SCHOOL OF COMPUTING

Structured Systems Analysis and Design

Tutorial letter 102/2009

for

INF206D
# Contents

## Introduction

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
</tbody>
</table>

## Chapter 1 – The World of the Modern Systems Analyst

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>8</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>8</td>
</tr>
<tr>
<td>Notes on Chapter Opening Case and Case Studies</td>
<td>9</td>
</tr>
<tr>
<td>Additional Notes and Quick Quizzes</td>
<td>9</td>
</tr>
<tr>
<td>Solutions to End of Chapter Material</td>
<td>17</td>
</tr>
<tr>
<td>Case Study</td>
<td>19</td>
</tr>
<tr>
<td>Key Terms</td>
<td>20</td>
</tr>
</tbody>
</table>

## Chapter 2 – Approaches to System Development

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>22</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>23</td>
</tr>
<tr>
<td>Notes on Chapter Opening Case and Case Studies</td>
<td>23</td>
</tr>
<tr>
<td>Additional Notes and Quick Quizzes</td>
<td>23</td>
</tr>
<tr>
<td>Solutions to End of Chapter Material</td>
<td>32</td>
</tr>
<tr>
<td>Case Studies</td>
<td>34</td>
</tr>
<tr>
<td>Key Terms</td>
<td>38</td>
</tr>
</tbody>
</table>

## Chapter 3 – Investigating System Requirements

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>40</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>40</td>
</tr>
<tr>
<td>Notes on Chapter Opening Case and Case Studies</td>
<td>40</td>
</tr>
<tr>
<td>Additional Notes and Quick Quizzes</td>
<td>41</td>
</tr>
<tr>
<td>Solutions to End of Chapter Material</td>
<td>55</td>
</tr>
<tr>
<td>Case Studies</td>
<td>59</td>
</tr>
<tr>
<td>Key Terms</td>
<td>60</td>
</tr>
</tbody>
</table>

## Chapter 4 – Modeling System Requirements

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>This chapter corresponds with chapter 5 of the textbook.</td>
<td>62</td>
</tr>
<tr>
<td>Chapter Overview</td>
<td>62</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>62</td>
</tr>
<tr>
<td>Notes on Chapter Opening Case and Case Studies</td>
<td>63</td>
</tr>
<tr>
<td>Solutions to End of Chapter Material</td>
<td>70</td>
</tr>
<tr>
<td>Review Questions</td>
<td>70</td>
</tr>
<tr>
<td>Case Studies</td>
<td>72</td>
</tr>
<tr>
<td>Key Terms</td>
<td>77</td>
</tr>
</tbody>
</table>

## Chapter 5 – The Traditional Approach to Requirements

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter Overview</td>
<td>79</td>
</tr>
<tr>
<td>Learning Objectives</td>
<td>79</td>
</tr>
<tr>
<td>Notes on Chapter Opening Case and Case Studies</td>
<td>79</td>
</tr>
<tr>
<td>Notes on RMO Memo</td>
<td>80</td>
</tr>
<tr>
<td>Solutions to End of Chapter Material</td>
<td>89</td>
</tr>
<tr>
<td>Review Questions</td>
<td>89</td>
</tr>
<tr>
<td>Case Studies</td>
<td>91</td>
</tr>
<tr>
<td>Key Terms</td>
<td>96</td>
</tr>
</tbody>
</table>

## Chapter 6 – Elements of Systems Design

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 6 – Elements of Systems Design</td>
<td>98</td>
</tr>
</tbody>
</table>
This tutorial letter contains additional material for the module Structured Systems Analysis and Design (INF206D)

Introduction

The content of the module INF206D is covered by the prescribed textbook, Systems Analysis and Design in a Changing World (5th edition) by Satzinger, Jackson and Burd, which provides a balanced coverage of systems analysis and design activities and tasks using both

* structured analysis and design, with database design, input/output/controls design and dialog (interface) design

and

* object oriented analysis, with database design, input/output/controls design and dialog (interface) design.

Structured systems analysis and design (INF206D)

Outcomes:

• Demonstrate sound knowledge of the components of a systems development methodology.
• Gather information about the client’s system, its additional stakeholders, and its requirements as an activity of the SDLC.
• Analyse client needs and interpret and document it according to the system’s functional requirements applying the structured/traditional approach, as part of the systems analysis phase of the SDLC.
• Develop information from the analysis phase into Design Models.
• Design a normalised database model from conceptual Data Models.
• Identify system interfaces, design user interfaces, system inputs, -outputs, integrity- and security controls.
• Ensure that implementation requirements are defined and support activities are planned/structured.

Specific topics include:

• The world of the information systems analyst
• Approaches to system development
• Investigating system requirements
• Modelling system requirements – the structured approach
• The traditional/structured approach to requirements
• Elements of systems design
• The traditional approach to design
• Designing databases – the structured approach
• Making the system operational
Object-Oriented systems analysis (INF206D)

This module mainly emphasizes the object-oriented approach with regard to systems analysis. It will be accepted that you are familiar with this knowledge.

Outcomes:
- Analyse client needs and interpret and document it according to the system’s functional requirements applying the object-oriented approach, as part of the systems analysis phase of the SDLC
- Develop information from the analysis phase into Design Models
- Design an object database schema based on a class diagram
- Design system inputs and outputs

Specific topics include:
- Modelling system requirements — the object-oriented approach
- The object-oriented approach to requirements
- Evaluating alternatives for requirements, environment, and implementation
- Designing databases – the object-oriented approach
- Designing the user interface
- Designing system interfaces, controls, and security
- Current trends in system development

HOW TO USE THE INFORMATION IN THIS LETTER

1. This letter is designed to accompany the textbook, Systems Analysis and Design in a Changing World, fifth edition, by Satzinger, Jackson and Burd.
   It cannot be used independently of the above textbook.
   It is clearly stated with which chapter in the textbook the chapter in the letter corresponds.

2. Each chapter in this letter has a learning guide which contains some or all of the following types of material:
   - Chapter overview
   - Learning objectives
   - Notes on chapter opening case (where applicable)
   - Additional notes and quick quizzes
   - Review questions and answers
   - Additional examples
   - Case study(ies) with applicable questions and answers.
   - Key terms

3. Page numbers, figure numbers and table numbers used in this letter refer to pages, figures, and tables in the textbook.

4. The following is a possible approach for using this letter together with the textbook:
   For each chapter do the following:
   - Read through the **Chapter overview** and **Learning objectives** at the beginning
of each chapter in this letter to find out what knowledge and abilities you will have acquired after having successfully studied the chapter material.

- Read through the **notes on the chapter opening case** (where applicable) in this letter.
- Read the **overview** at the beginning of the chapter in the **textbook**.
- Study each topic in the **textbook**. Take special note of the keywords in the **textbook**, presented in bold writing, and of their definitions in the left-hand margin.
- Work through the examples, applicable to the case studies, which are presented in the **textbook**.
- Read and reread paragraphs in the **textbook** until you reach a proper understanding of the content.
- Work through the review questions and answers, additional examples and case studies with answers in **this letter**. If you experience problems with any of the questions, review the applicable material in the **textbook** and redo the questions until the answers are clear to you.
- Try to do some of the remaining questions, exercises and case studies in the textbook.

**This letter emphasizes the traditional or structured approach to systems analysis and design.**
Chapter 1 – The World of the Modern Systems Analyst

Chapter Overview

This chapter describes the role of the systems analyst—the nature of the work, the knowledge and skills that are important, and the types of systems and projects that analysts work on.

Information systems (IS) are crucial to the success of modern business organizations. New information systems are constantly being developed to make businesses more competitive. People are attracted to IS careers because information technology can have a dramatic impact on productivity and profits. People develop information system solutions to apply information technology to obtain business benefits.

The analyst’s work is defined as organizational problem solving, and the analyst follows a standard problem-solving process. The key to successful system development is thorough systems analysis and design to understand what the business requires from the information system.

This chapter surveys several types of business information systems that are used by organizations today. The information systems listed in the text are the types of systems that IS, MIS, and CIS graduates should be able to develop and deploy.

A systems analyst is a business professional who requires extensive technical, people, and business skills. There are numerous career options for people who have graduated with a degree in information systems. Job titles range from programmer analyst to business consultant to Web developer.

This chapter also introduces Rocky Mountain Outfitters (RMO), a regional sports clothing distributor headquartered in Park City, Utah. The RMO case is used throughout the text to illustrate analysis and design techniques. RMO is following a strategic information systems plan that calls for a series of information system development and integration projects over the next several years. The RMO project that we will study is a system development project for a new customer support system that will integrate phone orders, mail orders, and direct customer orders using the Internet.

Learning Objectives

After studying this chapter, you should be able to:

- Explain the key role of a systems analyst in business.
- Describe the various types of systems an analyst might work on.
- Explain the importance of technical, people, and business skills for an analyst.
- Explain why ethical behaviour is crucial for a systems analyst’s career.
- Describe various job titles in the field and places of employment where analysis and design work is done.
- Discuss the analyst’s role in strategic planning for an organization.
- Describe the analyst’s role in a systems development project.
Notes on Chapter Opening Case and Case Studies

A Systems Analyst at Consolidated Refineries: This case introduces the idea that a systems developer, even just a few years out of school, is very much involved with important organizational problems – not just programming or working with technical people. Mary has to understand the problems the organization faces, its strategic plan (and the need to constantly update that plan), and the importance of information system solutions to the future success of the organization. Mary rapidly became a business professional applying knowledge of information technology to the problems faced by the organization.

'This is not at all the way I thought it would be' is the message to students who primarily think of computer information systems as programming all day alone in a cubicle.

Note also the way the end of chapter case, Association for Information Technology Professional Meeting, reinforces this view. There, three professionals are talking about the importance of problem solving skills, communication skills, and understanding the business when they interview job candidates.

Rethinking Rocky Mountain Outfitters: Each chapter has an add-on case for RMO. These cases are frequently discussion cases that extend the RMO concepts that are presented in the chapter.

Focusing on Reliable Pharmaceutical Service: Each chapter has a Reliable Pharmaceutical Service case to parallel the RMO running case. The Reliable case is used throughout the text as an example of a smaller systems project compared to the larger system project of Rocky Mountain Outfitters. Their strategic plan is focused on an effort to streamline operations to improve service and reduce costs in order to be more competitive.

Additional Notes and Quick Quizzes

• Systems analysis: The process of understanding and specifying in detail what the information system should do.

• Systems design: The process of specifying in detail how the many components of the information system should be physically implemented.

• Systems analyst: A business professional who uses systems analysis and design techniques to solve business problems that require technology support.

The Analyst as a Business Problem Solver

How Systems Solve Organizational Problems
Systems analysts need to know about computers and programming, but they also should know and have a desire to use computers to solve problems. The solution to the “problem” is generally a new information system. Systems analysts solve problems for business organizations, such as the following:
• Problems getting orders from customers twenty-four hours a day.
• Problems planning production amounts to satisfy customer demands.
• Problems reducing inventory holding costs and obtaining supplier discounts.
• Problems anticipating customer needs by tracking buyer trends.
• Problems limiting complete information about the organization’s financial position.
• Problems limiting employee flexibility in benefits plans.

The Approach to Solving Problems
A systems analyst uses a generic problem-solving approach. The analyst uses a series of steps to systematically understand and solve the problem. These steps include the following:

1. Research and understand the problem.
2. Verify that the benefits of solving the problem outweigh the costs.
3. Define the requirements for solving the problem.
4. Develop a set of possible solutions (alternatives).
5. Decide which solution is the best, and make a recommendation.
6. Define the details of the chosen solution.
7. Implement the solution.
8. Monitor to make sure the desired results are obtained.

When a new information system will be a solution to a problem, it is important to understand the problem itself. This is the essence of systems analysis—understanding and defining what it takes to solve the problem. A business case must be made for solving the problem—if the benefits don’t outweigh the costs, then why should the problem be solved at all?

There are often many alternative solutions that will solve the problem. These solutions must be identified and carefully evaluated. A solution is chosen based on a variety of factors. The best solution has the greatest benefits and the fewest risks. The chosen solution is defined in detail, and then it is implemented. While the new system is being used, it is important to monitor the new system to be sure it is doing what is needed to solve the problem. Over time, the new system will need to be supported and perhaps modified.

Quick Quiz: The Analyst as a Business Problem Solver

What is the primary long-term value that an analyst can bring to an organization?

ANSWER: Tools and technologies continually change, but good problem solving skills are always required in every organization.

Systems that Solve Business Problems

Although the approach to problem solving presented above can be applied to solving any type of problem, this text is about problems that are solved with information systems.

Information Systems
The prospective analyst should be familiar with the following concepts:
• System: A collection of interrelated components that function together to achieve some outcome.

• Information system: A collection of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks.

• Subsystem: A system that is part of a larger system.

• Functional decomposition: Dividing a system into components based on subsystems that in turn are further divided into subsystems.

• System boundary: The separation between a system and its environment that inputs and outputs must cross.

• Automation boundary: The separation between the automated part of a system and the manual part of a system.

**Types of Information Systems**

Various types of information systems are found in business organizations. These systems are often integrated through the use of shared data. The types of systems include the following:

• Customer relationship management (CRM) system: a system that supports marketing, sales, and service operations involving direct and indirect customer interaction

• Supply chain management (SCM) system: a system that seamlessly integrates product development, product acquisition, manufacturing, and inventory management

• Accounting and financial management (AFM) system: a system that records accounting information needed to produce financial statements and other reports used by investors and creditors

• Human resource management (HRM) system: a system that supports employee-related tasks such as payroll, benefits, hiring, and training.

• Manufacturing management system: a system that controls internal production processes that turn raw materials into finished goods

• Knowledge management system (KMS): a system that supports the storage of and access to documents from all parts of the organization

• Collaboration support system (CSS): a system that enables geographically distributed personnel to collaborate on projects and tasks

• Business intelligence system: a system that supports strategic planning and executive decision making
Quick Quiz: Systems that Solve Business Problems

1) How are systems boundaries defined?
ANSWER: Boundary is the separation between a system and its environment that inputs and outputs must cross.

2) What are the types of information systems found in most organizations?
ANSWER: Knowledge management system (KMS), Human resource management (HRM) system, Customer relationship management (CRM) system, Supply chain management (SCM) system, Accounting and financial management (AFM) system, Collaboration support system (CSS), and Business intelligence system.

3) What are some examples of commercial knowledge management systems that are commonly used by businesses today?
ANSWER: Microsoft Office, Lotus Notes

Required Skills of the Systems Analyst

Because of the complexity of business problems and the information systems that solve them, a systems analyst needs extensive technical, business, and people skills.

Technical Knowledge and Skills
These are the most obvious areas of expertise: computers, peripheral devices, communications networks, connectivity, databases, programming languages, and operating systems. Analysts also use tools and techniques to build systems.

Tools: Software products that are used to develop analysis and design specifications and completed system components. Examples include development packages (such as Microsoft Access, Oracle Developer, and IBM Websphere Studio), integrated development environments (IDEs), computer-aided system engineering (CASE) tools, program code generators, documentation support tools, testing tools, and project management tools.

Techniques: Strategies for completing specific system development activities. Examples include project planning techniques, cost/benefit analysis techniques, interviewing techniques, requirements modelling techniques, architectural design techniques, network configuration techniques, and database design techniques.

Business Knowledge and Skills
Systems analysts need to understand business organizations and how they operate. It is important to understand how organizations are structured and managed in addition to understanding the business functions that are performed in finance, accounting, manufacturing, marketing, human resources, and customer service.

It is also important to understand the specific organization involved. What does it do? What makes it successful? What are its strategies and plans? What are its traditions and values? A system solution is tailored specifically for the needs of a specific organization.

People Knowledge and Skills
Because systems analysts often work on development teams with other employees, they need to understand a lot about people and possess many interpersonal skills. Analysts
need to understand how people think, learn, react to change, communicate, and work in a variety of jobs and levels.

A Few Words about Integrity and Ethics
It is also important for systems analysts to recognize the importance of ethical behaviour. They are trusted with private information, such as salary, health, and job performance information. They might also work with confidential corporate information about products, strategic plans, business tactics, and security systems and processes.

Quick Quiz: Required Skills of the Systems Analyst

1) What is the primary difference between tools and techniques?
   ANSWER: Tools are software products, e.g. things that are used. Techniques are strategies, e.g. methods or ways, for completing development tasks or deliverables.

2) List the six fundamental technologies an analyst needs to understand.
   ANSWER: Computers, devices that interact with computers, communication networks, databases and database management systems, programming languages, and operating systems/utilities.

3) How can an individual's lack of integrity or ethics impact an organization? Describe some current examples of unethical behaviour.
   ANSWER: An organization can be impacted when trade secrets or customers are lost. Financial organizations, such as banks and insurance companies, can have major losses if employees are unethical. Examples include Enron, Arthur Anderson, and the New York Times.

Analysis-Related Careers

Rapid changes in technology, business practices, and the structure of the global economy have changed related jobs. Typical information system graduates of the late twentieth century were employed as programmer analysts. Job tasks consisted primarily of programming with some analysis and design. The employment picture is much more complex in the twenty-first century. The number and nature of the jobs, their titles, and the organizations that fill those positions are much more complex than in the past. Changes in software development, technology, and business practices have created many new career opportunities for analysts, including:

- Sales and support of ERP software
- Business analysts for user organizations
- Auditing, compliance, and security
- Web development

People doing systems analysis and design work have many different job titles. Sometimes analysis and design work is only a part of their job responsibilities. Sometimes systems analysts also manage the project and are referred to as project leaders or project managers. Other job titles include:

- Programmer analyst
- Business systems analyst
• System liaison
• End-user analyst
• Business consultant
• Systems consultant
• Systems support analyst
• Systems designer
• Software engineer
• System architect
• Webmaster
• Web developer

The career prospects for analysts are bright, but the nature of related jobs, their location, and the typical career development path for analysts and other information system professionals has changed significantly over the last two decades.

Quick Quiz: The Environment Surrounding the Analyst

1) List ten job titles that involve systems analysis and design work.
ANSWER: Programmer analyst, business systems analyst, system liaison, end-user analyst, business consultant, systems consultant, systems support analyst, systems designer, software engineer, system architect, Webmaster, and Web developer. The following titles might also be used: project leader, project manager, lead analyst, team lead, and chief architect.

2) Based on the diversity of titles, do titles have any meaning or relevant value?
ANSWER: Titles have less importance than the role and nature of the work.

The Analyst’s Role in Strategic Planning

Systems analysts can become involved in strategic issues relatively early in their careers. They might meet with top-level executives and present recommendations to achieve corporate goals.

Special Projects
An analyst might work closely with executives to develop an executive information system (EIS). An analyst might also work on projects of interest to executives using business process reengineering, a technique that seeks to alter the nature of the work done in a business function with the objective of radically improving performance.

Strategic Planning
All businesses complete strategic planning, a process in which executives try to answer questions about the company, such as where they are now, where they want to be, and what they have to do to get there. Analysts are often involved with issues related to the role of information systems.

Information Systems Strategic Planning
Analysts might also be involved in developing the information systems strategic plan, the plan defining the technology and applications the information systems function needs to provide to support the organization’s strategic plan. It includes the applications architecture plan, a description of the integrated information systems needed by the organization to
carry out its business functions, and the technology architecture plan, a description of the hardware, software, and communications networks required to implement planned information systems.

**Quick Quiz: The Analyst’s Role in Strategic Planning**

1) Describe the strategic planning, information systems strategic planning, and project planning process that is typically followed by organizations.
   ANSWER: The organizational strategic planning drives the information systems strategic plan. The information systems strategic plan is used to identify individual projects.

2) What is the difference between the application architecture plan and the technology architecture plan?
   ANSWER: The application architecture plan is more of a strategic plan, which identifies the major types of systems that an organization needs. The technology architecture plan identifies specific systems, including hardware and software that are needed.

**Rocky Mountain Outfitters and Its Strategic Systems Plan**

This text uses the Rocky Mountain Outfitters (RMO) case study to demonstrate important systems analysis and design techniques. RMO requires a comprehensive information systems strategic plan. This text describes one information system development project that is included in the information systems strategic plan.

**Introducing Rocky Mountain Outfitters (RMO)**
RMO was created in 1978 by John and Liz Blankens in Park City, Utah. They started manufacturing and selling winter sports clothing to local ski shops. They then expanded their services by offering a more complete line of clothing through a mail order catalogue. RMO is now a large regional sports clothing manufacturer and distributor in the Rocky Mountain and Western states. Interest in winter and summer sports, such as skiing, snowboarding, mountain biking, water skiing, river running, jogging, hiking, camping, mountain climbing, and rappelling, led to the company’s rapid growth.

RMO employs 600 people and produces almost $180 million in annual sales. They now offer phone sales, catalogue sales, and retail stores. In 2004, John and Liz contracted with an outside firm to develop and host a Web-based ordering system, which has been functional since early 2005.

**RMO Strategic Issues**
In 2006 RMO performed a detailed market analysis that showed that RMO was growing more slowly than the industry average and the age of its customers was substantially older than its competitors.

By 2007 they realized that the Web-based ordering system was substantially underperforming against the competition. Some of the more critical problems included the following:

- Slow and cumbersome updates to online content
- Poor coordination with in-house customer service functions
- Poor coordination between Web-based ordering and supply chain management functions
RMO’s Organizational Structure and Locations
RMO is managed by John Blankens (president and CEO) and Liz Blankens (vice president of merchandising). Top managers also include the vice president of marketing and sales and the vice president of finance and systems. Working under the vice president of finance and systems are the assistant vice president of accounting and finance and the assistant vice president with the title of chief information officer (CIO).

Manufacturing, distribution, and other facilities are located in Oregon, Utah, Colorado, and New Mexico. Corporate offices are located in Park City, Utah.

The RMO Information Systems Department
The information systems department is headed by Mac Preston, chief information officer, and includes about fifty employees. The information systems department is organized into system development and system support.

Existing RMO systems
- Supply Chain Management
- Mail order
- Phone order
- Retail store systems
- Office systems
- Human resources
- Accounting/finance
- Web-based Catalogue and Order System

The Information Systems Strategic Plan
The current RMO strategic plan was developed with the help of consultants based on the strategic plans of RMO. The information systems strategic plan defines a technology architecture plan and an application architecture plan to support the corporate plan.

The technology architecture plan calls for distributing applications across multiple locations and computer systems, conducting more business processes via the Internet including SCM and CRM systems, and moving toward intranet applications for business functions such as human resources, accounting, finance, and information management.

The application architecture plan includes future use of packages for retail store systems (RSS), accounting/finance, and human resources and strategic information management systems (SIMS). New system development will be done for a customer support system (CSS) and supply chain management (SCM).

The time line for the systems plan is as follows (example in text):
- 2009-2010: New development of supply chain management (SCM)
- 2010-2011: New development of customer support system (CSS)
- 2011: Package solution for strategic information management system (SIMS)
- 2011: Replace retail stores system with new package
- 2012: Replace existing accounting/finance system with intranet package
- 2013: Replace human resources system with intranet package

The Customer Support System
The information system project followed in this text is the customer support system (CSS)
project, a system development project begun shortly after the SCM system project. The CSS includes integrating the mail order and phone order processes and adding direct customer interaction via the Internet.

**The Analyst as a System Developer (The Heart of the Course)**

This section provides an overview of the text, which focuses on the systems analyst as a system developer working on a system development project. Topics include:

**Part 1: The Systems Analyst**
1. The World of the Systems Analyst (this chapter)
2. Approaches to System Development
3. The Analyst as a Project Manager

**Part 2: Systems Analysis Tasks**
4. Beginning the Analysis: Investigating System Requirements
5. Modelling System Requirements
6. The Traditional Approach to Requirements
7. The Object-Oriented Approach to Requirements
8. Evaluating Alternatives for Requirements, Environment, and Implementation

**Part 3: Systems Design Tasks**
9. Moving to Design
10. The Traditional Approach to Design
11. Object-Oriented Design: Principles
12. Object-Oriented Design: Use Case Realizations
13. Designing Databases
14. Designing the User Interface
15. Designing System Interfaces, Controls, and Security

**Part 4: Implementation and Support**
16. Making the System Operational
17. Current Trends in System Development

**Solutions to End of Chapter Material**

**Review Questions**

1. **Give an example of a business problem.**
   See page 5 for some examples. The answer should focus on the business needs of the problem. Business problems that are applicable for information systems include either a business process that requires or processes information. The other type of applicable business problem entails producing reports or providing critical information to the organization. For example, the self-insured health insurance program needs to project expenses for the next year based on past claims, changes in employee demographics, and estimated increases in health care costs. Notice that this example has an extensive requirement for information that can best be provided by a computer system.
2. **What are the main steps followed when solving a problem?**
Research and understand the problem, verify that the benefits of solving the problem outweigh the costs, define requirements for solution, develop a set of possible solutions (alternatives), decide which solution is best, define the details of the chosen solution, monitor to make sure that you obtain the desired benefits. (See Figure 1-1)

3. **Define system.**
A collection of interrelated components that function together to achieve some outcome.

4. **Define information system.**
A collection of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks.

5. **What are the types of information systems found in most organizations?**
Customer relationship management system, Accounting and financial management system, Manufacturing management system, and Human resources management system, Knowledge management system, Business intelligence system, Supply chain management system, Collaboration support system (see Figure 1-5).

6. **List the six fundamental technologies an analyst needs to understand.**
Computers and how they work; file, database, and storage technologies; input and output hardware; computer networks and protocols; programming languages and operating systems; and communication and collaboration.

7. **List four types of tools the analyst needs to use to develop systems.**
Software packages used to develop systems, integrated development environments (IDEs), computer-aided system modelling and code generation tools, testing tools, and documentation support tools.

8. **List five types of techniques used during system development.**
Project planning techniques, cost/benefit analysis techniques, interviewing techniques, requirements modelling techniques, architectural design techniques, network configuration techniques, and database design techniques.

9. **What are some of the things an analyst needs to understand about businesses and organizations in general?**
The activities and processes organizations perform, how they are structured, how they are managed, the type of work that goes on – finance, manufacturing, marketing, customer service, and so on - and the organizational structure.

10. **What are some of the things an analyst needs to understand about people?**
Most important is communication skills. Other skills include being able to build rapport, be able to understand and translate people's problems into business requirements. The analyst must play many roles, including teacher, mentor, confidante, collaborator, and manager.

11. **What are some of the types of technology an analyst might encounter?**
Desktop systems, networked desktop systems, client-server systems, large-scale centralized mainframe systems, and systems using Internet technology.
12. **List ten job titles that involve analysis and design work.**
   Programmer analyst, business systems analyst, system liaison, end-user analyst, business consultant, systems consultant, system support analyst, system designer, software engineer, system architect, webmaster, web developer, etc. Also, project leader, project manager, lead analyst.

13. **How might an analyst become involved with executives and strategic planning relatively early in his or her career?**
   Working on special projects, such as business process management, and working on the information systems strategic plan brings the analyst into contact with top management.

**Case Study**

**Case Study: Association for Information Technology Meeting**

Three information systems professionals discuss what they look for when interviewing college students for positions in their firms.

1. **Do you agree with Alice and the others about the importance of problem solving skills? Industry specific insight? Communications skills? Discuss.**
   We agree with Alice that asking students how they go about solving problems and questions about problems facing the banking industry reveals much about the maturity and potential of the job applicant.

2. **Should you research how a hospital is managed before interviewing for a position with an information systems manager at a hospital? Discuss.**
   We think you should research the industry of a company before interviewing. First, you should know whether the industry is appealing. Since much time is spent on the job working with people involved in the industry, it helps if it is interesting. Second, it gives the interviewer the feeling that the applicant cares about the job. Third, it allows the applicant to ask meaningful questions about problems and opportunities in the company. When the interviewer starts talking about a specific system, the applicant will know something about the problem solved by the system.

3. **It terms of your career, do you think it really makes a difference whether you go to work for a bank, a hospital, or a retail chain in terms of your career? Or is any information systems job going to be the same no matter where you work? Discuss.**
   We think this is important for you understand. If you go to work for a bank, you eventually might think of yourself as a banker. If you go to work for a retail chain, you might begin to think of yourself as a retailer. Information systems employees have loyalties to the IS field but also to the industry and company they work in. Since the expertise about the industry is so important and takes time to develop, it is often difficult to switch industries mid career. You should give this some thought.
Key Terms

- **Applications architecture plan** – A description of the integrated information systems that the organization needs to carry out its business functions.
- **Automation boundary** – The separation between the automated part of a system and the manual part of a system.
- **Business process reengineering** – A technique that seeks to alter the nature of the work done in a business function, with the objective of radically improving performance.
- **Communication support systems** – Support systems that allow employees to communicate with each other and with customers and suppliers.
- **Customer relationship management (CRM)** – Processes that support marketing, sales, and service operations involving direct and indirect customer interaction.
- **Decision support systems (DSS)** – Support systems that allow a user to explore the impact of available options or decisions.
- **Enterprise resource planning (ERP)** – A process in which an organization commits to using an integrated set of software packages for key information systems.
- **Executive information systems (EIS)** – Information systems for executives to use for monitoring the competitive environment and for strategic planning.
- **Functional decomposition** – Dividing a system into components based on subsystems that in turn are further divided into subsystems.
- **Information system** – A collection of interrelated components that collect, process, store, and provide as output the information needed to complete business tasks.
- **Information systems strategic plan** – The plan defining the technology and applications that the information systems function needs to support the organization’s strategic plan.
- **Management information systems (MIS)** – Information systems that take information captured by transaction processing systems and produce reports that management needs for planning and control.
- **Office support systems** – Support systems that help employees create and share documents, including reports, proposals, and memos.
- **Strategic planning** – A process during which executives try to answer questions about the company such as where the business is now, where they want the business to be, and what they have to do to get there.
- **Subsystem** – A system that is part of a larger system.
- **Supersystem** – A larger system that contains other systems.
- **Supply chain management (SCM)** – A process that seamlessly integrates product development, product acquisition, manufacturing, and inventory management.
- **System** – A collection of interrelated components that function together to achieve some outcome.
- **System boundary** – The separation between a system and its environment that inputs and outputs must cross.
- **Systems analysis** – The process of understanding and specifying in detail what the information system should do.
- **Systems analyst** – A business professional who uses analysis and design techniques to solve business problems using information technology.
- **Systems design** – The process of specifying in detail how the many components of the information system should be physically implemented.
- **Techniques** – Strategies for completing specific system development activities.
- **Technology architecture plan** – A description of the hardware, software, and
communications networks required to implement planned information systems.

- **Tools** – Software products used to help develop analysis and design specifications and completed system components.
- **Transaction processing systems (TPS)** – Information systems that capture and record information about the transactions that affect the organization.
Chapter 2 – Approaches to System Development

Chapter Overview

There are two closely related yet independent concepts in this chapter. The first important concept is that there are two types of Systems Development Life Cycle approaches—a predictive approach and an adaptive approach. The second important concept is that there are two types of development methodologies—a structured approach and an object-oriented approach. These are two separate concepts. Projects can be any mix of these two approaches, the approach to the life cycle and the approach to the methodology—predictive with structured, predictive with object-oriented, adaptive with structured, or adaptive with object-oriented.

The chapter first presents and explains the differences in the life cycle approach—the predictive and the adaptive approaches. These two approaches are really a continuum and any given project may have elements of both approaches. The predictive approach to the SDLC is used for projects that are well understood and low risk. The adaptive approach to the SDLC is used for projects that are not well understood and are higher risk. Adaptive SDLCs are more iterative and allow the project team to adapt the project to changing circumstances.

Within each approach it is necessary to perform planning, systems analysis, systems design, systems implementation. In the predictive approach, these are called phases. The SDLC phases are project planning, analysis, design, implementation, and support. The activities within each “phase” are defined and explained.

The other important concept that should be learnt from this chapter is the difference between the two main methodologies to system development that are currently used to develop business systems: the traditional approach and the object-oriented approach. The traditional approach refers to both structured system development (structured analysis, structured design, and structured programming) and information engineering (IE). The object-oriented approach refers to system development using newer object technologies that require a different approach to analysis, design, and programming.

The SDLC as well as models, techniques, and tools comprise a system development methodology.

Most variations of the SDLC use iterations across phases. Current trends, which are becoming system development best practices, include the Unified Process (UP), Extreme Programming (XP), Agile Modelling, and Scrum.

