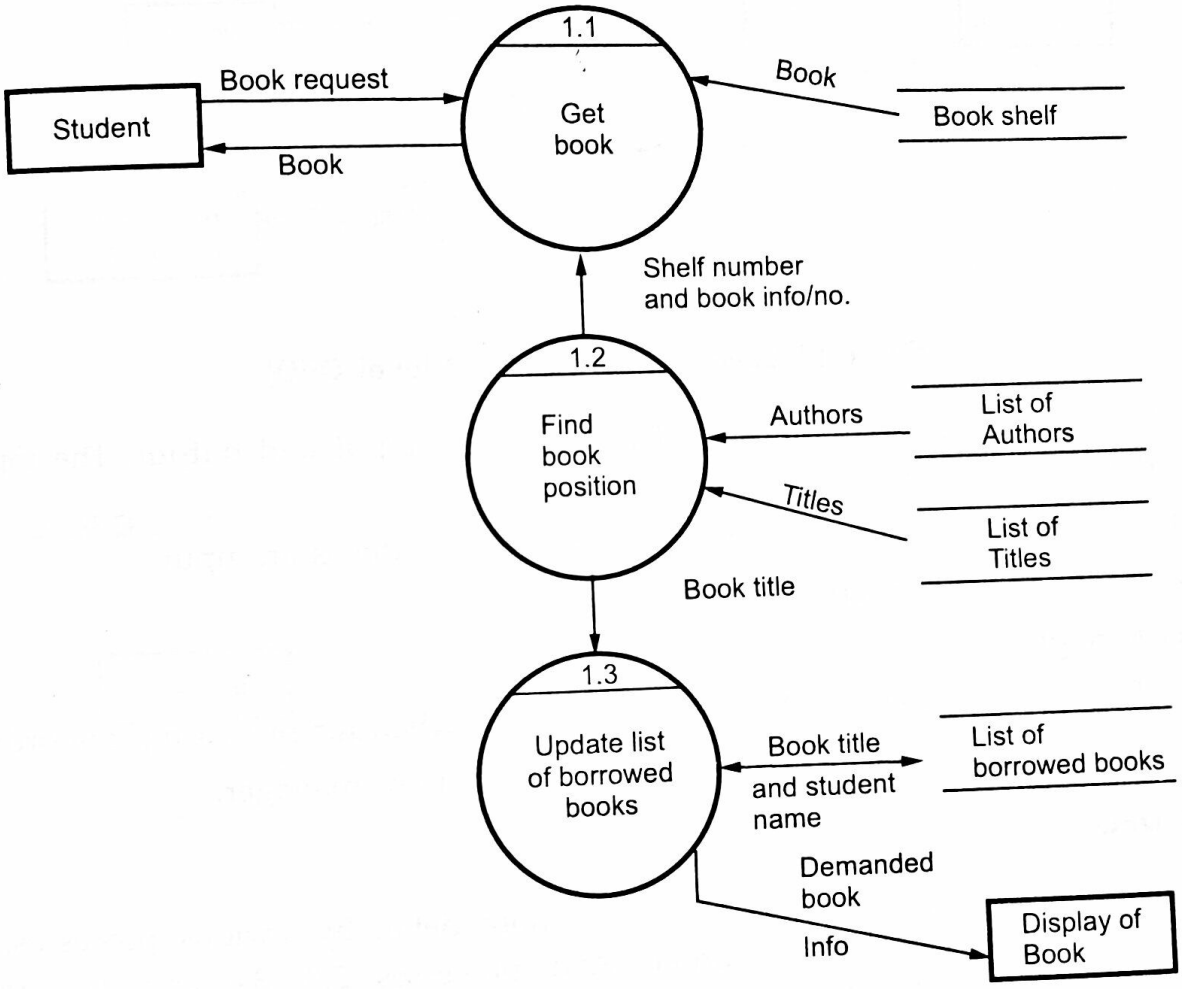


Fig. 6.17

Out of scope : The purchasing of new books/replacement of old books or charging a fine all these maintenance activities are not considered in this system.

Example 2 DFD for food ordering system.

A **customer** goes to a restaurant and orders for the food. The food order is noted down carefully and this order is sent to kitchen for preparing the required food.

This restaurant has to manage one housekeeping department which maintains **sold items** and **inventory data.** The daily information about sold items and inventory deplition amount is used to generate a **management report**. Finally this management report is given to **restaurant manager**.

Level 0 DFD

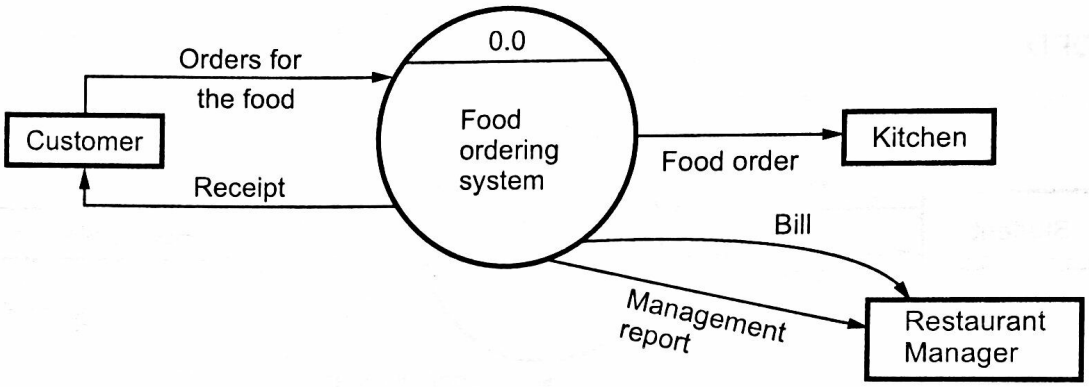


Fig. 6.18 Level 0 DFD (Context level DFD)

In this level, the system is designed globally with input and output. The input to Food ordering system are -

1. As customer orders for the food. Hence food order is an input.

The output to food ordering system are -

1) Receipt.

2) The food order should be further given to kitchen for processing the order.

3) Bill and management report is given to restaurant manager.

Level 1 DFD

In this level, the bubble 0.0 is shown in more detail by various processes. The process 1.0 is for processing an order. And processes 2.0, 3.0 and 4.0 are for housekeeping activities involved in food ordering system. To create a management report there should be some information of daily sold items. At the same time inventory data has to be maintained for keep track of 'instock' items. Hence we have used two **data stores** in this DFD -

1. Database of sold items

2. Inventory database.

Finally management report can be prepared using daily sold details and daily inventory deplition amount. This management report is given to restaurant manager.

