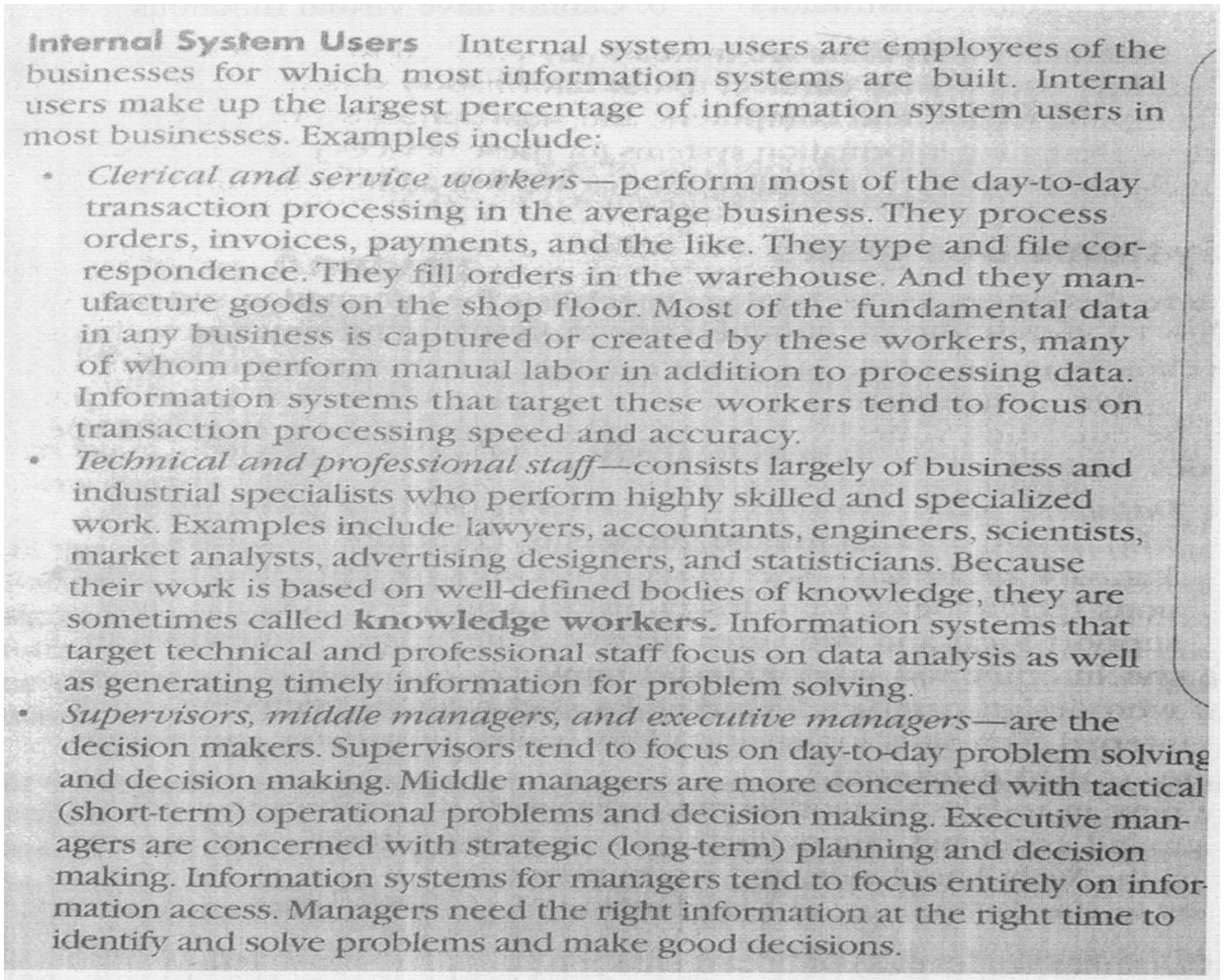
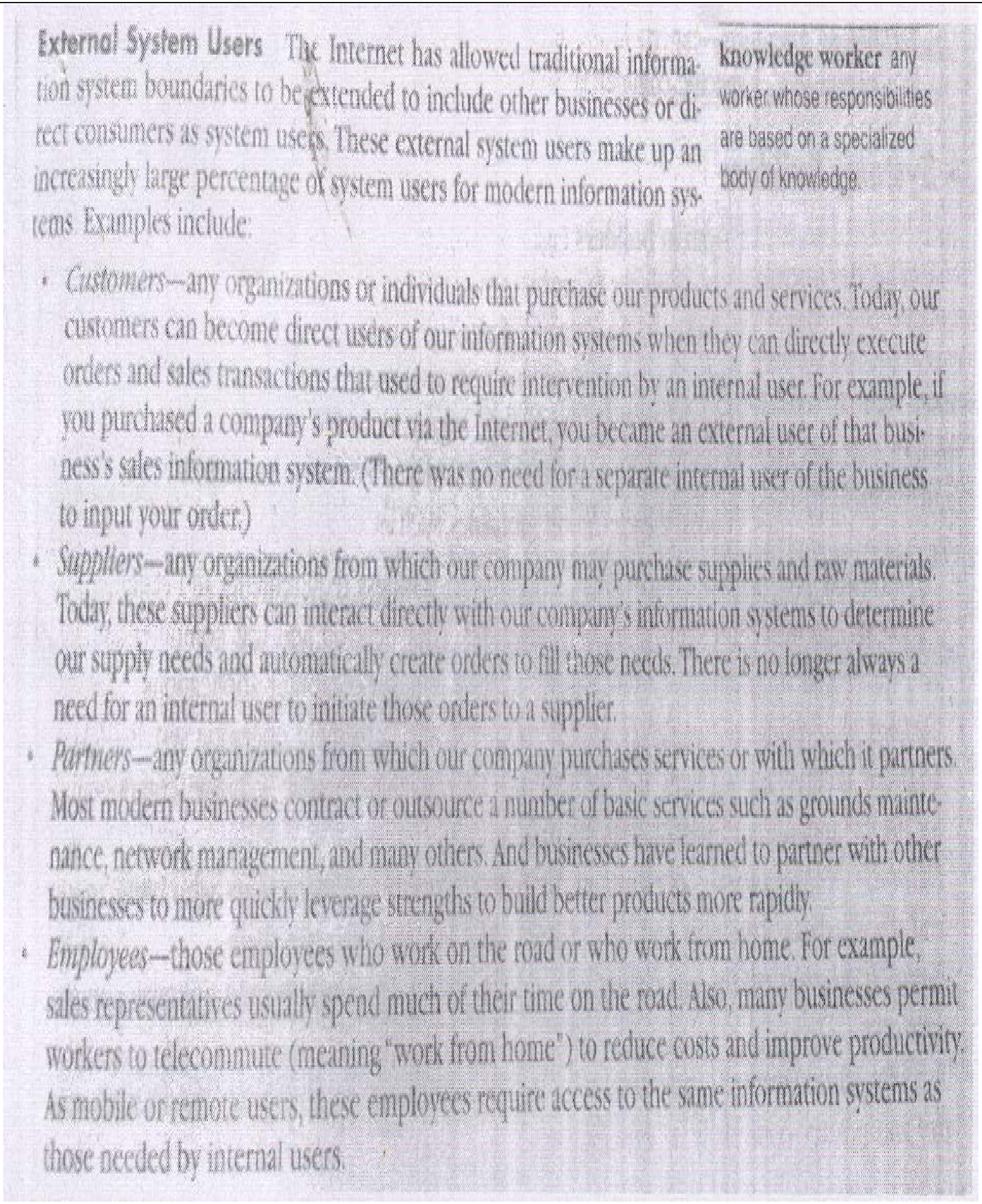
Q.2a. What are the differences between internal and external users of an Information System ? Give examples.