This chapter concludes with a discussion of the tools used to support system development, such as, integrated application development tools, visual modelling tools, and round-trip engineering tools.
Learning Objectives

After studying this chapter, you should be able to:

• Explain the purpose and various phases of the traditional system development life cycle (SDLC).
• Explain when to use an adaptive approach to the SDLC in place of the predictive traditional SDLC.
• Explain the differences between a model, a tool, a technique, and a methodology.
• Describe the two overall approaches used to develop information systems: the traditional approach and the object-oriented approach.
• Describe the key features of current trends in system development: the Unified Process (UP), Extreme programming (XP), and Scrum.
• Explain how automated tools are used in system development.

Notes on Chapter Opening Case and Case Studies

Development Approaches at Ajax Corporation, Consolidated Concepts, and Pinnacle Manufacturing: This case shows that there are many different names for the same basic system development concepts used in industry. Despite what many textbooks might imply, there is not very much industry standardization. Therefore, it is a good idea to be aware of synonyms used for key concepts. Also there are two major approaches to system development: the traditional approach (structured and information engineering) and the object-oriented approach. This text emphasizes the diversity of approaches in the real world, although it is expected that each course will emphasize one approach for the hands-on exercises in analysis and design.

Rethinking Rocky Mountain Outfitters: One of Barbara Halifax’s initial jobs as project manager is to make decisions about the approach used to develop the customer support system. All of the options contained within this chapter should be considered and note that this also includes current trends in systems development. Her final decision is not described because the customer support system example will be used throughout the text to present more details about all approaches.

Focusing on Reliable Pharmaceutical Service: This case is similar to the Rocky Mountain Outfitters case. One of initial jobs as project manager is to make decisions about the approach used to develop the web-based application to connect client facilities with Reliable. This system may have a heavier emphasis on designing controls due to compliance requirements with HIPAA and intranet security controls.

Additional Notes and Quick Quizzes

The Systems Development Life Cycle

Project: A planned undertaking that has a beginning and an end, and which produces a predetermined result or product.

The term system development project describes a planned undertaking, which is
normally a large job that produces a new information system. Success depends heavily on having an organized, methodical sequence of tasks and activities that culminate with an information system that is reliable, robust, and efficient.

One of the key concepts in system development is the systems development life cycle (SDLC). The SDLC refers to the entire process of building, deploying, using, and updating an information system.

The other major concept in this chapter that students should learn well relates to the two types of SDLC approaches. The nature of the project determines the best approach to use. A predictive approach to the SDLC assumes that the development project is planned in advance and that the new information system can be developed according to the plan. An adaptive approach to the SDLC is used when the exact requirements or needs of the users are not well understood. A more flexible approach is needed that allows the plan to be modified as the project progresses.

The Traditional Predictive Approaches to the SDLC
In every project there must be activities associated with project planning, analysis and understanding the requirements, design and structuring of a solution, implementation, and support when the system goes into production. This text teaches the basic concepts associated with the planning, analysis, design, implementation, and support phases of an SDLC.

The five phases of the SDLC are quite similar to the general problem-solving approach described in Chapter 1. The organization recognizes a problem and plans a project, studies the problem in detail, designs a solution, implements the solution, and then monitors the solution to make sure the intended benefits are achieved.

The SDLC that is the most predictive is called the waterfall approach, an SDLC that assumes the various phases of a project can be completed sequentially—one phase falls into the next phase, and there is no going back. Other modified waterfall approaches are more flexible. The five phases are followed sequentially but overlap. For example, during the analysis phase, some design and even implementation work might be completed.

It is important to note that all large development projects that are successful use some type of organized system development method. Small projects also require analysis, design, and implementation, but may not be as rigorous or include as much planning. The concept of relating similar activities into phases and defining the steps is what makes the SDLC an appropriate development method. Without the structure and organization provided by an SDLC method, development projects are at severe risk for missed deadlines, escalating budgets, and producing a low-quality system.

The Newer Adaptive Approaches to the SDLC
In contrast to the predictive SDLC, the adaptive SDLC assumes project activities have to be adjusted as the project progresses. This is necessary because aspects of the project are not well understood at the beginning. An early version of an adaptive SDLC is called the spiral model, which cycles over and over again throughout development activities and makes adjustments until the project is complete. After each cycle or iteration, a working prototype is available as a preliminary model that shows some aspect of the system that is ready for testing and user evaluation. An iteration is one cycle in the spiral model during which work activities—analysis, design, implementation—are used to complete a version
of the working prototype. Completing a project using multiple iterations breaks the problem
of system development into more manageable mini-projects.

Risk is a key concept from the spiral model. An iteration is planned based on an
assessment of the risk the projected faces at the time. Sometimes the greatest risk is
technology. Sometimes the greatest risk is user acceptance. In an adaptive approach, the
project manager focuses early on the greatest risk to the project and works to mitigate the
risk in an iteration.

Incremental development is a type of iterative development in which you complete parts of
the system in a few iterations and then put the system into operation for the users. You
then perform additional iterations to complete the next part of the system.

**Activities of Each SDLC “Phase”**

In the traditional SDLC the grouping of similar activities is called a phase. In the adaptive
approach similar activities are required, but are usually not called phases. Hence we group
related activities together but do not call them phases.

**Planning activities:** The initial set of activities of the SDLC, whose objective is to identify
the new system scope and plan the project.

These activities are grouped together because they help to achieve the objective of a
planning, not because they occur at a specific time in the project. Planning a project
involves the following five activities:

- Define the problem.
- Produce the project schedule.
- Confirm project feasibility.
- Staff the project.
- Launch the project.

**Analysis activities:** The activities of the SDLC whose objective is to understand the user
needs and develop requirements.

**Problem domain:** The area of the user’s business for which a system is being developed.

Analysis is essentially a discovery process. The key words to drive the activities during
analysis are discovery and understanding. Six primary activities are part of this phase:

- Gather information.
- Define system requirements.
- Build prototypes for discovery of requirements.
- Prioritize requirements.
- Generate and evaluate alternatives.
- Review recommendations with management.

Gathering information is a fundamental part of analysis. Information about the problem is
obtained by observing users performing the business procedures, interviewing and asking
the users questions, reading existing documents about procedures, business rules, and
job responsibilities, and reviewing existing automated systems. This activity is the core
activity for discovery and understanding.
The information obtained must be reviewed, analyzed, and structured so that an overall understanding of requirements for the new system can be developed. In defining the system’s requirements, analysts use drawing diagrams to model the processing requirements for the new system. Prototyping during analysis is done as a discovery activity. In developing requirements for new systems, users often find it easier to express the needs of the system by reviewing working prototypes of processing alternatives. This method is similar to the “a picture is worth a thousand words” approach in communication.

**Design activities:** The activities of the SDLC during which the system and programs are designed.

**Application:** The portion of the new system that satisfies the user needs in the problem domain. This term is somewhat overused in the industry. This text does not try to distinguish between those activities that produce a general design and those that produce a detailed design. Instead, a general distinction is made between the two.

The following are the primary activities of the design phase:

- Design and integrate the network.
- Design the application architecture.
- Design the user interfaces.
- Design the system interfaces.
- Design and integrate the database.
- Prototype for design details.
- Design and integrate the system controls.

The inclusion of prototyping permits a discussion of those types of development methods that are prototype-driven. During design, the accuracy and feasibility of the proposed design is verified. One verification method that is used is to build working prototypes of the parts of the system to ensure that it will function correctly in the operating environment. Sometimes, if the prototypes are built correctly, they can be saved and used as part of the final system. System controls are designed into the new system to protect the information and assets of the organization.

**Implementation activities:** The activities of the SDLC during which the new system is programmed and installed.

The implementation phase includes all those activities that occur between the design of the system and placing the new system in production. The text emphasizes the difference between the terms implementation and installation. Implementation consists of the following six activities that are needed to bring a system into production:

- Construct software components.
- Verify and test.
- Convert data.
- Train users and document the system.
- Install the system.

The objective of the activities of this phase is not only to have a reliable, well-working information system, but also to ensure that the users are trained and that the business
benefits are obtained. All the prior activities must come together during this phase to culminate in an operational production system.

**Support activities:** The activities of the SDLC whose objective is to keep the system running productively after it is installed. The activities that occur during this phase fall into two general categories: providing support to end users, and maintaining and enhancing the computer system. End-user support is an ongoing effort that is carried out through the implementation of help desks and training programs. Enhancements and upgrades to the system occur throughout the life of the system to maintain and extend its productive life.

**Quick Quiz: The Systems Development Life Cycle**

1) What are the six activities done during analysis?
   **ANSWER:** Gather information, define system requirements, build prototypes for discovery of requirements, prioritize requirements, generate and evaluate alternatives, and review recommendations with management. See pages 45 and 46 for a brief description of each.

2) Why is support not typically included in iterations?
   **ANSWER:** Iteration is the process of looping through the same analysis and design activities during various times to extract more detail in the analysis or to refine and extend the design and newly constructed system. Support begins when implementation is complete.

**Methodologies, Models, Tools, and Techniques**

A systems analyst must decide which approach to use when developing a system. Several choices are available. It is important to distinguish between methodologies, models, tools, and techniques to organize the choices.

Methodologies provide comprehensive guidelines to follow for completing every activity in the systems development life cycle, including specific models, tools, and techniques. A methodology might be home-grown or purchased from a consulting firm.

Models are representations of some important aspect of the real world. A collection of different types of models is needed to depict the requirements for an information system or the design of an information system. Some examples are flow charts, data flow diagrams, entity-relationship diagrams, use case diagrams, class diagrams, and sequence diagrams. Some models are used to manage the development process, such as PERT charts. Gantt charts and financial analysis models are used to calculate net present value or return on investment.

Tools provide software support that helps the analyst create models or other components required in the project. These include drawing tools, project management tools, integrated development environments (IDEs), reverse engineering tools, code generating tools, and debugging tools.

Techniques provide a collection of guidelines that help the analyst complete a system development activity or task. These include project planning techniques, user interviewing techniques, data modelling techniques, structured analysis techniques, structured design techniques, structured programming techniques, and software testing techniques, and object-oriented analysis and design techniques.

How do these fit together? A methodology includes a collection of techniques that are
used to complete activities within each phase of the systems development life cycle. The activities include the completion of a variety of models as well as other documents and deliverables. As with any other occupation, system developers use software tools to help them complete the activities.

**Quick Quiz: Methodologies, Models, Tools, and Techniques**

1) Describe the differences between models and tools.
**ANSWER:** A model is a representation of some important aspect of the real world. A tool provides software support that helps create models or other components required in the project.

2) Describe the differences between methodologies and techniques.
**ANSWER:** A methodology provides comprehensive guidelines to follow for completing every activity in the systems development life cycle, including specific models, tools, and techniques. A technique is a collection of guidelines that help the analyst complete a system development activity or task.

**Two Approaches to System Development**

Systems analysts should be familiar with the following two approaches to system development. Methodologies used in the industry are based on these approaches.

**The Traditional Approach**
The traditional approach includes many variations based on the techniques that are used to develop information systems using structured and modular programming. Structured approaches and information engineering are two terms used for this approach.

**Structured System Development**
The structured approach refers to system development using structured programming, structured design, and structured analysis techniques. These techniques are sometimes collectively referred to as the structured analysis and design technique (SADT).

In the 1960s, structured programming was the first technique used to provide guidelines for producing quality programs. A structured program has one beginning and one end, and each step in the program execution consists of sequence, selection (or decision), or repetition constructs. A structured program is easy to read and debug. Another concept related to structured programming is top-down programming, which divides more complex programs into a hierarchy of program modules. One module “calls” or “invokes” another module below it in the hierarchy. Sometimes each module is a separate program or sub-program, which is referred to as modular programming.

**Structured design** is a technique that provides guidelines for deciding what the set of programs should be what each program should accomplish, and how the programs and program modules should be organized into a top-down hierarchy. The structured design technique was developed in the 1970s. The structure chart, a graphical model showing the hierarchy of program modules produced in a structured design, is the main model used in structured design.

Two principles guide the design of the structure chart: the modules should be loosely
coupled and highly cohesive. By the 1980s, structured design included provisions for integration with database management systems and was done in conjunction with user interface design to produce interactive software designs. Structured design as a technique, however, focuses on software design only and not on all design phase activities.

**Structured analysis** was the last of the structured approaches to be developed, although it is used before the other techniques in the SDLC. Structured analysis is a technique that helps the developer define what the system needs to do (the processing requirements), what data the system needs to store and use (data requirements), what inputs and outputs are needed, and how the functions work together overall to accomplish tasks. These detailed requirements, which are defined during systems analysis, are used to produce the system design.

The key model used with structured analysis is the **data flow diagram (DFD)**, which shows the inputs, processes, storage, and output requirements of a system in graphical format. A collection of DFDs is used to represent the system. The most recent version of structured analysis defines system-processing requirements by identifying all of the events that require the system to react in some way. Structured analysis also uses the **entity-relationship diagram (ERD)**, a graphical model of the data needed by the system. The ERD is also used in the information engineering approach.

Even with the many variations of the structured approach, some people consider it limiting because only parts of analysis and design are addressed.

**Information Engineering**

**Information engineering** (a methodology to some, mainly credited to James Martin) focuses on strategic planning, data modelling, and automated CASE tools. Information engineering (IE) is thought to be more rigorous and complete than the structured approach. Critics of the structured approach wanted more emphasis on planning the information systems to be built so they would function together. The strategic information systems planning discussed in Chapter 1 comes from the information engineering approach. Data is thought to be more stable than processes in a system, so information engineering focuses more on the data model (ERD) than on process models, such as the DFD. A process dependency diagram is used instead. Finally, information engineering provides guidelines for the complete life cycle, and provides CASE tool support for every activity. Code generation is possible if the CASE tool is used carefully throughout the phases. Flexibility is sacrificed to gain rigor and automation.

This text combines structured development and information engineering into what is referred to as the **traditional approach** because of the similarities between the two. Both focus on process, data, and the interaction of the two approaches. The other approach is the object-oriented approach, which is quite different in systems analysis, systems design, and programming.

**The Object-Oriented Approach**

The **object-oriented approach** views an information system as a collection of interacting objects that work together to accomplish tasks. Data and processes are not separate. Instead, there are only **objects**, things in the computer system that can respond to messages. An object has data and is capable of actions. This is a very different approach
The object-oriented approach originated with the development of the Simula programming language in Norway in the 1960s, but became better known with the development of the Smalltalk language that led to graphical user interfaces. Graphical user interfaces include objects the user interacts with, and now all parts of the computer system can be developed as objects. C++ and Java are popular object-oriented programming languages. Other languages are not adding object-oriented capabilities. These languages include PHP and Visual Basic dot Net.

Object-oriented analysis (OOA) means defining all of the types of objects that do the work in the system and showing what use cases are required to complete tasks. Object-oriented design (OOD) means defining all of the types of objects necessary to communicate with people and devices in the system and refining the definition of each type of object so it can be implemented with a specific programming language or environment. Object-oriented programming (OOP) means writing statements in a programming language to define what each type of object does, including the messages the objects send to each other.

An object is a type of thing—a customer or an employee as well as a button or a menu. A class represents a collection of similar objects that are “classified” as the same type of thing. Therefore, a class diagram is used as a graphical model that shows all classes of objects in the system in the object-oriented approach. Other key graphical models are the use case diagram, sequence diagram, communication diagram, and state machine diagram.

Two key benefits of the object-oriented approach are its ability to seem natural to people and its ability to be reused. The object-oriented approach is natural because people tend to think of the world around them as objects. It is less natural to think of complex procedures. Because many systems in the organization use the same classes of objects, the classes can be reused over and over again without the need to reinvent the wheel. Although the object-oriented approach is newer and quite different from the traditional approach, there are common concepts between the two methodologies that will be explored later in this text.

Quick Quiz: Two Approaches to System Development

Where would an organization be more likely to use structured development instead of object-oriented development?

ANSWER: For those projects that are enhancements to existing systems, in C or COBOL, would most likely require a structured approach. Also projects for systems that require very little user interface, such as scientific systems, are sometimes still developed in a structured language. However, almost all Web development as well as any system with a graphical user interface (GUI) will most likely be developed using object-oriented techniques and languages.

Current Trends in Development

The information systems field is constantly changing. New techniques and tools are always appearing. A few of the current trends are discussed in this section. They are discussed in greater detail in Chapter 16.
The Unified Process (UP)
Booch, Rumbaugh, and Jacobson of Rational Software, and now IBM, are the originators of the Unified Modelling Language (UML). UML is a modelling notation, not a methodology. Booch, Rumbaugh, and Jacobson have also developed a system development methodology, called the Unified process (UP), that they advocate in their books and training programs. UP is not a standard, but because of the stature of Booch, Rumbaugh, and Jacobson, it is receiving much attention. UP is based on the six best practices of system development that are similar to the practices outlined in this text—iteration, defining and managing system requirements, component architectures, visual models, quality assurance, and change control. The four life cycle phases that parallel the generic SDLC are called inception, elaboration, construction, and transition (see Figure 2-20). UP uses the term development process to mean development methodology. It is explicitly an OO development methodology.

Extreme Programming (XP)
Extreme Programming (XP) is a “lightweight” development methodology popularized by Kent Beck. Lightweight methodologies are those that are kept simple and focus on the needs of the developers. XP begins by working with users to discover what are called “user stories” (similar to use cases) that define the requirements of the system. Developers create a series of “releases” of the software (similar to incremental development). XP also specifies that the programmers work in teams of two to co-author code.

Scrum
Scrum is an adaptive development methodology that is reminiscent of the sport of rugby because both the system development approach and the sport are quick, adaptive, and self-organizing.

Quick Quiz: Current Trends in Development
1) What are some features of the Unified Process (UP)?
   ANSWER: UP is a development methodology proposed by Booch, Rumbaugh, and Jacobson of Rational Software (who originated UML). UML is a standard modelling notation for OO. The Unified Process has four life cycle phases, assumes the use of UML models, and is based on accepted best practices of system development.

2) What are some features of Extreme Programming (XP)?
   ANSWER: XP is a lightweight methodology (simple and focused on helping the developer) that involves iterations, multiple releases, user stories to define requirements, and programmer teams working together on code.

Tools to Support System Development
Visual modelling tools are used to assist in documenting system requirements and system design. These tools include a repository, a database that stores information about the system, including models, descriptions, and references that link the various models together. Drawing on the repository, the modelling tool can generate program code based on the models.

This section shows examples of Microsoft Visio, Visible Analyst, Embarcadero Describe, and Rational XDE Professional. Products are frequently upgraded and vendors commonly provide Web sites with product descriptions and additional product-related information.
Solutions to End of Chapter Material

Review Questions

1. **What are the five phases of the SDLC?**
   The five phases of the SDLC are planning phase, analysis phase, design phase, implementation phase, and support phase.

2. **What characteristics of a project call for a predictive approach to the SDLC?**
   Predictive: Projects are well understood—technology is well known; user requirements are well known; development methodology is well known; project team is experienced and familiar with the system; and there are few known risks.
   Adaptive: Projects are not well understood—technology is new or unfamiliar; requirements are not very clear; team is not experienced with the type of system or methodology.

3. **How is the SDLC based on the problem-solving approach described in Chapter 1?**
   The five phases are similar to the steps in the general problem-solving approach outlined in Chapter 1. First, the organization recognizes it has a problem to solve (Project Planning). Next, the project team investigates and thoroughly understands the problem and the requirements for a solution (Analysis). Once the problem is understood, a solution is specified in detail (Design). The system that solves the problem is then built and installed (Implementation). As long as the system is being used by the organization, it is maintained and enhanced to make sure it continues to provide the intended benefits (Support).

4. **What is the objective of each phase of the SDLC? Describe each objective briefly.**
   See Figure 2-3 page 40
   The **planning** phase involves identifying the scope of the new system to ensure a solution is feasible, develop a schedule, and obtain approval for funding. The **analysis** phase involves understanding and documenting the business needs and processing requirements. The **design** phase involves designing a solution system, both at an architectural level and a detailed level. The **implementation** phase involves building, testing, and installing a working system. The **support** phase involves supporting and maintaining the system over its lifetime.

5. **How is iteration used across phases?**
   Iteration means you do some analysis then some design then some implementation, and then you go back and do more analysis, more design, and more implementation. You have to do some design and some implementation before you can go back and complete the analysis. This is different than overlapping the phases.

6. **What is the difference between a model and a tool?**
   A model is a representation of some important aspect of the real world. A tool provides software support that helps create models or other components required in the project.
7. **What is the difference between a technique and a methodology?**
   A methodology provides comprehensive guidelines to follow for completing every activity in the system development life cycle, including specific models, tools, techniques. A technique is a collection of guidelines that help the analyst complete a system development activity or task.

8. **Which of the two approaches to system development was the earliest?**
   The structured approach was the earlier approach, beginning with structured programming in the 1960s, structured design in the 1970s, and structured analysis in the late 1970s and early 1980s.

9. **Which of the two approaches to system development is the most recent?**
   The object-oriented approach to information systems is the most recent approach, although OO programming languages go back to the 1960s and 1970s.

10. **Which of the traditional approaches focuses on overall strategic systems planning?**
    Information engineering begins with a strategic systems planning activity to define the systems that need to be developed.

11. **Which of the traditional approaches is a more complete methodology?**
    Information engineering includes guidelines for all phases, including planning, analysis, design, and implementation.

12. **What are the three 'constructs' used in structured programming?**
    The three ‘constructs’ used in structured programming are sequence of instructions, selection or decision of one direction or the other, and iteration or repetition of instructions.

13. **What graphical model is used with the structured design technique?**
    The structure chart showing modules organized hierarchically with 'calls' from modules at the top to modules below.

14. **What graphical model is used with the modern structured analysis technique?**
    The data flow diagram (DFD).

15. **What model is the central focus of the information engineering approach?**
    The data model, specifically the entity relationship diagram (ERD). Note that the ERD is also used in the structured analysis technique along with the DFD.

16. **Explain what is meant by a 'waterfall' life cycle model.**
    When a phase in the lifecycle is completed, the results fall down to the next phase and there is no going back.

17. **What concept suggests repeating activities over and over until you achieve your objective?**
    Iteration or iterative development.

18. **What concept suggests completing part of the system and putting it into operation before continuing with the rest of the system?**
19. **What are some features of the Unified Process (UP)?**
UP is an object-oriented system development methodology proposed by Booch, Rumbaugh, and Jacobson of Rational Software (who originated UML). In the UP, the term development process is synonymous with development methodology. It has four life cycle phases, assumes the use of UML models, and is based on accepted best practices of system development.

20. **What are some features of Extreme Programming (XP)?**
XP is a lightweight methodology (simple and focused on helping the developer) that involves iterations, multiple releases, user stories to define requirements, and programmer teams working together only on code.

21. **What are some of the features of SCRUM?**
SCRUM is based on rugby's football idea of a very intense, focused, and energetic activity to capture the ball and move it forward. In software development a SCRUM team is a self-organizing group that responds to a highly changing environment with intense focus and work.

22. **What are visual modelling tools? Why are they used?**
Visual Modelling tools refer to software tools that support the work of the system developer creating models and other system documents and components. Visual Modelling tools are used because system developers, like any other professionals, can benefit from computer support of their work. Because system developers create models, much of the Visual Modelling tools support is for creating models and storing system information in a repository.

**Case Studies**

**Case Study: A 'College Education Completion' Methodology**

Think of completing college as a project, and like any other project, you should follow some sort of 'college education completion' methodology.

1. **What might be the phases of your personal college education completion life cycle?**
2. **What are some of the activities of each phase?**
3. **What are some techniques you use to help complete the activities? What models might you create during the process of completing college? Differentiate between models you create that get you through college from those that help you plan and control the process of completing college.**
4. **What are some of the tools you use to help you complete the models?**

Answers will vary. One approach is to think of it as planning for going to college, analyzing the requirement for what you want out of college, designing a specific program of study, and implementing the schedule and completing the classes for the program of study until college is complete. Another approach is to make up more specific phases: getting in, moving away from home, meeting new friends, deciding on a major, establishing a social life, completing courses, planning for the
job hunt, job hunting, final graduation activities (social and academic), and moving on after graduation. Remember:

**Technique:** a strategy or guidelines for getting an activity or task completed
**Model:** something created as an outcome of a task or an activity
**Tool:** something that is used by the technique to create the model

**Case Study: Factory System Development Project**

The case describes a factory automation project assigned to Sally Jones

1. **Is the proposed system an accounting system? A factory operations system? Or both?**
   Systems today are very integrated – it is difficult to define where one system ends and the other one begins. The factory operations system draws on inventory accounting data when items are manufactured, and then it creates updates to the finished goods inventory accounting data. One system cannot work without the other. An analyst increasingly needs to be aware of the overall business processes of the company. Additionally, an analyst needs to be aware of the overall strategic plan to see how the systems fit together.

2. **Which life cycle variations might be appropriate for Sally to consider using?**
   Sally should use extensive user participation to get as much input from users in the factory as possible. She doesn’t know much about the factory, so she needs to rely on the experts. Prototyping, extensive user interviews, questionnaires, factory worker membership on the development team, and a great deal of user testing are appropriate strategies to use.

3. **Which activities of analysis and of design discussed in this chapter should involve factory workers as well as factory management?**
   All analysis activities should involve factory workers and management. The following design activities should also involve these individuals: prototyping, designing the user interface, and designing and integrating controls.

**Case Study: Rethinking Rocky Mountain Outfitters**

Barbara has lots of choices for managing her project in terms of methodology, tools and techniques.

1. **What if Barbara decides to use the object-oriented (OO) approach for the CSS project? How extensive would the required training be?**
   Training must include an introduction to the OO approach, especially for developers who are steeped in the traditional approach. Models, such as class diagrams, sequence diagrams, statechart diagrams, and package diagrams must be mastered. Only then can OO programming be mastered, but there is much more to the approach than just learning the language. A decision about using the OO approach should be made very early.

2. **Does a Web development project require using the OO approach? What types of projects require an OO approach?**
Web development does not require using the OO approach overall. Some OO tools and models might help define and construct some aspects of the system, but the system can still be traditional in that it is constructed of procedural programs and separate data. In reality, many systems today are a blend of traditional and OO technologies. Perhaps the user interface is created using OO tools and techniques, but the business logic and database are more traditional. Some projects are implemented using a pure OO approach, which is required if the team wants to obtain the benefits of OO in the long run.

3. **What life cycle variations is Barbara considering because she wants to involve users throughout the project? What else might she do to speed development? What else might she consider adapting from Extreme Programming, from the Unified Process, from Agile Modelling, or from Scrum?**

Involving users requires a sociotechnical approach, sometimes called user-centred design. Techniques to help involve users are prototyping, joint application development (JAD) meetings, focus groups, extensive interviews, adding users explicitly to the development team, and so on.

Development can be faster using incremental development, JAD, and prototyping. Ideas from the spiral model can focus on risk (as all project management techniques suggest). XP could include programming in pairs. UP might be useful if she is interested in using an OO approach, but it also includes some good project management ideas useful for any system project. Agile Modelling’s lightweight approach may be more efficient. In other words, a good project manager draws from a variety of approaches and techniques to put together a plan for completing the project.

**Case Study: Focusing on Reliable Pharmaceutical Service**

Management has placed a high priority on developing a Web-based application to connect client facilities with Reliable.

1. **One approach to system development that Reliable might take is to start one large project that uses a waterfall approach to the SDLC to thoroughly plan the project, analyze all requirements in detail, design every component, and then implement the entire system, with all phases completed sequentially. What are some of the risks of taking this approach? What planning and management difficulties would this approach entail?**

Risks of a pure waterfall approach arise from the long time that it would take to follow that approach for the entire project. The scope of the project is large, encompassing every aspect of the business. A pure waterfall approach would take years to complete. During that time, Reliable would continue to lose market share and profitability and might possibly cease to exist as a viable business. In addition, requirements and underlying technology might change in ways that would make the system obsolete or irrelevant between the time analysis was completed and the time the system was finally delivered.

Planning difficulties arise from the large system scope and the interconnection interdependence among components of the final system. To tackle the entire
system at once, many stakeholders would need to participate at one time during planning and analysis. Design and implementation would be very complex due to the large number of interconnected programs and other system components that would need to be designed and implemented in the same time frame. In short, the project would simply be too large to effectively manage, especially given Reliable's relative lack of internal expertise and poor starting position.

2. Another approach to system development might be to start with the first required component and get it working. Later, other projects could be undertaken to work on the other identified capabilities. What are some of the risks of taking this approach? What planning and management difficulties would this approach entail?

The risks of completing the project on a component-by-component or subsystem-by-subsystem basis follow from the tightly integrated nature of the system requirements. Because Reliable wants to modernize all of its systems, dependencies among those systems need to be addressed. For example, the core database, inventory management system, and order fulfillment system must be tightly integrated with order entry and purchasing. All operational systems must be tightly integrated with accounting and financial management systems to enable Reliable managers to effectively manage the business.

Working on the subsystems one at a time makes it more likely that a design decision made early for one subsystem will later turn out to be suboptimal or possibly unworkable for other dependent subsystems. In other words, working on subsystems independently increases the risk of failure for subsystems completed relatively late.

Project planning and management would need to be geared toward minimizing the risks associated with later subsystems. At minimum, planning, analysis, and design activities would have to address the overall architecture into which each subsystem will fit. Thus, some activities will have to be performed 'up front' for all subsystems to ensure that they will work together optimally once all are implemented. Deciding what activities need to be performed early versus later and ensuring the proper flow of knowledge and decisions throughout the project are difficult project management issues.

3. A third approach to system development might be to define one large project that will use an iterative approach to the SDLC. Briefly describe what you would include in each iteration. Describe how incremental development might apply to this project. How would an iterative approach decrease project risks compared with the first approach? How might it decrease risks compared with the second approach? What are some risks the iterative approach might add to the project?

These issues are partly addressed in answers to the previous questions and the answer to the Chapter 1 questions. Early iterations will have to define an application and technical architecture in sufficient detail to ensure that all subsystems will operate correctly once implemented. The database may also need to be fully designed at the start of the project to ensure that it can adequately support all subsystems. Doing so will mitigate the risk of making early decisions that later prove to be suboptimal or unworkable. Working in an interactive fashion will mitigate the risks associated with a pure waterfall approach.
As described in the answer to the previous question, striking the appropriate balance between activities and decisions that must be performed or made now versus later complicates project management. In particular, planning activities are spread throughout the project and day-to-day management becomes more complex since there tends to be more activities occurring in parallel. There is also an increased probability that work completed early in the project will need to be redone or thrown away later in the project due to issues such as changing requirements, shifting technology, or later discovery of incompatibilities in the application or technical architecture.

Key Terms

- **Analysis phase** – The phase of the SDLC whose objective is to understand the user needs and develop requirements.
- **Application** – The portion of the new information system that satisfies the user’s needs in the problem domain.
- **CASE tool** – A computer-aided system engineering tool designed to help a systems analyst complete development tasks.
- **Class diagram** – A graphical model used in the object-oriented approach to show all of the classes of objects in the system.
- **Data flow diagram (DFD)** – A graphical model showing the inputs, processes, storage, and outputs of a system produced in structured analysis.
- **Design phase** – The phase of the SDLC during which the system and programs are designed.
- **Entity-relationship diagram (ERD)** – A graphical model of the data needed by a system, including things about which information is stored and the relationships among them, produced in structured analysis and information engineering.
- **Help desk** – The availability of support staff to assist users with any technical or processing problem associated with an information system.
- **Implementation phase** – The phase of the SDLC during which the new system is programmed and installed.
- **Incremental development** – A development approach that completes part of a system in one or more iterations and puts them into operation for users.
- **Information engineering** – A traditional system development methodology thought to be more rigorous and complete than the structured approach, because of its focus on strategic planning, data modelling, and automated tools.
- **Model** – A representation of an important aspect of the real world.
- **Object** – A thing in the computer system that can respond to messages.
- **Object-oriented analysis (OOA)** – Defining all of the types of objects that do the work in the system and showing what user interactions are required to complete tasks.
- **Object-oriented approach** – An approach to system development that views an information system as a collection of interacting objects that work together to accomplish tasks.
- **Object-oriented design (OOD)** – Defining all of the types of objects necessary to communicate with people and devices in the system, showing how objects interact to complete tasks, and refining the definition of each type of object so it can be implemented with a specific language or environment.
- **Object-oriented programming (OOP)** – Writing statements in a programming
language to define what each type of object does, including the messages that the objects send to each other.