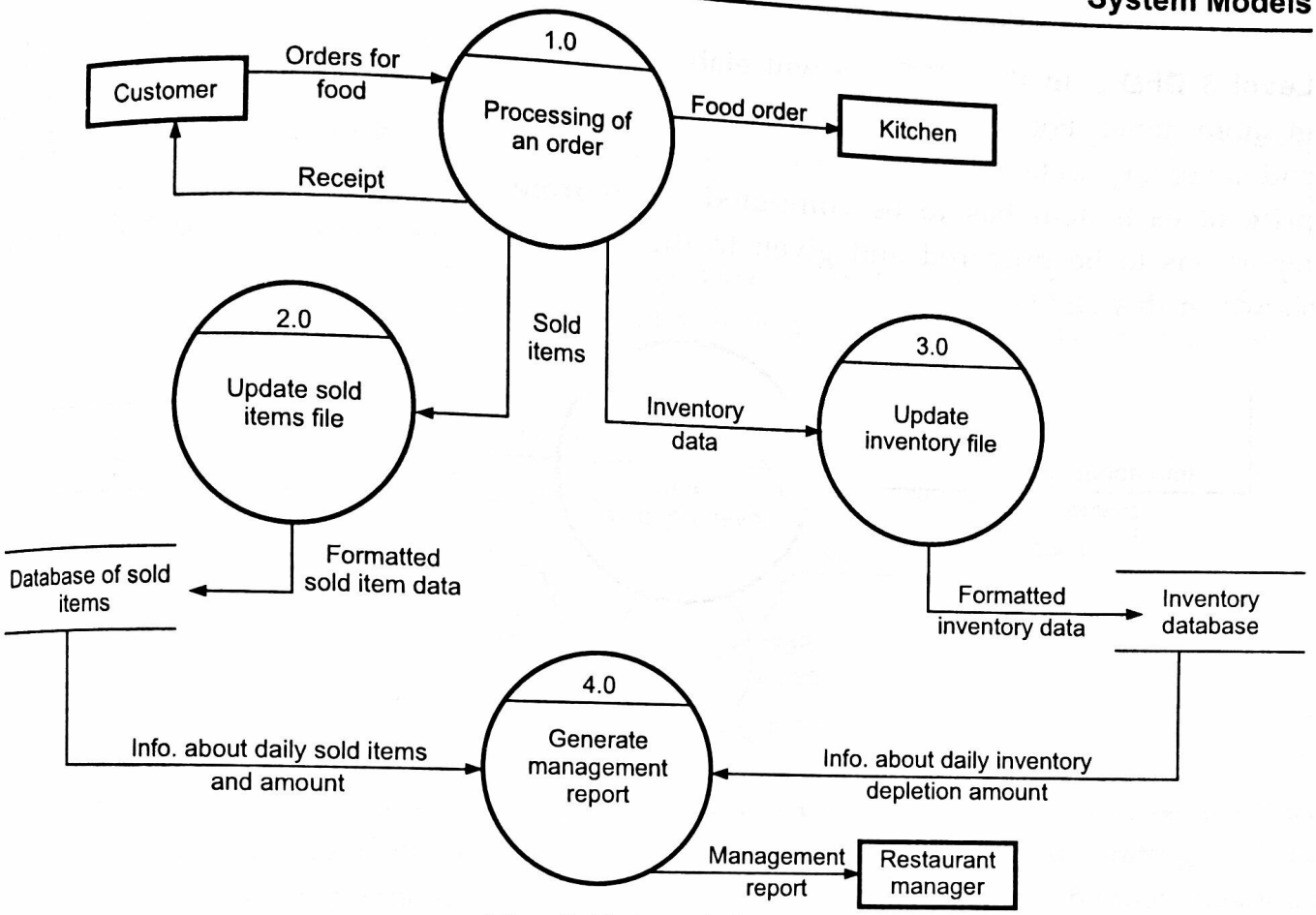


Fig. 6.19 Level 1 DFD

Level 2 DFD : "Processing of an order" is shown in detail.

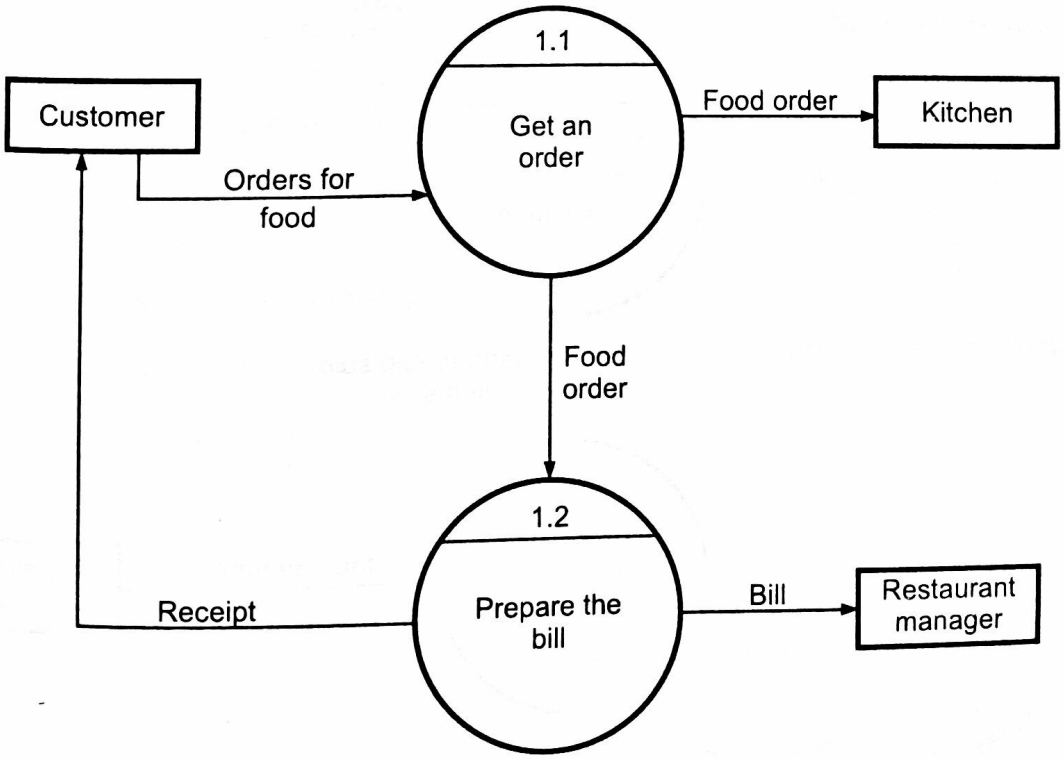


Fig. 6.20 Level 2 DFD

Level 3 DFD : In this DFD we will elaborate "Generate management report" activity in more detail. For generating management report we have to access sold items data and inventory data. Then aggregate both solid items data and inventory data. Total price of each item has to be computed. Then from these calculation a management report has to be prepared and given to the restaurant manager. These details can be shown in this DFD -