Internal System Users Internal system users are employees of the businesses for which most information systems are built. Internal users make up the largest percentage of information system users in most businesses. Examples include:

- *Clerical and service workers*—perform most of the day-to-day transaction processing in the average business. They process orders, invoices, payments, and the like. They type and file correspondence. They fill orders in the warehouse. And they manufacture goods on the shop floor. Most of the fundamental data in any business is captured or created by these workers, many of whom perform manual labor in addition to processing data. Information systems that target these workers tend to focus on transaction processing speed and accuracy.
- *Technical and professional staff*—consists largely of business and industrial specialists who perform highly skilled and specialized work. Examples include lawyers, accountants, engineers, scientists, market analysts, advertising designers, and statisticians. Because their work is based on well-defined bodies of knowledge, they are sometimes called **knowledge workers.** Information systems that target technical and professional staff focus on data analysis as well as generating timely information for problem solving.
- *Supervisors, middle managers, and executive managers*—are the decision makers. Supervisors tend to focus on day-to-day problem solving and decision making. Middle managers are more concerned with tactical (short-term) operational problems and decision making. Executive managers are concerned with strategic (long-term) planning and decision making. Information systems for managers tend to focus entirely on information access. Managers need the right information at the right time to identify and solve problems and make good decisions.

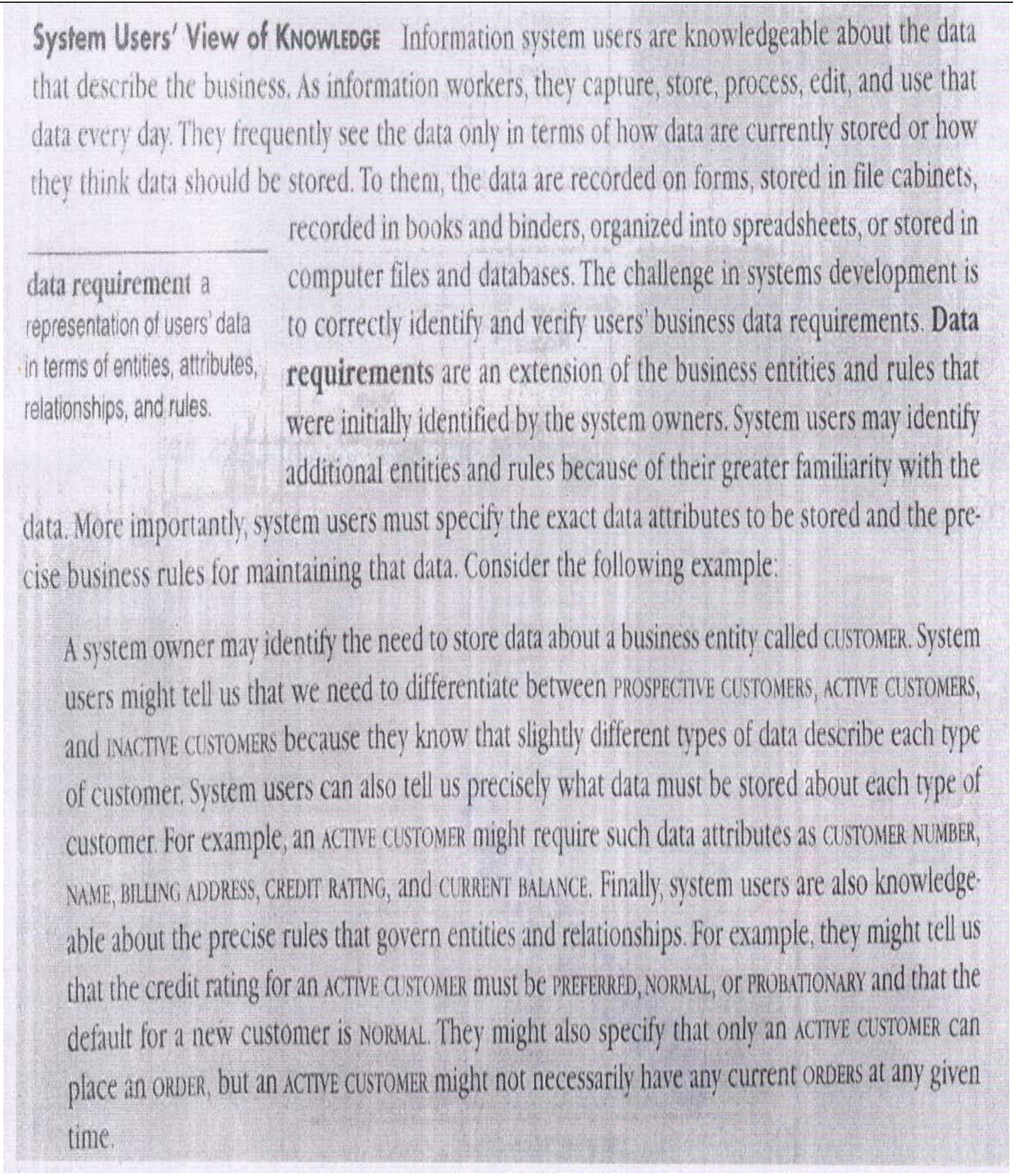


External System Users The Internet has allowed traditional information system boundaries to be extended to include other businesses or direct consumers as system users. These external system users make up an increasingly large percentage of system users for modern information systems. Examples include:

knowledge worker any worker whose responsibilities are based on a specialized body of knowledge.

- *Customers*—any organizations or individuals that purchase our products and services. Today, our customers can become direct users of our information systems when they can directly execute orders and sales transactions that used to require intervention by an internal user. For example, if you purchased a company's product via the Internet, you became an external user of that business's sales information system. (There was no need for a separate internal user of the business to input your order.)
- *Suppliers*—any organizations from which our company may purchase supplies and raw materials. Today, these suppliers can interact directly with our company's information systems to determine our supply needs and automatically create orders to fill those needs. There is no longer always a need for an internal user to initiate those orders to a supplier.
- *Partners*—any organizations from which our company purchases services or with which it partners. Most modern businesses contract or outsource a number of basic services such as grounds maintenance, network management, and many others. And businesses have learned to partner with other businesses to more quickly leverage strengths to build better products more rapidly.
- *Employees*—those employees who work on the road or who work from home. For example, sales representatives usually spend much of their time on the road. Also, many businesses permit workers to telecommute (meaning "work from home") to reduce costs and improve productivity. As mobile or remote users, these employees require access to the same information systems as those needed by internal users.