- **Phase** – Related systems development activities, which are grouped into categories of project planning, analysis, design, implementation, and support.
- **Planning phase** – The initial phase of the SDLC, whose objective is to identify the scope of the new system and plan the project.
- **Problem domain** – The area of the users’ business for which a system is being developed.
- **Project** – A planned undertaking that has a beginning, and an end, and that produces a desired result or product.
- **Repository** – A database that stores information about the system in a CASE tool, including models, descriptions, and references that link the various models together.
- **Socio technical systems** – Information systems that include both social and technical subsystems designed to work well together.
- **Structure chart** – A graphical model showing the hierarchy of program modules produced by the structured design technique.
- **Structured analysis** – A technique that helps the developer define what the system needs to do (the processing requirements), what data the system needs to store and use (data requirements), what inputs and outputs are needed, and how the functions work together as a whole to accomplish tasks.
- **Structured approach** – System development using structured analysis, structured design, and structured programming techniques.
- **Structured design** – A technique providing guidelines for deciding what the set of programs in an IS should be, what each program should accomplish, and how the programs should be organized into a hierarchy.
- **Structured program** – A program or program module that has one beginning and one ending, and for which each step in the program execution consists of sequence, decision, or repetition constructs.
- **Support phase** – The phase of the SDLC whose objective is to keep the system running productively after it is installed.
- **Systems development life cycle (SDLC)** – A project management framework organized into phases and activities.
- **Systems development methodology** – Comprehensive guidelines to follow for completing every activity in the systems development life cycle, including specific models, tools, and techniques.
- **Technique** – A collection of guidelines that help an analyst complete a system development activity or task.
- **Tool** – Software support that helps create models or other components required in the project.
- **Top-down programming** – Dividing more complex programs into a hierarchy of program modules.
- **Unified Process (UP)** – An object-oriented system development methodology offered by Rational Software.
- **Waterfall approach** – An approach to executing and SDLC in which one phase leads (falls) sequentially into the next phase.
Chapter 3 – Investigating System Requirements

This chapter corresponds with chapter 4 of the text book

Chapter Overview

During the fact-finding and investigation activities a systems analyst learns the details of the business processes and daily operations. In fact, a major objective of systems analysts during these activities is to try to become as knowledgeable in how the business operates as the users. Why become an expert? Because only then can the project team ensure that the system meets the needs of the business. The primary contribution of systems analysts is to be able to add value to the enterprise through advances in information technology. This requires a thorough understanding of the business coupled with a high level of problem solving and technical skills.

This chapter provides an overview of the activities of the analysis phase. The different types of systems requirements are defined and explored. The techniques that analysts use to learn about the business processes and to gather information using traditional and newer methods are explained. This chapter ends with a discussion of the need for quality control of the information gathered and the models built during the activities of the analysis phase.

Learning Objectives

After studying this chapter, you should be able to:

• Describe the activities of the systems analysis.
• Explain the difference between functional and non-functional system requirements.
• Describe three types of models and the reasons for creating models.
• Identify and understand the different types of users who will be involved in investigating system requirements.
• Describe the kind of information that is required to develop system requirements.
• Determine system requirements through review of documentation, interviews, observation, prototypes, questionnaires, vendor research, and joint application design sessions.
• Discuss the need for validation of system requirements to ensure accuracy and completeness and the use of a structured walkthrough.

Notes on Chapter Opening Case and Case Studies

Mountain States Motor Sports: This opening case highlights the benefits of collecting information and defining requirements and design concepts using joint application design (JAD), which is discussed in this chapter. In the intensive JAD meeting, many key project aspects are discussed and refined. Scope is defined. Decisions are made. In this case, one firm, MSMS, is more experienced with systems, but the other firm, ACC, is not. Their
businesses differ—one is a retailer, and the other is a manufacturer. They enter a business relationship and need to better understand each other and their joint needs. This case emphasizes the importance of information gathering, requirements, priorities, prototypes, alternatives, and recommendations.

**John and Jacob, Inc. On-Line Trading System:** This case describes the process of identifying the people that should be included in system requirements development. The project manager has some uncertainties and issues that need to be resolved in order to be inclusive. Most of the issues concern how to select users, customers, and other stakeholders to assist in defining systems requirements.

**Rethinking Rocky Mountain Outfitters:** The project manager in this case has several senior executives involved as stakeholders on the project and also several department managers and senior users to provide detailed processing requirements. This manager has carefully organized the process to be followed and has provided early planning to communicate with users, manage user expectations, and manage the project scope.

**Focusing on Reliable Pharmaceutical Service:** This case describes requirements definition when the system users are primarily outside the organization. Compared to the other cases in this chapter, the questions are open-ended and are asked earlier in the planning process. A key point is that suppliers and clients may not want to share information on their internal operations.

### Additional Notes and Quick Quizzes

#### Analysis Activities in More Detail

This chapter describes the activities done as part of analysis. Different life cycle methodologies recommend using different techniques for completing these activities, but all methodologies include them in some way. Recall that predictive approaches to the SDLC use a modified waterfall approach to the life cycle, so the analysis phase activities are done earlier in the project. With an adaptive approach to the SDLC, these activities are done in several individual iterations.

Analysis activities of the SDLC involve defining in great detail what the information system needs to accomplish to provide the organization with the desired benefits. There are six activities that must be completed, oftentimes in parallel:

- Gather information.
- Define system requirements.
- Prioritize requirements.
- Prototype for feasibility and discovery.
- Generate and evaluate alternatives.
- Review recommendations with management.

The objective of the analysis phase is to understand the user needs and document those needs in the form of system requirements.

#### Gather Information
The analyst gathers information from people who will use the system, from management, from strategic plans, from existing documentation, and so on. The analyst needs to become an expert in the business area the system will support. The analyst gathers information on current and future users, locations where work occurs, system interfaces, and software packages that might be used to satisfy system requirements. The key question to be answered is, “Do we have all of the information (and insight) we need to define what the system must do?”

**Define System Requirements**
The information gathered is used to document both functional and non-functional requirements. Functional requirements define what the system is required to do. The functional requirements are documented by creating system models, and much of Part 2 of this text involves teaching students how to create the requirements models (see chapters 5, 6, and 7 in particular). Non-functional requirements include system performance expectations, usability, and supportability.

Requirements models are a type of **logical model**, a model that shows what the system is required to do without committing to any one technology. **Physical models** show how the system will actually be implemented. Examples of logical models depend on what technique is being used for analysis. Modern structured analysis uses data flow diagrams (DFDs) and entity-relationship diagrams (ERD). Information engineering uses process dependency diagrams and entity-relationship diagrams. The object-oriented approach uses class diagrams and use case diagrams. The key question to be answered is, “What exactly do we need the system to do?”

**Prioritize Requirements**
Once the system requirements are well understood and detailed models are completed, it is important to establish the priorities for the new system, because not all requirements are crucial. Resources to build systems that meet requirements are always limited. There is a tendency toward scope creep, where the system scope increases to include more than originally intended. The key question to be answered is, “What are the most important things the system must do?”

**Prototype for Feasibility and Discovery**
Creating prototypes of parts of the new system can be valuable during systems analysis, but the purpose is not to prototype design ideas. If the system involves new technology, it is important for the team to be sure the technology works as expected. This is called prototyping for feasibility. Prototyping might also be done to help the users visualize possibilities with the new technology, helping to uncover requirements the users might not otherwise have thought of. This is called prototyping for discovery. The key questions to be answered when prototyping for feasibility and discovery are, “Have we proven that the technology proposed can do what we think it will do?” and “Have we built some prototypes to ensure the users fully understand the potential of the new technology?”

**Generate and Evaluate Alternatives**
Many alternatives exist for the final design of the system. It is important to define and then evaluate all of the possibilities. Once the requirements are prioritized, the analysis can propose alternatives by including only the high priority requirements rather than all of the requirements. Different types of technology proposed can lead to alternatives. The analyst can also propose alternatives based on the approach used to build the system (including using a package instead of custom development). A high-level physical model of each
alternative is created, and then all of the alternatives are evaluated. Each has its own costs, benefits, and other feasibility factors. The key question to be answered is, “What is the best way to create the system?”

**Review Recommendations with Management**

All of the preceding activities are done in parallel. Reviewing recommendations with management is usually done when the other analysis activities are complete. The questions the analyst must answer for management are: Should the project continue at all? If so, which alternative is best? Given that recommendation, what are the revised budget and schedule for the project? Making the recommendation in this activity is a major management checkpoint and an opportunity for management to decide what should happen next with this project. Appendix D (online) explains in more detail the techniques for making presentations. The key question to be answered by this activity is, “Should we continue with the design and implement the system we propose?”

**Quick Quiz: The Analysis Phase in More Detail**

1. What is the difference between a logical model and a physical model?

   ANSWER: A logical model shows what the system is required to do without committing to any one technology. A physical model shows how the system will actually be implemented with specific technology.

**System Requirements**

When we speak of system requirements, we refer to the capabilities and constraints that the new system must address. You will remember that one of the activities during the planning phase was to identify the system scope and to identify a set of system capabilities. During the analysis phase, we define those capabilities in greater detail by describing them as system requirements. Generally, we divide system requirements into two categories, functional requirements and non-functional requirements.

**Functional requirements** are the activities that the system must perform—that is, the business uses to which the system will be applied. Functional requirements are derived directly from the capabilities identified in the planning phase. For example, if we are developing a payroll system, the required business uses might include the following functions: write paychecks, calculate commission amounts, calculate payroll taxes, maintain employee dependent information, and report year-end tax deductions to the IRS. These are the functional requirements of the new system.

**Non-functional requirements** are all the operational objectives related to the environment, hardware, and software of the organization. Examples include: must run in a client/server environment with Windows NT, must have one-half second response time on all screens, and must be able to support 100 terminals at once (with same response time). Specific types of non-functional requirements include technical requirements, performance requirements, usability requirements, reliability requirements, and security requirements.

- **Technical requirements** describe operational characteristics related to the environment, hardware, and software of the organization.
• **Performance requirements** describe operational characteristics related to measures of workload, such as throughput and response time.

• **Usability requirements** describe operational characteristics related to users, such as the user interface, related work procedures, online help, and documentation.

• **Reliability requirements** describe the dependability of a system—how often a system exhibits behaviours such as service outages and incorrect processing.

• **Security requirements** describe which users can perform what system functions under what conditions.

Some organizations clearly define a business analyst as someone who focuses on functional requirements and business users, and a technical analyst as someone who focuses on technical requirements and operational objectives.

**Quick Quiz: System Requirements**

1) What is the difference between functional requirements and non-functional requirements?

**ANSWER:** Functional requirements define the functions or user services that will be supported by the system. For example, support of business processes and reports are included in functional requirements. Non-functional requirements are often called technical specifications or operating specifications and include such things as response time or operating platform and environment. Non-functional requirements include technical requirements, performance requirements, usability requirements, reliability requirements, and security requirements.

2) How are functional requirements and non-functional requirements documented?

**ANSWER:** Functional requirements are documented in graphical and textual models (see Chapters 5, 6, and 7) and non-functional requirements are typically documented in narrative descriptions that accompany the models.

**Models and Modelling**

Recall from Chapter 2 that a model is a representation of some aspect of the system being built, and the analyst needs to create a variety of models to represent all aspects of the system. Some models are high-level overviews; some are detailed views; some focus on one aspect of the system, such as inputs, processes, outputs, or data storage; some show how the other models fit together; and some show the same problem from a different perspective.

**The Purpose of Models**
Models and the process of creating models are important to system development for the following reasons:

• Learning from the modelling process.
• Reducing complexity by abstraction.
• Remembering all of the details.
• Communicating with other development team members.
• Communicating with a variety of users and stakeholders.
• Documenting what was done for future maintenance/enhancement.

Types of Models
The following types of models are used:

• Mathematical models are a series of formulas that describe the technical aspects of a system.
• Descriptive models are narrative memos, reports, or lists that describe some aspect of a system.
• Graphical models are diagrams or schematic representations of some aspect of a system.

Models are used extensively in a wide variety of technical occupations beyond information technology. Architecture, scientific research, and engineering are some of the fields that use modelling.

Overview of Models Used in Analysis and Design
All three types of models are used during analysis and design. The requirements models created during the systems analysis phase are logical models because they define in detail what is required without committing to one specific technology. Figure 4-5 on page 128 shows examples of commonly used models, including event lists, data flow diagrams (DFD), entity-relationship diagrams (ERD), data flow definitions, data element definitions, process descriptions, location diagrams, class diagrams, use case diagrams, sequence diagrams, communication diagrams, and state machine diagrams.

Many models are also created during the design phase. These physical models indicate how the system component is implemented using a specific technology. Some are high-level models and some are very detailed. Examples of physical models, include screen layouts, report layouts, system flowcharts, structure charts, database schema, network diagrams, and deployment diagrams. Note that with the object-oriented approach, some models are used in both analysis and design (including the class diagram, sequence diagram, communication diagram, and state machine diagram).

Carefully go over the models created during the analysis phase and the models created (or modified) during the design phase to set the stage for Chapter 6: “The Traditional Approach to Requirements”.

Quick Quiz: Models and Modelling

1) What are some of the reasons for creating models during systems development?

ANSWER: Learning from the modelling process, reducing complexity by abstraction, remembering all of the details, communication with team members and stakeholders, documentation for future enhancements and maintenance.

2) What are three types of models?
ANSWER: Mathematical models, descriptive models, and graphical models.

Stakeholders — The Source of System Requirements

Your primary source of information for functional system requirements is from the various stakeholders of the new system. By stakeholders, we mean all people who have an interest in the successful implementation of the system. Stakeholders are generally categorized into three groups: users (those who actually use the system on a daily basis), clients (those who pay for and own the system), and technical staff (those who must ensure that the system operates in the computing environment of the organization). Figure 4-6 illustrates the various kinds of stakeholders who have an interest in a new system.

Users as Stakeholders
User roles (types of system users) should be identified in two dimensions: horizontally and vertically. By horizontally, we mean that the analyst must look for information flow across business departments or functions. For example, a new inventory system may affect receiving, warehousing, sales, and manufacturing. You must be sure that individuals from each of these departments have the opportunity to describe their requirements.

By vertically, we mean the information needs of clerical staff, middle management, and senior executives. Each of these users will have different information requests from the system that must be included in the design.

Business Users
Business users are the people who use the system to perform the day-to-day operations of an organization. We often call these operations transactions. A transaction is a single occurrence of a piece of work or an activity done in an organization, such as “enter an order.”

Information Users
Information users need information from the system, either detailed or summary. Often they will receive reports or be allowed to run queries against the database.

Management Users
Managers are responsible for seeing that the company is performing its daily procedures efficiently and effectively. Consequently, they need statistics and summary information from a system.

Executive Users
The top executives of an organization are interested in strategic issues, as well as the daily issues just described.

External Users
More and more systems today allow customers and external users to have access to the data or to directly enter transactions. To ensure that the system is “user friendly” to external users requires their involvement in the system development project.

Client Stakeholders
In Chapter 3 we defined the client as the person or group who is providing the funding for
the project. In many cases, the client is the same group as the executive users. However, clients may also be a separate group of people, such as a board of trustees or executives in a parent company.

**Technical Stakeholders**

The technical staff includes the people who establish and maintain the computing environment of the organization. These people provide guidance in such areas as programming languages, computer platforms, and other equipment.

**Stakeholder Determination**

Because stakeholders are the primary source of information for system requirements, the selection and participation of stakeholders strongly determines the probability of project success or failure.

There is also a direct trade-off. More stakeholders can provide additional information, but each additional person adds to the effort to gather and analyze that information. When selecting stakeholders there is a balance between quality and quantity.

**The Stakeholders for Rocky Mountain Outfitters**

This material describes the process that was used to identify stakeholders for the proposed customer support system for Rocky Mountain Outfitters. The set of system requirements is incomplete if all stakeholders are not consulted as information is being gathered.

**Quick Quiz: Stakeholders – The Source of System Requirements**

1) What categories of stakeholders should be included in fact finding?

**ANSWER:** The type and scope of the system will dictate which stakeholders are required. User stakeholders are always required. Client stakeholders are sometimes needed. Technical stakeholders sometimes also need to be involved.

2) What is meant by vertical and horizontal dimensions when determining stakeholders?

**ANSWER:** Horizontal dimension means that various departments in a company may be affected by a system. For example, a marketing system may have information that might be useful to the production plant. Therefore, both marketing users and production users need to be interviewed.

Vertical dimension means that within each department there are clerical users and middle management users and that both should be included. The vertical dimension also goes higher, up to top management and executives who may have an interest in the information provided by the system.

**Techniques for Information Gathering**

The objective of the analysis phase of system development is to understand the business
functions and develop the system requirements. The question that always arises is whether to study and document the existing system, or whether to only document the requirements of the new system.

The objective of analysis has not changed; however, the approach to developing system requirements has improved. It is still critically important to have a complete, correct set of system requirements, but in today’s fast-paced world, there is neither the time nor the money to develop documentation for old systems. Today we use a more accelerated approach by balancing the review of current business functions with the new system requirements. As shown in Figure 4-8, the focus of analysis activities today is to develop a set of logical system requirements for the new system. In other words, the objective is to immediately develop the logical model of the new system. Review of the current system is done as much as necessary to understand the business needs but not to define the specific processes.

**Question Themes**

The first questions that new systems analysts ask are, “What kind of information do I need to collect?” and “What is a requirement?” Basically, you want to obtain information that will enable you to build the logical model of the new business system (see Chapters 5 and 6 for information about modelling). As shown in Figure 4-9, the following three major themes should guide you as you pursue your investigation:

- What are the business processes and operations? (What do you do?)
- How should those operations be performed? (How do you do it? What steps are followed?)
- What information is needed to perform those operations? (What information do you use? What forms or reports do you use?)

**What Are the Business Processes?**
In the first question—What do you do?—the focus is on understanding the business functions. In most cases, the users will provide answers in terms of the current system. As an analyst, you must carefully discern which of those functions are fundamental business functions, which will remain, and which may possibly be eliminated with an improved system.

**How is the Business Process Performed?**
The second question—How can it be done?—moves the discussion from the current system to the new system. The focus is on how the new system should support the function rather than on how it is performed under the existing system. Thus, the first two questions go hand-in-hand to discover the need and begin the definition of the system requirements in terms of the new system.

**What Information is Required?**
The final question—What information is needed?—defines specific information that the new system must provide. The answers to the second and third questions form the basis for the definition of the system requirements.

If you focus your investigation around these three themes, you will be able to ask intelligent, meaningful questions in your investigation. Later, as you learn about models, you will be able to formulate additional meaningful detailed questions to ask.
In the next sections, we present the various methods of information gathering to identify requirements. All these methods have been proven to be effective, although some are more efficient than others are. In most cases, we use a combination of methods to increase both our effectiveness and our efficiency. Together, they provide a comprehensive fact-finding method that is widely used for the development of large, complex systems.

**Review Existing Reports, Forms, and Procedure Descriptions**

The review of existing documents and documentation serves two purposes. First, it is a good way to get a preliminary understanding of the processes. Often new systems analysts will not know much about the industry or the specific application that they are studying. A preliminary review of existing documentation will bring them up to speed fairly rapidly. Second, forms and reports are useful during interviews to help users understand the business processes and the information requirements. Discussions of the data content of the forms and reports as well as how they are initiated and used will help the analyst probe into processing details.

**Conduct Interviews and Discussions with Users**

Interviewing stakeholders is by far the most effective way to understand business functions and business rules. It is also the most time consuming and expensive. Using this technique, members of the project team (systems analysts) meet with individuals or groups of users. A list of detailed questions is prepared and discussion continues until all of the processing requirements are understood and documented by the project team. Obviously, this may take some time, so it usually requires multiple sessions with each of the users or user groups.

To gather information effectively, analysts need to do the following: (1) prepare the interview material carefully, (2) conduct the interview, and (3) follow up the interview. Figure 4-9 on page 129 is a sample checklist for preparing for and conducting an interview.

**Preparing for the Interview**

Every successful interview requires preparation. The first and most important step in preparing for an interview is to establish the objective of the interview. In other words, what do you want to accomplish in this interview? Write down the objective so that it is firmly established in your mind. The second step is to determine which users should be involved in the interview. Frequently, the first two steps are intertwined and must be done together. The objective and the participants drive everything else in the interview.

The third step is to prepare detailed questions to be used in the interview. Write down a list of specific questions and prepare notes based on the forms or reports received earlier. It is generally recommended to prepare a list of questions that are consistent with the objective of the interview. Both open-ended and closed-ended questions are appropriate. Open-ended questions, such as “How do you do this function?” encourage discussion and explanation. Close-ended questions, such as “How many forms a day do you process?” are used to get specific facts. Open-ended questions often help get the discussion started and encourage the user to explain all the details of the business process.
The last step is to make the final interview arrangements and to communicate those arrangements to all participants.

**Conducting the Interview**

New systems analysts are usually quite nervous about conducting interviews. However, in most cases, the users are excited about getting a better system to help them do their jobs. Practicing good manners usually ensures that the interview will go well. Here are a few guidelines:

- Dress appropriately.
- Arrive on time.
- Limit the time of the interview.
- Look for exception and error conditions.
- Probe for details.
- Take careful notes.

**Following up the Interview**

Follow-up is an important part of each interview. The first task is to absorb, understand, and document the information that was obtained. Analysts often document the details of the interview by constructing models of the business processes. Review your findings with the other project members in the interview and document the results (that is, build the models) within a day or two to avoid forgetting important details.

**Observe and Document Business Processes**

The old adage that a picture is worth a thousand words also applies to systems analysis. Early in the investigation activities, time should be scheduled to observe the business procedures that the new system will support. There is no better way to learn how users use a system and what information they need than by observing first-hand the work being done.

**Observing**

You can observe the work in several ways, from conducting a quick walk-through of the office or plant to doing the work yourself. A quick walk-through gives a general understanding of the layout of the office, the need and use of computer equipment, and the general workflow. Scheduling several hours to observe a user doing his or her job provides an understanding of how the computer system is used and how business functions are carried out. By learning and performing the user’s job, you can better understand the difficulties of learning new procedures, the importance of a system that is easy to use, and the stumbling blocks and bottlenecks of existing procedures and information sources.

**Documenting with Activity Diagrams**

A *workflow* is a sequence of processing steps necessary to completely handle one business transaction or customer request. As an analyst, you may try to depend only on your memory to remember and understand the workflow (that is a bad idea), you may write it down in a long description, or you may document it with a diagram. The advantages of a simple diagram are that you can visualize it better, and you can review it with the users to make sure it is correct. In fact, the act of developing the diagram will often help you discover parts of the business process that you do not understand or that are missing. One
other benefit of using diagrams is that they become a powerful communication mechanism between the project team and the users.

Methodologies commonly used to model workflows include flowcharts, data flow diagrams, and activity diagrams. An activity diagram is a workflow diagram that describes the various user (or system) activities, the person who does each activity, and the sequential flow of these activities. The activity diagram is one of the Unified Model Language (UML) diagrams associated with the object-oriented approach. It is generally replacing the data flow diagram for many traditional analysts. Figure 4-14 shows the symbols used in an activity diagram.

The first step to create an activity diagram is to identify the agents and make the appropriate swim lanes. Next, identify the various steps of the workflow and make appropriate ovals for these activities. Connect the activity ovals with arrows to show the flow of the workflow. Use a decision symbol to represent an either/or situation and synchronization bars to document parallel concurrent paths (both paths are taken).

Swim lane: A rectangular area on an activity diagram representing the activities done by a single agent.

Synchronization bar: A symbol in an activity diagram to control the splitting or uniting of sequential paths.

**Activity Diagrams**

Regardless of methodology, activity diagrams are an important part of the business process documentation. Once the stakeholders have agreed on the workflow, then everyone involved has a shared understanding of the sequence of events for this business process. Activity diagrams are a foundation for the transition from analysis to design.

**Build Prototypes**

A prototype is an initial working model of a larger, more complex entity. Prototypes are used for many different purposes, and the following names are used to describe these uses: throwaway prototypes, discovery prototypes, design prototypes, and evolving prototypes. As discussed in Chapter 2, the purpose of a discovery prototype used during analysis is to have a working model to test a concept or verify an approach.

Mock-up: An example of a final product. It is for viewing only; it is not executable.

**Distribute and Collect Questionnaires**

Questionnaires have a limited and specific use in information gathering. The benefit of a questionnaire is that it permits the collection of information from a large number of stakeholders. The stakeholders can be widely distributed in geographical location and still participate in the requirements definition.

Frequently, a questionnaire can be used to obtain preliminary insight on the information needs of stakeholders. This information can then be used to help determine the areas that need further research with document reviews, interviews, and observation. Questionnaires are also helpful to answer quantitative questions, such as: “How many orders do you enter a day?” and “How long does it take to enter one order?” Finally, questionnaires can be used to determine the users’ opinions about various aspect of the system. Questionnaires
can answer questions, such as: “On a scale of 1 to 7, how important is it to be able to access a customer’s past purchase history?”

**Closed-ended questions:** Questions that have a simple definitive answer.

**Open-ended questions:** Questions that require discussion and do not necessarily have a simple short answer.

**Conduct Joint Application Design Sessions**

**Joint application design (JAD)** is a technique used to expedite the investigation of system requirements. The normal interview and discussion approach, as explained earlier, requires a substantial amount of time. The analyst meets with the users, documents the discussion by writing notes and building models, and reviews and revises the models. Unresolved issues are placed on an open items list and may require several additional meetings and reviews to be finalized. This process can extend from several weeks to several months, depending on the size of the system and the availability of user and project team resources.

The objective of JAD is to compress all of these activities into a shorter series of sessions with users and project team members. An individual JAD session may last from a single day to a week. During the session, all of the fact-finding, model-building, policy-making, and verification activities are completed for a particular aspect of the system. If the system is small, the entire analysis may be completed during the JAD session. The critical factor in a successful JAD session is to have all of the important stakeholders present and available to contribute and make decisions. The actual participants will vary depending on the objective of the specific JAD session. Those who are involved may include the following:

- JAD session leader
- Users
- Technical staff
- Project team members

**Joint application design:** A technique to define requirements or design a system in a single session by having all necessary people participate together.

**Group support systems (GSSs),** which also run on the network of computers, allow all participants to post comments (anonymously if desired) in a common working chat room. This approach helps participants who may be shy in a group discussion to become more active and contribute to the group decisions. GSSs also enable the team to store final requirements as decisions are made. Normally JAD sessions are conducted with everyone in the same room. However, GSSs on wider networks provide the opportunity for virtual meetings with participants at geographically dispersed locations.

**JAD Facilitation**

The strength of the JAD session leader can make or break the quality of the JAD sessions. If an organization does not have an experienced facilitator, either the planned session leaders have to be trained, or an outside facilitator needs to be hired to lead the sessions.
Facilitation is a specialized skill and being a good analyst does not guarantee that the person will be a good facilitator.

Research Vendor Solutions

Many of the problems and opportunities that companies want to address with new information systems have already been solved by other companies. In many instances, consulting firms have experience with the same problems, and sometimes software firms have packaged solutions for a particular business need. Directories, such as Data Sources, list thousands of hardware, software, consulting, and solution developers.

There are three positive contributions and one danger in exploring existing solutions. First, researching alternative solutions will frequently help users generate new ideas of how to better perform their business functions. Second, some of these solutions are in fact the "best of the breed." That is, they demonstrate best practices in the industry. Third, it is often cheaper, and less risky, to buy a solution rather than to build it.

The first difficulty in researching vendor alternatives is simply to find out who has solutions that fit the business need. Many of the large software and hardware companies, such as Oracle, IBM, Microsoft and Computer Associates have specific solution systems. There are also directories of system solutions—of software, hardware, and developer companies.

The danger in this process is that sometimes the users, or the systems analysts, want to buy one of the alternative solutions too early in the process before the company’s needs are thoroughly investigated.

Quick Quiz: Techniques for Information Gathering

1) List and describe the three fact-finding themes.

ANSWER: What are the business operations and processes? The purpose of the first theme is to identify the various business processes that must be supported in the new system. The standard question to users is, “What do you do?” or “What are your job responsibilities?”

How should those operations be performed? The purpose of the second theme is to understand the details of the business process in the context of a new system. The focus is not necessarily on how the operation is done now, but on how should it be done.

What information is needed to perform the operation? The purpose of the third theme is directly tied to the development of the new system, and what information it must maintain and provide.

2) What is the purpose of an activity diagram?

ANSWER: An activity diagram is used to document the detailed steps of a business process or workflow. A workflow is a sequence of processing steps necessary to completely handle one business transaction or customer request.

3) What is the difference between a JAD facility and a GSS?
ANSWER: JAD sessions are conducted with everyone in the same room (JAD facility). GSS systems on wider networks provide the opportunity for virtual meetings with participants at geographically dispersed locations.

Validating the Requirements

A **structured walkthrough**, sometimes called a **walkthrough**, is a review of the findings from your investigation and of the models built based on those findings. We call it a “structured” walkthrough because analysts have formalized the review process into a set procedure. The objective of a structured walkthrough is to find errors and problems. The fundamental concept is one of documenting the requirements as you understand them, and then reviewing them for any errors, omissions, inconsistencies, or problems. A review of the findings can be done informally with colleagues on the project team, but a structured walkthrough, must be more formal.

What and When

The first item to be reviewed during a structured walkthrough is the documentation that was developed as part of the analysis phase. The documentation can be a narrative describing a process, a flowchart showing a workflow, or a model diagram documenting an entire procedure. Normally, it is better to conduct several smaller walkthroughs that review three to six pages of documentation than to cover thirty pages of details. Any written work that is a fairly independent package can be reviewed in a walkthrough.

Who

The two main parties involved in walkthroughs are the person or persons who need their work reviewed and the group who reviews it. For verification—that is, internal consistency and correctness—it is best to have other experienced analysts, who are used to looking for inconsistencies and problems, involved in the walkthrough. For validation—that is, that the system satisfies all the needs of the various stakeholders—the appropriate stakeholders should be involved.

How

As with an interview, a structured walkthrough requires preparation, execution, and follow-up.

**Preparation**
The analyst whose work is being reviewed gets the material ready for review. Next, the analyst identifies the appropriate participants and provides them with copies of the material to be reviewed. Finally, he or she schedules a time and place for the walkthrough and notifies all participants.

**Execution**
During the walkthrough, the analyst presents the material point by point. If the material is a diagram or flowchart, the analyst walks through the flow, explaining each component. The reviewers look for inconsistencies or problems and point them out. A librarian (a helper for the presenter) documents the comments made by the reviewers.
Follow-up
Follow-up consists of making the required corrections. If the reviewed material had major errors and problems, an additional walkthrough may be necessary. Otherwise, the corrections are made, and the project continues to the next activities.

Quick Quiz: Validating the Requirements

What is the objective of a structured walkthrough?

ANSWER: A structured walkthrough is a formalized review of the findings and models from your investigation to find errors and problems.

Solutions to End of Chapter Material

Review Questions

1. List the six activities of systems analysis.
   - Gather information
   - Define system requirements
   - Prioritize requirements
   - Prototype for feasibility and discovery
   - Generate and evaluate alternatives
   - Review recommendations with management

2. What are three types of models?
   - Mathematical models
   - Descriptive models
   - Graphical models

3. What is the difference between functional requirements and non-functional requirements?
   Functional requirements define the functions or user services that will be supported by the system. Such things as support of business processes and reports are included in functional requirements.

   Non-functional requirements are often called technical specifications or operating specifications and include such things as performance, usability, reliability, and security requirements.

4. Explain the use of a discovery prototype and an evolutionary prototype.
   A discovery prototype is used to help understand user requirements. Often the users are not sure what capabilities are needed in a system, or how certain functions should work. A discovery prototype can be built to help define and refine the business activity as a combined manual/computer process. Usually discovery prototypes are considered throwaway prototypes.

   An evolutionary prototype, on the other hand, is one that is expected to grow and evolve until it eventually becomes the final working system. It may be used for discovery, but its main objective is to become the final system.
5. **List and describe the three fact-finding themes.**

What are the business operations and processes? The purpose of the first theme is to identify the various business processes that must be supported in the new system. The standard question to users is 'What do you do?' or 'What are your job responsibilities?'

How should those operations be performed? The purpose of the second theme is to understand the details of the business process in the context of a new system. The focus is not necessarily on how the operation is done now, but how should it be done.