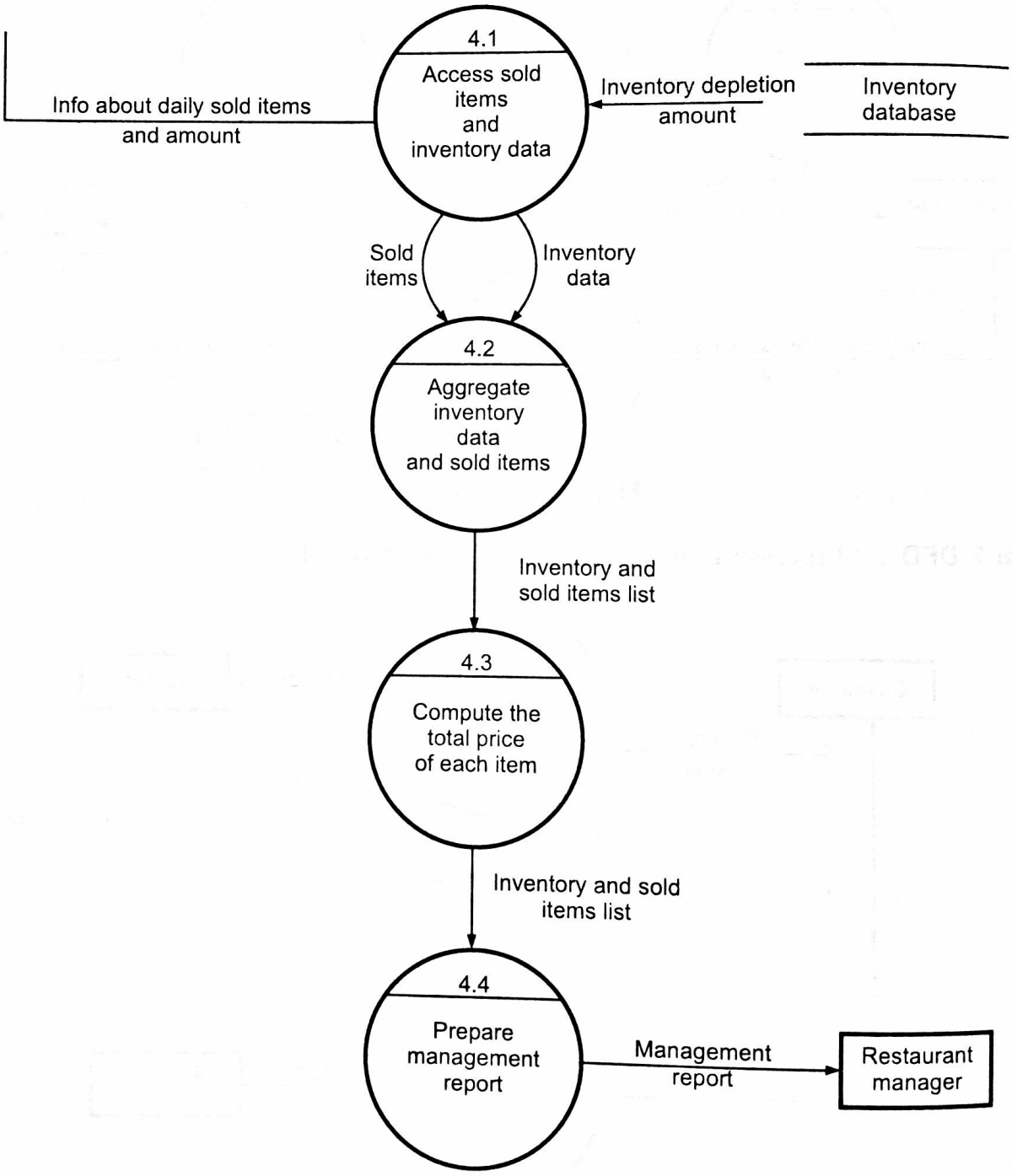


Fig. 6.21 Level 3 DFD

Example 3

Prepare a CFD for a Books order processing system. The **customers** in this system are book sellers who do not make a stock of books. As the **orders** come the books are demanded from **publishers** directly.

As per the problem description it is clear that the input to this system is 'customer' who places an order and output will be purchase order given to publisher.

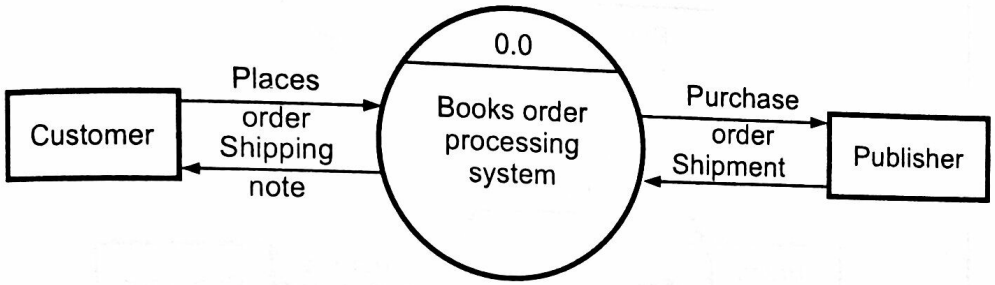


Fig. 6.22 Level 0 CFD

Now, we will do more detailing for order processing when an order is received by the system, it will be first verified using the books database. Credit rating will be decided by checking customer details (i.e. whether the customer is regular customer or whether he is new). For submitting a batch of orders we have to maintain pending orders list as well. There may be the order for books which are published by different publishers, in such a case database for different publishers need to be maintained by the system. This list of purchase orders is maintained by the system and then after verifying the shipment a shipping note will be given to the customer. The level 1 decomposition is as shown in Fig. 6.23.

We can further decompose the system by elaborating the process **Assemble order.** In assemble order we

- First get details of total copies per title.
- Then prepare purchase order accordingly for corresponding publishers.
- Send these purchase orders to publishers.

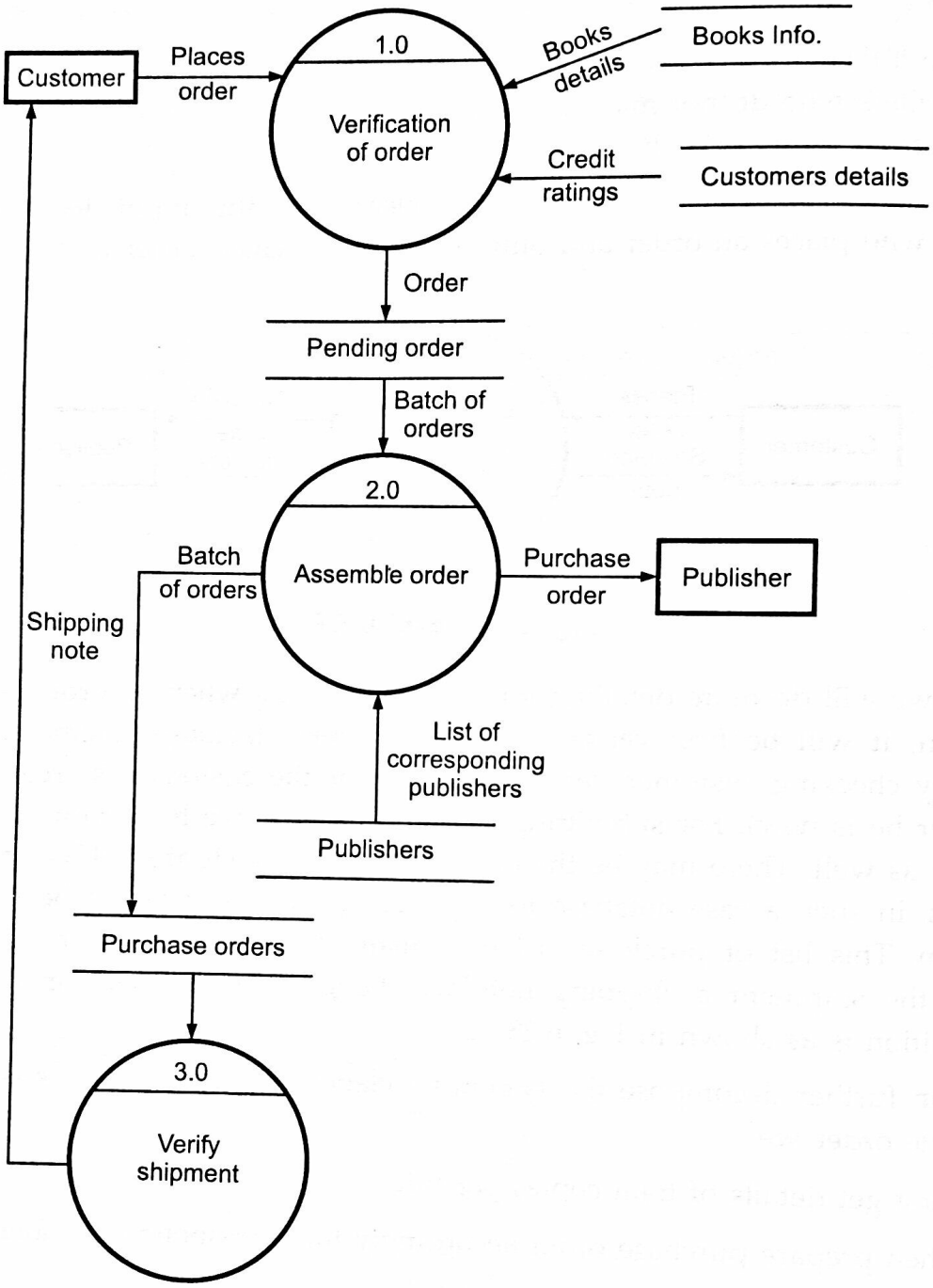


Fig. 6.23 Level 1 CFD

- These purchase order details should be stored in the system. Let us draw level 2 CFD for these details.