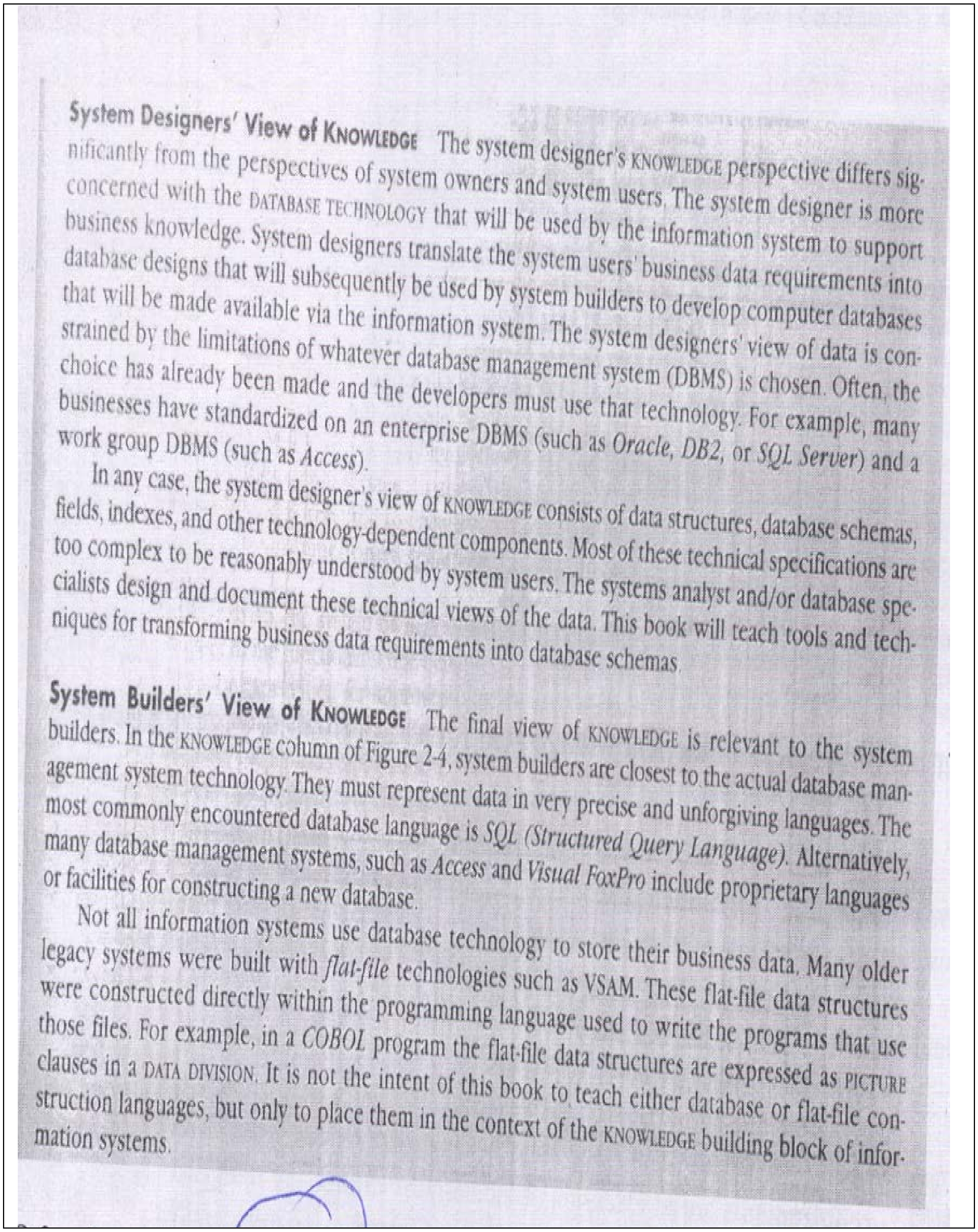
b. Describe the four knowledge building blocks of an information system.



System Users' View of Knowledge Information system users are knowledgeable about the data that describe the business. As information workers, they capture, store, process, edit, and use that data every day. They frequently see the data only in terms of how data are currently stored or how they think data should be stored. To them, the data are recorded on forms, stored in file cabinets, recorded in books and binders, organized into spreadsheets, or stored in computer files and databases. The challenge in systems development is to correctly identify and verify users' business data requirements. **Data requirements** are an extension of the business entities and rules that were initially identified by the system owners. System users may identify additional entities and rules because of their greater familiarity with the data. More importantly, system users must specify the exact data attributes to be stored and the precise business rules for maintaining that data. Consider the following example:

data requirement a representation of users' data in terms of entities, attributes, relationships, and rules.

A system owner may identify the need to store data about a business entity called CUSTOMER. System users might tell us that we need to differentiate between PROSPECTIVE CUSTOMERS, ACTIVE CUSTOMERS, and INACTIVE CUSTOMERS because they know that slightly different types of data describe each type of customer. System users can also tell us precisely what data must be stored about each type of customer. For example, an ACTIVE CUSTOMER might require such data attributes as CUSTOMER NUMBER, NAME, BILLING ADDRESS, CREDIT RATING, and CURRENT BALANCE. Finally, system users are also knowledgeable about the precise rules that govern entities and relationships. For example, they might tell us that the credit rating for an ACTIVE CUSTOMER must be PREFERRED, NORMAL, or PROBATIONARY and that the default for a new customer is NORMAL. They might also specify that only an ACTIVE CUSTOMER can place an ORDER, but an ACTIVE CUSTOMER might not necessarily have any current ORDERS at any given time.



System Designers' View of Knowledge The system designer's KNOWLEDGE perspective differs significantly from the perspectives of system owners and system users. The system designer is more concerned with the DATABASE TECHNOLOGY that will be used by the information system to support business knowledge. System designers translate the system users' business data requirements into database designs that will subsequently be used by system builders to develop computer databases that will be made available via the information system. The system designers' view of data is constrained by the limitations of whatever database management system (DBMS) is chosen. Often, the choice has already been made and the developers must use that technology. For example, many businesses have standardized on an enterprise DBMS (such as *Oracle, DB2,* or *SQL Server*) and a work group DBMS (such as *Access*).

In any case, the system designer's view of KNOWLEDGE consists of data structures, database schemas, fields, indexes, and other technology-dependent components. Most of these technical specifications are too complex to be reasonably understood by system users. The systems analyst and/or database specialists design and document these technical views of the data. This book will teach tools and techniques for transforming business data requirements into database schemas.

System Builders' View of Knowledge The final view of KNOWLEDGE is relevant to the system builders. In the KNOWLEDGE column of Figure 2-4, system builders are closest to the actual database management system technology. They must represent data in very precise and unforgiving languages. The most commonly encountered database language is *SQL (Structured Query Language)*. Alternatively, many database management systems, such as *Access* and *Visual FoxPro* include proprietary languages or facilities for constructing a new database.

Not all information systems use database technology to store their business data. Many older legacy systems were built with *flat-file* technologies such as VSAM. These flat-file data structures were constructed directly within the programming language used to write the programs that use those files. For example, in a *COBOL* program the flat-file data structures are expressed as PICTURE clauses in a DATA DIVISION. It is not the intent of this book to teach either database or flat-file construction languages, but only to place them in the context of the KNOWLEDGE building block of information systems.