What information is needed to perform the operation? The purpose of the third theme is directly tied to the development of the new system, and what information it must maintain and provide.

6. **What is the objective of a structured walkthrough?**

The objective of a structured walkthrough is to find errors in a piece of work. In other words, it is a quality control activity to improve the quality of a piece of work such as a data flow diagram or data model.

The objective is not to be a performance review or an evaluation of an employee.

7. **Explain the steps in preparing for an interview session.**

See Figure 4-11.

- Establish an objective for the interview session.
- Determine the persons who should be involved in the interview, both users and team members.
- Develop a list of questions and issues to be discussed.
- Review any related documentation. Existing reports or procedure manuals are a good source of questions and issues to discuss.
- Set time and location.
- Inform all participants of objective, time, and location. It is especially important that the users be informed of the objectives so that they can come to the meeting prepared.

8. **What are the benefits of doing vendor research during information-gathering activities?**

Doing vendor research, both for vendors of packaged solutions and consulting firms that develop solutions, allows the project team the opportunity to see what other companies have done. It is less expensive to buy a solution than to build one, if a viable solution is available. It helps the project team see other solutions that may be state-of-the-art. It may widen the horizon of possible approaches to provide a competitive advantage to the enterprise.

9. **What categories of stakeholders should you include in fact finding?**

Obviously the type and scope of the system will dictate which stakeholders are required. However, user stakeholders are always required. Client stakeholders are sometimes needed. Technical stakeholders sometimes also need to be involved.
10. **What is meant by vertical and horizontal dimensions when determining users to involve?**

Horizontal dimension means that various departments in a company may be affected by a system. For example, a marketing system may have information that might be useful to the production plant. Therefore, not just marketing users need to be interviewed, but production users also.

Vertical dimension means that within each department there are clerical users and middle management users. Both should be included. The vertical dimension also goes higher, up to top management and executives who may have an interest in the information provided by the system.

11. **What is JAD? When is it used?**

JAD stands for Joint Application Design. It is a technique to speed up the fact finding or design process. It can be used when the requirements for a system are fairly standard and can be defined in detail with only a few working sessions of the right stakeholders. The approach is to have meeting sessions with all the necessary stakeholders so that requirements and final decisions can be made quickly.

12. **What is BPR? What does it have to do with systems analysis?**

BPR stands for Business Process Reengineering. Business systems are always a combination of manual and automated processes. A change in the level or approach of automation will have an important impact on the total business process. Increasing levels of automation have also provided opportunities to shift many manual processes to automated processes. Additional information and services is also possible with increased automation. Thus information technology has provided opportunities for companies to reengineer their business processes.

13. **What technique is used to validate user requirements?**

Since user requirements cannot be 'tested' in the normal programming sense of the word, then another method must be used to validate the requirements. A widely used method is with a structured walkthrough. When a piece of work is completed that describes some user requirements, either in the form of a diagram or textual description, then a walkthrough is conducted. In the walkthrough the work is reviewed with the objective of finding missing pieces, checking for accuracy, and verifying that the diagram is internally consistent.

14. **Describe the open-items list and explain why it is important.**

The open-items list is a list of issues that have not been resolved. It is important so that requirements will be complete and those undefined specifications are not forgotten.

15. **What do correct, complete, and comprehensive mean with regard to systems analysis?**

Sets of requirements that are developed during systems analysis need to be correct, complete, and comprehensive. Correct means that there are no errors in the requirements specifications. Complete means that each requirement is completely specified with adequate detail. Comprehensive means that the requirements cover all possible processing needs.
16. List and describe the seven information-gathering techniques.
   - Review existing reports, forms, and procedure descriptions
   - Conduct interviews and discussion with users
   - Observe and document business processes
   - Build prototypes
   - Distribute and collect questionnaires to stakeholders
   - Conduct joint application design (JAD) sessions
   - Research vendor solutions

17. What is the purpose of a workflow diagram?
   A workflow diagram is used to document the detail steps of a business process.

18. Draw and explain the symbols used on a workflow diagram.
   Refer to Figure 4-14.

Example of an activity diagram:

Develop an activity diagram based on the following narrative. Note any ambiguities or questions that you have as you develop the model. If you need to make assumptions, also note them.

The purchasing department handles purchase requests from other departments in the company. People in the company who initiate the original purchase request are the “customers” of the purchasing department. A case worker within the purchasing department receives that request and monitors it until it is ordered and received. Case workers process requests for the purchase of products under $1,500, write a purchase order, and then send it to the approved vendor. Purchase requests over $1,500 must first be sent out for bid from the vendor that supplies the product. When the bids return, the case worker selects one bid. Then, he or she writes a purchase order and sends it to the vendor.
Case Studies

Case Study: Jacob and Jacob, Inc. On-Line Trading System

1. What is the best method for Edward to involve the brokers (users) in development of the new on-line trading system? Should he use a questionnaire? Should he interview the brokers in each of the company’s 30 offices, or would one or two brokers representing the entire group be better? How can Edward ensure that the information about requirements is complete, yet not lose too much time doing so?

This situation is probably a good one to start with a questionnaire. The users are dispersed and probably diverse. The questionnaire should focus on needs and preferences and can also help to establish which topics need further refinement.

It will probably cost too much to interview the brokers in all of the offices. One way to select offices is to develop a set of characteristics that distinguish the various offices. Then to select a representative office from each set of similar offices. As interviews progress, if the answers to questions are very similar, then it may be possible to abbreviate or shorten the visits to later offices. If there is a wide variation between needs and procedures, then additional interviews can be scheduled.
2. Concerning customer input for the new system, how can Edward involve customers in the process? How can he interest them in participating? What are some ways that Edward can be sure that the customers he does involve are representative of Jacob and Jacob's entire customer group? This may also be a viable candidate for a questionnaire. Statistically sampling can guarantee that a large enough sample can be studied, at least for the questionnaire. The questionnaire should focus on the types of services and reports (statements) that the end customer receives from the system. As with the offices, if interviews are needed, then some distinguishing characteristics should first be identified. Then representative samples of customers could also be interviewed. The cost of interviewing can also be controlled through the use of telephone interviews.

3. As Edward considers what other stakeholders he should include, what are some criteria he should use? Develop some guidelines to help him build a list of people to include.

Guidelines include:
- Look at all the existing reports and destinations. All of the destination persons will have an interest in the information provided by the system.
- Look at all the different departments in the company to see if they currently receive or need to receive information from the new system.
- Consider senior management to see if strategic information needs to be maintained and reported.

Key Terms

- **Activity diagram** – A type of workflow diagram that describes the user activities and their sequential flow.
- **Closed-ended questions** – Questions that have a simple, definitive answer.
- **Functional requirement** – A system requirement that describes an activity or process that the system must perform.
- **Group Support System (GSS)** – A computer system that enables multiple people to participate with comments at the same time, each on the user’s own computer.
- **Joint application design (JAD)** – a technique to define requirements or design a system in a single session by having all necessary people participate.
- **Logical model** – Any model that shows what the system is required to do without committing to any one technology.
- **Mock-up** – An example of a final product that is for viewing only, and not executable.
- **Open-ended questions** – Questions that require discussion and do not necessarily have a simple, short answer.
- **Performance requirement** – A system requirement that describes an operational characteristic related to workload measures, such as throughput and response time.
- **Physical model** – Any model that shows how the system will actually be implemented.
- **Prototype** – A preliminary working model of a larger system.
- **Reliability requirement** – A system requirement that describes the dependability of a system, such as service outages, incorrect processing, and error detection and recovery.
- **Security requirement** – A system requirement that describes user access to certain functions and the conditions under which access is granted.
- **Stakeholders** – All the people who have an interest in the success of a new system.
- **Structured walkthrough** – A review of the findings from your investigation and of the models built based on those findings.
- **Swimlane** – A rectangular area on an activity diagram representing the activities done by a single agent.
- **Synchronization bar** – A symbol in a activity diagram to control the splitting or uniting of sequential paths.
- **System requirements** – Specifications that define the functions to be provided by a system.
- **Technical requirement** – A system requirement that describes an operational characteristic related to an organization’s environment, hardware, and software.
- **Transaction** – A single occurrence of a piece of work or an activity done in an organization.
- **Usability requirement** – A system requirement that describes an operational characteristic related to users, such as the user interface, work procedures, on-line help, and documentation.
- **Workflow** – A sequence of steps to process a business transaction.
Chapter 4 – Modelling System Requirements

This chapter corresponds with chapter 5 of the text book.

Chapter Overview
Note: chapters mentioned in the text below refer to the chapters in the text book.

Chapter 4 described the activities of the systems analysis phase of the SDLC and then introduced tasks and techniques for gathering information about the system, its stakeholders, and its requirements. Chapter 4 also introduced the concept of models and their importance in defining system requirements. This chapter, along with Chapters 6 and 7, presents techniques for documenting the system’s functional requirements by creating a variety of models. This analysis activity is named Define system requirements, and concepts and techniques that apply to both the traditional approach and the object-oriented approach are covered in this chapter.

The first part of the chapter focuses on use cases and business events. Use cases are activities that the system performs to enable a user to complete a task or goal. Business events are those events or activities that are external to the system and that initiate a use case. Thus there is always a one-to-one correspondence to business events and use cases. The explanations in the chapter teach how to identify and model business events and use cases.

The latter part of the chapter deals with the data items within an information system. This concept is introduced with a discussion of “things” in the real world, and which of those things are important for the data model of the new system. Explanations are provided for both the Entity-Relationship Diagram model (ERD) and the Class Diagram model. The ERD has its roots in the traditional approach, and the class diagram is part of the object-oriented approach. The two models have many similarities. However the notation is different between the two. We recommend that students have an exposure to both sets of notation, but with emphasis on the selected approach for the course (Traditional or OO).

Chapter 6 continues the discussion of requirements models for the traditional approach, and Chapter 7 continues the object-oriented approach.

Learning Objectives

After studying this chapter, you should be able to:

• Understand why identifying cases is the key to defining functional requirements.
• Use three techniques for identifying use cases.
• Write brief, intermediate, and fully developed use case descriptions.
• Explain how the concept of things in the problem domain also defines requirements.
• Identify and analyze data entities and objects needed in the system.
• Read, interpret, and create an entity-relationship diagram.
• Read, interpret, and create a class diagram.
Notes on Chapter Opening Case and Case Studies

Waiters on Call Meal-Delivery System: This case shows how the analyst uses events and the use cases they trigger - and things to summarize what the users require in their order and delivery system. As the users discuss their business, the analyst notes what events occur that require the system to respond in some way. Additionally, the analyst notes what types of things the system needs to store information about (data entities or classes).

Key points to note include:

- Events (and use cases) and things can be discovered by talking with the users.
- Talking about events and things is natural for business-oriented users.
- The analyst can focus on requirements independent of current or proposed technology (a logical model), which is something that students often find difficult. Even if the users want to talk about technology, it is important to define the requirements in this way.

Note also that the approach taken by the analyst is the same whether the traditional structured approach or the object-oriented approach is being used, a key point in this chapter.

The Spring Breaks 'R' Us Travel Service Booking System: This case uses a common and easily understandable business problem to practice OO modelling of systems requirements with event tables, data entity (or class) attributes and relationships, and a generalization/specialization hierarchy.

The Real Estate Multiple Listing Service System: This case also uses a common and easily understandable business problem to practice modelling of systems requirements with event tables, an entity-relationship diagram for data storage, and class diagram.

The State Patrol Ticket Processing System: This case is similar to the Real Estate Multiple Listing Service System case. It also uses a common business problem to practice modelling of systems requirements with event tables, an entity-relationship diagram for data storage, and class diagram.

Rethinking Rocky Mountain Outfitters: This case builds on the Chapter 5 material to address the work that needs to be done when additional requirements are included in a project. Beyond the stakeholders and scope, these requirements need to be captured within the existing event tables, entity-relationship diagrams, and class diagrams.

Focusing on Reliable Pharmaceutical Service: The Reliable case includes a subset of the system requirements and a corresponding subset of events, data entities, and classes. Future chapter exercises will add to the scope and complexity of these requirements. Similar to other cases, systems requirements are captured with event tables, entity-relationship diagrams, and class diagrams.
Additional Notes and Quick Quizzes

User Goals, Events, and Use Cases

Virtually all approaches to system development begin modelling functional requirements with the concept of a use case. A use case is an activity the system performs, usually in response to a request by a user. There are several techniques recommended for identifying use cases. One technique is called the user goal technique, which is to identify users (actors) and list the goals each type of user has when using the system. These user goals are named using a verb-noun syntax. For example, in Figure 5-1 on page 161, the goals of the order clerk are to look up item availability, create a new order, and update an order.

A second technique is called the CRUD technique, where CRUD stands for Create, Read or Report, Update, and Delete. With this technique the analyst analyzes each persistent object in the system to ensure that use cases have been defined to perform all of the CRUD processes on those objects. Figure 5-2 gives an example.

Another comprehensive technique is called the Event Decomposition technique and is explained in the next section.

The appropriate level of detail for identifying use cases is one that focuses on elementary business processes (EBPs). An EBP is a task performed by one person in one place in response to a business event that adds measurable business value and leaves the system and its data in a consistent state.

Note that each EBP (and so each use case) occurs in response to a business event. These events drive or trigger the system's processing, and they provide the basis for decomposing the system into separate activities or use cases. Therefore, this technique that helps identify use cases is based on identifying business events and then analyzing the processing the system does when triggered by the event.

The Event Decomposition

The event concept originated with the modern structured analysis technique to help model real-time systems in the early 1980s. Information engineering adopted the event concept later. The object-oriented approach has more recently adopted the event concept as an approach for decomposing the system into use cases. Regardless of which approach you are using in your course, the event concept is very important.

Types of Events

Analysts should look for three types of events—external events, temporal events, and state events. See the opening case for examples of events and how to name them. The analyst begins by listing as many of the potential events as possible.

External events are events that occur outside the system, and that are usually initiated by an external agent or actor that supplies data inputs to the system. When describing external events, it is important to name the event so that the external agent is clearly defined. For example, in an order processing system a customer places an order, a customer returns a product, and a customer pays a bill. External events always result in an input to the system that must be processed in some way.
Temporal events are events that are triggered by reaching a point in time, and that produce an output from the system that goes to an external agent or actor. Many information systems produce end-of-month reports, statements, bills, late notices, and so on when a point in time is reached. When describing temporal events, it is important to name them by stating the time to, the output produced, and the external agent or actor who receive them (for example, time to produce end-of-month exception report for management).

State events are events that occur when some internal state changes and triggers the need for processing, such as reorder point is reached. There is no input, but there usually is an output. State events are similar to temporal events, but the point in time at which they occur is not known.

Identifying Events
It is not always easy to identify the events that affect the system. For external events, you should focus on the external agents outside of the system. This helps to keep the focus on the requirements from the user’s point of view, not on the internal technology. Sometimes it helps to distinguish between an event and the sequence of conditions that leads up to the event. For example, a person decides that he or she needs a new shirt, drives to the store, and looks through the racks. But the event occurs only when the individual walks up to the cashier with a shirt and says, “I want to buy this shirt.” At that point, the system responds. The event name might be something like customer buys a shirt.

Sometimes it is useful to think about all of the actions an external agent or actor might perform over time. For example customer requests a catalogue, the customer checks item availability, the customer places an order, the customer cancels or changes an order, the customer checks order status, the customer updates account information, and the customer returns an item. Consider all the possible ways an external agent might interact with a system to identify all possible events.

Some events are related to specific technologies or system controls. System controls are checks or safety procedures put in place to protect the integrity of the system. These are important in the design phase but not during analysis. For example, a user logs into the system, the user changes his or her password, the system requires a certain amount of time to back up the database, and so on, are not events that define the functional requirements for the system. Rather, they involve the need to design system controls to protect the system.

The perfect technology assumption is a concept to help the analyst determine if an event is required in the analysis phase. If the system must respond to the event even if the system is implemented with perfect technology, then the event should be included. Under perfect conditions, equipment never breaks down, the capacity for processing and storage is unlimited, and people operating the system are completely honest and never make mistakes.

However, if a customer buys a shirt, the system must process the purchase and produce a receipt even if the system is implemented with perfect technology. Therefore, Customer buys a shirt is an event of importance that defines the requirements of the system.

Events in the Rocky Mountain Outfitters Case

Figure 5-13 on page 165 lists fourteen external events identified for the Rocky Mountain
Outfitters customer support system. These events are similar to the ones discussed earlier (for example, customer wants to check item availability and customer places an order). Other external agents are part of RMO but are external to the customer support system that interacts with the system. For example, shipping fulfils order results in an input to the system from the shipping department. Merchandising updates catalogue results in an input to the system from the merchandising department. Figure 5-10 lists six temporal events (Time to), most of which result in the production of reports that go to external agents.

Looking at Each Event and the Resulting Use Case

Once an event has been identified, the analyst should record additional information about the event, most importantly the use case that is required. An event table is used to list events in rows with key pieces of information about each event in columns. The columns include the event, the trigger, the source, the use case (activity the system carries out), the response, and the destination for the response.

**Trigger:** A signal that tells the system that the event has occurred, either the arrival of data needing processing for an external event (a data flow input) or a point in time for a temporal event.

**Source:** An external agent or actor that supplies data to the system for an external event.

**Activity:** Behaviour that the system performs when an event occurs (similar to a use case).

**Use case:** A series of actions that a system performs that result in a defined outcome (similar to an activity).

**Response:** An output produced by the system that goes to a destination.

**Destination:** An external agent or actor that receives data from the system.

Once the analyst has recorded this information about each event, the event table becomes an important source of information for building the more detailed models that define the system functional requirements. The completed event table helps answer the following questions: What happens that the system must handle? How do we know it is happening? What’s the source of the data coming in when it happens? What does our system need to do when it happens? What outputs are produced by our system as a response? Where do the outputs go?

Quick Quiz: Events and System Requirements

1) **What is an event?**
   ANSWER: An event is an occurrence at a specific time and place that can be described and is worth remembering by the system.

2) **What are the three types of events?**
   ANSWER: External events, temporal events, and state events.

3) **What is the perfect technology assumption?**
   ANSWER: A concept that helps the analyst decide whether or not an event is required in
the analysis phase. If the system must respond to the event even when the system is implemented with perfect technology and perfect users, then the event should be included.

**Use Case Descriptions**

A use case shows that an actor interacts with the computer system to carry out a business activity. We document use cases in several ways. One common method of documenting the internal details of the use case is with a use case description. Implied in all use cases is a person or user who is carrying out the use case. This role is called an actor.

A use case is a high-level description and may include a whole sequence of steps to complete a business process. We describe these individual steps with a narrative that is called a scenario or use case instance. A scenario is a unique set of internal activities within a use case and represents a unique path through the use case. A use case may have several different scenarios.

Use case descriptions can be brief, intermediate, or fully developed. The brief description merely provides a short overview of the major steps in a use case. The intermediate description goes somewhat further in that it identifies step-by-step the processing required. An intermediate description also frequently documents any exception conditions that occur. Finally the fully developed description includes all components of the first two descriptions, but also documents other critical facts about the use case such as the preconditions and postconditions. Figures 5-13 through 5-17 show examples of each type of description.

**Quick Quiz: The System Activities: A Use/Case Scenario View**

- **What does an actor represent in a use case diagram?**
  ANSWER: An actor is the role played by a particular person when that person interacts with the system.

- **What is the difference between a fully developed description and an intermediate description?**
  ANSWER: Both contain a step-by-step description of the workflow. However, a fully developed description also documents many characteristics of the use case such as objective, users, preconditions and postconditions.

- **What are preconditions? What are postconditions?**
  ANSWER: Preconditions state which conditions must be true before a use case begins. Postconditions identify which conditions must be true upon completion of the use case.

**“Things” in the Problem Domain**

Another key concept used to define system requirements involves understanding and modelling things the system needs to store information about. To users, these are the things they deal with when they work—products, orders, invoices, and customers. In the traditional approach to development, these things make up the data the system needs to store. In the object-oriented approach, these things begin to define the types of objects that interact in the system.
Types of Things
As with events, the analyst should ask users about several types of things. Some are tangible and, therefore, easy to identify. Some are intangible. Types of things to look for include roles played, organizational units, devices, incidents/events/interactions, and sites/locations. Remember it is the information about the things (customers, products, orders, shipments, and so on) that is important.

Procedure for Developing an Initial List of Things
A useful procedure for identifying the things that the system needs to store information about is to list all of the nouns that users mention when talking about the system. See page 178 for the specific steps to follow. Figure 5-19 lists some of the nouns from the RMO customer support system event table and other sources.

Relationships Among Things
The system needs to remember many important relationships among things. A relationship is a naturally occurring association among specific things, such as an order is placed by a customer, and an employee works in a department. The system needs to remember what customer placed a particular order and which department the employee works in.

Relationships between things apply in two directions. For example, a customer places an order describes the relationship in one direction, and an order is placed by a customer describes the relationship in the other direction.

It is also important to understand the nature of each relationship in terms of the number of associations for each thing. For example, a customer might place many different orders, but an order is placed by only one customer. Cardinality defines the number of associations that occur between specific things. UML (Unified Modelling Language) and the object-oriented approach call this concept multiplicity. Each relationship (in each direction) can have a minimum and a maximum cardinality/multiplicity. For example, a particular customer might not ever place an order. In this case, there are zero associations. Alternatively, the customer may place one order, meaning one association exists. Finally, the customer might place two, three, or even more orders. The relationship for a customer placing an order can have a range of zero, one, or more, usually indicated as zero or more. The zero is the minimum cardinality, and “more” is the maximum cardinality.

Binary relationships are relationships between two different types of things, such as a customer and an order.

Unary (recursive) relationships are relationships between two things of the same type, such as one person being married to another person.

Ternary relationships are relationships between three different types of things. (An n-ary relationship means a relationship between n, any number, of different types of things.)

Attributes of Things
Most information systems store and use specific pieces of information about each thing. One piece of information about a thing is called an attribute. For example, a customer has a name, address, phone number, and so on. Each customer has a specific value for the attribute that is important for the system to remember. An attribute that uniquely identifies
a specific thing is called an **identifier** or **key**. For example, an identifier for a customer is a customer number, for a vehicle it is a vehicle ID, and for a product it is a product ID. Sometimes these identifiers already exist (social security number, vehicle ID number) but sometimes they need to be created and assigned by the system (invoice number, transaction number).

Not all identifiers are ID numbers. For example, a name of a university could be an identifier. A **compound attribute** is an attribute that contains a collection of related attributes. For example, a Customer full name compound attribute might represent a customer’s first, middle, and last names. Sometimes the identifier or key might be a compound attribute (for example, Department code plus course number might be used to identify a course, such as CIS321 or ART 20).

**Quick Quiz: Things and System Requirements**

1) What is a “thing” called in models used in the traditional approach?
   ANSWER: A data entity.

2) What is a “thing” called in models used in the object-oriented approach?
   ANSWER: An object.

**The Entity Relationship Diagram**

The traditional approach to development emphasizes data storage requirements (what data must be stored by the system for later use?). The processing requirements are defined separately. The entity-relationship diagram (ERD) is used to model the data storage requirements.

**Examples of ERD Notation**

Because the data requirements include entities, relationships among entities (including cardinality), and attributes of entities, the ERD includes notations for these aspects of the model. An entity is a rectangle with the name inside. The relationships are shown with the lines that connect the rectangles. Cardinality is shown using the “crow’s feet” symbol and small circles or lines to indicate minimum cardinality. See the Figures 5-23 and 5-24 of the text. Remember to consider each direction of each relationship separately. In this text, an asterisk is used to indicate the identifier or key, usually at the top of the list of attributes. In practice, variations of this notation are used.

Many-to-many relationships have “crow’s feet” on both ends of the relationship line. Sometimes this means there is more information that needs to be stored about the relationship. For example, a course section enrolls many students, and a student is enrolled in many course sections. An **associative entity** called Course enrolment should replace the many-to-many relationship to allow the system to store the grade for each section that is taken by the student. See Figure 5-28.

**The Rocky Mountain Outfitters Case ERD**

The Rocky Mountain Outfitters ERD shows eleven data entities (see Figure 5-29). Be sure to read and understand each relationship in each direction. For example, a shipper makes many shipments, and each shipment contains many order items, each of which is included in one order. An order has many order items, each of which is included in a shipment that
is shipped by one shipper. It is important to “debug” a model such as this by checking to make sure it provides the required data storage capability. Test the data model just as you would test the program logic. The attributes are not shown in the figure, but they usually are shown in an ERD. You can view the attributes in the Rocky Mountain Outfitters class diagram in Figure 5-38.

Quick Quiz: The Entity-Relationship Diagram

1) What is a relationship?
   ANSWER: A naturally occurring association among specific things, such as an order is placed by a customer, and an employee works in a department.

2) What are the symbols shown in an entity-relationship diagram?
   ANSWER: Rectangle for entity, line for relationship, minimum and maximum cardinality symbols, including “crow’s feet,” on the line of a relationship.

Solutions to End of Chapter Material

Review Questions

1. What are the two key concepts used to begin defining system requirements?
   Events the system needs to respond to and things that the system needs to store information about.

2. What is a use case?
   A use case is an activity the system performs.

3. What are three techniques used to identify use cases?
   The user goal technique, the CRUD technique, and the Elementary Business Process technique.

4. What is an event and what is an elementary business process (EBP)?
   An event is an occurrence at a specific time and place that should be remembered. An EBP is a task that is performed by one person in one place in response to a business event.

5. What is the three types of events?
   A temporal event, a state event, and an external event.

6. Which type of event results in data entering the system?
   External event.

7. Which type of event occurs at a defined point in time?
   Temporal event.

8. Which type of event does not result in data entering the system but always results in an output?
   Temporal event.
9. **What type of event would be named Employee quits job?**
   External event, as the data entering the system would be data about the termination of the employee (who, when, and why).

10. **What type of event would be named Time to produce paychecks?**
    Temporal event because the system knows it is time to produce paychecks because it is the end of the month.

11. **What are some examples of system controls?**
    Validating user input, requiring user IDs and passwords for logging on to the system, backing up data regularly, encrypting data that is transmitted, and so on.

12. **What does the perfect technology assumption state?**
    The perfect technology assumption says to only consider events during analysis when defining requirements that the system is required to respond to even if the system is assumed to be implemented with perfect technology. Basically, this keeps the analysts from worrying about systems controls until later during the design phase.

13. **What are the columns in an event table?**
    The event, trigger, source, activity/use case, response and destination.

    See margin definitions. Basically, the trigger is a data input for an external event and a definition of a point of time for a temporal event. A source is what external agent or actor supplies the data input for an external event. The activity/use case is what the system does when the event occurs (the process). A response is a data output from the system. The destination is the external agent or actor that receives the data output.

15. **What is the difference between a use case and a scenario? Give an example of each.**
    A scenario is a particular instance or set of steps for one path through the use case. A use case is a goal oriented process done by the system.

16. **What are the three types of use case descriptions? Which one is usually sufficient for a simple use case?**
    Brief description, Intermediate description, and a Fully Developed description.

17. **What are preconditions and postconditions? Give an example of each.**
    Preconditions are conditions that must exist in the system before the use case begins. Post conditions are those that must exist after the use case completes its processing.

18. **What are exceptions conditions? Give an example of each.**
    Exception conditions are steps that are outside of the ordinary or normal process of steps for a use case. An example would be out of stock condition.

19. **What communicates back and forth in a two-column flow of activities?**
    In a flow of activities, the communication is between an external actor and the
automated system. Normally this is some sort of data communications.

20. **What is a “thing” called in models used in the traditional approach?**
A data entity.

21. **What is a “thing” called in the object-oriented approach?**
An object.

22. **What is a relationship?**
A naturally occurring association among specific things, such as an order is placed by a customer, and an employee works in a department.

23. **What is cardinality of a relationship (also called multiplicity)?**
The number of associations that occur between specific things, such as a customer places many orders, and an employee works in one department.

24. **Describe how an entity-relationship diagram shows the minimum and maximum cardinality.**
A circle on the relationship line means zero, and a short slash line means one. A “crow’s feet” symbol means many.

25. **What are unary, binary, and n-ary relationships?**
A unary relationship is a relationship between two things of the same type; a binary relationship is a relationship between two things of different types; and an n-ary relationship is a relationship among n things or n different types.

26. **What are attributes and compound attributes?**
An attribute is one piece of specific information about a thing, such as the first name of a person. A compound attribute is made up of more than one specific piece of information. For example a Full name compound attribute could represent the first, middle, and last names of a person.

27. **What is an associative entity?**
An associative entity is a data entity that represents a many-to-many relationship between two entities when the relationship includes information needing to be stored.

28. **What are the symbols shown on an entity-relationship diagram?**
Rectangle for entity, line for relationship, minimum and maximum cardinality symbols including a “crow’s feet” symbol are shown on the line of a relationship.

**Case Studies**

**Case Study: Spring Breaks 'R' Us Travel Service Booking System (SBRU)**

1. To what events must the SBRU booking system respond? Create a complete event table listing the event, trigger, source, activity, response, and destination for each event. Be sure to consider only the events that trigger processing in the booking system, not the SBRU accounting system or the systems operated by the resorts.
List of events and resulting use cases with explanations:

1. Event: Resort submits availability information.
   Use case: Record resort availability information.
   This is the information sent to SBRU in the fall indicating what rooms are available and what the rates are for each week. The system records this information.

2. Event: Time to produce brochure information for college representatives.
   Use case: Produce brochure information.
   As soon as December arrives, the system takes all of the resort information that has come in and produces the brochure information. Note that the format of the brochure might be a printed document or a Web site. A logical model would not indicate how the brochure was implemented or how it gets to the college representative.

3. Event: Student group requests a reservation.
   Use case: Book student group.
   A group of students submits a reservation request for a specific resort during a specific week. Note that the college representative might interact with the students, but that interaction is outside the scope of this system as it is defined. No information is captured until the student group actually makes a reservation.

4. Event: Time to produce booking information for resorts.
   Use case: Produce booking information for resorts.
   Before each spring break week begins, the system produces a list for each resort that shows who has booked rooms for that week. Each resort uses the information to check in arriving students. What the resort does is outside the scope of this system. A student paying the resort is also outside the scope of this system. Resorts later send commissions to SBRU accounting, which again is outside the scope of this system.

5. Instructors might include an additional event for producing booking reports every week for management of SBRU, although nothing was specifically mentioned in the case. Similarly, reports might be sent to the resorts or college reps to indicate how well bookings were going. It might be assumed that accounting can read data about bookings from the database to reconcile commission checks (accounting is a separate system), or the booking system might generate a report for accounting each spring break week or at the end of the spring break season.

The event table follows on the next page
Event Table

<table>
<thead>
<tr>
<th>Event</th>
<th>Trigger</th>
<th>Source</th>
<th>Essential Activity</th>
<th>Response</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Resort submits availability info</td>
<td>Avail info</td>
<td>Resort</td>
<td>Record resort availability info</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Time to produce brochure info for college reps</td>
<td>'December'</td>
<td>Produce brochure information</td>
<td>Brochure info</td>
<td>College Rep</td>
<td></td>
</tr>
<tr>
<td>3. Student group requests a reservation</td>
<td>Reservation request</td>
<td>Student group</td>
<td>Book student group</td>
<td>Confirmation</td>
<td>Student</td>
</tr>
<tr>
<td>4. Time to produce booking info for resorts</td>
<td>'Right before each week'</td>
<td>Produce booking info for resorts</td>
<td>Booking info</td>
<td>Resort</td>
<td></td>
</tr>
</tbody>
</table>

2. List the data entities (or classes) that are mentioned. List the attributes of each data entity (or class). List the relationships between data entities (or classes).
The data entities (or classes) might include Resort, Week, Room Type, Recreational Facilities, College Rep, College, Student Group, Student, and Reservation. Note that the data stored included information about external agents/actors as well as information about availability and reservations.

3. Which classes might be refined into a generalization/specialization hierarchy? List the superclass and any subclasses for each of them.
There might be special types of rooms, types of resorts, types of recreational facilities, types of student groups, or types of college representatives. The key to deciding if a generalization/specialization hierarchy is required is to identify whether special information needs to be stored for each type (subclasses have additional attributes).

Case Study: Real Estate Multiple Listing Service System

1. To what events must the multiple listing service system respond? Create a complete event table listing the event, trigger, source, activity, response, and destination for each event.