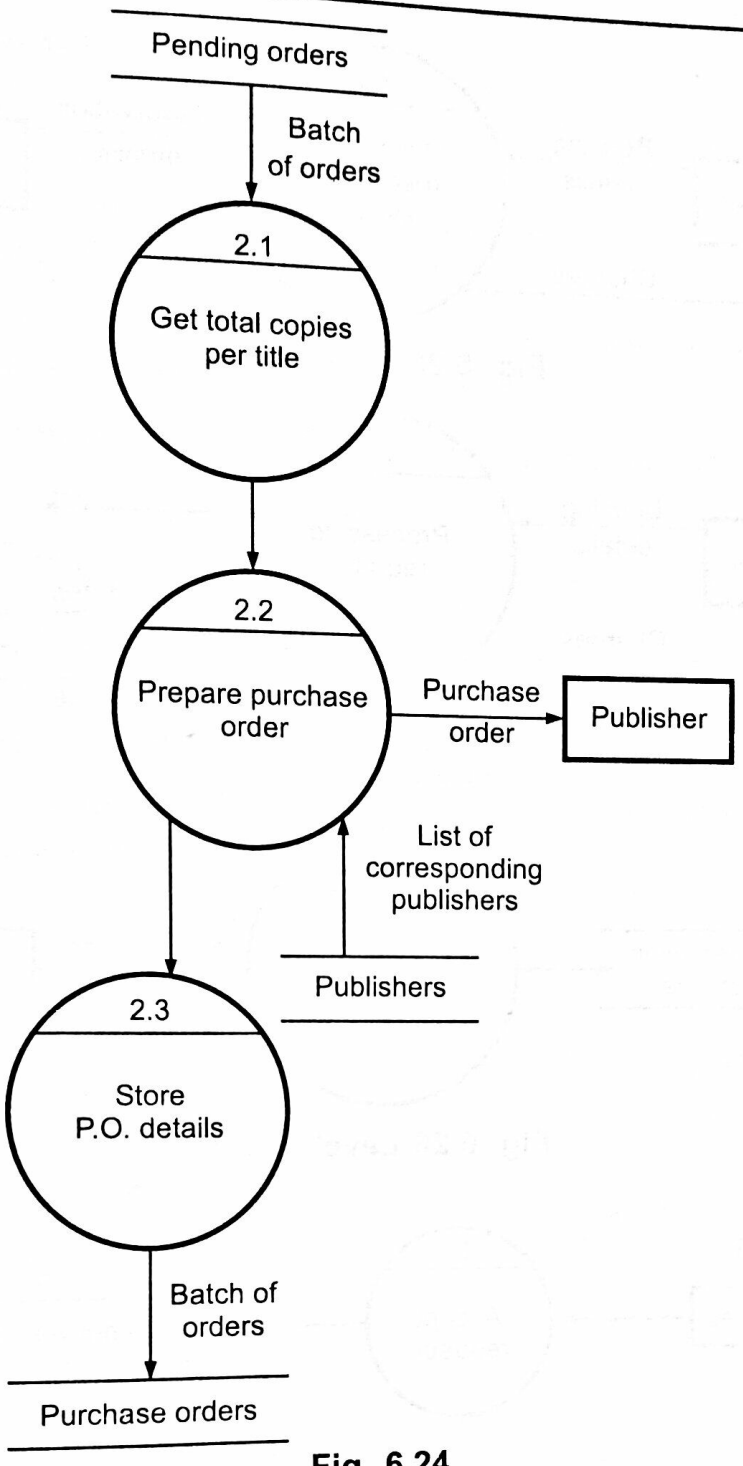


Fig. 6.24

Example 4

In a hotel reservation system, a **customer** can make online booking for a hotel, by specifying the accomodation requirements such as type of room (AC/Non AC/One bed/Two bed), total number of rooms, duration of stay. The system then **selects** a suitable hotel as per customer's requirements. If such a hotel is found then the **availability** of rooms in that hotel is **checked.** The **charges are calculated** for the selected requirement and these are acknowledged to the customer. If the customer is satisfactory about the selection made by the system then he **confirms** the reservation.

The system then gives these booking details to the corresponding **hotel**. Design DFD for this system.

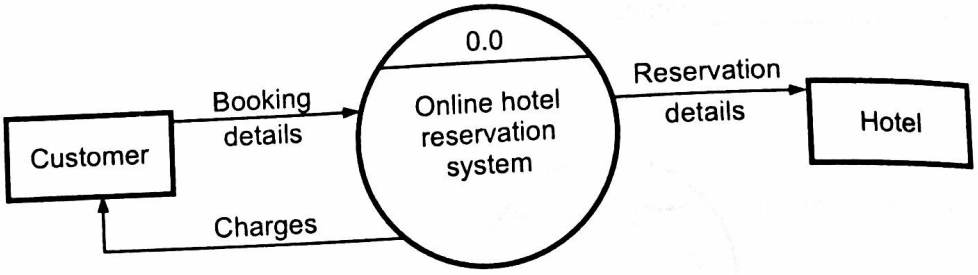


Fig. 6.25 Level 0 DFD

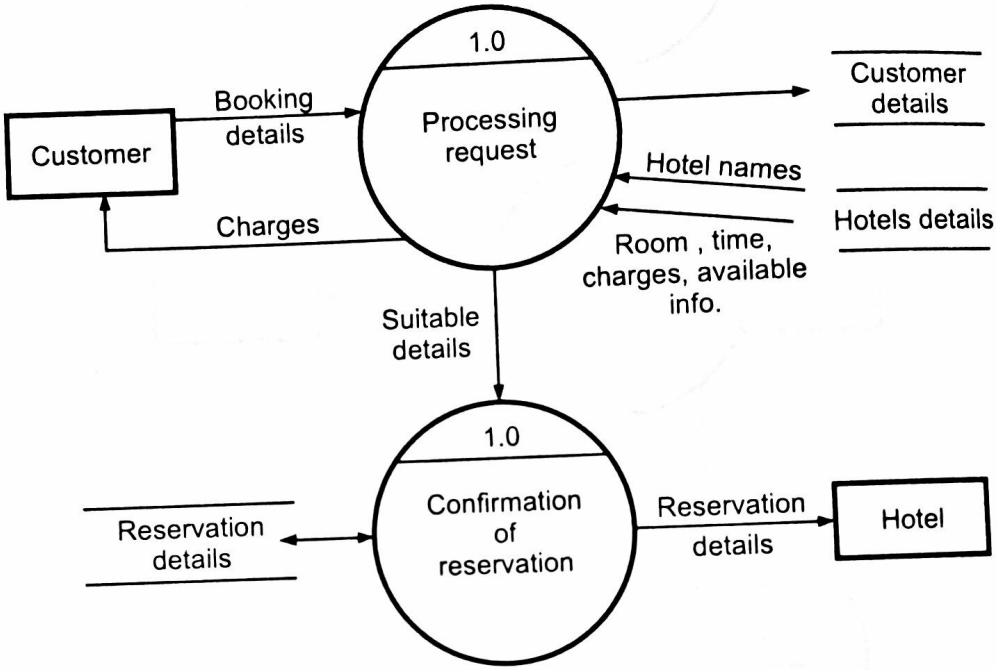
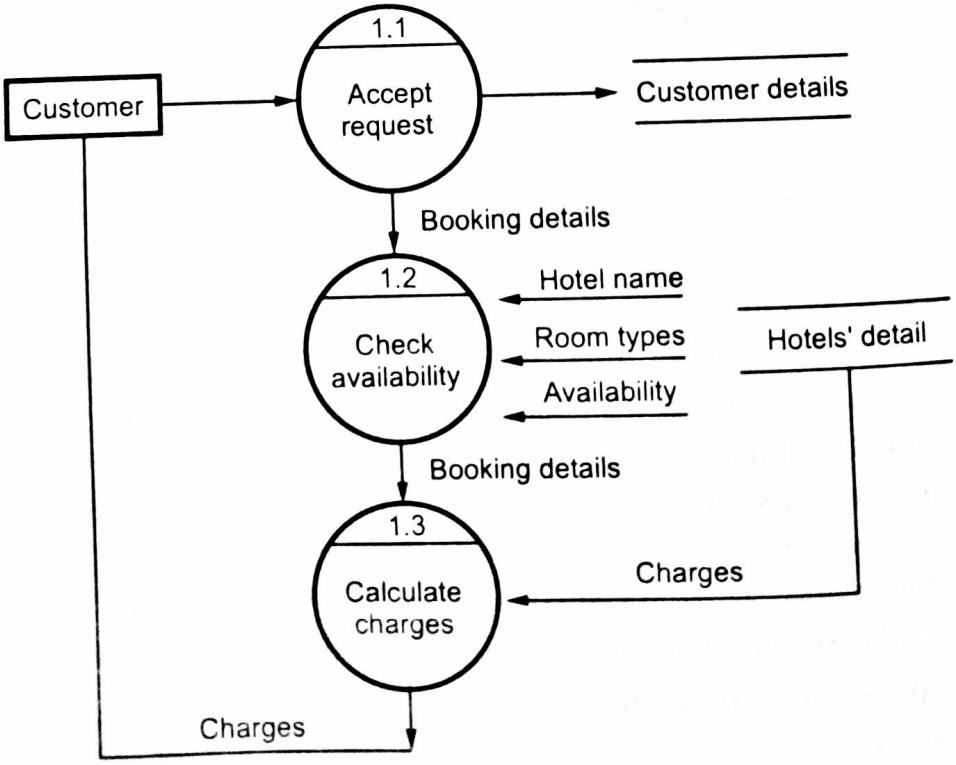