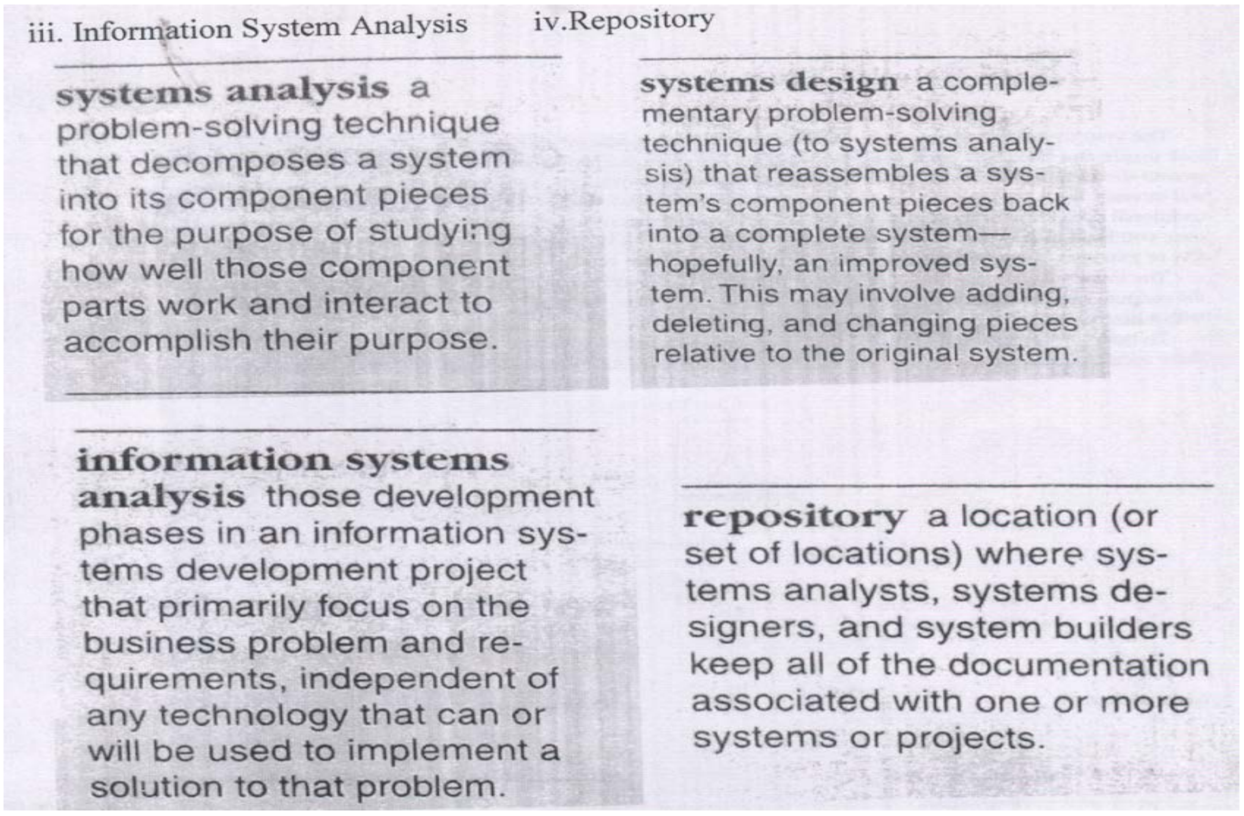
Q.3a. Explain any two strategies for each of the following:-
(i) Alternative routes and strategies
(ii) Automated tools and technology

Ans Page (89-111)

b. Explain the following system development process :-
(i) Cross life-cycle activities
(ii) Sequential versus iterative development

Ans: Page (88 – 92) book I

Q.4 a. Define briefly :-
(i) System Analysis
(ii) System Design
(iii) Information System Analysis
(iv) Repository



iii. Information System Analysis iv.Repository

systems analysis a problem-solving technique that decomposes a system into its component pieces for the purpose of studying how well those component parts work and interact to accomplish their purpose.

systems design a complementary problem-solving technique (to systems analysis) that reassembles a system's component pieces back into a complete system—hopefully, an improved system. This may involve adding, deleting, and changing pieces relative to the original system.

information systems analysis those development phases in an information systems development project that primarily focus on the business problem and requirements, independent of any technology that can or will be used to implement a solution to that problem.

repository a location (or set of locations) where systems analysts, systems designers, and system builders keep all of the documentation associated with one or more systems or projects.

b.Explain various tasks for any two of the following:- **(8)**
(i) Requirements analysis phase
(ii) Logical design phase
(iii) Decision analysis phase

Page 185 — 199 [Book-1)

Q.5 a. Explain the four steps required in the process of Use Case Modeling.

Ans. Page No.251

b.Define Data Modeling. Explain the process of Logical Data Modeling and give an illustration.