List of events and resulting use cases with explanations:


   The event is the real estate office sending in the information after the agent signs the listing.
2. Event: Agent requests listing information.  
   Use case: Provide listing information.

   This is the query facility where the agent can get listing information that meets search criteria.

   Use case: Produce multiple listing books.

   This is a temporal event triggered twice a month, resulting in the multiple listing book being sent to the real estate agents. Note the “book” does not have to be printed on paper, so this can remain a logical model. The class might debate this, though.

4. Event: Real estate office submits listing change request.  
   Use case: Record listing change.

   This represents changes in the listings to correct errors, update terms such as price, and mark the status of a listing when it is under contract, sold, or withdrawn.

5 & 6. There really should be two additional events, although they were not explicitly stated in the case. When real estate offices are added or changed, the system needs to update real estate office data. Additionally, when an agent is added or needs updating, the system needs to update agent data. These two events could be added to the event table below.

### Event Table

<table>
<thead>
<tr>
<th>Event</th>
<th>Trigger</th>
<th>Source</th>
<th>Essential Activity</th>
<th>Response</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Real estate office submits new listing</td>
<td>New listing</td>
<td>RE Office</td>
<td>Add new listing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Agent requests listing info</td>
<td>Listing query</td>
<td>Agent</td>
<td>Provide listing information</td>
<td>Listing information</td>
<td>Agent</td>
</tr>
<tr>
<td>3. Time to produce multiple listing book</td>
<td>'15th and 30th of the month'</td>
<td>Produce multiple listing book</td>
<td>Multiple listing book</td>
<td>Agent</td>
<td></td>
</tr>
<tr>
<td>4. Real estate office submits listing change request</td>
<td>Listing change request</td>
<td>RE Office</td>
<td>Record listing change</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Draw an entity relationship diagram to represent the data storage requirements for the multiple listing service system, including the attributes mentioned. Does your model include data entities for offer, buyer, and closing? If so, reconsider. Include information the multiple listing service needs to store, which might be different from information the real estate office needs to store.

Data Entities and Attributes for ERD

This is limited to three data entities, just to store info on the listings and the agent and office responsible for the listing. Buyers and sellers and closings would not be included. The data entities and attributes are:

Real Estate Office
- Office Number
- Name
- Office Manager Name
- Address
- Phone
- FAX

Real Estate Agent
- Agent Number (license number?)
- Name
- Office Phone
- Home Phone
- E-mail Address
- Cell Phone

Real Estate Listing
- Listing Number
- Address
- Year Built
- Square Feet
- Number Bedrooms
- Number Baths
- Owner Name
- Owner Phone
- Asking Price
- Date Listed
- Date Last Updated
- Status Code

Relationships are:
- Real Estate Agent : Real Estate Listing  1:M
- Real Estate Agent : Real Estate Office  M:1

From all the above information the ERD can now be drawn
Key Terms

- **Abstract class** – A class that cannot be instantiated (no objects can be created), existing only to allow subclasses to inherit its attributes, methods, and associations.
- **Activity** – Behaviour that the system performs when an event occurs (similar to a use case).
- **Aggregation** – Whole-part relationship between an object and its parts.
- **Association class** – A class that represents a many-to-many relationship between two other classes.
- **Associative entity** – A data entity that represents a many-to-many relationship between two other data entities.
- **Attribute** – One piece of specific information about a thing.
- **Binary relationships** – Relationships between two different types of things, such as a customer and an order.
- **Cardinality** – The number of associations that occur among specific things, such as a customer places many orders and an employee works in one department.
- **Class** – the type or classification to which all similar objects belong.
- **Composition** – Whole-part relationship in which the parts cannot be dissociated from the object.
- **Compound attribute** – An attribute that contains a collection of related attributes.
- **Concrete Class** – A class that can be instantiated (objects can be created).
- **Data entities** (175) – The things the system needs to store information about in the traditional approach to information systems.
- **Descriptive model** – Narrative memos, reports, or lists that describe some aspect of a system.
- **Destination** – an external agent or actor that receives data from the system.
- **Domain Model** – A class diagram without methods, which is created as a requirements model.
- **Encapsulation** – Covering or protecting each object so that it contains values for attributes and methods for operating on those attributes, making the object a self-contained (and protected) unit.
- **Event** – An occurrence at a specific time and place that can be described and is worth remembering.
- **Event table** – a table that lists events in rows and key pieces of information about each event in columns.
- **External event** – an event that occurs outside the system, usually initiated by an external agent or actor.
- **Generalization/specialization hierarchies** – Hierarchies that structure or rank classes from the more general superclass to the more specialized subclasses; sometimes called inheritance hierarchies.
- **Graphical model** – Diagrams and schematic representations of some aspect of a system.
- **Identifier(key)** – An attribute that uniquely identifies a thing.
- **Inheritance** – A concept that allows subclasses to share characteristics of their superclasses.
- **Mathematical model** – A series of formulas that describe technical aspects of a system.
- **Methods** – The behaviours all objects of the class are capable of doing.
- **Multiplicity** – a synonym for cardinality (used with the object-oriented approach).
• **N-ary relationship** – A relationship among N(any number of) different types of things.
• **Perfect technology assumption** – The assumption that events should be included during analysis only if the system would be required to respond under perfect conditions.
• **Relationship** – A naturally occurring association among specific things, such as an order is placed by a customer and an employee works in a department.
• **Response** – An output, produced by the system that goes to a destination.
• **Source** – an external agent or actor that supplies data to the system.
• **State event** – an event that occurs when something happens inside the system that triggers the need for processing.
• **System controls** – Checks or safety procedures put in place to protect the integrity of the system.
• **Temporal event** – an event that occurs as a result of reaching a point in time.
• **Ternary relationship** – A relationship among three different types of things.
• **Trigger** – A signal that tells the system that an event has occurred, either the arrival of data needing processing or a point in time.
• **Unary (recursive) relationship** – A relationship among two things of the same type, such as one person being married to another person.
• **Use case** – A series of actions that a system performs that result in a defined outcome (similar to an activity)
• **Whole-part hierarchies** – Hierarchies that structure classes according to their associated components.
Chapter 5 – The Traditional Approach to Requirements

This chapter corresponds with chapter 6 of the textbook.

Chapter Overview
Note: chapters mentioned in the text below refer to the chapters in the textbook.

Chapter 5 described two key concepts associated with modeling system requirements in both the traditional and the object-oriented (OO) approaches to information system development: events and things. In this chapter, the focus turns to activities and interactions—what the system does when an event occurs.

This chapter describes the traditional structured approach to representing activities and interactions. Chapter 7 describes details of the OO approach to representing activities and interactions. With both approaches, analysts and users must jointly evaluate model completeness, correctness, and quality.

Learning Objectives

After studying this chapter, you should be able to:

- Explain how the traditional approach and the object-oriented approach differ when modeling the details of a use case.
- List the components of a traditional system and the symbols representing them on a data flow diagram.
- Describe how data flow diagrams can show the system at various levels of abstraction.
- Develop data flow diagrams, data element definitions, data store definitions, and process descriptions.
- Develop tables to show the distribution of processing and data access across system locations.

Notes on Chapter Opening Case and Case Studies

San Diego Periodicals: Following the Data Flow: The case describes a realistic interaction scenario between a system analyst and a user. The case depicts the heart of the analysis task—eliciting user requirements and accurately representing them in analysis models.

Key points to emphasize about the case include:
- Systems analysis requires communication with people. Success as an analyst depends heavily on interpersonal communication skills (unlike many other computer-related jobs).
- Developing an accurate model requires iterations of eliciting requirements, documenting them in models, and reviewing the models for accuracy. There is no one-step 'cookbook' approach to developing complete and accurate analysis models.
- Part of the analyst's job is to train the users to read and understand the models. If the
user can’t understand the model then he or she can’t assist the analyst in identifying errors.

- Developing accurate models requires close attention to detail (note the filled notepad, the large number of corrections, and the 'drained brains' of the participants).
- It is worth mentioning that the above points are equally applicable to documenting requirements with object-oriented models.

Notes on RMO Memo

The memo on page 204 provides a status report on the analysis phase and describes some of the problems encountered in specifying requirements. Key points to stress about the memo include:

- Documenting the requirements of a large-scale system typically requires hundreds of pages.
- Coordinating analysis phase activities with users can be problematic because many users must be consulted. Also, users sometimes disagree about system requirements and those differences must be resolved.
- The need for user involvement is constant throughout the analysis phase. Users typically provide input for multiple parts of the system and many consultations are required to iron out all of the details.
- Analysts rely on user input though they have no direct authority over those users. Users and their superiors must be committed to the project’s success and to its schedule.

The Real Estate Multiple Listing Service System: The event list and ERD for the Chapter 5 case study is used as a starting point to develop a context DFD, an event-partitioned DFD, and required process decomposition DFDs.

The State Patrol Ticket Processing System: Similar to the Real Estate Multiple Listing Service System case study, the event list and ERD for the Chapter 5 case study are used as a starting point to develop a context DFD, an event-partitioned DFD, and required process decomposition DFDs. This case also asks for data flow definitions for data flows that are fully described in the written system description.

Rethinking Rocky Mountain Outfitters: The RMO event table (Figure 5-12) and ERD (Figure 5-29) is used as a starting point to develop DFD fragments for events not documented in Figure 6-12, a single DFD for all events, and a data flow definition for the RMO customer order form in Figure 6-35.

Additional Notes and Quick Quizzes

Traditional and Object-Oriented Views of Activities/Use Cases

The traditional and object-oriented approaches to software development differ in how a system’s response to an event is modelled and implemented. The traditional approach views data and processes as two separate and distinct things. Data is a passive thing that is stored in data stores (one per entity on the ERD) and is passed back and forth between processes and external agents. Processes represent activities.
The OO approach views data entities and their related activities as a single thing called a class or object. Objects are containers for both data and activities (processes). Objects interact with one another by sending and responding to messages.

Figure 6-1 summarizes the differences between traditional and OO approaches to systems. Figure 6-2 summarises the requirements models for the traditional and OO approaches. Chapter 6 focuses on how activities and data are viewed using the traditional approach. Chapter 7 focuses on these same concepts using the OO approach.

Quick Quiz: Traditional and Object-Oriented Views of Activities

1) How is a system viewed with the traditional approach?
   ANSWER: A system is viewed as a collection of processes performed by people and by computers.

2) How is a system viewed with the object-oriented approach?
   ANSWER: A system is viewed as a collection of interacting objects.

3) What is involved (and modelled) with the traditional approach?
   ANSWER: Processes, stored data, inputs, and outputs.

4) What is involved (and modelled) with the object-oriented approach?
   ANSWER: Models that show objects, their behaviour, and their interactions with other objects.

Data Flow Diagrams

A data flow diagram (DFD) represents the flow of data among internal processes and external agents. Five symbols are used in a DFD (see Figure 6-3):

- **Process** – A symbol in a DFD that represents an algorithm or procedure by which data inputs are transformed into data outputs.
- **Data flow** – An arrow in a DFD that represents the movement of data among processes, data stores, and external agents.
- **External agent** – A person or organization, outside the boundary of a system, that supplies data inputs or accepts data outputs.
- **Data store** – A place where data is held pending future access by one or more processes.
- **Real-time link** – A symbol on a data flow that represents data exchange between a process and an external agent as the process is executing.

Students typically have problems interpreting the difference between:

- A data flow and a data store
- A data flow and a real-time link

A common question is, “How long does data have to remain at rest to be represented as a data store?” Often examples such as an in-basket for incoming mail or a stack of forms on a desk that are awaiting data entry or some other form of processing are given. The answer, of course, is not always clear. A key indicator that a data store is required is when
the same data is read by two different processes each operating (or potentially operating) at different times. In general, data stores that represent temporary stopping (or batching) points between two processes or between an external agent and a process should be avoided because they imply physical knowledge of the implementation technology (for example, knowledge of batch processing to improve efficiency).

The difference between a data flow and a real-time link is subtle. Real-time links always represent data movement in both directions (typically a request followed by an immediate response). They also connect a process and an external agent.

A DFD is a graphical representation of information from the event table and ERD (as defined in Chapter 5). The correspondence is summarized as follows (see Figure 6-5):

- **Source** – An external agent
- **Trigger** – A data flow from an external agent to a process
- **Activity** – A process
- **Response** – A data flow from a process to an external agent
- **Destination** – An external agent
- **Data entity from ERD** – A data store

### Data Flow Diagrams and Levels of Abstraction

A DFD can model a system or part of a system’s various levels of detail. Different levels of detail are sometimes called levels of abstraction. Data flow diagrams can show either higher-level or lower-level views of the system. The high-level processes on one DFD can be decomposed into separate lower-level, detailed DFDs. Processes on the detailed DFDs can also be decomposed into additional diagrams to provide multiple levels of abstraction.

A single DFD can represent either extreme or any level of abstraction in between. DFDs at different levels of abstraction have different names and are used for various purposes.

At first, this section is usually difficult for students. An appropriate analogy is the hierarchical grouping of processing functions within a large organization or military unit. The high number of activities precludes representing them in a single diagram or model. What is needed is a hierarchical partitioning (perhaps one could use the term *grouping*) of the hundreds or thousands of functions into relatively small groups that can be represented on a single page. One can later refer to the context level as the CEO’s or general’s view, the event-partitioned model as the branch manager’s or captain’s view, and the lowest-level DFD as the office manager’s or sergeant’s view. The views are all related by hierarchical lines of authority in much the same fashion that DFDs at different abstraction levels are hierarchically related.

#### Context Diagrams

A context diagram describes an entire system at the highest level of abstraction (see Figure 6-6). A context diagram is a DFD that summarizes all processing activity within the system in a single process symbol. The primary role of the context diagram is to show the relationship (data flows) between the system and external agents. A context diagram is a useful tool for describing the scope of a system to an external party (for example, an IS steering committee member).
Context diagrams are often created for a single subsystem, especially when the entire system is very large. Figure 6-9 shows the RMO customer support system and Figure 6-11 shows the context diagram for the order-entry subsystem. All RMO examples in the rest of the chapter are drawn from the order-entry subsystem.

**DFD Fragments**

A DFD fragment (see Figure 6-12) is a DFD that represents the system response to one event within a single process symbol. Therefore, one DFD fragment is created for each event/use case in the event table. Figure 6-12 shows five separate DFD fragments. DFD fragments are the most direct link between the entity-relationship diagram and the other parts of a traditional analysis model. DFD fragments are “bite-sized” pieces of the analysis model that can be constructed, analyzed, validated, and dissected independently. The first pass at modelling a large system is often performed bottom up, with the DFD fragments being the bottom-most layer.

A single DFD fragment can hide a lot of complexity (for example, a process named Prepare Federal Income Tax Return!). Just as the event-partitioned system model is decomposed into a set of DFD fragments, a single DFD fragment can, if necessary, be decomposed into one or more DFDs at lower levels of abstraction (that is, higher levels of detail).

The simplest form of decomposition is a DFD that represents the details of a single DFD fragment. A hierarchical numbering scheme relates detailed DFDs to DFD fragments and processes on an event-partitioned system model. For example, the detailed DFD of Process 5 on an event-partitioned system model is called Diagram 5. Processes within Diagram 5 are numbered 5.1, 5.2, and so forth. If any process (for example, Process 5.3) in the DFD requires further decomposition, the corresponding detailed DFD is called Diagram 5.3 and its processes are number 5.3.1, 5.3.2, and so forth. Decomposition can continue for any number of levels.

---

Students often ask the question, “When do you stop making lower-level DFDs?” The short answer is to stop when a process can be described in a single-page structured English process description, decision table, or decision tree. Of course, the definition of page varies from person to person. Because process descriptions are similar to code, screen-sized pages (24 lines) are probably a more suitable maximum length than are standard-sized paper pages (66 lines).

Another common question is, “How many processes do you put on a detailed DFD?” This question is tough to answer because it depends on the specifics of the process and how you handle complexity limits (as covered in a later section). The key principles are readability and minimization of interfaces. But the answer to this question usually requires several examples (both good and bad) to drive the point home.

---

**The Event-Partitioned System Model**

An event-partitioned model (diagram 0) combines all of a system’s or subsystem’s DFD fragments on a single DFD (see Figure 6-13). Each context diagram can have a corresponding event-partitioned model (see Figure 6-13 and Figure 6-11). Diagram 0 is primarily a presentation tool. Analysts often avoid creating diagram 0 because it can be very complex in large systems and because its information content is redundant with the DFD fragments.
Physical and Logical DFDs
A physical DFD models one particular implementation of a system (see Figure 6-15). A logical DFD models the system without bias to any particular implementation details. When looking at a logical DFD, the reader shouldn’t be able to tell whether the system is automated or manual, centralized or distributed, or how its various pieces are distributed among locations, organizations, computer hardware, or programs.

With respect to DFDs, the terms physical and logical describe two ends of a continuum. It is difficult to create a DFD that is completely logical. Many assumptions about technology or other implementations are subtle and difficult to identify or remove from a DFD. Specific things to look for when trying to identify physical DFD features include:

- Technology-specific processes
- Actor-specific process names
- Technology- or actor-specific process orders
- Redundant processes, data flows, and data stores

Physical DFDs are usually avoided during the analysis phase. By documenting system requirements in a logical way, the analyst leaves all design and implementation options open. Creating physical DFDs during analysis limits thinking about technology and other implementation choices.

Perfect technology is a useful concept for separating physical and logical requirements. Perfect technology is a hypothetical technology (or set of technologies) that is error-free and has infinite processing speed and data transmission/storage capacity. If a requirement “disappears” when perfect implementation technology is assumed, then it is a physical (not a logical) requirement. Note that perfect technology is only assumed for things inside the system.

If you master the perfect technology concept, you’ll have little trouble differentiating between logical and physical models. Think of common desktop computer concepts when describing perfect technology, such as infinitely fast processors, CDs with infinite capacity, hard drives that never crash, files that can always be recovered if accidentally erased, and modems with infinitely fast speed that never drop their connections or garble data transmission.

Students sometimes go too far in conceptualizing perfection. Statements such as “in a perfect world customers wouldn’t pay for a product” are common. Remember that there are distinctions between perfect internal technology and a perfect world.

Redundancy implies a less-than-perfect technology. For example, copies of files exist to protect against accidental data loss and process redundancy exists to reduce undetected errors. Neither situation is possible in a system implemented with perfect technology.

Evaluating DFD Quality
Once a first draft of a DFD is prepared, it must be evaluated for quality. (Quality control can also be performed while the DFD is being prepared.) DFD quality checks fall into two primary categories:

- Readability/intelligibility
Logical consistency (or lack thereof)

Readability and intelligibility are a matter of information or cognitive overload. The analyst always walks a fine line between presenting too much information in a single chunk and having too many chunks to keep track of. The rule of seven plus or minus two (also known as Miller’s Number) and the principle of minimization of interfaces are useful but imperfect attempts to define the proper balance.

Miller’s Number is a limit on the number of information chunks that a typical human can accurately “process” at one time. Some applications of the rule of 7 ± 2 to DFDs include the following:

- A single DFD should have no more than 7 ± 2 processes.
- No more than 7 ± 2 data flows should enter or leave a process, data store, or data element on a single DFD.

Minimization of interfaces is related to Miller’s Number. The principle controls complexity by limiting the number of interfaces between DFD components. One application is to limit the number of data flows into and out of each process to 7 ± 2.

Black holes and miracles are logical inconsistencies. A black hole (see Figure 6-16) is a process or data store that data enters but never leaves. A miracle (see Figure 6-17) is a data flow that leaves a process or data store without ever having entered that process or data store. Both conditions can be discovered by a straightforward (but tedious) application of logical rules of DFD construction (for example, every outflow must have a corresponding inflow).

Quick Quiz: Data Flow Diagrams

1) List at least three different types of data flow diagrams (DFDs).
   ANSWER: Types of DFDs include context diagrams, event-partitioned system models, subsystem DFDs, diagram 0, DFD fragments, process decomposition, physical DFDs, and logical DFDs.

2) Describe a physical data flow diagram (DFD).
   ANSWER: A physical DFD is any DFD that shows the implementation specifics of one particular way of implementing a system.

3) Describe a logical data flow diagram (DFD).
   ANSWER: A logical DFD is any DFD that shows system requirements under the assumption of perfect technology.

Documentation of DFD Components

This section describes various techniques for further documenting the details of DFD components. Subsections cover methods for describing processes, data flows, and data stores.

Remember that a DFD lacks many details that a system designer or programmer needs to know to implement a complete system. For example, would you be able to write a program to
implement the subsystem in Figure 6-11 without asking the user any questions. You should quickly realize that additional model components are needed to provide the details needed to write programs.

**Process Descriptions**
The three primary techniques for documenting process logic and detail are:

- Process descriptions (structured English)
- Decision tables
- Decision trees

Regardless of which method is used, a process description must be specific enough to allow programs to be written. A process description, decision table, or decision tree should also fit on one page (otherwise, a process decomposition DFD is drawn).

With each method, the analyst chooses the most appropriate presentation format by determining which is most compact, readable, and unambiguous. The best format will vary from process to process. The analyst may have to prepare process descriptions in multiple formats to determine which is the most appropriate.

A structured English process description (see Figures 6-20, 6-21, and 6-22) is similar to a program. It uses a precise subset of English to describe an algorithm or procedure. As with program source code, indentation is used to enhance readability and emphasize control structure. Structured English is well-suited to processes containing many simple sequential steps and those with relatively simple control structures (for example, a single if statement or loop).

Decision tables (see Figure 6-23) and decision trees (see Figure 6-24) are graphical representations of a process. They are well-suited to representing processes with complex decision logic and large numbers of decision variables or decision variables values in a relatively compact space.

Students sometimes assume that decision trees are preferred over decision tables because they provide a graphical process description. In fact, decision trees are used less frequently than decision tables because a complex process can’t usually be represented in one page. Decision tables can more compactly represent the same decision (for example, compare the size of Figures 6-24 and 6-25). However, neither decision tables nor decision trees are useful when a process contains relatively simple logic and sequential processing steps.

**Data Flow Definitions**
Data flow definitions describe the content and structure of data flows. The two most common formats of data definitions are:

- Data element lists (see Figure 6-26).
- Algebraic definitions (see Figures 6-27 and 6-29).

Data flows without a complex internal structure (for example, embedded repeating groups) can be represented by simply listing their component data element names. Data flows with
a complex internal structure require a specific notation to represent that structure. The notation uses the following symbols:

- + to represent concatenation of data elements.
- {} to represent a repeating group of data elements.

**Data Store Definitions**

Data store definitions are generally omitted because they are redundant with the ERD.

**Data Element Definitions**

Data elements can be defined by short textual descriptions (see Figure 6-30). The description usually specifies the content type (for example, numeric or alphanumeric), maximum length (if necessary), and allowable values (if the data element contains coded data). The analyst should avoid excessively detailed data element definitions because defining appropriate format and content is usually a design and implementation decision.

Students usually have trouble with this section because they do not know when a data element needs a definition and when it does not. The general rule is that a definition is not needed when everyone involved in the project (including programmers that haven’t yet been brought on board) understands data element content. This is a difficult rule to apply in practice because students invariably start to think in terms of programming (for example, alphanumeric strings of specific lengths).

A well-chosen name (for example, First-Name, Street-Address) is usually sufficient. Analysis is not the time to be arguing about whether a separate field is needed for Mr., Mrs., or Ms. and whether four or five lines are required for a street address. Such decisions are design decisions best left for later development phases. Exceptions to the above guidelines include externally imposed formats (for example, outputs flowing to a governmental agency) and data elements with coded content (for example, the “special” data element in Figure 6-30).

**DFD Summary**

Together, DFDs, process descriptions, data definitions, and the ERD form an interlocking set of models (see Figure 6-31). An ideal set of analysis models is mutually exclusive and collectively exhaustive—that is, each analysis “fact” is represented by only one of the modelling techniques, and all relevant facts are (or can be) represented.

**Quick Quiz: Documentation of DFD Components**

Why might an analyst describe a structured process with a decision table or tree instead of structured English?

**ANSWER:** Decision tables and trees are used when they improve readability more effectively than structured English. This is often the case for processes with a large number of decision variables and relatively simple processes for each combination of decision variable values.

**Locations and Communication through Networks**

IE represents information about process and data storage locations using three different
models:

• Location diagrams
• Activity-location matrices
• Activity-data matrices

The location diagram (see Figure 6-32) is simply a map (or set of maps) that shows each location where data is stored or processed. The activity-location matrix (see Figure 6-33) is a table that shows each location where an activity on the event table is performed. The activity-data matrix (see Figure 6-34) shows how data in each data store (entity) is used at each location. Possible uses include create (abbreviated as C), read (abbreviated as R), update (abbreviated as U) and delete (abbreviated as D). The table is sometimes called a CRUD matrix based on the abbreviations.

This section discusses the most important contribution of IE to traditional structured development. Structured development considers issues such as location to be unimportant during analysis. Philosophically, the location of a process or data store is just another design decision that is best left with all of the other design decisions.

IE takes a less philosophical view of geography. The reality of most complex business operations is that businesses and business units don’t pick up and move on a whim. IE considers geographic location a fixed requirement instead of a design decision. Thus, IE gathers extensive information about location during analysis.

If the analyst knows that the designer will have carte blanche to address the issue of geography, then the “purist” approach of structured analysis may be preferred. But in the more typical case where geography is not negotiable, it is important to gather and model location information during analysis because this information will drive many high-level design decisions.

Geography can be correctly considered as a requirement or a design decision. Even if there is some design flexibility in determining processing and data storage locations, it is still a good idea to build activity-location and activity-data matrices during analysis because it is a natural and convenient time to gather that information. But the analyst can’t always assume that the information is cast in stone. Those tables are subject to change during the design phase. Furthermore, geography should not be used as a basis for partitioning processes in upper-level DFDs.

Quick Quiz: Locations and Communications through Networks

1) What is an activity-location matrix?
   ANSWER: An activity-location matrix is a table that shows the location(s) where each processing activity is performed.

2) What is an activity-data matrix?
   ANSWER: An activity-data matrix is a table that shows each processing location and what items of stored data are accessed from that location. The matrix is only loosely related to data flow diagrams.
Solutions to End of Chapter Material

Review Questions

1. List at least three different types of DFDs. What is each diagram type used to represent?
   Types of DFDs include context diagrams, event-partitioned system models, subsystem DFDs, diagram 0, DFD fragments, process decompositions, physical DFDs, and logical DFDs.

   A context diagram contains a single process representing the entire system. The diagram shows important interactions between the system and external agents.

   An event-partitioned system model contains one process per event. The model shows important interfaces with external agents. If no subsystem DFD is created then the event-partitioned system model is also called diagram 0.

   A subsystem DFD contains one process per major subsystem. The DFD shows important interfaces with external agents. Each process is further represented by another DFD – an event-partitioned DFD (or diagram zero) for that subsystem.

   A DFD fragment is a portion of an event-partitioned system model that shows the process, external agents, data stores, and data flows needed to respond to a single event.

   A process decomposition is a DFD that shows the internal implementation details of a single process on another DFD.

   A physical DFD is any DFD that shows the implementation specifics of one particular way of implementing a system.

   A logical DFD is any DFD that shows system requirements under the assumption of perfect technology.

2. List the five component parts (symbols) of a DFD. Briefly describe what each symbol represents.
   Process (a rectangle with rounded corners) – represents an algorithm for transforming data input into data output.

   Data flow (a one or two-headed arrow) – represents the movement of data among processes, data stores, and external agents.

   External agent (a square) – represents a person or organization outside the scope and control of the system that provides data inputs and/or accepts data outputs.

   Data store (a shallow rectangle missing either its left or right side) – data at rest, awaiting future access by a process or between process invocations.

   Real-time link (two-headed arrow with a kink or double bend) – a special type of data flow representing two-direction data movement between a process and an
3. **How does an analyst determine whether a person or organization should be represented on a DFD as an external agent or one or more processes?**

The key issues are system scope and degree of control. If the system has little or no control over the actions of a person, program, or organization then that person or organization should be represented as an external agent (e.g., a customer, bank, or governmental agency). If the system has substantial control over the person, program, or organization then representation using processes, data flows, data stores, and real-time links are usually appropriate. However, the issue of control may be ignored to limit the scope of system development in which case things outside the limited scope are represented as external agents regardless of the issue of control.

4. **Processes on an event-partitioned DFD can be described by a detailed DFD or a process specification. How does an analyst determine which is the most appropriate form of description?**

The key issues are clarity of presentation and size of the model. In theory, the representation that is most readable and understandable is preferred. As a practical matter, processes that can be described in a single decision table or tree or in a short (e.g., less than 20 lines) structured English process description should not be represented with a detailed DFD.

5. **Describe how each column of an event table is represented on a DFD (that is, what symbols are used?).**

See Figure 6-5 for a summary of this answer.

- **Event** – there is no direct representation of the event as a whole.
- **Trigger** – data inflow to a process
- **Source** – external agent from which data inflows arrive.
- **Activity** – process that responds to the data inflow (event).
- **Response** – data outflow from a process.
- **Destination** – external agent(s) to which data outflows are sent.

6. **How are entities from the ERD represented on a DFD? How are relationships from the ERD represented on a DFD?**

Entities are represented by data stores. Relationships are not directly represented but are implicitly represented by the data stores of the related entities.

7. **What features may be present on a physical DFD that should never be present on a logical DFD?**

In general, features that are technology-specific may appear on a physical DFD but not a logical DFD. Also, processes, data flows, and data stores that would not be required if the system were implemented with perfect technology are never present on a logical DFD. Direct examples include processes that represent specific existing programs or persons (roles). Indirect examples include duplicate processes, data stores, and data flows and partitioning of processes or data flows
8. **What DFD characteristics does an analyst examine when evaluating DFD quality?**
Quality of a DFD boils down to accuracy and readability. Accuracy is best evaluated during a walkthrough with the user. Potential inaccuracies can be discovered by looking for logical inconsistencies, such as black holes and miracles. Readability can be evaluated by measuring overall complexity of each model component and evaluating it with the rule of seven plus or minus two.

9. **What is a black hole? What is a miracle? How can each be detected?**
A black hole is a process or file into which data enters but never leaves. Black holes are discovered by comparing the content of data outflows to data inflows. Data flowing in must either flows out or be used to generate data that flows out.

A miracle is a data outflow that appears from a process or data store without any corresponding data inflow(s). Miracles are discovered by comparing the data content of data outflows to data inflows. Data outflows with no corresponding inflow and data outflows that can’t be generated from data inflows are miracles.

10. **Why might an analyst describe a process with a decision table or tree instead of structured English?**
An analyst uses decision tables and trees are used when they improve readability. This is often the case for processes with a large number of decision variables and relatively simple processing for each combination of decision variable values.

11. **What is an activity-location matrix? How is it related to DFDs?**
An activity-location matrix is a table that shows the location(s) in which each processing activity is performed. Each process (or group of processes) on a DFD should appear as a row or column of the activity-location matrix.

12. **What is an activity-data matrix? How is it related to DFDs and the ERD?**
An activity-data matrix is a table that lists each processing location and what items of stored data are accessed from that location. The matrix is only loosely related to data flow diagrams. Data stores on DFDs represent entities in the IE entity-relationship. Each process on a DFD that reads or write stored data must be specified as to location and that information is used to generate the activity-data matrix.

**Case Studies**

**Case Study: The Real Estate Multiple Listing Service System**

1. **Draw a context DFD**
2. Draw an event-partitioned DFD.

3. Draw any required process decomposition DFDs. No process decomposition DFDs are required. The most complex processes are #2 and #3 and it should be possible to describe each in one page or less of structured English.

Case Study: Rethinking Rocky Mountain Outfitters

1. Develop DFD fragments for all of the events not documented in Figure 6-12.
RMO Order Fulfillment Subsystem DFD Fragments

1. Customer or Management
   - Customer Status Inquiry
     - Order Status Details
     - Order Look up Order Status
       - Customer
       - Order
       - Order Item
       - Shipment
       - Shipper

2. Shipping
   - Order Fulfillment Notice
     - Record Order Fulfillment
       - Order
       - Order Item
       - Shipment
       - Shipper

3. Customer
   - Backorder Notice
     - Record Backorder
       - Customer
       - Order
       - Order Item

4. Management
   - Fulfillment Summary Reports
     - Produce Fulfillment Summary Reports
       - Order
       - Order Item
       - Product Item
       - Return Item
       - Shipment
       - Shipper

5. Bank
   - Transaction
     - Create Order Return
       - Customer
       - Return Item

6. Customer Return Notice
   - Return Confirmation
2. Develop a single DFD that shows processing for all events, using one process for each subsystem and showing all needed data stores. To simplify the diagram, place all external agents along the outer edge and duplicate them as necessary to minimize long or crossing data flows. Place all data stores in the middle of the diagram.