Fig. 6.26 Level 1 DFD



6.6.3 Designing Control Flow Diagrams

- There are certain applications which are event driven rather than being data driven. They produce control information rather than producing data(may be report, displays). Such applications can be modeled with the control information along with data flow modelling.
- A graphical model used to represent the control information along with the data flow model is called control flow model.
- The following guideline is used while drawing the control flow diagrams.

1. List all the sensors that can be read.
2. List all the interrupt conditions.
3. List all the data conditions.
4. List all the switches actuated by the operator.
5. Use noun/verb parsing technique to identify the control information.
6. Describe behaviour of the system by identifying the states. Define the transition between the states.
7. Avoid common errors while specifying the control

Rules for designing DFD

1. No process can have only outputs or only inputs. The proces must have both outputs and inputs.

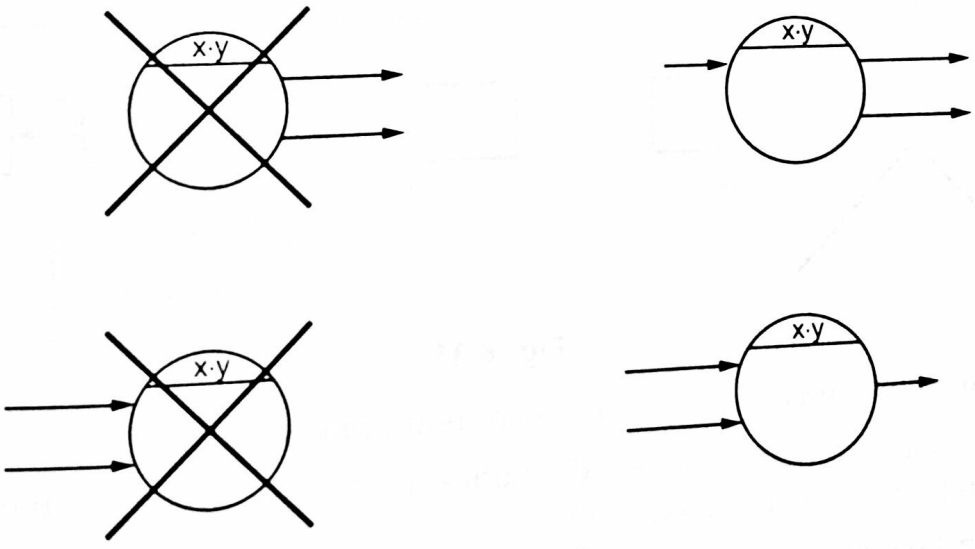


Fig. 6.28

2. The verb phrases in the problem description can be identified as processes in the system.
3. There should not be a direct flow between data stores and external entity. This flow should go through a process.

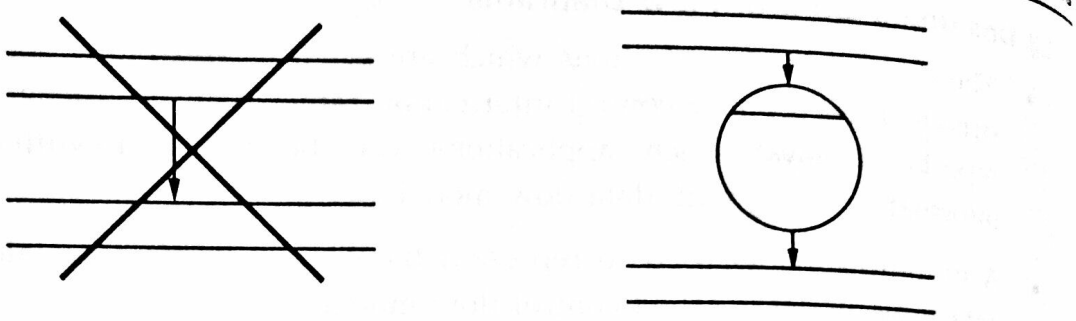


Fig. 6.29

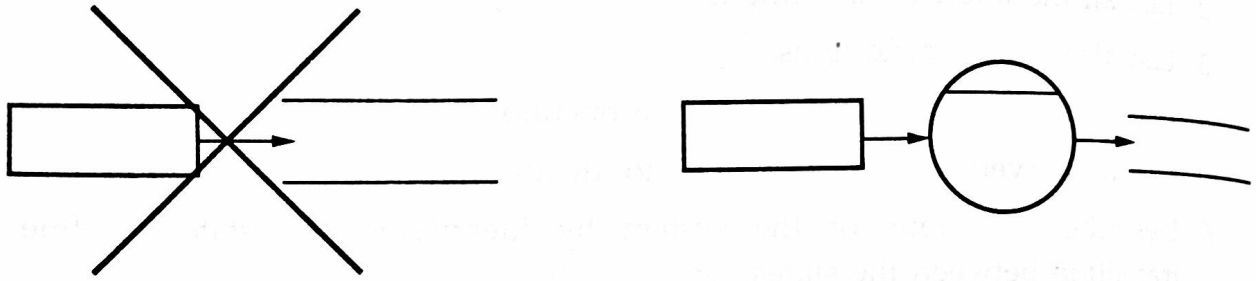


Fig. 6.30

4. Data store labels should be noun phrases from problem description.
5. No data should move directly between external entities. The data flow should go through a process.

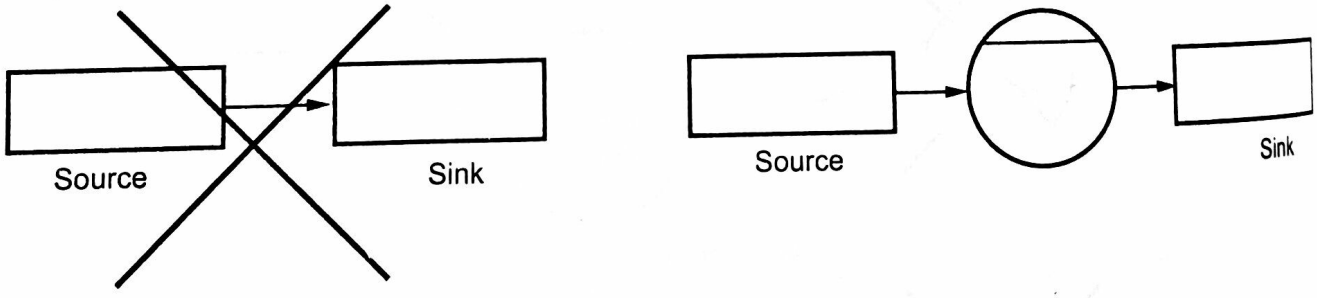


Fig. 6.31

6. Generally source and sink labels are noun phrases.
7. Interactions between external entities is outside scope of the system and therefore not represented in DFD.
8. Data flow from process to a data store is for updation/insertion/deletion.
9. Data flow from data store to process is for retrieving or using the information.
10. Data flow labels are noun phrases. from problem description.