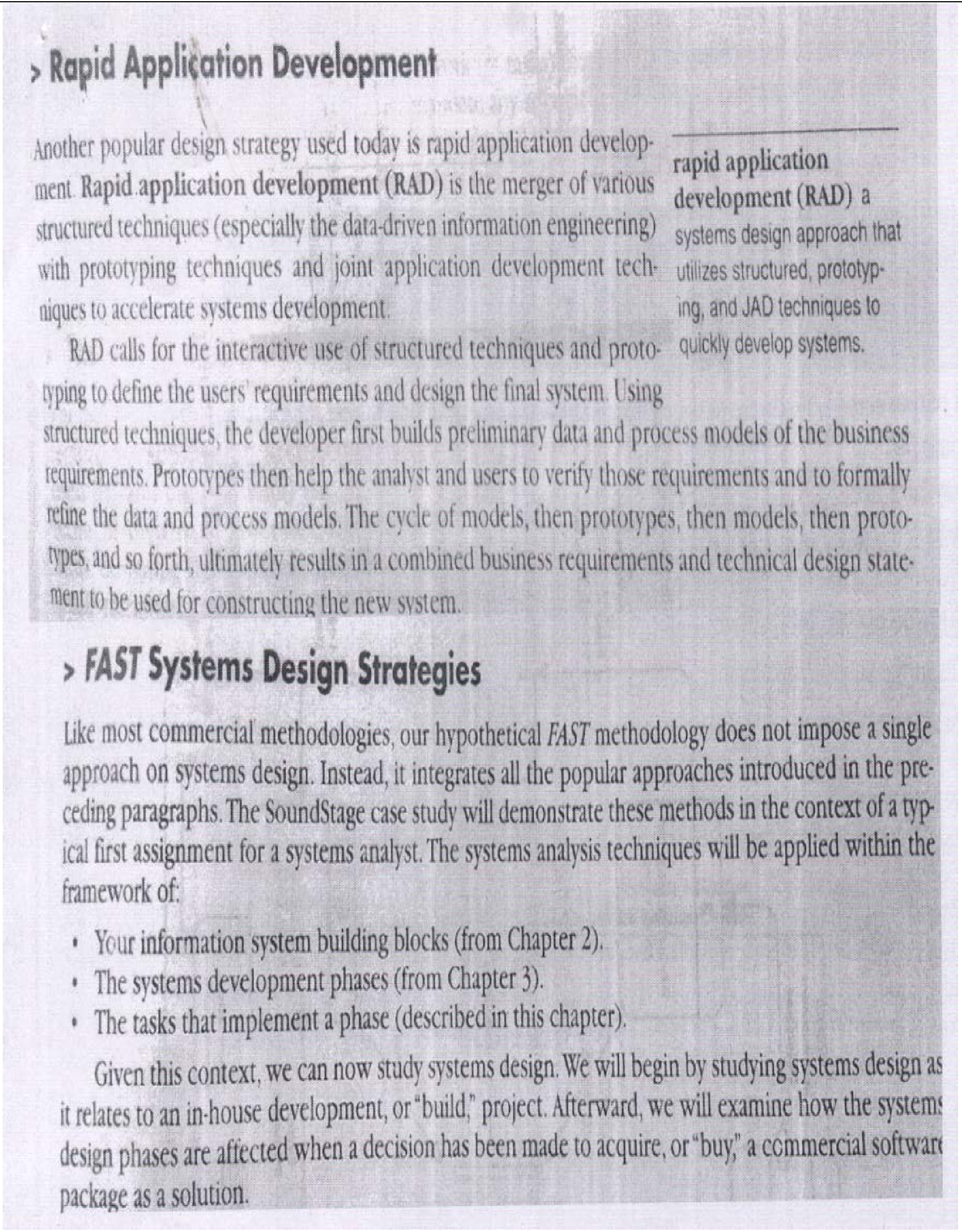
Ans. Page No. 270, 283-288

Q.6a. Describe the four steps for organizing the objects & identifying their relationships using class diagram. Give an illustration.

Ans. Page 400

6 b. Explain the following System design approaches :-
(i) Model – Driven Approach
(ii) Rapid Application Development

Ans Page- 447-452 of textbook



> Rapid Application Development

Another popular design strategy used today is rapid application development. **Rapid application development (RAD)** is the merger of various structured techniques (especially the data-driven information engineering) with prototyping techniques and joint application development techniques to accelerate systems development.

rapid application development (RAD) a systems design approach that utilizes structured, prototyping, and JAD techniques to quickly develop systems.

RAD calls for the interactive use of structured techniques and prototyping to define the users' requirements and design the final system. Using structured techniques, the developer first builds preliminary data and process models of the business requirements. Prototypes then help the analyst and users to verify those requirements and to formally refine the data and process models. The cycle of models, then prototypes, then models, then prototypes, and so forth, ultimately results in a combined business requirements and technical design statement to be used for constructing the new system.

> FAST Systems Design Strategies

Like most commercial methodologies, our hypothetical *FAST* methodology does not impose a single approach on systems design. Instead, it integrates all the popular approaches introduced in the preceding paragraphs. The SoundStage case study will demonstrate these methods in the context of a typical first assignment for a systems analyst. The systems analysis techniques will be applied within the framework of:

- Your information system building blocks (from Chapter 2).
- The systems development phases (from Chapter 3).
- The tasks that implement a phase (described in this chapter).

Given this context, we can now study systems design. We will begin by studying systems design as it relates to an in-house development, or "build," project. Afterward, we will examine how the systems design phases are affected when a decision has been made to acquire, or "buy," a commercial softwar package as a solution.

Q.7 a. Explain the type of users, human factors and human engineering guidelines that effect the user interface design. (6)

Ans. 7a Page 614-617 of text book 1

b.Explain the features of any three Graphical User Interface Styles and Considerations.

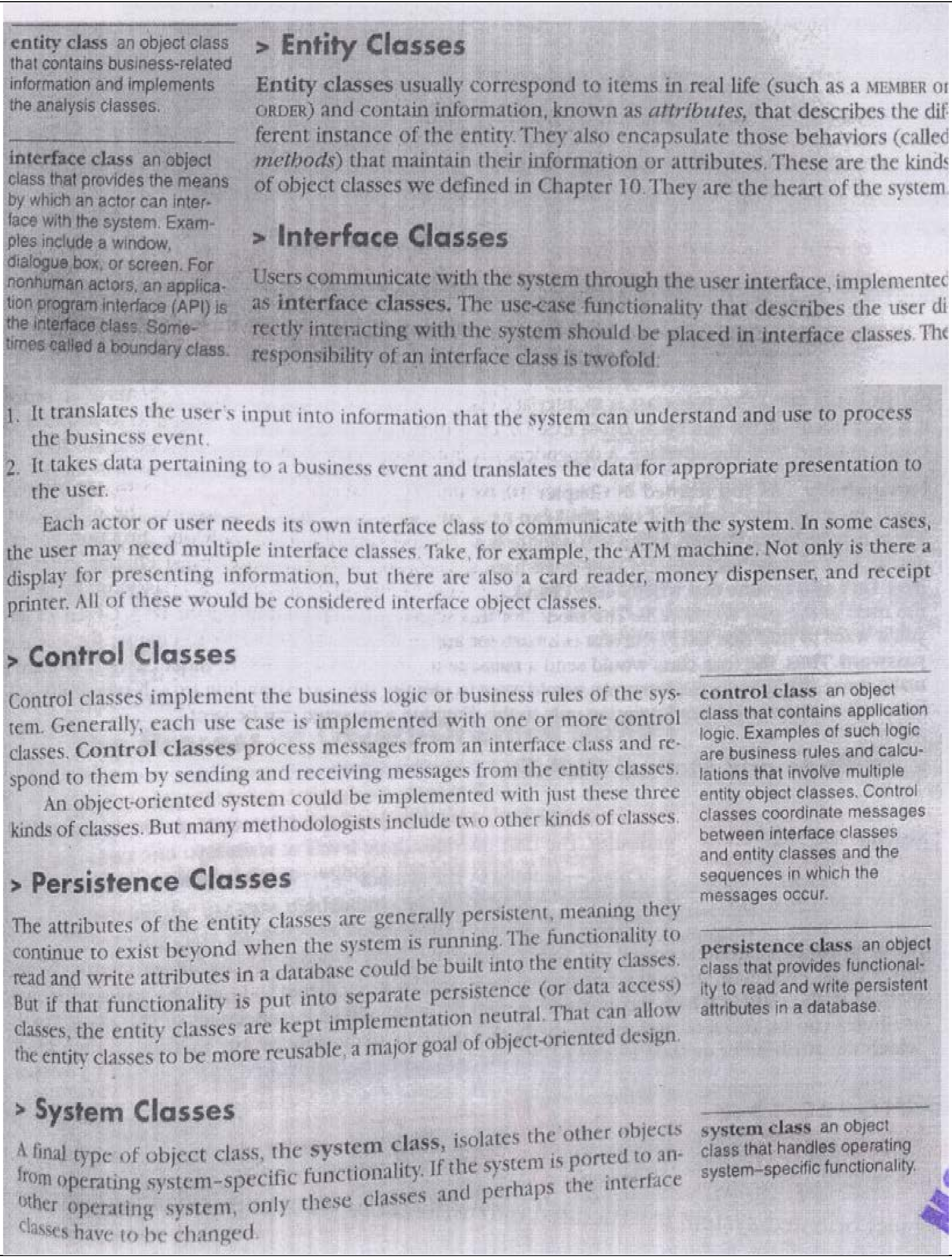
Ans. Page 618

c.Explain in brief the role of automated tools for User Interface Design and Prototyping.