3. Develop a data flow definition for the RMO customer order form in Figure 6-35.

Customer-Order-Form = Order-Date + Name-And-Address + Day-Phone + Evening-Phone + (Ship-To-Name-And-Address + Gift-Check-Box + [Address-For-This-Shipment-Only-Check-Box | Permanent-Change-Of-Address-Check-Box ] + Delivery-Phone ) + 1{Item-Detail} + Payment-Detail + Totals

Name-And-Address = Name + Address-Line-1 + Apartment-Number + Address-Line-2 + City + State + Zip
Ship-To-Name-And-Address = Name + Address-Line-1 + Apartment-Number + Address-Line-2 + City + State + Zip

Item-Detail = Item-Number + Description + Style + Colour + Size + Sleeve-Length + Quantity + Monogram + Monogram-Style + Price-Each + Item-Total

Payment-Detail = [ Check-Money-Order-Check-Box | Gift-Certificate-Check-Box | American-Express-Check-Box | MasterCard-Check-Box | Visa-Check-Box | Other-Check-Box + Other-Description ] + (Account-Number + Expiration-Date + Signature)

Totals = Merchandise Total + Total-Monogramming-Charges + Sales-Tax + Minimum-Shipping-Charge + Additional-Address-Shipping-Charge + FedEx-Standard-Overnight-Charge + Additional-Freight-Charges + International-Shipping-Charge

Key Terms

- **Activity-data matrix** – A table that describes stored data entities, the locations from which they are accessed, and the nature of the accesses.
- **Activity-location matrix** – A table that describes the relationship between processes and the locations in which they are performed.
- **Balancing** – Equivalence of data content between data flows entering and leaving a process and data flows entering and leaving a process decomposition DFD.
- **Black hole** – A process or data store with a data input that is never used to produce a data output.
- **Context diagram** – A DFD that summarizes all processing activity within the system in a single process symbol.
- **CRUD** – Acronym of create, read, update, and delete.
- **Data dictionary** – A repository for definitions of data flows, data elements, and data stores.
- **Data flow** – An arrow on a DFD that represents data movement among processes, data stores, and external agents.
- **Data flow definition** – A textual description of a data flow’s content and internal structure.
- **Data flow diagram (DFD)** - A diagram that represents system requirements as processes, external agents, data flows, and data stores.
- **Data store** – A place where data are held pending future access by one or more processes.
- **Decision table** - A tabular representation of processing logic containing decision variables, decision variable values, and actions or formulas.
- **Decision tree** – A graphical description of process logic that uses lines organized like branches of a tree.
- **DFD fragment** – A DFD that represents the system response to one event within a single process symbol.
- **Event-partitioned system model, or diagram 0** – A DFD that models system requirements using a single process for each event in a system or subsystem.
- **External agent** – A person or organization, outside the system boundary, that supplies
data inputs or accepts data outputs.

- **Information Engineering (IE)** – A system development methodology that focuses on strategic planning, data modelling, and automated tools, and is thought to be more rigorous and complete than the structured approach.

- **Information overload** – Difficulty in understanding that occurs when a reader receives too much information at one time.

- **Level of abstraction** – Any modelling technique that breaks the system into a hierarchical set of increasingly more detailed models.

- **Location diagram** – A diagram or map that identifies all of the processing locations of a system.

- **Minimization of interfaces** – A principle of model design that seeks simplicity by limiting the number of connections among model components.

- **Miracle** – A process or data store with a data element that is created out of nothing.

- **Process** – A symbol on a DFD that represents an algorithm or procedure by which data inputs are transformed into data outputs.

- **Process decomposition diagram** – A model that represents the hierarchical relationship among processes at different levels of abstraction.

- **Process dependency diagram** – A model that describes the ordering of processes and their interaction with stored entities.

- **Rule of 7+/− 2** – The rule of model design that limits the number of model components or connections among components to no more than nine.

- **Structured English** – A method of writing process specifications that combines structured programming techniques with narrative English.
Chapter 6 – Elements of Systems Design

This chapter corresponds with chapter 9 of the text book.

Chapter Overview
Note: chapters mentioned in the text below refer to the chapters in the text book.

Chapter 8 described the activities that complete the analysis phase and begins the transition from analysis to design. This chapter completes that transition and discusses issues related to the design of the new system. While analysis focuses on what the system should do – that is, the requirements – design is oriented toward how the system will be built – on defining structural components. For example, defining the application deployment environment and determining levels of automation are tasks begun during analysis but completed in more detail during design.

Learning Objectives
After studying this chapter, you should be able to:

• Discuss the issues related to managing and coordinating the design phase of the SDLC.
• Explain the major components and levels of design.
• Describe each design phase activity.
• Develop a simple network diagram.
• Describe common development environments and matching application architectures.

Notes on Chapter Opening Case and Case Studies

Fairchild Pharmaceuticals: Finalizing Architectural Design for a Production System: This case describes the transition from analysis (detailed knowledge of what the user wants and needs) to design (precise blueprint of a system that will satisfy those wants and needs). Design decisions are constrained by available time, budget, existing systems, skills, and infrastructure.

The focus of the case is on the architectural design of infrastructure and development tools in order to finalize hardware and operating systems, data storage and data access, and development languages and tools. Choices need to be made regarding Web support services, database design, application software design, and user interface design.

The summer intern in the case recently completed a study on infrastructure requirements and communications protocols for Web services using COM+, COBRA, and SOAP. The intern and project manager concluded the case with a plan for the two to work together on architectural design and then detailed design.

The Real Estate Multiple Listing Service System: This case builds on the Chapter 8 three-layer.
Rethinking Rocky Mountain Outfitters: This case uses the Chapter 8 alternative deployment scenario for RMO based on Apache Web servers running under Linux and an Oracle database server.

Focusing on Reliable Pharmaceutical Service: This case uses the Chapter 8 case exercise and a selected vendor description (RxTechSys) to develop a design plan and schedule to manage and control the project.

Additional Notes and Quick Quizzes

Project Management Revisited: Execution and Control of Projects

In earlier chapters of the book, particularly in Chapter 3, you learnt the concepts relating to the planning and initiation of a project. In subsequent chapters, you learnt specific skills for fact finding and modelling user requirements. In this section we introduce additional project management concepts related to the control and execution of development projects. These are the concepts and skills that are required to actually manage an alive, ongoing development project. We address four areas of running a project:

- Organizing teams and assigning work
- Communicating status and information
- Monitoring and controlling project progress
- Controlling project issues and risks

Organizing project teams and assigning work

The best project managers view themselves as resources to clear roadblocks so that the team members can get the work of the project completed. The best development teams can outperform mediocre teams by as much as ten times. Hence it is good practice to do all those things to build a good development team.

Team Organization

Different project managers have different approaches to organizing teams. Some of the key questions to be answered when organizing a team include:

- Assign a team leader for each sub team or let the sub teams organize themselves?
- Assign members permanently to a team or have floating team assignments?
- Assign team members to sub teams to achieve a balance based on (1) skills, (2) experience, or (3) personality traits?
- Balance team membership based on permanent (core) members or transient members?

Assigning Tasks to Team Members

Project managers also have preferences to assigning tasks. Different projects require different approaches. Here are a few issues that must be addressed as team members receive assignments.

- What is the formality (versus informality) of the project (schedule, assignments, status, and so on)?
- Should tasks be assigned to sub teams or to key individuals?
• Should tasks be assigned well in advance or using a just-in-time approach?
• Is the project schedule stable or is it a changing schedule?
• How do the number and duration of critical-path tasks compare to the number of tasks that are not on the critical path?
• Should tasks be assigned based on specific skills or on availability?
• Should tasks be assigned so that team members are fully scheduled or should open times be provided on people’s schedules?

Managing the Communication Process
Communication requires a good balance to keep the project team informed yet not wasting time with many inefficient and unproductive meetings. Communication management is a two step process. The first step is to determine what information is important to collect, store, and distribute. Two types of information are important: project status information and information about the solution system. Once it is determine what information is important, then the mechanisms for collecting, storing, and distributing the information need to be designed.

In today’s high-tech world, most project managers utilize database systems and electronic ways to collect, store and distribute information. A centralized data repository is a good way to maintain all the information. A project dashboard, accessible on the Internet is an effective way to distribute project status information to all interested parties.

Monitoring the Project Plan
Assigning work and keeping the project on schedule is one of the toughest responsibilities of the project manager. The theory is rather simple, assign tasks and make sure they get done. Unfortunately the reality of it is much more complex and difficult. However, without the basic process in place, a project manager will soon be in serious trouble. The flowchart in the chapter illustrates the fundamental process: assign work, collect status; if not on schedule determine why; and take corrective action. This process, of course, is dependent on having some type of project plan and schedule which identifies all the tasks that need to be done. Without a plan, then project execution is simply doing work as someone thinks of them. That may be a fun way to work, but it does not allow any predictions about completion or cost.

Controlling Issues and Risks
Every project has unexpected issues arise, or problems that do not get resolved immediately. A mechanism must be put in place to track these issues so that first they don't get forgotten and second they get resolved in a timely fashion. The solution is usually nothing more than a simple tracking log, maintained on a spreadsheet or some type of tracking system. There are many Internet based bug and issue tracking systems available on the market today.

The Project Team at RMO
As the customer support system project moves forward into design at RMO, the project team has added new team members and has initiated two subprojects, one for data conversion and another for the system and acceptance test plans. To integrate new people into the team, the project manager reorganized the structure of the project team. Those who had been on the team throughout the analysis phase are now key players in getting the new team members up to speed.
Quick Quiz: Executing and Controlling the Project

1) What are the four principle processes that must be done to execute and control the project?
ANSWER: (1) Organize the teams and assign the work, (2) Collect and communicate status and project information, (3) Monitor and control project progress, and (4) Control project issues and risks.

2) What are two critical questions that must be answered to set up project communications?
ANSWER: First what information needs to be collected and stored, and second how to do it, e.g. the mechanisms and techniques to actually collect, store, and distribute the information.

Understanding the Elements of Design

Systems design is the process of describing, organizing, and structuring the components of a system at both the architectural level and a detailed level with a view toward constructing the proposed system. Systems design is like a set of blueprints used to build a house. The blueprints are organized by the different components of the house and describe the rooms, the stories, the walls, the windows, the doors, the wiring, the plumbing, and all other details. We do the same organizing in systems design, although the components we are describing are the components of the new system. We design and specify various components of the final solution.

To understand the various elements of systems design, we must consider two questions:

- What are the components that require systems design?
- What are the inputs to and outputs of the design process?

Major Components and Levels of Design

To do design, analysts first partition the total system into its major components because an information system is much too complex to design all at once. Each of the design phase activities develops a specific portion of the final set of design documents. Just as a set of building blueprints consists of several different documents, a systems design package consists of several different documents that specify the entire system. Figure 9-6 illustrates the following components of design: application design, database design, user interface design, system interface design, and network design.

As you begin to work in industry, you will find that various names are given to the design at the highest level, including architectural design, general design, and conceptual design. During architectural design, you first determine the overall structure and form of the final solution before trying to design the details. Designing the details is usually called detail design. It is not so important at this point to distinguish which activities are architectural design and which activities are detail design. What is important is to recognize that design should begin in a top-down fashion.

Inputs for System Design

You should study Figure 9-8 in detail and use it as a reference when studying the
The original definition of design indicates that design involves describing, organizing, and structuring. The output of the design activities is a set of diagrams and documents that achieve that objective. Figure 9-8 illustrates the design models for both traditional structured design and object-oriented design. The set of models for structured design includes system flowcharts, structure charts, and relational database schema. The set of models for object-oriented design includes package diagrams, design class diagrams, and object database schema. Other models and design components, such as user interface design, system security controls, and nodes and locations diagrams, are common to both structured design and object-oriented design.

A frequently asked question is, “Can structured techniques and object-oriented techniques be mixed? In other words, is it possible to do structured analysis and then object-oriented design of the application or vice versa?” In some situations, it might be possible to mix and match, such as when designing and implementing the interface using OO after completing traditional structured analysis. But generally, such mixing and matching does not work well for application design, because the basic philosophies of the two techniques are so fundamentally different. The design of the application software using a traditional approach provides an architectural structure based on the top-down procedural functions of the system. A system designed using object-oriented techniques has an architectural structure based on a set of interacting objects for each use case.

Quick Quiz: Understanding the Elements of Design

1) What is the difference between architectural design and detailed design?
   ANSWER: Architectural design (also called general design or conceptual design) first determines the overall system structure and form of the final solution. Detailed design uses the architectural design artefacts to design the lower-level details.

2) What is the primary objective of systems design?
   ANSWER: The primary purpose of systems design is to develop a structure or roadmap that can be used for programming—that is, to take the requirements that were defined in analysis and organize them in a way that allows programming to occur.

Design Activities

Figure 9-9 identifies the activities that are associated with the design phase. The design phase involves specifying in detail how a system will work using a particular technology. Some of the design details will have been developed during systems analysis when the alternatives were described. But much more detail is required. Sometimes systems design work is done in parallel with the analysis phase, and usually the activities within systems design are done in parallel. For example, the database is typically designed at the same time the user interface is designed.

Each of the activities develops a specific portion of the final set of design documents. Just as a set of building blueprints has several different documents, a systems design package also consists of several different sets of documents that specify the entire system. In addition, just as all of the blueprints must be consistent and integrated to describe the same physical building, the various systems design documents also must be integrated to provide a comprehensive set of specifications for the total system.
An iterative approach to the SDLC is commonly used, in which some of the design phase activities are completed in the first iteration, and other design phase activities are completed in subsequent iterations.

**Design and Integrate the Network**

Sometimes a new system is implemented along with a new network. In this case, the network needs to be designed. More often, though, network specialists have established the network based on an overall strategic plan. The key question to be answered when completing the Design and integrate the network activity is, “Have we specified in detail how the various parts of the system will communicate with each other throughout the organization?”

**Design the Application Architecture and Software**

Designing the application architecture involves specifying in detail how all system activities will actually be carried out. These activities are described in great detail during systems analysis as logical models, without indicating what specific technology would be used. After a specific design alternative is chosen, the detailed computer processing—the physical models—can be designed. Models created include physical data flow diagrams, structure charts, interaction diagrams, and other physical models. The key question to be answered when completing the ‘Design the application architecture’ activity is, “Have we specified in detail how each system activity is actually carried out by the people and computers?”

When an iterative approach to the SDLC is used, designing the application architecture and necessary design activities are completed in the first iteration, and other design phase activities are completed in subsequent iterations.

**Design the User Interfaces**

A critical aspect of the information system is the quality of the user interface. The design of the user interface defines how the user will interact with the system. To most users, the interface is a graphical user interface with windows, dialog boxes, and mouse interaction. Increasingly, it can include sound, video, and voice commands.

Analysts should remember that to the user of the system, the user interface is the system. The user interface is more than just the screens—it is everything the user comes into contact with while using the system, conceptually, perceptually, and physically. Therefore, the user interface is not just an add-on to the system. It is something that needs to be considered throughout the development process. The key question to be answered when completing the ‘Design the user interfaces activity’ is, “Have we specified in detail how all users will interact with the system?”

**Design the System Interfaces**

No system exists in a vacuum. Many other information systems are affected by a new information system. Sometimes one system provides information that is later used by another system. Other times systems exchange information continuously as they run. The component that enables systems to share information is the system interface, and each system interface needs to be designed in detail. The key question to be answered when completing the ‘Design the system interfaces’ activity is, “Have we specified in detail how the system will work with all other systems inside and outside our organization?”
Design and Integrate the Database
Designing the database for the system is another key design activity. The data model (a logical model) created during systems analysis is used to create the physical model of the database. Sometimes the database is a collection of traditional computer files. More often, it is a relational database consisting of dozens or even hundreds of tables. Sometimes files and relational databases are used in the same system. Sometimes object-oriented databases are used instead of relational databases. The key question to be answered when completing the ‘Design and integrate the database’ activity is, “Have we specified in detail how and where the system will store all of the information needed by the organization?”

Prototype for Design Details
During the design phase, it is important to continue creating and evaluating prototypes. Often associated with interface design, prototyping can also be used to confirm design choices about the database, network architecture, controls, or even programming environments being used. Therefore, when analysts consider all of the design activities, they think about how prototypes might be used to help understand a variety of design decisions. The key question to be answered when completing the ‘Prototype for design details’ activity is, “Have we created prototypes to ensure that all detailed design decisions have been fully understood?”

As rapid application development becomes more prevalent, prototypes that prove that design concepts are practical and evolve into the finished system becomes more common.

Design and Integrate the System Controls
A final design activity involves ensuring that the system has adequate safeguards to protect organizational assets. These safeguards are referred to as system controls. User-interface controls limit access to the system to authorized users. System interface controls ensure that other systems cause no harm to this system. Application controls ensure that recording transactions and other work done by the system are done correctly. Database controls ensure that data is protected from unauthorized access and from accidental loss due to hardware failure. Finally, and of increasing importance, network controls ensure that communication through networks is protected.

Control issues are addressed in several chapters but more explicitly in Chapter 15. The key question to be answered when completing the Design and integrate the systems controls activity is, “Have we specified in detail how we can be sure that the system operates correctly and the data maintained by the system is safe and secure?”

Quick Quiz: Design Phase Activities

1) What are the six activities associated with system design?
   ANSWER: Design the network, design the software, design the user interface, design the system interface, design the database, prototype if necessary, design system controls.

2) What is the difference between user interfaces and system interfaces?
   ANSWER: User interface design requires human-computer interface evaluations and often includes things like story boards and workflow analysis. System interfaces are those interfaces that go to other computer systems and are normally electronic in nature.
Network Design

Networks are used throughout organizations, and many new development projects involve network design. Network planning and design are critical issues that must be dealt with early in the design phase for any multitiered system. The key design issues are as follows:

- Integrating network needs of the new system with existing network infrastructure.
- Describing the processing activity and network connectivity at each system location.
- Describing the communication protocols and middleware that connect layers.
- Ensuring that sufficient network capacity is available.

Computer Networks

A computer network is a set of transmission lines, specialized hardware, and communication protocols that allow communication among different users and computer systems. Computer networks are divided into two classes depending on the distance they span. A local area network (LAN) is typically less than one kilometre in length and connects computers within a single building or floor. A wide area network (WAN) describes any network over one kilometre, though the term typically implies much greater distances spanning cities, countries, continents, or the entire globe.

There are many ways to distribute information system resources across a computer network. Users, application programs, and databases can be placed on the same computer system, on different computer systems on the same LAN, or on different computer systems on different LANs. Application programs and databases can also be subdivided and each distributed separately.

The Internet, Intranets, and Extranets

The Internet and World Wide Web are becoming increasingly popular frameworks for implementing and delivering information system applications. The Internet is a global collection of networks that are interconnected using a common low-level networking standard—TCP/IP (Transmission Control Protocol/Internet Protocol). The World Wide Web (WWW), also called simply the Web, is a collection of resources (programs, files, and services) that can be accessed over the Internet by a number of standard protocols. Resources of the Web are delivered to users over the Internet.

An intranet is a private network that uses Internet protocols but is accessible only by a limited set of internal users (usually members of the same organization or workgroup). The term also describes a set of privately accessible resources that are organized and delivered via one or more Web protocols over a network that supports TCP/IP. An intranet uses the same protocols as the Internet and Web but restricts resource access to a limited set of users. An extranet is an intranet that has been extended to include directly related business users outside the organization (for example, suppliers, large customers, and strategic partners). An extranet allows separate organizations to exchange information and coordinate their activities, thus forming a virtual organization.

Virtual private network: A network that has security and controlled access for a private group but is built on top of a public network such as the Internet.

Network Integration

Modern organizations rely on networks to support many different applications. Therefore,
the majority of new systems must be integrated into existing networks without disrupting existing applications. Network design and management are highly technical tasks, and most organizations have permanent in-house staff, contractors, or consultants to handle network administration. The systems analyst begins network design by consulting with the organization’s network administration to determine whether the existing network capacity and configuration can accommodate the new systems.

**Network Description**
Location-related information gathered during analysis may have been documented using location diagrams, activity-location matrices, and activity-data matrices. During network design, the analyst expands the information content of this analysis to include processing locations, communication protocols, middleware, and communication capacity. Figure 9-12 shows a network diagram that describes how application layers are distributed across locations and computer systems for the RMO customer support system.

The network diagram summarizes key architectural decisions and combines them with specific assumptions about where application software will execute, where servers and workstations will be located, and how network resources will be organized. Decisions such as server locations, communication routes, and network security options are determined both by application requirements and organization-wide policies.

**Communication Protocols**
The network diagram is also a starting point for specifying protocol and middleware requirements. For example, the private WAN connections must support protocols required to process Microsoft Active Directory logins and queries.

**Network Capacity**
Information from the activity-location and activity-date matrices (Figure 9-13) is the starting point for estimating communication capacity requirements for the various LAN, WAN, and Internet connections. Data size per access type is an educated guess at this point in the systems design. After the software layers, interlayer communication dialogs, or databases have been designed in more detail or implemented, analysts can refine their estimates and sample and measure real data transmissions. Actual data transmission capacity will include communication protocols in addition to raw data.

**Quick Quiz: Network Design**

1) What is a network diagram?
**ANSWER:** A network diagram is a model that shows how application layers are distributed across locations and computer systems.

2) What information does a network diagram convey and where does the analyst gather that information?
**ANSWER:** The network diagram summarizes key architectural decisions and combines them with specific assumptions about where application software will execute, where servers and workstations will be located, and how network resources will be organized.
Deployment Environment and Application Architecture

The application architecture involves the structure and organization of the new software system.

**Single-Computer and Multitier Architecture**

As its name implies, single-computer architecture employs a single computer system and its directly attached peripheral devices. Users interact with the system via simple input/output devices that are directly connected to the computer. Single-computer architecture requires all system users to be located near the computer. The primary advantage of single-computer architecture is its simplicity.

**Clustered architecture** employs a group (or cluster) of computer systems to provide needed processing or data storage and retrieval capacity. Computers from the same manufacturer and model family are networked together. Similar hardware and operating systems allow application programs to execute on any machine in the cluster without modification. In effect, a cluster acts as a single large computer system. Clustered computer systems are normally located near one another so that they can be connected with short high-capacity communication links.

**Multicomputer architecture** also employs multiple computer systems, but hardware and operating systems are not required to be as similar as in clustered architecture. Hardware and software differences make it impractical to move application programs from one machine to another. Instead, a suite of application programs and data resources is exclusively assigned to each computer system. Even though this architecture is similar to a distributed configuration (discussed in the next section), we classify it as a centralized system because it functions as a single large computer.

**Centralized and Distributed Architecture**

**Centralized architecture** is generally used for large-scale batch and real-time processing applications. These applications are common in industries such as banking, insurance, and catalogue sales. Information systems in these industries often have the following characteristics:

- Some input transactions do not need to be processed in real-time (for example, out-of-state checks delivered in large nightly batches from central bank clearinghouses).
- Online data-entry personnel can be centrally located (for example, a centrally located group of telephone order takers can serve geographically dispersed customers).
- Large numbers of periodic outputs are produced by the system (for example, monthly credit card statements mailed to customers).
- A high volume of transactions occurs between high-speed computers (for example, business-to-business processing for supply chain management).

Any application that has two or three of these characteristics is a viable candidate for implementation on a centralized mainframe.
Components of a modern information system are typically distributed across many computer systems and geographic locations. For example, corporate financial data might be stored on a centralized mainframe computer. Personal computers in many locations might be used to access and view periodic reports as well as to directly update the central database. Such an approach to distributing components across computer systems and locations is generically called **distributed architecture**.

**Client/Server Architecture**

Client/server architecture is the dominant architecture for modern software. Client/server architecture divides information system processes into two types: client and server. A **server** manages one or more system resources and provides access to those resources through a well-defined communication interface. A **client** uses the communication interface to request resources, and the server responds to those requests.

The architectural issues to be addressed when designing client/server software are as follows:

- Decomposing the application into client and server programs, modules, or objects. The key to decomposing the application is to identify resources or services that can be centrally managed by independent software units.
- Determining which clients and servers will execute on which computer systems. The most typical arrangement is to place server software on separate server computer systems and to distribute client software to computer systems “close” to end users, such as desktop workstations.
- Describing the communication protocols and physical networks that connect clients and servers.

The primary advantage of client/server architecture is deployment flexibility (location, scalability, and maintainability).

The primary disadvantages of client/server architecture are the additional complexity introduced by the client/server protocols and the potential performance, security, and reliability issues that may arise when communicating over networks.

**Three-Layer Client/Server Architecture**

Application software can be divided into a set of client and server processes or layers independent of hardware or locations. This approach to client/server architecture is sometimes called **three-layer architecture**. The most common set of layers includes:

- **Data layer**: Manages stored data, usually in one or more databases.
- **Business logic layer**: Implements the rules and procedures of business processing.
- **View layer**: Accepts user input and formats and displays processing results.

Three-layer architecture is currently a widely applied architectural design pattern with both the traditional approach and the object-oriented approach. The key design tasks are decomposing the application into layers, clients, and servers, distributing the components across hardware platforms, and defining the physical network and protocols.

**Internet and Web-Based Application Architecture**

Internet and Web technologies present an attractive alternative for implementing
information systems. For example, to allow buyers to access applications remotely, RMO could construct an application that uses a Web browser interface, executes on a Web server, and is accessible from any computer with an Internet connection. Buyers could then connect to the system from remote locations to gain access to the application server, make queries to the database, and enter data.

Implementing an application using the Web, an intranet, or an extranet has a number of advantages over traditional client/server approaches, including accessibility, low-cost communication, and widely implemented standards.

Of course, there are negative aspects of application delivery via the Internet and Web technologies, including security, reliability, throughput, and volatile standards.

Web Services Architecture

Web services architecture packages software into server processes that can be accessed via Web protocols. Web services architecture provides a flexible mechanism for making software services available to both internal and external clients. It is widely used to create a unified system by combining software services distributed across multiple organizations and computers. For example, RMO could employ external credit verification, shipment, and inventory replenishment Web services in its new online ordering system.

Middleware

Client/server and three-layer architecture relies on specific software to enable communications between the various layers. Middleware connects parts of an application and enables requests and data to pass between them. Common types of middleware include teleprocessing monitors, transaction processing monitors, and object request brokers (ORBs). Each type of middleware has its own set of protocols to facilitate communication between the components of an information system. When specifying the protocols to be used for client/server or interlayer communication, the designer usually relies on standard frameworks and protocols incorporated into middleware.

Quick Quiz: Deployment Environment and Application Architecture

1) What is the difference between centralized architecture and distributed architecture?
   ANSWER: Centralized architecture is architecture that locates all computing resources in a central location. Distributed architecture is architecture that deploys computing resources in multiple locations connected by a computer network.

2) What is an intranet?
   ANSWER: An intranet is a private network that uses Internet protocols but is accessible only by a limited set of internal users (usually members of the same organization or workgroup). The term also describes a set of privately accessible resources that are organized and delivered via one or more Web protocols over a network that supports TCP/IP.

3) What are the primary advantages and disadvantages of client/server architecture?
   ANSWER: The primary advantage of client/server architecture is deployment flexibility (location, scalability, and maintainability). The primary disadvantages of client/server architecture are the additional complexity introduced by the client/server protocols and the
potential performance, security, and reliability issues that arise from communication over networks.

4) What is the business logic layer in three-layer architecture?
ANSWER: The business logic layer in three-layer architecture is the part of three-layer architecture that contains the programs that implement the business rules of the application.

**Solutions to End of Chapter Material**

**Review Questions**

1. **What is the primary objective of system design?**
   Its primary purpose is to develop a structure or roadmap that can be used for programming. It is to take the requirements that were defined in analysis and organize them in a way so that programming can occur.

2. **What is the difference between analysis and design? List the activities of the design phase of the SDLC.**
   Analysis is for discovery and understanding. Its purpose is to develop requirements. Design is for structuring and developing a solution. While analysis looks at problem issues and problem domain, design looks at solution issues and the solution system.

   The activities of the design phase of the SDLC include:

   - Design and integrate the network
   - Design the application architecture
   - Design the user interfaces
   - Design the system interfaces
   - Design and integrate the database
   - Prototype for design details
   - Design and integrate the system controls

3. **Why is project management so critical during the design phase? What tools can a project manager use during the design phase?**
   As design moves forward, the development teams begin to generate a tremendous amount of detailed information about the system. Project management and coordination is needed to plan the design and to keep track of all of these pieces of information.

   The most common technique for recording and tracking project information is to use a CASE tool. Most CASE tools have a central repository to capture information. Tracking open items and unresolved issues is an especially difficult part of project development. Keeping an open items control log can help with this task.

4. **Explain the difference between centralized architecture and distributed architecture.**
   Centralized architecture describes deployment of all computer systems in a single location.
Distributed architecture deploys computing resources in multiple locations connected by a computer network.

5. **Explain the difference between clustered architecture and multicomputer architecture in a centralized system.**

Clustered architecture is a group of computers of the same type that share processing load and act as a single large computer system.

Multicomputer architecture is a group of dissimilar computers that share processing load through specialization of function.

6. **How are the Internet, intranets, and extranets similar? How are they different?**

All three are a collection of networks that are interconnected using TCP/IP (Transmission Control Protocol / Internet Protocol).

The Internet is a global collection of networks, intranets are private networks that are accessible to a limited number of users, and extranets are intranets that have been extended outside the organization to include directly related business users.

7. **Describe client-server architecture and list the key architecture design issues that must be addressed when developing a client-server information system.**

Client-server architecture is the dominant architecture for modern software. Client-server architecture divides information system processes into two types: client and server. A server manages one or more system resources and provides access to those resources through a well-defined communication interface. A client uses the communication interface to request resources, and the server responds to those requests.

The architectural issues to be addressed when designing client-server software are:

- Decomposing the application into client and server programs, modules, or objects. The key to decomposing the application is identifying resources or services that can be centrally managed by independent software units.
- Determining which clients and servers will execute on which computer systems. The most typical arrangement is to place server software on separate server computer systems and to distribute client software to computer systems 'close' to end users, such as desktop workstations.
- Describing the communication protocols and physical networks that connect clients and servers.

8. **List and briefly describe the function of each layer in three-layer architecture.**

Information system application software can be divided into a set of client and server processes or layers independent of hardware or locations. Three-layer architecture includes:

- **Data layer:** Interacts with the database.
- **Business logic layer:** Contains the programs that implement the business rules of the application.
- **View layer:** Contains the user interface and other components to access the system.
9. **What role does middleware play?**
Client-server and three-layer architecture relies on middleware to enable communications between the various layers. Middleware connects parts of an application and enables requests and data to pass between them. Middleware includes teleprocessing monitors, transaction processing monitors, and object request brokers (ORBs).

10. **Describe the process of network design.**
The key network design issues for any multi-tiered systems are:
- Integrating network needs of the new system with existing network infrastructure
- Describing the processing activity and network connectivity at each system location
- Describing the communication protocols and middleware that connect layers.
- Ensuring that sufficient network capacity is available.

11. **What roles do systems analysts and network administrators play in network design?**
Systems analysts consult with network administrators to determine whether the existing system can accommodate the new system by adding connections for servers or modifying routing and firewall configuration to enable new application layers to communicate.

Network administrators assume the responsibility for acquiring new capacity and making configuration changes to support the new system.

12. **What is a network diagram? What information does it convey and where does the analyst gather that information?**
A network diagram is a model that shows how application layers are distributed across locations and computer systems. Analysts gather location-related information from location diagrams, activity-location matrices, and activity-data matrices.

The analyst expands the information content of these documents during network design to include processing locations, communication protocols, middleware, and communication capacity.

13. **How does the analyst generate estimates of required communication capacity? What analysis phase models are used as input?**
Information gathered during analysis from activity-location matrices and activity-data matrices are used as inputs to estimating communication capacity.