6.7 Data Dictionary

The data dictionary can be defined as an organized collection of all the data elements of the system with precise and rigorous definitions so that user and system analyst will have a common understanding of inputs, outputs, components of stores and intermediate calculations.

- The data models are less detail hence there is a need for data dictionary.
- Data dictionaries are lists of all of the names used in the system models.
- Descriptions of the entities, relationships and attributes are also included in data dictionary.
- Typically, the data dictionaries are implemented as a part of structured analysis and design tool
- The data dictionary stores following type of information

Name	Description
Name	The primary name of data or control item, the data store or external entity.
Alias	Other name used for the Name
Where-used or how is used	It describes where the data or control item is used. It also describes how that item is used(that means input to the process, output to the process)

The notations used in data dictionary are

Data construct	Notation	Meaning
Composition	=	Is composed of
Sequence	+	And
Selection	[|]	Or
Repetition	$\{\ \}^n$	Repetition for n times
	( )	Optional data
	* ... *	Commented information

For example :

Consider the some reservation system. The data item "passenger" can be entered in the data dictionary as

6.9 Structured Methods

Applying the structured methods means designing different models to represent that particular system. Various structured models can be Context models, Process models, Data flow diagrams, State chart diagrams, Entity relationship diagrams, Object models. The structured modelling provides the systematic framework for **system modelling** as a part of requirement elicitation and analysis. Along with these models **documentation** is essential in order to provide line certain guideline in system development. Not only this use of **CASE tool** support also helps to generate *reports* or *code generation* from the system model.

Drawbacks of structured methods –

1. **Non functional** requirements can not be represented effectively using structured methods.
2. It is hard to **predict** whether the structured methods that are applied to particular **problem** are suitable for solving that problem or not. Even it is not possible to decide whether the structured method is suitable for particular environment or not.
3. Sometimes **too much documentation** hides the basic requirements of the system. Unnecessary detailing is done in structured methods.
4. The system models are represented in **more detail** so casual user can not **understand** the system because he gets lost in the unnecessary details.

Thus in this chapter we have discussed various ways by which the system can be graphically modelled.

Solved Exercise

Q.1 *Compare functional and behavioural models.*

Ans. :

Functional model	Behavioural model
The functional model depicts all the essential functionalities of the system.	The behavioural model represents how system behaves.
The functional model is represented by data flow and control flow diagrams.	The behavioural model is represented by state chart diagrams.
The DFDs can be represented by levels.	The state chart diagram has some number of states and transitions.
The functional diagram gives detailed scenario of system which has to be developed.	The behavioural model gives the abstract representation of the system

Q.4 *Differentiate data flow diagram and state transition diagram.*

Ans. :

Data flow diagram	State transition diagram
Data flow diagram is a graphical representation for representing the information flow and the transforms that are applied as data move from input to output.	State transition diagram is a graphical representation for representing the behavior of a system by depicting its states and the events that cause the system to change state.
The Data flow diagram is a collection of process, data store, flow of data(transitions) and external entity.	The state transition diagram is a collection of states and events.
The Data flow diagrams are used to represent the system at any level of abstraction, and the increasing levels are used to expose more and more functionalities in the system.	The state transition diagrams are typically drawn at single level. They are intended to expose the overall behavior of the system.

Q.5 *Draw an ER diagram for the relationship of manufacturer and dealership. Also specify the association, cardinality and modality.*

Ans. : **ER Diagram**

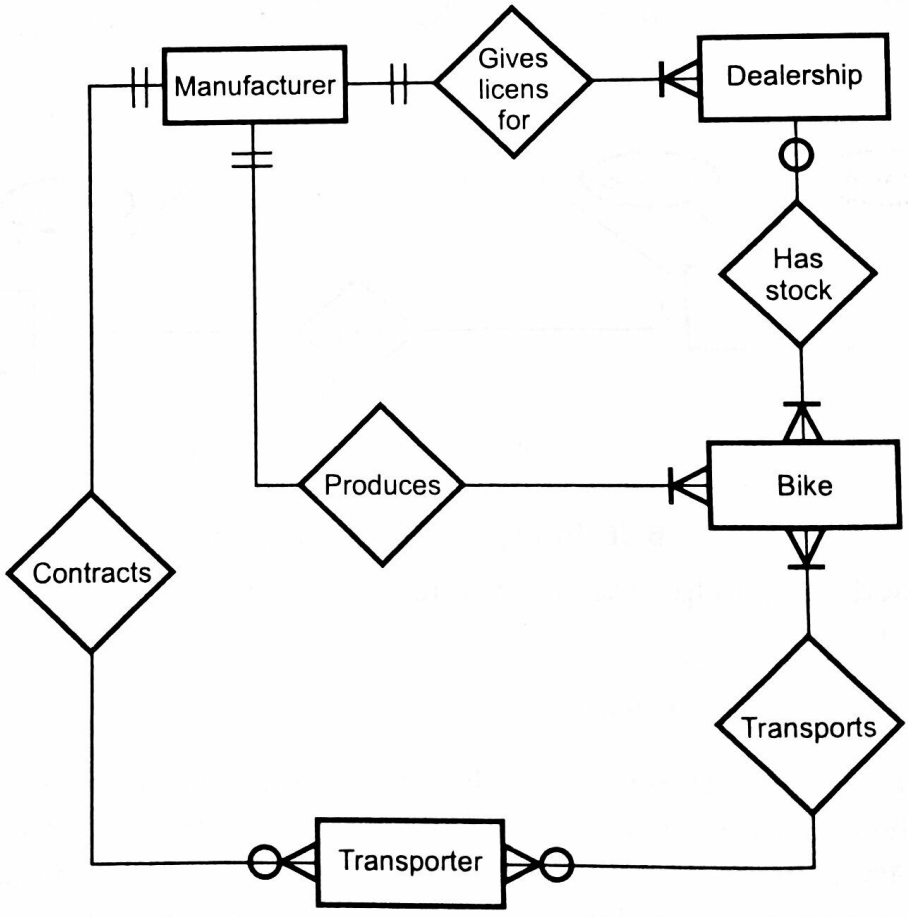


Fig. 6.37 ER diagram

Associations

In above ER diagram various relations are "Gives licens for", "Has stock", "Produces", "Transports" which are used to associate entries : Manufacturer, Dealership and Transporter.

Manufacturer *gives licens for* dealership.

Dealership *has stock* for bikes.

Manufacturer *produces* bikes.

Manufacturer *contracts* transporter.

Transporter *transports* bikes.

Cardinality and Modality

- One manufacturer can give licens to many dealers. But to get dealership licens from manufacturer is a must.
- There may be a situation that there is no stock of bikes with Dealers. (Bikes can be out of stock!)
- There may be a situation that there may not be a single bike for transport.
- Manufacturer can contract with many transporters.

6.3 The Web Engineering Process

The attributes of Web-based systems and applications have a profound influence the WebE process that is chosen. In Chapter 3 we noted that a software engin chooses a process model based on the attributes of the software that is to be dev oped. The same holds true for a Web engineer.

If immediacy and continuous evolution are primary attributes of a WebApp, a W engineering team might choose an agile process model (Chapter 4) that produ WebApp releases in rapid-fire sequence. On the other hand, if a WebApp is to be veloped over a longer time period (e.g., a major e-commerce application), an inc mental process model (Chapter 3) might be chosen.