Ans. Page 634 – 636

Q.8 a. Explain the following object-oriented system features :
(i) Persistance class
(ii) Interface class
(iii) Method Visibility
(iv) Object Responsibility

Ans. Page 650-651



entity class an object class that contains business-related information and implements the analysis classes.

interface class an object class that provides the means by which an actor can interface with the system. Examples include a window, dialogue box, or screen. For nonhuman actors, an application program interface (API) is the interface class. Sometimes called a boundary class.

> Entity Classes

Entity classes usually correspond to items in real life (such as a MEMBER or ORDER) and contain information, known as *attributes*, that describes the dif ferent instance of the entity. They also encapsulate those behaviors (called *methods*) that maintain their information or attributes. These are the kinds of object classes we defined in Chapter 10. They are the heart of the system.

> Interface Classes

Users communicate with the system through the user interface, implementec as **interface classes.** The use-case functionality that describes the user di rectly interacting with the system should be placed in interface classes. The responsibility of an interface class is twofold:

1. It translates the user's input into information that the system can understand and use to process the business event.
2. It takes data pertaining to a business event and translates the data for appropriate presentation to the user.

Each actor or user needs its own interface class to communicate with the system. In some cases, the user may need multiple interface classes. Take, for example, the ATM machine. Not only is there a display for presenting information, but there are also a card reader, money dispenser, and receipt printer. All of these would be considered interface object classes.

> Control Classes

Control classes implement the business logic or business rules of the system. Generally, each use case is implemented with one or more control classes. **Control classes** process messages from an interface class and respond to them by sending and receiving messages from the entity classes.

An object-oriented system could be implemented with just these three kinds of classes. But many methodologists include two other kinds of classes.

control class an object class that contains application logic. Examples of such logic are business rules and calculations that involve multiple entity object classes. Control classes coordinate messages between interface classes and entity classes and the sequences in which the messages occur.

> Persistence Classes

The attributes of the entity classes are generally persistent, meaning they continue to exist beyond when the system is running. The functionality to read and write attributes in a database could be built into the entity classes. But if that functionality is put into separate persistence (or data access) classes, the entity classes are kept implementation neutral. That can allow the entity classes to be more reusable, a major goal of object-oriented design.

persistence class an object class that provides functionality to read and write persistent attributes in a database.

> System Classes

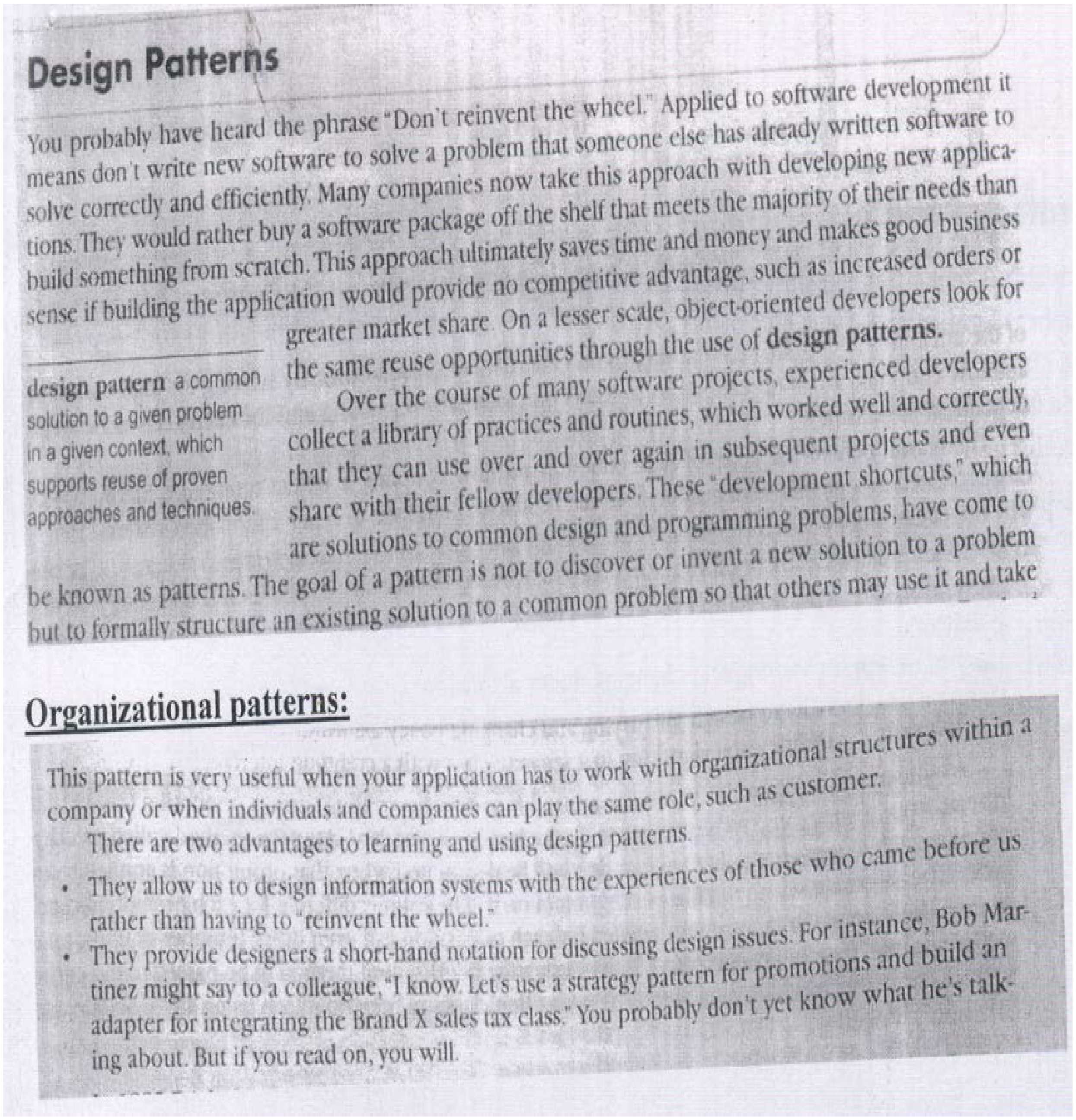
A final type of object class, the **system class,** isolates the other objects from operating system–specific functionality. If the system is ported to another operating system, only these classes and perhaps the interface classes have to be changed.

system class an object class that handles operating system–specific functionality.

b.Explain the features of design patterns.

Ans Page 668, 89, 90

c.Explain briefly the following and give an illustration for each:
(i) Strategy Pattern
(ii) Organization Pattern
(iii) Adapter Pattern



Design Patterns

You probably have heard the phrase "Don't reinvent the wheel." Applied to software development it means don't write new software to solve a problem that someone else has already written software to solve correctly and efficiently. Many companies now take this approach with developing new applications. They would rather buy a software package off the shelf that meets the majority of their needs than build something from scratch. This approach ultimately saves time and money and makes good business sense if building the application would provide no competitive advantage, such as increased orders or greater market share. On a lesser scale, object-oriented developers look for the same reuse opportunities through the use of **design patterns.**

design pattern a common solution to a given problem in a given context, which supports reuse of proven approaches and techniques.