Data size per access type and the average and peak number of access per minute or hour are estimates until the software layers, interlayer communication protocols, or databases have been designed. After those components have been designed in more detail or implemented, analysts can refine their estimates or actually sample and measure real data transmissions.

14. **What is Web services architecture? What are some examples of its potential use for business systems?**
Web services architecture packages software into server processes that can be accessed via Web protocols. RMO could employ external credit verification, shipment, and inventory replenishment Web services in its new online ordering system.

Key Terms
- **Architectural design** – Broad design of the overall system structure; also called general design or conceptual design.
- **Business logic layer**
- **Centralized architecture** – Architecture that locates all computing resources in a central location.
- **Client**
- **Clustered architecture** – A group of computers of the same type that share processing load and act as a single large computer system.
- **Computer network** – A set of transmission lines, equipment, and communication protocols to permit sharing of information and resources.
- **Data layer**
- **Detail design** – Low level design that includes the design of specific program details.
- **Distributed architecture** – Architecture that deploys computing resources in multiple locations connected by a computer network.
- **Extranet** – an intranet that has been extended outside the organization to facilitate the flow of information.
- **Interface designers** – Specialists in user-interface design; also called usability consultants or human factors engineers.
- **Internet** – A global collection of networks that use the same networking protocol—TCP/IP.
- **Intranet** – A private network that is accessible to a limited number of users, but which uses the same TCP/IP protocol as the Internet.
- **Local area network (LAN)** – A computer network in which the distances are local, such as within the same building.
- **Middleware**
- **Multicomputer architecture** – A group of dissimilar computers that share processing load through specialization of function.
- **Multitier architecture** – Architecture that distributes application-related software or processing load across multiple computer systems.
- **Network diagram**
- **Router** – Network equipment that directs information within the network.
- **Server**
- **Single-computer architecture** – Architecture that employs a single computer system executing all application-related software.
- **Three-layer architecture**
- **View Layer**
- **Virtual organization**
- **Virtual private network (VPN)**
- **Wide area network (WAN)** – A computer network spread across large distances, such as a city, state, or nation.
- **World Wide Web (WWW), or Web** – A collection of resources such as files and programs that can be accessed over the Internet using standard protocols.
Chapter 7 – The Traditional Approach to Design

This chapter corresponds with chapter 10 of the text book.

Chapter Overview
Note: chapters mentioned in the text below refer to the chapters in the text book.

This chapter describes the traditional approach to designing software. It provides information about the following topics:
- An overview of the structured models, model development process, and related terminology.
- How data flow diagrams (DFD) are annotated with automation boundary information.
- How information from analysis phase models is transformed into design models using system flowcharts, structure charts, and module pseudocode.
- How traditional software design is integrated with other design phase activities.
- How the traditional approach is applied to design a three-layer architecture.

Traditional software design and structured design models have been in use for many years. They are commonly used with systems developed using procedural programming languages. Most new systems are developed with object-oriented programming languages, so traditional system design models are decreasing in popularity. However, many older systems in use today were designed and documented using traditional methods and models. In addition, traditional design concepts such as coupling, cohesion, and top-down partitioning underlie both traditional and object-oriented design methods. Therefore, analysts should be knowledgeable about the traditional approach to design.

Learning Objectives

After studying this chapter, you should be able to:
- Describe the steps involved in the traditional approach to designing the application architecture.
- Develop a system flowchart.
- Develop a structure chart using transaction analysis and transform analysis.
- Write pseudocode for structured modules.
- Explain how to use three-layer design with the traditional approach.

Notes on Chapter Opening Case and Case Studies

Theatre Systems, Inc.: Something Old, Something New: The case illustrates a tug of war between old and new approaches to building system design models. The fresh, young upstart wants to burn down all of the “old stuff,” and the seasoned veteran tries to quell his fire. The case also describes bridging the gap between a structured top-down view of software and a component- and Web-based view of that same software. This information is worth returning to later in the chapter in the section that covers three-layer-design.
On a more general level, the case can serve as a starting point to see how analysts match modelling tools to the intended development and deployment environment. The new system will not be based exclusively on Web, component, or OO technologies; however, it will have a Web-based user interface, and C programs will execute within a Web/application server—presumably via CGI or similar calls from within Web pages. Because the C programming language is not object-oriented, OO modelling techniques are probably not the best way to represent its design, though they may be the best way to represent other aspects of the system design such as the Web-based user interface.

**The Real Estate Multiple Listing Service System:** Develop a structure chart for the system using the case study description in Chapter 5 and the DFDs developed in Chapter 6.

**Rethinking Rocky Mountain Outfitters:** Using the deployment environment and design approaches described in Chapters 8 and 9 and the related traditional design models discussed in Chapter 10, answer the following questions: What are the advantages and disadvantages of designing software using traditional methods and models? What are the advantages and disadvantages of designing software using object-oriented methods and models? These questions could also be used during an in-class discussion.

**Focusing on Reliable Pharmaceutical Service:** Assume that the system will be designed and deployed using three-layer architecture. Develop a system flowchart and a structure chart using the case study description in Chapter 5 and the DFDs developed in Chapter 6.

**Additional Notes and Quick Quizzes**

**The Structured Approach to Designing the Application Architecture**

The application architecture consists of the application software programs that will carry out the system functionality. Application design must be done in conjunction with the design of both the database and the user interface. However, this chapter focuses exclusively on the design of computer software itself for ease of understanding.

Analysts should be knowledgeable about the traditional approach to design because older systems still in use were designed and documented using traditional methods and models.

Many of you have already had experience writing business programs using a third-generation language such as Visual Basic, C, COBOL, or Pascal. Third-generation programming languages are organized around modules that are arranged in a tree-like hierarchy. The top module is often called the boss module or the main module. The middle-level modules are control modules, and the leaf modules are the detail modules that contain most of the algorithms and logic for the program. A module is a small section of program code that carries out a single function. A computer program is the set of these modules that are compiled into a single executable entity.

Frequently, analysts are asked to convert traditional-approach systems to the object-oriented approach to take advantage of object-oriented languages such as Java or .NET.
**Module**: An identifiable component of a computer program that performs a defined function.

**Computer program**: An executable entity made up of a set of modules.

Both the structure of the overall system and any individual subsystems are documented using a system flowchart. The system flowchart identifies each program, along with the data it accesses. The system flowchart also shows the relationships between the various programs, subsystems, and their files or databases. It documents the architectural structure of the overall system.

Finally, the project team must design the internal logic of individual modules. The internal algorithms that compose the logic of the module are usually documented using pseudocode.

**System flowchart**: A diagram that describes the overall flow of control between computer programs in a system.

**Pseudocode**: Structured-programming-like statements that describe the logic of a module.

The inputs for the development of the design models and documents are the data flow diagrams and their detailed documentation, in structured English, and the detailed data flow definitions. Figure 10-1 illustrates the structured design modelling process.

**Quick Quiz: The Structured Approach to Designing the Application Architecture**

1) What is a module?
   ANSWER: A module is an identifiable component of a computer program that performs a defined function.

2) What is a system flowchart used for?
   ANSWER: A system flowchart is used to depict the overall flow of information and control in a total system that consists of several programs.

**The Automation System Boundary**

The automation system boundary partitions the data flow diagram processes into manual processes and those that are to be included in the computer system. During the analysis phase, we looked at the business events and all of the processes that were triggered by those events. At that time, we did not try to distinguish between manual and automated processes. However, to develop the computer system’s design, we must identify the processes that will be automated.

Processes can either be inside or outside the system boundary. Processes that are outside are manual processes, such as sorting and inspecting paper documents, entering customer orders, or visually inspecting incoming shipments. In some cases, the system boundary goes right through a process, which indicates that the process is a mid- or high-
level process and is exploded on a more detailed drawing. Some of the processes in the exploded detail will be inside and some outside the system boundary.

Data flows are found inside, outside, or crossing the system boundary and the program boundaries. The data flows that cross the system boundary are particularly important—they represent inputs and outputs of the system. In other words, the design of the program interfaces, both user interface and transmittals to other systems, are defined by the boundary-crossing data flows. In the final system, these data flows will be forms or reports in the user interface, or files or telecommunication transmittals between systems. Data flows that cross the boundaries between programs represent program-to-program communication. In the final system, these data flows will also be files or telecommunication transmittals between programs.

Figure 10-2 represents the high-level data flow diagram with an automated system boundary and shows all of the major processes for a payroll system.

The automated system boundary, although needed for both structured and object-oriented development, needs to be defined explicitly for the structured approach. The object-oriented approach already has the automated system boundary defined in the use case diagram.

**Quick Quiz: The Automated System Boundary**

1) What is the purpose of the automation system boundary?
**ANSWER:** The automation system boundary defines which processes are going to be automated within the computer system, and which processes are parts of the environment, or manual system. The system boundary, although needed for both structured and object-oriented development, needs to be defined explicitly for the structured approach. The object-oriented approach already has the system boundary defined in the use case diagram.

2) How do you develop an automation system boundary?
**ANSWER:** The automation system boundary is developed at the DFD-fragment level or below by defining which processes are manual and which are automated. The boundary line is a line between these two processes.

**The System Flowchart**

The system flowchart is the computer system representation of various computer programs, files, databases, and associated manual procedures. Processes are grouped into programs and subsystems based on similarities such as shared timing (for example, a process performed monthly), shared accesses to stored data (for example, all processes that update employee data), and shared users (for example, processes that produce reports for the marketing department). The programs and subsystems thus created have complex interdependencies including flow of data, flow of control, and interaction with permanent data stores.

A system flowchart graphically describes the organization of the subsystems into automated and manual components, showing the flow of data and control among them. System flowcharts are primarily used to describe large information systems consisting of
distinct subsystems and dozens or more programs. Figure 10-4 shows a sample system flowchart for a payroll system, and Figure 10-5 shows the system flowchart for Rocky Mountain Outfitters.

**Quick Quiz: The System Flowchart**

What is a system flow chart used for?
**ANSWER:** A system flow chart is used to depict the overall flow of information and control in a total system that consists of several programs.

**The Structure Chart**

The primary objective of structured design is to create a top-down decomposition of the functions to be performed by a given program in a new system. Each independent program shown in the system flowchart performs a set of functions. Using structure chart techniques always provides a hierarchical organization of program functions. A structure chart hierarchy describes the functions and the sub functions of each part of a system.

**Structure chart:** A hierarchical diagram showing the relationships between the modules of a computer program.

A module is the basic component of a structure chart and is used to identify a function. Modules are relatively simple and independent components. Higher-level modules are “control” modules that control the flow of execution. Lower-level modules are “worker bee” modules and contain the program logic to actually perform the functions. Figure 10-6 shows a simple structure chart for the Calculate pay amounts module.

The vertical lines connecting the modules indicate the calling structure from the high-level modules to the lower-level modules. The little arrows next to the lines show the data that is passed between modules and represent the inputs and outputs of each module. At the structure chart level, we are not yet concerned with what is happening inside the module. We only want to know that somehow the module performs the function indicated by its name, using the input data and producing the output data. For our purposes, each module is a functioning “black box.”

A program call occurs when one module invokes a lower-level module to perform a needed service or calculation.

The arrows with the open circle, called data couples, represent data being passed into and out of the module. A data couple can be an individual data item (for example, a customer account number) or a higher-level data structure (for example, an array, record, or other data structure). The arrow with the darkened circle is a “flag.” A flag is purely internal information that is used between modules to indicate some result.

**Program call:** The transfer of control from a module to a subordinate module to perform a requested service.

**Data couples:** The individual data items that are passed between modules in a program call.
A basic idea of structured programming is that each module only has to do a very specific function. The module at the very top of the tree is the boss module. Its functions are to call the modules on the next tier, pass information to them, and receive information back. The function of each middle-level module is to control the processing of the modules below it. Each module has control logic and any error-handling logic that is not handled by the lower-level module. The modules at the extremities, or the leaves, contain the actual algorithms to carry out the functions of the program.

**Developing a Structure Chart**

Structure charts are developed to design a hierarchy of modules for a program. A structure chart is in the form of a tree with a root module and branches. A subtree is simply a branch that has been separated from the overall tree. When the subtree is placed back in the larger tree, the root of the subtree becomes just another branch in the overall tree. Why is this important? The structure chart can be developed in pieces and combined for the final diagram.

There are two methods to develop structure charts: transaction analysis and transform analysis.  

**Transaction analysis** uses as input the system flowchart and the event table to develop the top level of the tree—that is, the main program boss module and the first level of called modules.  

**Transform analysis** uses as input the DFD fragments to develop the subtrees, one for each event in the program. Each subtree root module corresponds to the first-level branch of the main program structure chart.

The method used to develop structure charts is frequently determined by the quality of the data flow diagrams.

**Transaction analysis:** The development of a structure chart based on a DFD that describes the processing for several types of transactions.

**Transform analysis:** The development of a structure chart based on a DFD that describes an input-process-output data flow.

In transaction analysis, the first step is to examine the system flowchart and identify each major program. Transaction analysis is the process of identifying each separate transaction that must be supported by the program and constructing a branch for each one. In essence, this program, at least at the highest level, is simply a module to display a screen for the user to enter a transaction choice and then to invoke the appropriate module to process the transaction.

Transform analysis is based on the idea that input data is "transformed" into output information by the computer program. Structure charts developed with transform analysis usually have three major subtrees: an input subtree to get the data, a calculate subtree to perform the logic, and an output subtree to write the results.

The method to develop a structure chart from a data flow diagram fragment consists of the following steps:

1. Determine the primary information flow.
2. Find the process that represents the most fundamental change from an input stream to an output stream. The input data stream is called the afferent data flow. The output data stream is called the efferent data flow. The centre process is called the central transform.

3. Redraw the data flow diagram with the input to the left and the output to the right. The central transform process goes in the middle.

4. Based on the redrawn data flow, generate the first-draft structure chart with the calling hierarchy and the required data couples.

5. Add other modules as necessary to get input data via the user interface screens, read and write to the data stores, and write output data or reports.

6. Based on any structured English or decision table documentation, add other required intermodal relationships such as looping and decision symbols.

7. Make the final refinements to the structure chart based on the quality-control concepts discussed in the following section.

Evaluating the Quality of a Structure Chart

The process of developing structure charts from DFDs can become rather involved. However, certain rules and guidelines can be used to test the quality of the final structure chart. Two measures of quality are module coupling and module cohesion. Generally, it is desirable to have highly cohesive and loosely coupled modules.

The principle of coupling is a measure of how a module is connected to the other modules in the program. The objective is to make modules as independent as possible. An independent module can execute in almost any environment and has a well-defined interface of several predefined data fields.

Cohesion refers to the degree to which all of the code within a module contributes to implementing one well-defined task. Modules with high cohesion implement a single function. All of the instructions within the module are part of that function, and all are required for the function.

The concepts of coupling and cohesion carry forward into many aspects of traditional and object-oriented design.

Module coupling: The manner in which modules relate to each other. The preferred method is data coupling.

Module cohesion: A measure of the internal strength of a module.

Quick Quiz: The Structure Chart

1) What is the purpose of a structure chart?
   ANSWER: A structure chart depicts the hierarchical structure of modules that make up a computer program. It shows the calling hierarchy as well as the data parameters passing through the program.

2) What is a program call?
   ANSWER: A program call is the transfer of control from a module to a subordinate module to perform a requested service.
3) Explain transaction analysis.
ANSWER: Transaction analysis is the development of a structure chart based on a data flow diagram (DFD) that describes the processing for several types of transactions. Transaction analysis is the process of identifying each separate transaction that must be supported by the program.

4) Explain transform analysis.
ANSWER: Transform analysis is the development of a structure chart based on a data flow diagram (DFD) that describes an input-process-output dataflow. Transform analysis is based on the idea that input data is “transformed” into output information by the computer program.

Module Algorithm Design: Pseudocode

The next requirement of design is to describe the internal logic within each module. Three common methods are used to describe module logic: flowcharts, structured English, and pseudocode. All three methods are equivalent in their ability to describe logic. Flowcharting is a visual method that uses boxes and lines to describe the flow of logic in a program. In the early days of computing, flowcharting was used almost exclusively. Today, however, it has been replaced with versions of pseudocode and structured English.

Quick Quiz: Module Algorithm Design: Pseudocode

What are the three common methods used to describe module internal logic?
ANSWER: Flowcharts, structured English, and pseudocode.

Integrating Structured Application Design with Other Design Tasks

The user interface consists of a set of input forms, output forms, and reports. Interactive user interfaces are usually based on a dialog between the user and the system and include a series of input and output forms. Every form must be displayed and the data retrieved somewhere in a module in the structure chart. As these forms are developed, the structure chart needs to be evaluated from three aspects:

• Do additional user interface modules need to be added?
• Does the pseudocode in the interface modules need to be modified?
• Do additional data couples need to be added to pass data?

The same three aspects—modules, pseudocode, and data couples—need to be evaluated for the database. If a database management system is being used, a common interface is usually provided. Finally, the structure charts and system flowcharts must be checked for correspondence to the existing or planned network architecture. An important final step in detailed design is to re-evaluate its correspondence to the network architecture, particularly with respect to required protocols, capacity, and security.

Quick Quiz: Integrating the Structured Application Design with Other Design Tasks

What does the user interface consist of?
ANSWER: The user interface consists of a set of input forms, output forms, and reports. Interactive user interfaces are usually based on a dialog between the user and the system and include a series of input and output forms.

### Three-Layer Design

Chapter 9 described three-layer design and the division of application software into the view layer, business logic layer, and data access layer. Structure charts and system flowcharts can be used to describe design decisions and software structuring based on three-layer architecture. Structure charts should be reworked to clearly show the modules that belong to each layer. Module pseudocode can be used to show the exact format and content of messages that are passed among layers.

The example in Figures 10-20 through 10-23 show an alternate approach to using structure charts for interactive applications. The top module is the activity or use case controller. Each form used in the dialog of the activity/use case is listed as a module of the view layer. Each form in turn calls a business logic layer procedure or function. Business layer procedures call data access procedures or the DBMS. Using this approach, the three-layer design architecture can be modelled for interactive, GUI applications.

Figure 10-19 and Figure 10-20 demonstrate how a system flow chart and structure chart can be useful for showing three-layer architecture for a traditional, GUI application. Often, students study the structure chart and wonder how it applies to Visual Basic applications and Web applications.

### Solutions to End of Chapter Material

#### Review Questions

1. **Explain the relationship and difference between a module and a program.**
   A program is made up of modules. A program is a set of modules that work together to perform all the required functions. A module is a relatively small set of statements that carry out a single function.

2. **What is the purpose of the automation system boundary? How do you develop one?**
   The automated system boundary defines which processes are going to be automated within the computer system, and which processes are part of the environment, or manual system. The system boundary, although needed for both structured and object-oriented development, needs to be defined explicitly for the structured approach. The object-oriented approach already has the system boundary defined in the use case diagram. It is developed at the DFD fragment level or below, by defining which processes are manual and which are automated. The boundary line is a line between these two processes.

3. **What is a system flow chart used for?**
   A system flow chart is used to depict the overall flow of information and control in a total system that consists of several programs.
4. **What are the symbols that are used on a system flow chart?**
The symbols include a rectangle for a program, a rectangle with curved sides or a disk for a file, a rectangle with a curved bottom line for a report, a circle with a tail for a file, a trapezoid for a manual operation, and arrow or lightning bolt for a transmission or movement of data.

5. **What is the purpose of a structure chart?**
A structure chart depicts the hierarchical structure of modules that make up a computer program. It shows the calling hierarchy as well as the data parameter passing between the modules.

6. **What are the symbols used on a structure chart?**
The symbols include a rectangle for the module, a connection line or arrow for the calling structure, and small arrows with open circles for data couples or with black circles for flags.

7. **Explain transaction analysis.**
Transaction analysis is the process of identifying a set of transactions (usually via DFD fragments) and developing a structure chart. Transaction analysis results in a structure chart with a calling structure to call a module for each transaction type.

8. **Explain transform analysis. What is meant by the term central transform?**
Transform analysis is the process of taking a DFD diagram and converting it to a structure chart. Transform analysis is based on the idea of 'transforming' an input data flow into an output information flow. The central transform is the central process that transforms the data.

9. **What is the difference between afferent and efferent data flow?**
Afferent data flow is the incoming data flow in a sequential set of process bubbles, while efferent data flow is: The outgoing data flow from a sequential set of process bubbles.

10. **Explain module coupling and module cohesion. Why are these concepts important?**
In structured analysis, the modules are defined and built ideally to have low coupling and high cohesion. With low coupling, the only knowledge that one module has about another module is its input parameters and output fields. Like a black box, a well-defined module is built this way so that other modules do not have to know its internal workings.
Module cohesion refers to the single purpose of a module. If a module carries out only one single function, it is self-contained and easier to maintain.

11. **Describe how structure charts for three-layer architecture are different from those for all-encompassing programs that execute on a single computer system.**
Structure charts for three-layer architecture shows all three layers but may not include modules to handle all operational aspects of the application tasks, such as data access modules included within the data access layer. All-encompassing programs that execute on a single computer system will include all modules.
Additional examples

Example 1
Given the data flow diagram shown in Figure 10-25, and using transaction analysis, develop a structure chart.

Answer:

```
Registration Program

Inquire Course Info  Verify Schedule  Add Class  Drop Class  Check Registration Status
```

Example 2
Given the data flow diagram shown in Figure 10-26, and using transform analysis, develop a structure chart.

Answer:

```
Add Class

Display Course Info  Get Course  Update Student Schedule  Display Schedule

Verify Student  Verify Sections  Update Schedule File
```

Example 3
Integrate the structure charts from problem numbers 2 and 3 together into a single structure chart.

Answer:
Replace the Add Class module from problem 2 with the tree from problem 3.
Key Terms

- **Afferent data flow** – The incoming data flow from a sequential set of DFD processes.
- **Central transform** – Set of DFD processes that are located between the input and output processes.
- **Computer program** – An executable entity made up of a set of modules.
- **Data couples** – The individual data items that are passed between modules in a program call.
- **Efferent data flow** – The outgoing data flow in a sequential set of DFD processes.
- **Module** – An identifiable component of a computer program that performs a defined function.
- **Module cohesion** – A measure of the internal strength of a module.
- **Module coupling** – The manner in which modules relate to each other; the preferred method is data coupling.
- **Program call** – The transfer of control form a module to a subordinate module to perform a requested service.
- **Pseudocode** – Structured-programming-like statements that describe the logic of a module.
- **Structure chart** – A hierarchical diagram showing the relationships between the modules of a computer program.
- **System flowchart** – A diagram that describes the overall flow of control between computer programs in a system.
- **Transaction analysis** – The development of a structure chart based on a DFD that describes the processing for several types of transactions.
- **Transform analysis** – The development of a structure chart based on a DFD that describes the input-process-output data flow.
Chapter 8 – Designing Databases

This chapter corresponds with chapter 13 of the text book.

Chapter Overview
Note: chapters mentioned in the text below refer to the chapters in the text book.

Databases and database management systems are important components of a modern information system. Databases provide a common repository for data so that it can be shared by the entire organization. Database management systems provide designers, programmers, and end users with sophisticated capabilities to store, retrieve, and manage data. Sharing and managing the vast amounts of data needed by a modern organization would not be possible without a database management system.

In Chapter 5, you learned to construct conceptual data models and to develop entity-relationship diagrams (ERDs) for traditional analysis and class diagrams for object-oriented (OO) analysis. To implement an information system, developers must transform a conceptual data model into a more detailed database model and implement that model within a database management system.

The process of developing a database model depends on the type of conceptual model and the type of data management software that will be used to implement the system. This chapter will describe the design of relational and OO data models and their implementation using database management systems.

Learning Objectives

After studying this chapter, you should be able to:

- Describe the differences and similarities between relational and object-oriented database management systems.
- Design a relational database schema based on an entity-relationship diagram.
- Describe the different architectural models for distributed databases.

Notes on Chapter Opening Case and Case Studies

Nationwide Books: Designing a New Database: The case presents a scenario in which a project development team discusses the most appropriate technology with which to implement a database for a new system. Key points to emphasize include:

- Although analysis was performed using object-oriented methods the database can be implemented with a non-object-oriented database management system.
- Interfacing object-oriented programs with traditional relational databases requires reconciling their different views of stored data.
- Changing an organization’s database technology is a complex matter because many applications share each database. Ensuring that existing applications continue
to function requires one or some combination of the following approaches:

- Existing applications must be converted to the new database technology.
- The new database technology must provide backward compatibility.
- Redundant databases and database management systems (DBMSs) must be implemented and supported.

- Failure to update database technology (or any other type of infrastructure technology) can minimize or eliminate the inherent advantages of newer application development tools and languages.

**Real Estate Multiple Listing Service System**: Use the ERD and class diagram from the Chapter 5 case study to develop a relational database schema in third normal form (3NF) and an Object Definition Language (ODL) database schema.

**State Patrol Ticket Processing System**: Use the ERD and class diagram from the Chapter 5 case study to develop a relational database schema in 3NF and an ODL database schema.

**Computer Publishing, Inc.**: Use this textbook as a template for CPI’s database content. Draw an ERD and class diagram to present the book and its key content elements. Expand the book to include related product content. Develop a list of data types required to store the content. Describe the features of an object database management system (ODBMS) that might be useful to implement CPI’s database.

**Rethinking Rocky Mountain Outfitters**: Use the ERD and class diagram from the Chapter 5 case study to update Figure 13-9 and develop a relational database schema in 3NF, as well as ODL schema specifications for all new classes and relationships that were added to the class diagram.

**Focusing on Reliable Pharmaceutical Service**: Use the ERD from the Chapter 5 case study and the design class diagram from the Chapter 7 case study to develop a relational database schema in 3NF and an ODL database schema, and to discuss the pros and cons of distributed database architecture.

**Notes on RMO Memo**

**Customer support system database design and DBMS**: The memo describes completed database design activities and goes on to discuss implementation alternatives – specifically, choosing a relational or object DBMS. The topics covered are similar to those covered in the opening case and the key points are the same.

**Additional Notes and Quick Quizzes**

**Databases and Database Management Systems**

A database is an integrated collection of stored data that is centrally controlled and managed. A database stores data describing dozens, hundreds, or thousands of entities or classes. Many different programs and information systems can access each database.

**DBMS Components**
A database management system (DBMS) is system software that manages and controls access to a database (see Figure 13-1). A DBMS stores raw data in a separate physical data store. Descriptive information about the data (for example, relationships among data elements and access controls) is stored in a separate data store called a schema. Application programs, end users, and database administrators use DBMS utilities to interact with the database. They access the data via names and relationships defined in the schema, and the DBMS accesses corresponding data within the physical data store.

Databases and DBMSs enable all of an organization’s data to be centrally managed and controlled. All users and applications use common data definitions, access controls, and access technology.

**Database Models**

Data in a database is organized according to a specific type of database model. Database models have evolved over time to reflect changes (improvements) in software development and data management technology. Database model types include:

- **Hierarchical** – Not used since the 1970s.
- **Network** – Not used since the 1980s.
- **Relational** – Currently the most common type.
- **Object-oriented** – Rapidly gaining in popularity.

The model types are incompatible as are the DBMSs that implement them.

**Quick Quiz: Databases and Database Management Systems**

1) What is a database schema?
   
   ANSWER: A database schema is a store of data that describes the content and structure of the physical data store (sometimes called metadata—data about data).

2) What information does a database schema contain?
   
   ANSWER: A database schema contains a variety of information about data types, relationships, indices, content restrictions, and access controls.

3) Why have databases become the preferred method of storing data that is used by an information system?
   
   ANSWER: Databases are a common point of access, management, and control. They allow data to be managed as an enterprise-wide resource while enabling many different users and application programs to access the data simultaneously. They solve many of the problems associated with separately maintained data stores, including redundancy, inconsistent security, and inconsistent data access methods.

**Relational Databases**

A relational DBMS (RDBMS) is a DBMS that organizes stored data into two-dimensional structures called tables or relations (see Figure 13-2). Table rows are sometimes called tuples or records. Table **columns** are sometimes called fields or attributes. The content of each table cell is called a field value, attribute value, or data element.
Each table row must have a primary key field (or set of fields) that contains a unique value (or set of values) for each row in the table. Primary keys may be natural or invented. Most keys used in business information systems are invented (for example, credit card, telephone, and social security numbers).

Entities on an entity-relationship diagram (ERD) are usually represented in separate tables. Relationships among entities (that is, among rows stored in different entity tables) are represented by storing the primary key of one entity table in a related entity table (see Figure 13-4). When the primary key fields of one table are added to another table to represent a relationship, they are called foreign keys.

**Designing Relational Databases**

To design a relational database schema from an ERD, follow these steps:

1. Create a table for each entity type.
2. Choose (or invent) a primary key for each table.
3. Add foreign keys to represent one-to-many relationships.
4. Create new tables to represent many-to-many relationships.
5. Define referential integrity constraints.
6. Evaluate schema quality and make necessary improvements.
7. Choose appropriate data types for each field.

**Representing Entities**

Each entity in an ERD (see Figure 13-5) is represented by a single table in a relational database (see Figure 13-6). The table contains a column for each attribute of the entity. After the tables are created, the designer chooses or adds a field to each table to be the table’s primary key.

**Representing Relationships**

Different representation methods are used for one-to-many and many-to-many relationships. One-to-many relationships are represented by adding the primary key field(s) of the “one” entity type to the table that represents the “many” entity type (see Figure 13-8). If no associative entity exists for the relationship, a new table is created to represent the relationship (see Figure 13-9). If an associative entity does exist, use its table to represent the relationship. The primary key field(s) of the related entity types is used as the primary key of the table that represents the relationship.

**Enforcing Referential Integrity**

Referential integrity is a consistent relational database state in which every foreign key value also exists as a primary key value. The DBMS enforces referential integrity by:

- Checking for updates to foreign keys.
- Updating foreign key values when corresponding primary key values are updated.

The database designer has some control over exactly how foreign key checking and updating is implemented.

**Evaluating Schema Quality**

A high-quality relational database schema has the following features:

- Unique table rows and primary keys.
Various formal and informal techniques for schema evaluation can be used. No one technique is sufficient by itself, but a combination of techniques can ensure a high-quality database design.

Assumptions about key uniqueness and procedures for generating invented key values must be carefully scrutinized. A typical database has a long life, and continued usefulness of the database depends on unique keys. Thus, key uniqueness must be evaluated over the long term, and the designer must consider possible changes over time. This is especially important for keys generated by external entities such as suppliers and government agencies. Few changes have the pervasive and disruptive impact of a database key change in a large information system.

The issues of data redundancy and ease of implementing future schema changes are intertwined. Redundant storage implies redundant maintenance, and schema changes are a form of maintenance. Thus, minimizing redundancy increases maintainability. Relational databases intentionally implement redundancy between primary and foreign keys to represent relationships. This form of redundancy is necessary and unavoidable. However, data redundancy among non-key fields must be avoided.

Relational database normalization is a process that ensures schema quality by minimizing redundancy. Normalization is based on a concept called functional dependency and a series of normal forms.

- A **functional dependency** is a one-to-one relationship between the values of two fields. Field A is functionally dependent on field B if for each value of B there is only one corresponding value of A.
- A table is in **first normal form (1NF)** if it contains no repeating fields or groups of fields.
- A table is in **second normal form (2NF)** if it is in first normal form and if each non-key field is functionally dependent on the entire key.
- A table is in **third normal form (3NF)** if it is in second normal form and if no non-key field is functionally dependent on any other non-key field.

Fields with a one-to-one correspondence are functionally dependent on one another. One example of a group of fields with functional dependency is the set of attribute values that describe a single entity, such as a person. These attributes (for example, name, age, gender) are functionally dependent on any field that uniquely identifies a person.

First normal form defines a restriction on schema structure. Repeating fields are not allowed within any table in a relational database. Repeating groups of fields are also prohibited. If such a group exists, it must be removed and stored in a separate table with each field in the repeating group stored as a separate row.

Second normal form requires that all non-key fields be functionally dependent on the entire key. The easiest way to determine functional dependency between two fields is to substitute their names into the boldface definition of functional dependency and ask if the resulting statement is true. For example:
PersonName is functionally dependent on PersonNumber if for each value of PersonNumber, there is only one corresponding value of PersonName.

If PersonNumber is unique for each person stored in the database, the statement is true, and the functional dependency holds.

If the primary key contains multiple fields, substituting field names in the definition becomes a bit more complex. For example:

AverageDayTemperature is functionally dependent on the combination of Day, Month, and Year if for each set of values for Day, Month, and Year, there is only one corresponding value of AverageDayTemperature.