> **"Web development is an adolescent . . . Like most adolescents, it wants to be accepted as an adult as it tries to pull away from its parents. If it is going to reach its full potential, it must take a few lessons from the more seasoned world of software development."**
>
> **Doug Wallace et a**

The network intensive nature of applications in this domain suggests a pop tion of users that is diverse (thereby making special demands on requirements itation and modeling) and an application architecture that can be highly special

(thereby making demands on design). Because WebApps are often content-driven with an emphasis on aesthetics, it is likely that parallel development activities will be scheduled within the WebE process and involve a team of both technical and non-technical people (e.g., copywriters, graphic designers).

16.3.1 Defining the Framework

Any one of the agile process models (e.g., Extreme Programming, Adaptive Software Development, SCRUM) presented in Chapter 4 can be applied successfully as a WebE process. The process framework that is presented here is an amalgam of the principles and ideas discussed in Chapter 4.

To be effective, any engineering process must be adaptable. That is, the organization of the project team, the modes of communication among team members, the engineering activities and tasks to be performed, the information that is collected and created, and the methods used to produce a high-quality product must all be adapted to the people doing the work, the project timeline and constraints, and the problem to be solved. Before we define a process framework for WebE, we must recognize that:

1. *WebApps are often delivered incrementally.* That is, framework activities will occur repeatedly as each increment is engineered and delivered.
2. *Changes will occur frequently.* These changes may occur as a result of the evaluation of a delivered increment or as a consequence of changing business conditions.
3. *Timelines are short.* This mitigates against the creation and review of voluminous engineering documentation, but it does not preclude the simple reality that critical analysis, design, and testing must be recorded in some manner.

In addition, the principles defined as part of the "Manifesto for Agile Software Development" (Chapter 4) should be applied. However, the principles are not the Ten Commandments. It is sometimes reasonable to adopt the spirit of these principles without necessarily abiding by the letter of the manifesto.

With these issues in mind, we discuss the WebE process within the generic process framework presented in Chapter 2.

Customer communication. Within the WebE process, customer communication is characterized by two major tasks: business analysis and formulation. *Business analysis* defines the business/organizational context for the WebApp. In addition, stakeholders are identified, potential changes in business environment or requirements are predicted, and integration between the WebApp and other business applications, databases, and functions is defined. *Formulation* is a requirements gathering activity involving all stakeholders. The intent is to describe the

problem that the WebApp is to solve (along with basic requirements for the WebApp) using the best information available. In addition, an attempt is made to identify areas of uncertainty and where potential changes will occur.

Planning. The project plan for the WebApp increment is created. The plan consists of a task definition and a timeline schedule for the time period (usually measured in weeks) projected for the development of the WebApp increment.

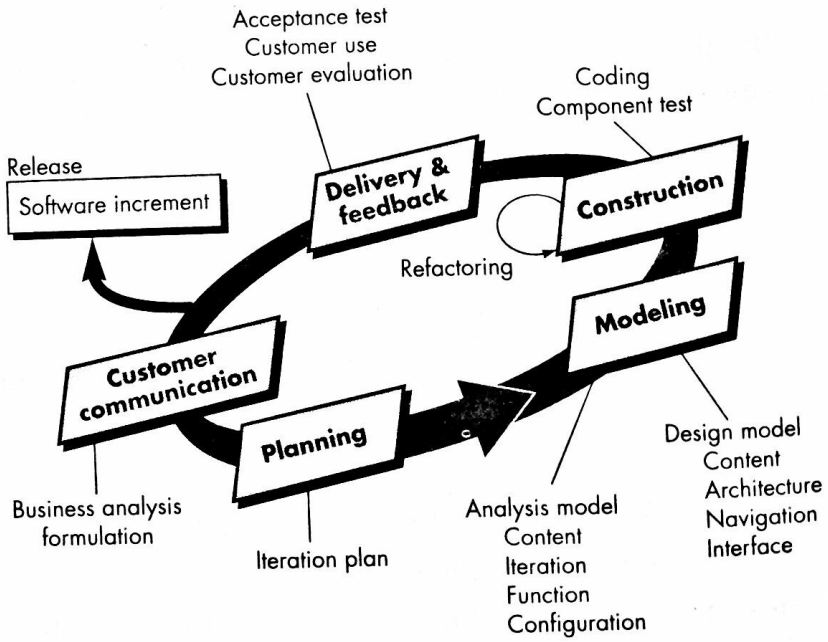
Modeling. Conventional software engineering analysis and design tasks are adapted to WebApp development, merged, and then melded into the WebE modeling activity (Chapters 18 and 19). The intent is to develop "rapid" analysis and design models that define requirements and at the same time represent a WebApp that will satisfy them.

Construction. WebE tools and technology are applied to construct the WebApp that has been modeled. Once the WebApp increment has been constructed, a series of rapid tests are conducted to ensure that errors in design (i.e., content, architecture, interface, navigation) are uncovered. Additional testing addresses other WebApp characteristics.

Deployment. The WebApp is configured for its operational environment, delivered to end-users, and then an evaluation period commences. Evaluation feedback is presented to the WebE team, and the increment is modified as required.

These five WebE framework activities are applied using an incremental process flow as shown in Figure 16.1.

FIGURE 16.1 The WebE process



16.3.2 Refining the Framework

We have already noted that the WebE process model must be adaptable. That is, a definition of the engineering tasks required to refine each framework activity is left to the discretion of the Web engineering team. In some cases, a framework activity is conducted informally. In others, a series of distinct tasks will be defined and conducted by team members. In every case, the team has responsibility for producing a high-quality WebApp increment within the time period allocated.

It is important to emphasize that tasks associated with WebE framework activities may be modified, eliminated, or extended based on the characteristics of the problem, the product, the project, and the people on the Web engineering team.

> "There are those of us who believe that the best practices for software development are practical and deserve implementation. And then there are those of us who believe that best practices are interesting in an academic sort of way, but are not for the real world, thank you very much."
>
> **Warren Keuffel**

16.4 Web Engineering Best Practices

Will every WebApp developer use the WebE process framework and task set defined in Section 16.3? Probably not. Web engineering teams are sometimes under enormous time pressure and will try to take short-cuts (even if these are ill-advised and result in *more* development effort, not less). But a set of fundamental best practices—adopted from the software engineering practices discussed throughout Part 2 of this book—should be applied if industry-quality WebApps are to be built.

Be sure that the business need for a WebApp has been clearly enunciated by someone. If it hasn't, your WebE project is at risk.

1. *Take the time to understand business needs and product objectives, even if the details of the WebApp are vague.* Many WebApp developers erroneously believe that vague requirements (which are quite common) relieve them from the need to be sure that the system they are about to engineer has a legitimate business purpose. The end result is (too often) good technical work that results in the wrong system built for the wrong reasons for the wrong audience. If stakeholders cannot enunciate a business need for the WebApp, proceed with extreme caution. If stakeholders struggle to identify a set of clear objectives for the product (WebApp), do not proceed until they can.
2. *Describe how users will interact with the WebApp using a scenario-based approach.* Stakeholders must be convinced to develop use-cases (discussed throughout Part 2 of this book) to reflect how various actors will interact with the WebApp. These scenarios can then be used (1) for project planning and tracking, (2) to guide analysis and design modeling, and (3) as important input for the design of tests.
3. *Develop a project plan, even if it is very brief.* Base the plan on a predefined process framework that is acceptable to all stakeholders. Because project timelines are very short, schedule granularity should be fine; i.e., in many instances, the project should be scheduled and tracked on a daily basis.
4. *Spend some time modeling what it is that you're going to build.* Generally, comprehensive analysis and design models are not developed during Web engineering. However, UML class and sequence diagrams along with other selected UML notation (e.g., state diagrams) may provide invaluable insight.
5. *Review the models for consistency and quality.* Formal technical reviews (Chapter 26) should be conducted throughout a WebE project. The time spent on reviews pays important dividends because it often eliminates rework and results in a WebApp that exhibits high quality—thereby increasing customer satisfaction.
6. *Use tools and technology that enable you to construct the system with as many reusable components as possible.* A wide array of WebApp tools are available for virtually every aspect of WebApp construction. Many of these tools enable

a Web engineer to build significant portions of the application using reusable components.