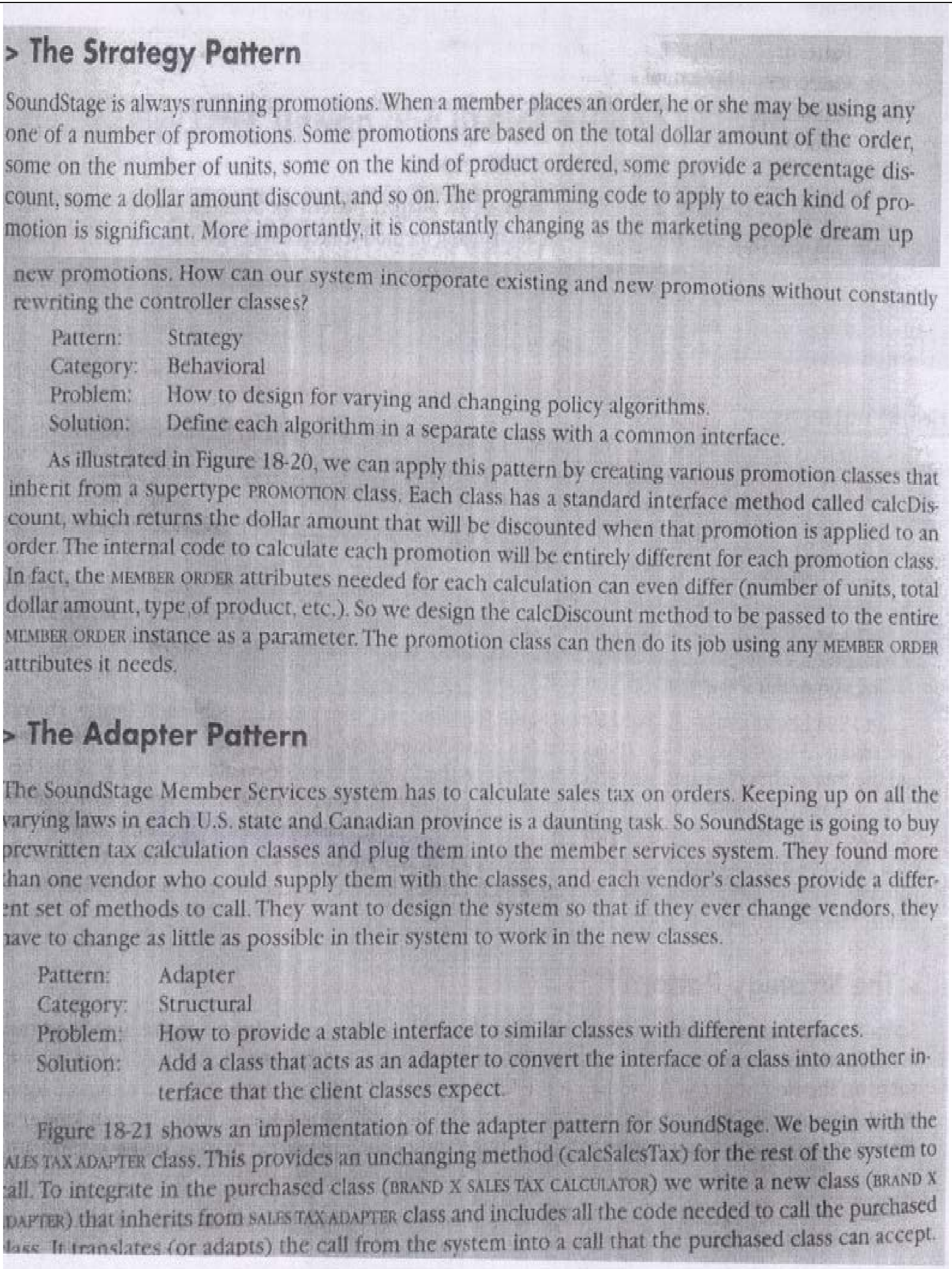
Over the course of many software projects, experienced developers collect a library of practices and routines, which worked well and correctly, that they can use over and over again in subsequent projects and even share with their fellow developers. These "development shortcuts," which are solutions to common design and programming problems, have come to be known as patterns. The goal of a pattern is not to discover or invent a new solution to a problem but to formally structure an existing solution to a common problem so that others may use it and take

Organizational patterns:

This pattern is very useful when your application has to work with organizational structures within a company or when individuals and companies can play the same role, such as customer.

There are two advantages to learning and using design patterns.

- They allow us to design information systems with the experiences of those who came before us rather than having to "reinvent the wheel."
- They provide designers a short-hand notation for discussing design issues. For instance, Bob Martinez might say to a colleague, "I know. Let's use a strategy pattern for promotions and build an adapter for integrating the Brand X sales tax class." You probably don't yet know what he's talking about. But if you read on, you will.



> The Strategy Pattern

SoundStage is always running promotions. When a member places an order, he or she may be using any one of a number of promotions. Some promotions are based on the total dollar amount of the order, some on the number of units, some on the kind of product ordered, some provide a percentage discount, some a dollar amount discount, and so on. The programming code to apply to each kind of promotion is significant. More importantly, it is constantly changing as the marketing people dream up new promotions. How can our system incorporate existing and new promotions without constantly rewriting the controller classes?

Pattern:	Strategy
Category:	Behavioral
Problem:	How to design for varying and changing policy algorithms.
Solution:	Define each algorithm in a separate class with a common interface.

As illustrated in Figure 18-20, we can apply this pattern by creating various promotion classes that inherit from a supertype PROMOTION class. Each class has a standard interface method called calcDiscount, which returns the dollar amount that will be discounted when that promotion is applied to an order. The internal code to calculate each promotion will be entirely different for each promotion class. In fact, the MEMBER ORDER attributes needed for each calculation can even differ (number of units, total dollar amount, type of product, etc.). So we design the calcDiscount method to be passed to the entire MEMBER ORDER instance as a parameter. The promotion class can then do its job using any MEMBER ORDER attributes it needs.

> The Adapter Pattern

The SoundStage Member Services system has to calculate sales tax on orders. Keeping up on all the varying laws in each U.S. state and Canadian province is a daunting task. So SoundStage is going to buy prewritten tax calculation classes and plug them into the member services system. They found more han one vendor who could supply them with the classes, and each vendor's classes provide a different set of methods to call. They want to design the system so that if they ever change vendors, they ave to change as little as possible in their system to work in the new classes.

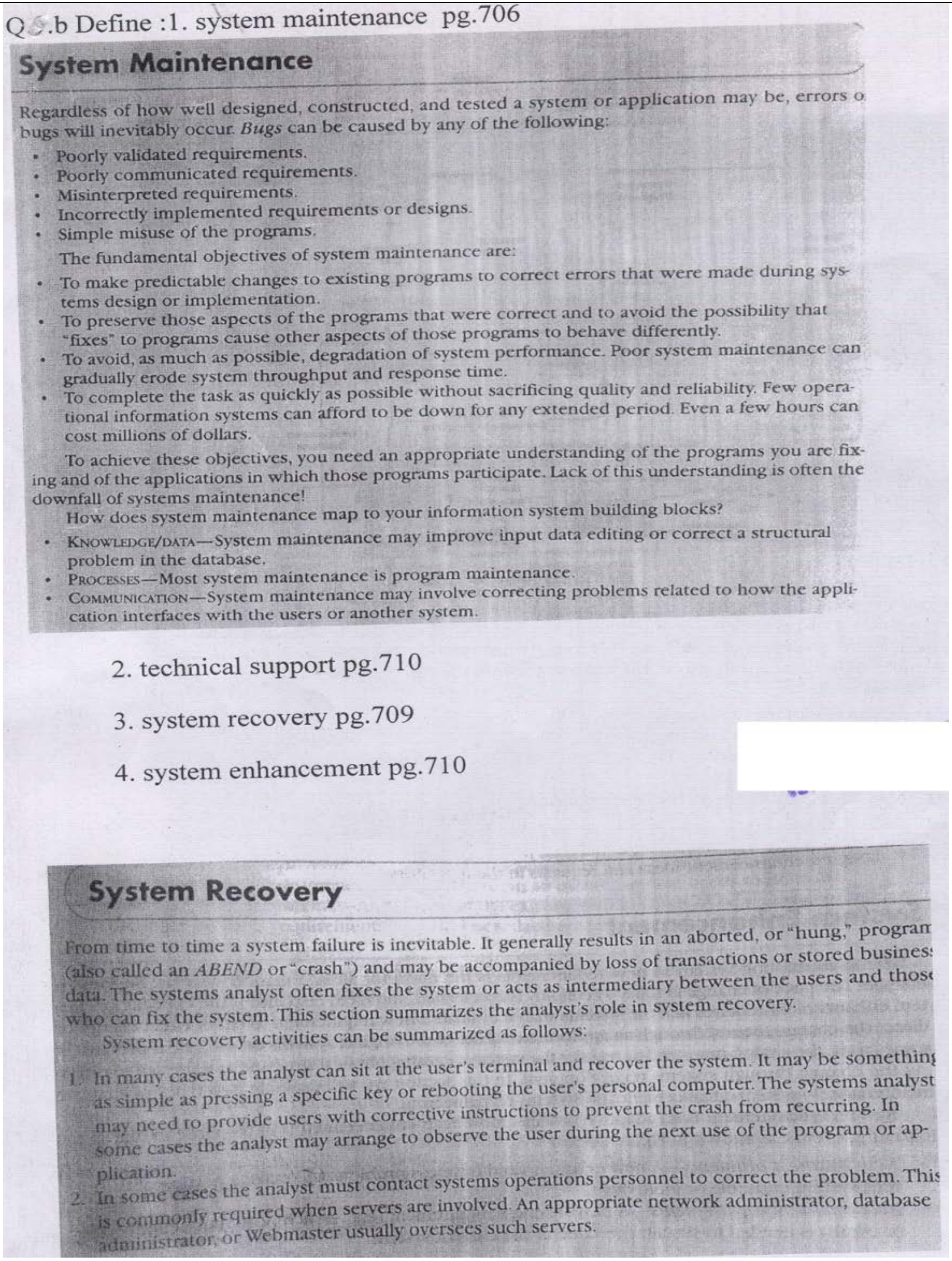
Pattern:	Adapter
Category:	Structural
Problem:	How to provide a stable interface to similar classes with different interfaces.
Solution:	Add a class that acts as an adapter to convert the interface of a class into another interface that the client classes expect.

Figure 18-21 shows an implementation of the adapter pattern for SoundStage. We begin with the ALES TAX ADAPTER class. This provides an unchanging method (calcSalesTax) for the rest of the system to all. To integrate in the purchased class (BRAND X SALES TAX CALCULATOR) we write a new class (BRAND X DAPTER) that inherits from SALES TAX ADAPTER class and includes all the code needed to call the purchased lass. It translates (or adapts) the call from the system into a call that the purchased class can accept.

9 a. Explain in detail various implementation phase tasks . Give the illustration of system implementation tasks.

Ans. Page 689 & 690 of Textbook I

b. Define : (i) system maintenance
(ii) technical support
(iii) system recovery
(iv) system enhancement



Q5.b Define :1. system maintenance pg.706

System Maintenance

Regardless of how well designed, constructed, and tested a system or application may be, errors o bugs will inevitably occur. *Bugs* can be caused by any of the following:

- Poorly validated requirements.
- Poorly communicated requirements.
- Misinterpreted requirements.
- Incorrectly implemented requirements or designs.
- Simple misuse of the programs.

The fundamental objectives of system maintenance are:

- To make predictable changes to existing programs to correct errors that were made during systems design or implementation.
- To preserve those aspects of the programs that were correct and to avoid the possibility that "fixes" to programs cause other aspects of those programs to behave differently.
- To avoid, as much as possible, degradation of system performance. Poor system maintenance can gradually erode system throughput and response time.
- To complete the task as quickly as possible without sacrificing quality and reliability. Few operational information systems can afford to be down for any extended period. Even a few hours can cost millions of dollars.

To achieve these objectives, you need an appropriate understanding of the programs you are fixing and of the applications in which those programs participate. Lack of this understanding is often the downfall of systems maintenance!

How does system maintenance map to your information system building blocks?

- KNOWLEDGE/DATA—System maintenance may improve input data editing or correct a structural problem in the database.
- PROCESSES—Most system maintenance is program maintenance.
- COMMUNICATION—System maintenance may involve correcting problems related to how the application interfaces with the users or another system.

2. technical support pg.710

3. system recovery pg.709

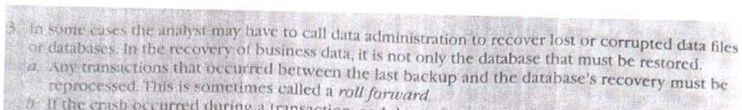
4. system enhancement pg.710

System Recovery

From time to time a system failure is inevitable. It generally results in an aborted, or "hung," progran (also called an *ABEND* or "crash") and may be accompanied by loss of transactions or stored busines data. The systems analyst often fixes the system or acts as intermediary between the users and thos who can fix the system. This section summarizes the analyst's role in system recovery.

System recovery activities can be summarized as follows:

1. In many cases the analyst can sit at the user's terminal and recover the system. It may be somethin as simple as pressing a specific key or rebooting the user's personal computer. The systems analyst may need to provide users with corrective instructions to prevent the crash from recurring. In some cases the analyst may arrange to observe the user during the next use of the program or application.
2. In some cases the analyst must contact systems operations personnel to correct the problem. This is commonly required when servers are involved. An appropriate network administrator, database administrator, or Webmaster usually oversees such servers.



3. In some cases the analyst may have to call data administration to recover lost or corrupted data files or databases. In the recovery of business data, it is not only the database that must be restored.
   a. Any transactions that occurred between the last backup and the database's recovery must be reprocessed. This is sometimes called a *roll forward*.
   b. If the crash occurred during a transaction

Textbook

1. Systems Analysis and Design Methods, Jeffrey L Whitten, Lonnie D Bentley, Seventh Edition, TMH, 2007