7. *Don't rely on early users to debug the WebApp—design comprehensive tests and execute them before releasing the system.* Users of a WebApp will often give it one chance. If it fails to perform, they move elsewhere—never to return. It is for this reason that "test first, then deploy" should be an overriding philosophy, even if deadlines must be stretched.

INFO

Quality Criteria/Guidelines for WebApps

WebE strives to produce high-quality WebApps. But what is "quality" in this context, and what guidelines are available for achieving it? In his paper on Web-site quality assurance, Quibeldey-Cirkel [QUI01] suggests a comprehensive set of on-line resources that address these issues:

W3C: Style Guide for Online Hypertext
www.w3.org/Provider/Style

The Sevloid Guide to Web Design
www.sev.com.au/webzone/design/guide.asp

Web Pages That Suck
www.webpagesthatsuck.com/index.html

Resources on Web Style
www.westegg.com/unmaintained/badpages

Gartner's Web Evaluation Tool
www.gartner.com/ebusiness/website-ings

IBM Corp: Web Guidelines
www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/572

World Wide Web Usability
ijhcs.open.ac.uk

Interface Hall of Shame
www.iarchitect.com/mshame.htm

Art and the Zen of Web Sites
www.tlc-systems.com/webtips.shtml

Designing for the Web: Empirical Studies
www.microsoft.com/usability/webconf.htm

Nielsen's useit.com
www.useit.com

Quality of Experience
www.qualityofexperience.org

Creating Killer Web Sites
www.killersites.com/core.html

All Things at Web
www.pantos.org/atw

SUN's New Web Design
www.sun.com/980113/sunonnet

Tognazzini, Bruce: Homepage
www.asktog.com

Webmonkey
hotwired.lycos.com/webmonkey/design/?tw=design

World's Best WebSites
www.worldbestwebsites.com

Yale University: Yale Web-Style Guide
info.med.yale.edu/caim/manual

Object Oriented Testing methods:

Testing is a continuous activity during software development. In object-oriented systems, testing encompasses three levels, namely, unit testing, subsystem testing, and system testing.

Unit Testing:

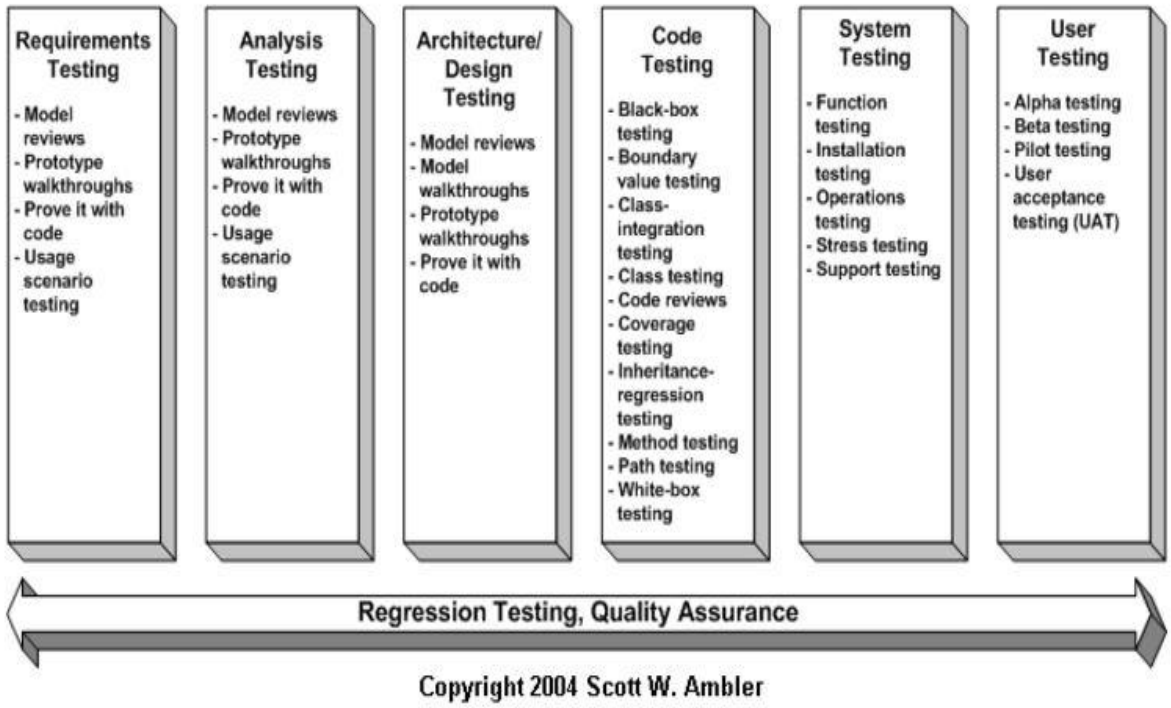
- In unit testing, the individual classes are tested. It is seen whether the class attributes are implemented as per design and whether the methods and the interfaces are error-free.
- Unit testing is the responsibility of the application engineer who implements the structure.

Subsystem Testing:

- This involves testing a particular module or a subsystem and is the responsibility of the subsystem lead. It involves testing the associations within the subsystem as well as the interaction of the subsystem with the outside.
- Subsystem tests can be used as regression tests for each newly released version of the subsystem.

System Testing:

- System testing involves testing the system as a whole and is the responsibility of the quality-assurance team. The team often uses system tests as regression tests when assembling new releases.



Object-Oriented Testing Techniques:

Grey Box Testing:

The different types of test cases that can be designed for testing object-oriented programs are called grey box test cases. Some of the important types of grey box testing are:

- **State model based testing:** This encompasses state coverage, state transition coverage, and state transition path coverage.
- **Use case based testing:** Each scenario in each use case is tested.
- **Class diagram based testing:** Each class, derived class, associations, and aggregations are tested.
- **Sequence diagram based testing:** The methods in the messages in the sequence diagrams are tested.

Techniques for Subsystem Testing:

The two main approaches of subsystem testing are:

- **Thread based testing:** All classes that are needed to realize a single use case in a subsystem are integrated and tested.
- **Use based testing:** The interfaces and services of the modules at each level of hierarchy are tested. Testing starts from the individual classes to the small modules comprising of classes, gradually to larger modules, and finally all the major subsystems.

Categories of System Testing:

- **Alpha testing:** This is carried out by the testing team within the organization that develops software.
- **Beta testing:** This is carried out by select group of co-operating customers.
- **Acceptance testing:** This is carried out by the customer before accepting the deliverables.

Black-box testing:

Testing that verifies the item being tested when given the appropriate input provides the expected results.

Boundary-value testing:

Testing of unusual or extreme situations that an item should be able to handle.

Class testing:

The act of ensuring that a class and its instances (objects) perform as defined.

Component testing:

The act of validating that a component works as defined.

Inheritance-regression testing:

The act of running the test cases of the super classes, both direct and indirect, on a given subclass.

Integration testing:

Testing to verify several portions of software work together.

Model review:

An inspection, ranging anywhere from a formal technical review to an informal walkthrough, by others who were not directly involved with the development of the model.

Path testing:

The act of ensuring that all logic paths within your code are exercised at least once.

Regression testing:

The acts of ensuring that previously tested behaviors still work as expected after changes have been made to an application.

Stress testing:

The act of ensuring that the system performs as expected under high volumes of transactions, users, load, and so on.

Technical review:

A quality assurance technique in which the design of your application is examined critically by a group of your peers. A review typically focuses on accuracy, quality, usability, and completeness. This process is often referred to as a walkthrough, an inspection, or a peer review.

User interface testing:

The testing of the user interface (UI) to ensure that it follows accepted UI standards and meets the requirements defined for it. Often referred to as graphical user interface (GUI) testing.

White-box testing:

Testing to verify that specific lines of code work as defined. Also referred to as clear-box testing.