Violations of 2NF can only occur in tables with multi-field primary keys. For example, a field called AverageMonthTemperature would be functionally dependent only on Month and Year. This condition is sometimes called a partial functional dependency.

To resolve a partial functional dependency, the designer has two choices:

- Move the “problem” field to an existing table with the appropriate primary key (for example, move AverageMonthTemperature to an existing table that uses Month and Year as its primary key).
- Create a new table for the "problem" field (for example, create a new table containing Month, Year, and AverageMonthTemperature).

If possible, choose the first alternative because it avoids redundancy among primary keys in different tables.

Third normal form is the most difficult normal form to verify because it requires checking functional dependency among all possible combinations of non-key fields. Any non-key field that is functionally dependent on another non-key field must be removed from the table and placed in a new table. The fields on which the removed field was functionally dependent must also be added to the new table, and they become the primary key of that table.

A common violation of 3NF occurs when a field can be computed from other non-key fields (for example, an invoice total). Note that this violation may cross table boundaries (that is, the computed field and the fields on which it is functionally dependent may be located in multiple tables). Regardless of whether the fields are located in one or multiple tables, the means of correcting the 3NF violation is the same—remove the computed field from the database.

Quick Quiz: Relational Databases

1) With respect to traditional databases, briefly define the terms row and field.
   ANSWER: **Row** – The portion of a table containing data that describes one entity, relationship, or object. **Field** – The portion of a table (a column) containing data that describes the same fact about all entities, relationships, or objects represented within the table.

2) What is a primary key? Are duplicate primary keys allowed?
ANSWER: A primary key is a field or set of fields, the values of which uniquely identify a row or a table. Duplicate key values aren’t allowed because that would destroy their ability to uniquely identify a row.

3) What is a foreign key? Are duplicate foreign keys allowed?
ANSWER: A foreign key is a value (or set of values) stored in one table that also exists as the value of a primary key in another table. Foreign keys are used to represent relationships among entities represented as tables. Duplicate foreign keys are not allowed within the same table because they would redundantly represent the same relationship. Duplicate foreign keys may exist in different tables because they would represent different relationships.

4) What is referential integrity?
ANSWER: Referential integrity is content constraint between the values of a foreign key and the values of the corresponding primary key. Values of foreign key fields must either exist as values of a primary key or must be null.

5) What is relational database normalization?
ANSWER: Relational database normalization is a process that increases schema quality by reducing data redundancy among tables. A schema with tables in third normal form has less non-key data redundancy than a schema with unnormalized tables. Less redundancy makes the schema and database contents easier to maintain over the long term.

Removing a computed field is not always desirable if one of the fields participating in the computation can change over time. For example, consider sales tax computations. Sales tax on an invoice can be computed by multiplying the pre-tax invoice total by the tax rate. If the tax rate is stored in a separate tax table, it represents the current tax rate. If the tax rate is changed in the future, the value in the tax table is modified. If the computed tax amount for “old” invoices has been removed from the database, the amount of tax actually assessed and paid on those old invoices can no longer be computed. Thus, leaving the computed field in the database is not really a 3NF violation because functional dependency of the computed field on the computational factors is not guaranteed over time.

In theory, the preferred method of solving this problem is to store a history of tax rates (for example, a tax rate table with a date field and a rate field). Each time the tax rate changes, it is added to the tax rate table as a new row. Tax for an old invoice is computed by extracting the invoice date and finding the tax rate table row with the latest date that isn’t later than the invoice date. In practice, most designers don’t remove a computed amount, such as sales tax, to avoid the complexity of storing a history of factor values. The issue of 3NF and time-varying fields is usually addressed in a database management course, but many students find it confusing. You must decide whether it is worth the time to discuss this topic in a systems analysis and design course.

Data Types

A data type defines the storage format and allowable content of a program variable, object state variable, or database field or attribute. A primitive data type is a data type that is supported directly by computer hardware, a programming language, or a DBMS. A complex data type (or user-defined data type) is a data type that is not supported directly by computer hardware, a programming language, or a DBMS. Complex data types are created using primitive data types as building blocks. Modern information systems
increasingly use complex data types. Examples include sounds, still and motion video, and Web links (URLs).

**Relational DBMS Data Types**
Each table column in a relational database must be defined as one of the primitive data types available in the DBMS. The set of RDBMS primitive data types is usually quite limited. Complex data, such as digital audio and compressed images, must usually be stored in a primitive binary data type. The generic name for this data type is a binary large object (BLOB), though names and exact specifications vary widely across RDBMSs.

Some RDBMSs allow the database designer to specify specific format or value constraints on each field. Examples include numeric range checks and format specifications for character strings. Specifying such constraints within the schema simplifies programs and eliminates the possibility of inconsistent constraint checking.

**Quick Quiz: Data Types**

1) What is a primitive data type?
   **ANSWER:** A primitive data type (for example, integer, real, and character) is supported directly by the CPU or a programming language.

2) What is a complex data type?
   **ANSWER:** A complex data type (for example, record, linked list, or object) contains one or more data elements that are constructed using the primitive data types as building blocks.

**Distributed Databases**

In most organizations, data is distributed across multiple databases and database servers. Reasons for data distribution include:

- Information systems may have been developed at different times using different DBMSs.
- Parts of an organization’s data may be owned and managed by different organizational units.
- System performance is improved when data is physically close to the applications that use it.

**Distributed Database Architectures**

There are several different approaches to distributing databases, including:

- Single database server
- Replicated database servers
- Partitioned database servers
- Federated database servers

Multiple approaches may be combined as needed.

Single database server architecture (see Figure 13-27) uses a single database and server to service all users and programs. Its chief advantage is simplicity—that is, it is relatively easy to create and administer. Its chief disadvantages are performance and susceptibility
to failure. Performance can suffer if the single server becomes overloaded. Network congestion or propagation delay from client to server can also result in decreased performance for remote clients.

Replicated database server architecture (see Figure 13-28) uses multiple database servers, each of which maintains a complete copy of the database. Replicas can be located on the same LAN segments as their clients to improve performance and spread database service load across multiple computers. Replicas can “fill in” for one another when one is inoperable.

Replicas must be synchronized to ensure that all database users see a consistent view of organizational data. Replicated servers must continuously or periodically exchange information about database updates to maintain data consistency. Determining and implementing an optimal synchronization strategy is complex. Maintaining replicas is relatively difficult.

To mitigate risk, the database synchronization strategy should be determined early in the database design.

Partitioned database server architecture (see Figure 13-29) divides a database into partitions (see Figure 13-30) and places each partition on a separate database server. Partitions can be located close to users and programs that need partitioned data. If partitions are mutually exclusive, no synchronization is required. If partitions overlap (which is usually the case), the overlapping portions of the database must be periodically synchronized. Maintenance is relatively difficult.

Federated database server architecture (see Figure 13-31) uses one database server as a front-end to other database servers. Typically, the back-end database servers are from different vendors and contain different subsets of organizational data. The front-end database server defines a schema that provides a unified view of all of the data. The front-end database server reformats queries to the unified schema into queries to the back-end servers. When all the responses are received, the front-end server combines them into a single response for the client. Federated database architecture is commonly used to implement data warehouses.

**RMO Distributed Database Architecture**

Review the RMO location diagram (see Figure 6-32), activity-location matrix (see Figure 6-33), and activity-data or CRUD matrix (see Figure 6-34). Information from these figures is summarized on page 521. RMO has already decided to house database management in Park City using the existing mainframe. Given the wide distribution of RMO’s processing activities, issues related to database distribution, network design, and system performance must be considered.

The text presents two completely different design alternatives. Figure 13-32 shows a single-server architecture, and Figure 13-33 shows a distributed architecture with partitioning or replication at all remote sites except corporate headquarters.

The single-server architecture offers a simple way to manage database content and structure. However, it requires substantial WAN capacity, and any failure of the database server or the data centre LAN to perform will effectively disable the entire RMO customer
support system (CSS). Given the strategic nature of the CSS, this option is probably too risky. There needs to be at least one database replica, possibly in a location other than the data centre, to ensure continuous CSS operation.

Figure 13-33 shows an architecture that is a complex combination of replication and partitioning. Each site, except corporate headquarters, has a local database server with all of the data needed to support operations at that site. The primary advantages of this architecture are its robustness in the face of failure at the data centre and its lower WAN communication requirements to support normal processing. However, this architecture also requires complex database management and the need to synchronize redundant database content. The latter issue is an interesting one with respect to WAN capacity requirements. If a significant portion of the CSS database changes daily, substantial WAN capacity will be required to synchronize changes across replicas. At the extreme, the WAN capacity requirements of both options are nearly identical.

Quick Quiz: Distributed Databases

Why might all or part of a database need to be replicated in multiple locations?

ANSWER: Database accesses between distant servers and clients must traverse one or more network links. This can slow the accesses due to propagation delay or network congestion. Access speed can be increased by placing a database replica close to clients.

Solutions to End of Chapter Material

Review Questions

1. List the components of a DBMS and describe the function of each.

   Application Program Interface – an interface engine or library of precompiled subroutines that enable application programs (e.g., those written in C or Java) to interact with the database.

   End-User Query Processor – a program or utility that allows end-users to retrieve data and generate reports without writing application programs.

   Data Definition Interface – a program or utility that allows a database administrator to define or modify the content and structure of the database (e.g., add new fields or redefine data types or relationships).

   Data Access and control Logic – the system software that controls access to the physical database and maintains various internal data structures (e.g., indices and pointers).

   Database – the physical data store (or stores) combined with the schema.

   Schema – a store of data that describes various aspects of the 'real' data including data types, relationships, indices, content restrictions, and access controls.

   Physical data store – the 'real' data as stored on a physical storage medium (e.g.,
magnetic disk).

2. **What is a database schema? What information does it contain?**
The database schema is a store of data that describes the content and structure of the physical data store (sometimes called 'metadata' – or data about data). It contains a variety of information about data types, relationships, indices, content restrictions, and access controls.

3. **Why have databases become the preferred method of storing data used by an information system?**
Databases are a common point of access, management, and control. They allow data to be managed as an enterprise-wide resource while providing for simultaneous access by many different users and application programs. They solve many of the problems associated with separately maintained data stores including redundancy, inconsistent security, and inconsistent data access methods.

4. **List four different types of database models and DBMSs. Which are in common use today?**
The four database models are hierarchical, network (CODASYL), relational, and object-oriented. Hierarchical and network models are technologies of the 1960s and 1970s – they are rarely found today. The relational model was developed in the 1970s and widely deployed in the 1980s and 1990s. It is currently the predominant database model. The object-oriented data model was developed in the 1990s and is still under development. It is expected to slowly supplant the relational model over the next decade.

5. **With respect to relational databases, briefly define the terms row and field.**
Row – The portion of a table containing data that describe one entity, relationship, or object.
Field – The portion of a table (a column) containing data that describes the same fact about all entities, relationships, or objects represented within the table.

6. **What is a primary key? Are duplicate primary keys allowed? Why or why not?**
A primary key is a field or set of fields, the values of which uniquely identify a row or a table. Duplicate key values aren’t allowed because that would destroy their ability to uniquely identify a row.

7. **What is the difference between a natural key and an invented key? Which type is most commonly used in business information processing?**
A natural key is a naturally occurring attribute of or fact about something represented in a database (for example, a human fingerprint or the atomic weight of an element). An invented key is one that is assigned by a system (for example, a social security or credit card number). Most keys used in business information processing are invented.

8. **What is a foreign key? Why are foreign keys used or required in a relational database? Are duplicate foreign key values allowed? Why or why not?**
A foreign key is a value (or set of values) stored in one table that also exists as the value of a primary key in another table. Foreign keys are used to represent
relationships among entities represented as tables. Duplicate foreign keys are not allowed within the same table since they would redundantly represent the same relationship. Duplicate foreign keys may exist in different tables since they would represent different relationships.

9. **Describe the steps used to transform an ERD into a relational database schema.**
   1. Create a table for each entity type.
   2. Choose a primary key for each table.
   3. Add foreign keys to represent one-to-many relationships.
   4. Create new tables to represent many-to-many relationships.
   5. Define referential integrity constraints.
   6. Evaluate schema quality and make necessary improvements.
   7. Choose appropriate data types and value restrictions for each field.

10. **How is an entity on an ERD represented in a relational database?**
    Each entity on an ERD is represented as a separate table.

11. **How is a one-to-many relationship on an ERD represented in a relational database?**
    A one-to-many relationship is represented by adding the primary key field(s) of the table that represents the entity participating in the 'one' side of the relationship to the table that represents the entity participating in the 'many' side of the relationship.

12. **How is a many-to-many relationship on an ERD represented in a relational database?**
    A many-to-many relationship is represented by constructing a new table that contains the primary key fields of the tables that represent each participating entity.

13. **What is referential integrity? Describe how it is enforced when a new foreign key value is created, when a row containing a primary key is deleted, and when a primary key value is changed.**
    Referential integrity is a content constraint between the values of a foreign key and the values of the corresponding primary key in another table. The constraint is that values of the foreign key field(s) must either exist as values of a primary key or must be NULL. A valid value must exist in the foreign key field(s) before the row can be added. When a row containing the primary key is deleted, the row with the foreign key must also be deleted for the data to maintain referential integrity. A primary key should never be changed; but in the event that it is, the value of the foreign key must also be changed.

14. **What types of data (or fields) should never be stored more than once in a relational database? What types of data (or fields) usually must be stored more than once in a relational database?**
    Non-key fields should never be stored more than once.

If a table represents an entity, then the primary key values of each entity represented within the table are redundantly stored (as foreign keys) for every relationship in which the entity participates.
15. **What is relational database normalization? Why is a database schema in third normal form considered to be of higher quality than an unnormalized database schema?**

Relational database normalization is a process that increases schema quality by reducing data redundancy among tables. A schema with tables in third normal form has less non-key data redundancy than a schema with unnormalized tables. Less redundancy makes the schema and database contents easier to maintain over the long term.

16. **Describe the process of relational database normalization. Which normal forms rely on the definition of functional dependency?**

The process of normalization modifies the schema and table definitions by successively applying higher order rules of table construction. The rules each define a “normal form” and the normal forms are numbered one through three. First normal form eliminates embedded repeating groups within tables.

Second and third normal forms are based on a concept called *functional dependency* – a one-to-one correspondence between two field values. Second normal form ensures that every field in a table is functionally dependent on the primary key. Third normal form ensures that no non-key field is functionally dependent on any other non-key field.

17. **What is an association class?**

An association class is an “artificial” class that is created to represent a many-to-many relationship among “real” classes. The association class has data members that represent attributes of the many-to-many relationship.

18. **Why might all or part of a database need to be replicated in multiple locations?**

Database accesses between distant servers and clients must traverse one or more network links. This can slow the accesses due to propagation delay or network congestion. Access speed can be increased by placing a database replica close to clients.

19. **Briefly describe the following distributed database architectures: replicated database servers, partitioned database servers, and federated database servers. What are the comparative advantages of each?**

**Replicated database servers** – An entire database is replicated on multiple servers, and each server is located near a group of clients. Best performance and fault tolerance for clients because all data is available from a “nearby” server.

**Partitioned database servers** – A database is partitioned so that each partition is a database subset used by a single group of clients. Each partition is located on a separate server, and each server is located close to the clients that access it. Better performance and less replication traffic than replicated servers if similar collocated clients use only a subset of database content.

**Federated database servers** – Data from multiple servers with different data models and/or DBMSs is pooled by implementing a separate (federated) server that presents a unified view of the data stored on all the other servers. The federated server constructs answers to client queries by forwarding requests to other servers.
and combining their responses for the client. Simplest and most manageable way to combine data from disparate DBMSs into a single unified data store.

20. What additional database management complexities are introduced when database contents are replicated in multiple locations?

Replicated copies are redundant data stores. Thus, any changes to data content must be redundantly implemented on each copy. Implementing redundant maintenance of data content requires all servers to periodically exchange database updates.

Case Studies

Case Study: The Real Estate Multiple Listing Service System

1. Develop a relational database schema in 3NF.

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>Number, OfficeNumber, Name, OfficePhone, HomePhone,</td>
</tr>
<tr>
<td></td>
<td>EmailAddress, CellPhone</td>
</tr>
<tr>
<td>Listing</td>
<td>Number, AgentNumber, Address, YearBuilt, SquareFeet,</td>
</tr>
<tr>
<td></td>
<td>NumberOfBedrooms, NumberOfBaths, OwnerName, OwnerPhone,</td>
</tr>
<tr>
<td></td>
<td>AskingPrice, DateListed, DateLastUpdated, StatusCode</td>
</tr>
<tr>
<td>Office</td>
<td>Number, Name, ManagerName, Address, Phone, FAX</td>
</tr>
</tbody>
</table>

Case Study: Rethinking Rocky Mountain Outfitters

1. Update the RMO relational database design in Figure 13-9 based on the changes that you made to the ERD. Be sure that all your database tables are in 3NF.

The database must now store information about customer charge accounts, charges, and payments. The requirement raises several third normal form issues since some of this information is or could be stored within the existing database structure:

- The existing AccountNo field can also be a customer's charge account number.
- One of the values of PaymentMethod inOrderTransaction can be 'RMO charge' or something similar. In that case, customer charges are already stored within the OrderTransaction table.

Under the above assumptions, the only data that must be added to the database are customer payments and whether the customer has as RMO charge account. The following additions and changes will store that information.

<table>
<thead>
<tr>
<th>Table</th>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>AccountNo, Name, BillingAddress, ShippingAddress,</td>
</tr>
<tr>
<td></td>
<td>DayTelephoneNumber, NightTelephoneNumber,</td>
</tr>
<tr>
<td></td>
<td>HasRMOChargeAccount</td>
</tr>
<tr>
<td>Payment</td>
<td>AccountNo, Date, Amount</td>
</tr>
</tbody>
</table>
One Boolean field, HasRMOChargeAccount, has been added to the Customer table. The Payment table stores information about customer payments. The new table’s primary key is the combination of AccountNo and Date, which assumes that a customer will never make two payments on the same day. AccountNo in the Payment table is also a foreign key.

Note that there is no field to represent the charge account balance. Strict application of the 3NF rule does not allow such a field since it can be computed by summing all amounts in the OrderTransaction table and subtracting the sum of all amounts in the Payment table for a specific customer. However, this assumes that transaction and payment information will be stored indefinitely in the database. A more reasonable assumption is that older data will be periodically removed from the database, in which case a current balance field should be added to the Customer table.

Key Terms

- **Complex data type** – A data type not directly supported by computer hardware or a programming language; also called user-defined data type.
- **Database (DB)** – An integrated collection of stored data that is centrally managed and controlled.
- **Database management system (DBMS)** – System software that manages and controls access to a database.
- **Database synchronization** – The process of ensuring consistency among two or more database copies.
- **Data type** – The storage format and allowable content of a program variable or database field.
- **Data warehouse** – A collection of data used to support structured and unstructured managerial decisions.
- **Field** – A column of a relational database table; also called an attribute.
- **Field value** – The data value stored in a single cell of a relational database table; also called an attribute value or data element.
- **First normal form (1NF)** – A relational database table structure that has no repeating fields or groups of fields.
- **Foreign key** – A field value stored in one relational database table that also exists as a primary key value in another relational database table.
- **Functional dependency** – A one-to-one correspondence between two field values.
- **Hybrid object-relational DBMS** – A relational database management system used to store object attributes and relationships; also called hybrid DBMS.
- **Key** – A field that contains a value that is unique within each row of a relational database table.
- **Multivalued attribute** – An attribute that contains zero or more instances of the same data type.
- **Navigation** – The process of accessing an object by extracting its object identifier from another (related) object.
- **Normalization** – A technique that ensures relational database schema quality by minimizing data redundancy.
• **Object database management system (ODBMS)** – A database management system that stores data as objects or class instances.

• **Object Definition Language (ODL)** – A standard object database description language promulgated by the Object Database management Group.

• **Object identifier** – A physical storage address or a reference that can be converted to a physical storage address at run time.

• **Persistent object** – An object that must store one or more attribute values between instantiations or method invocations.

• **Physical data store** – The storage area used by a database management system to store the raw bits and bytes of a database.

• **Primary key** – A key used to uniquely identify a row of a relational database table.

• **Primitive data type** – A storage format directly implemented by computer hardware or a programming language.

• **Referential integrity** – A consistent relational database state in which every foreign key value also exists as a primary key value.

• **Relational database management system (RDBMS)** – A database management system that stores data in tables.

• **Row** – The portion of a table containing data that describes one entity, relationship, or object; also called tuple or record.

• **Schema** – A description of the structure, content, and access controls of a physical data store or database.

• **Second normal form (2NF)** – A relational database table structure in which every non-key field is functionally dependent on the primary key.

• **Table** – A two-dimensional data structure containing rows and columns; also called a relation.

• **Third normal form (3NF)** – A relational database table structure in which no non-key field is functionally dependent on any other non-key field(s).

• **Transient object** – An object that doesn’t need to store any attribute values between instantiations or method invocations.
Chapter 9 – Making the System Operational

This chapter corresponds with chapter 16 of the textbook.

Chapter Overview

This chapter focuses on the activities of the implementation and support phases of the systems development life cycle (SDLC). Activities that occur before the system is turned over to its users are collectively called implementation, while activities that occur after the system becomes operational are collectively called support. Figure 16-1 describes the activities of the implementation and support phases.

Implementation consumes more time and resources than earlier phases of the SDLC – particularly software construction and testing. Project management complexity is at its greatest during the implementation phase because so many people and activities must be coordinated.

Supporting information systems is one of the most important jobs in an organization. Support activities ensure that the system and its users function efficiently and effectively for years after installation. Therefore, most organizations spend more money maintaining and supporting existing systems than they do building new ones.

Learning Objectives

After studying this chapter, you should be able to:

- Describe implementation and support activities.
- Choose an appropriate approach to program development.
- Describe various types of software tests and explain how and why each is used.
- List various approaches to data conversion and system installation and describe the advantages and disadvantages of each.
- Describe different types of documentation and the processes by which they are developed and maintained.
- Describe training and user support requirements for new and operational systems.

Notes on Chapter Opening Case and Case Studies

Tri-State Heating Oil: Juggling Priorities to Begin Operation: The case describes a status meeting for a project in the midst of the implementation phase. Participants discuss trade-offs and dependencies among the activities in progress and those not yet completed.

Key points to stress about the case include:

- Implementation contains many different activities with complex interdependencies.
- Implementation is often behind schedule due to delays in earlier project phases.
- Delays and unforeseen problems require at least a few schedule revisions (e.g., the delay starting user training).
• Implementation resource requirements are substantial (e.g., count the number of people mentioned in the case and consider the timeline).
• A large number of people work simultaneously on many different tasks.
• Schedule revisions may result in underutilizing human resources that were committed based on the original plans (e.g., having to pay temporary staff that can’t yet be productively used).

The implementation management process must be capable of handling the numerous tasks, dependencies, and revisions. Note that Kim needed time in his office to verify the feasibility of a revised schedule. Project management software is a must for project planning.

Notes on RMO Memo

Customer support system implementation milestones: The memo describes milestones and target dates for completing implementation phase activities. Barbara Halifax discusses possible delays, contingency plans, and regular meetings to monitor progress.

Key points to stress about the case include:
• Implementation contains many different activities with complex interdependencies (e.g., the sequential dependence among all the listed milestones).
• Delays and unforeseen problems may require additional resources (e.g., the cost of operating both systems in parallel during the holiday shopping season if the new system doesn’t complete acceptance testing by December 1).
• The importance of monitoring progress so that delays can be minimized or eliminated (e.g., the twice weekly meetings with attendance by senior management).

The key points in the memo are similar to those raised in the opening case.

HudsonBanc Billing System Upgrade: This case uses a bank merger to describe a direct installation. HudsonBanc had several problems with this approach that could have been avoided with improved planning, testing, and a different approach.

The DownTown Videos Rental System: Using the design class diagram you developed in Chapter 11 for the DownTown Videos rental system, develop an implementation and testing plan. Specify the order in which classes and their methods will be implemented and the groups of methods and classes that will be tested during integration testing.

Rethinking Rocky Mountain Outfitters: There are business risks involved with implementing close to the holiday sales peak period. Contingency planning is required and the installation plan and schedule must be carefully developed and closely monitored.

Focusing on Reliable Pharmaceutical Service: Using the structure chart you developed in Chapter 10 for Reliable Pharmaceutical Service, develop an implementation and testing plan. Specify the order in which modules will be implemented and the groups of modules that will be tested during integration testing.
Additional Notes and Quick Quizzes

Program Development

A complex system (for example, automobiles, buildings, and information systems) contains thousands of interconnected parts. Each part must be constructed, tested, integrated with other parts, and retested as part of the integrated whole. The construction and testing process is typically custom designed for each information system. The process must ensure that the following conditions are met:

- Resources are efficiently used.
- Development time is minimized.
- Product quality is maximized (or meets minimum standards).

Programming is not the only implementation phase activity, but it is one of the most important because of the following factors:

- Required resources
- Managerial complexity
- System quality

Program development (including unit testing) typically accounts for one-third of all development labour and between one-third and one-half of the development schedule. The magnitude of resources and time consumed during program development clearly warrants careful planning and management attention.

Order of Implementation

One of the most basic decisions to be made about program development is the order in which program components (such as modules, methods, and classes) will be developed. Several orders are possible, including the following:

- Input, process, output (IPO)
- Top-down
- Bottom-up

The IPO development order is based on data flow through a system or program. Programs or modules that obtain external input are developed first. Programs or modules that process the input (that is, transform it into output) are developed next. Programs or modules that produce output are developed last. For structured designs and programs, an analyst can determine IPO ordering by examining the system flowchart and structure charts. The analyst classifies programs on the system flowchart as input, process, or output and implements them in that order (see Figure 16-2). The analyst then similarly classifies and implements modules and groups of modules on a structure chart. Applying the IPO development order to a structure chart is straightforward if the structure chart was developed using transform analysis (see Figure 16-3).

IPO development order can also be applied to object-oriented designs and programs. The key issue to analyze is dependency—that is, which classes and methods capture or generate data that is needed by other classes or methods. Dependency information is documented in package (see Figure 16-4), sequence, and class diagrams. Data
dependencies among methods cannot be easily determined.

Advantages of using the IPO development order include the following:

- Simplifies testing. Because input programs and modules are developed first, they can be used to enter test data for process and output programs and modules. The need to write special-purpose programs to generate or create test data is reduced, thus speeding the development process.
- Important user interfaces are developed early, allowing for early testing and user evaluation. If changes are needed, there is plenty of time to make them.

The chief disadvantage of IPO development order is the late implementation of output units, which are needed to perform full system integration testing.

Top-down and bottom-up development orders are techniques that originated with the advent of structured design and structure charts. Top-down development order implements modules from the top of a structure chart to the bottom, one layer at a time. Top-down development order requires writing a stub for every module in the system. The advantage of top-down development order is that programs are easily built and tested. The disadvantage is that development staff might not be used efficiently at the start of programming.

Bottom-up development order implements modules from the bottom of the structure chart to the top. Bottom-up development order requires writing a driver for every group of lower-level modules and revising that driver each time a new lower-level module is added. The primary advantage of bottom-up development order is that many programmers can be put to work immediately. The disadvantage is that driver programs need to be written to test bottom-level modules, which adds additional complexity to the development and testing processes. Because the entire system is not assembled until the topmost modules are written, testing of the system as a whole is delayed.

Deciding on an IPO, top-down, or bottom-up development order is only a starting point in creating a program development plan. The analyst must consider other factors, including the following:

- Testing
- Training
- Documentation
- User feedback

Testing and program development are interdependent. For this reason, a formal implementation plan covering both program development and testing is usually created before any development begins. The plan typically details development order, testing order, test data, acceptance criteria, and personnel assignments.

**Framework Development**

When implementing a large object-oriented system, it is not unusual to build an object framework (or set of foundation classes) that covers most or all of the domain and data access layer classes. Foundation classes are typically reused in many parts of the system and across many different applications. Because of this reuse, they are a critical system component. In addition, later changes to foundation classes might require significant
changes throughout the system.

Foundation classes are typically developed and tested first, usually by the most skilled programmers. Early development and complete testing are important because framework code is reused throughout the system.

Framework development is carefully planned, closely managed, and fully tested during the first iteration of a project. This part of the project must be successful because every other class (and iteration) is dependent upon the quality of the foundation classes. Framework development can literally make or break a project.

Team-Based Program Development
Programmers are usually organized into small programming teams of no more than 10 members. Some commonly used organizational models include the following:

- Cooperating peer
- Chief developer
- Collaborative specialist

A cooperating peer team includes members of roughly equal skill and experience with overlapping areas of specialization. Cooperating peer teams are appropriate for tasks requiring experimentation and creative problem solving.

A chief developer team has a single leader who makes all of the important decisions. Chief developer teams are appropriate for projects with well-defined objectives and a well-defined path to completion.

A collaborative specialist team has members with wide variation in and minimal overlap of skills and experience. Collaborative specialist teams are appropriate for tasks that require diagnosis or experimentation, require creative and integrative problem solving, and span a wide range of technology.

One member of each team should be responsible for communicating and coordinating activities with other teams. Teams also need a mix of non-technical skills, including the ability to generate ideas, build consensus, manage details, and communicate with external constituencies. Figure 16-7 summarizes team characteristics and optimal task characteristics.

Source Code Control
Development teams need tools to help coordinate their programming tasks. A source code control system (SCCS) is an automated tool for tracking source code files and controlling changes to those files. An SCCS allows multiple programmers to check out a file in read-only mode, but only one programmer can check out a file in read/write mode. This prevents multiple programmers from attempting to modify the same file at the same time. Figure 16-8 shows a screen capture from the Microsoft Visual SourceSafe SCCS.

Versioning
Medium-scale and large-scale systems must be developed, tested, and deployed in versions. An alpha version is a version that is incomplete, but ready for some level of rigorous testing. A beta version is a version that is stable enough to be tested by end
users. A production version is a version that is formally distributed to users or made operational. A system typically goes through multiple alpha and beta versions before it becomes a production version (see Figure 16-9).

Version numbers are usually embedded in the application and can be displayed by end users (see Figure 16-11). This allows support personnel to provide version-specific help to users. Versioning support is typically integrated into SCCSs.

**Quick Quiz: Program Development**

1) Briefly describe the input, process, output (IPO) approach to program development order.
   ANSWER: IPO development order develops input programs or modules first, processing programs or modules next, and output programs or modules last.

2) Briefly describe the top-down approach to program development order.
   ANSWER: Top-down development order implements program modules in their order of invocation (procedure or function calling), which corresponds to the top-to-bottom order of modules as they are drawn on a structure chart.

3) Briefly describe the bottom-up approach to program development order.
   ANSWER: Bottom-up development order implements program modules in the reverse of their invocation order, which corresponds to the bottom-to-top order of modules as they are drawn on a structure chart.

4) List the three approaches to organizing programming teams.
   ANSWER: Programmers are usually organized into small (less than ten person) programming teams of the following types: cooperating peer, chief developer, collaborative specialist.

5) What is a source code control system (SCCS)?
   ANSWER: An SCCS is an automated tool for tracking source code files and controlling changes to those files. SCCSs prevent multiple programmers from making inconsistent or conflicting changes to source files. An SCCS ensures that a source code file cannot be modified by more than one programmer at a time.

**Quality Assurance**

Quality assurance (QA) is the process of ensuring that an information system meets minimal quality standards. QA is a set of interrelated activities, including technical reviews and testing. QA activities occur during all development phases. During analysis and design, QA is primarily concerned with identifying gaps or inconsistencies in user requirements and producing designs that satisfy requirements with minimal implementation effort. During implementation, QA is primarily concerned and integrated with software testing activities.

**Technical Reviews**

A technical review is a formal or informal review of design or construction details by a group of developers. Informal technical reviews by two or three people are often called walkthroughs. An inspection is a formal review of design or construction details by a group...
of developers, where each person plays a specific role (for example, presenter, critic, or secretary).

Technical reviews reduce the number of errors that reach testing by a factor of 5 to 10 and reduce testing costs by approximately 50 percent. Testing and technical reviews each find between 50 and 75 percent of errors. But some errors are more easily detected by one method or the other. Some errors are rarely found by one technique but are easily found by the other. Thus, the two techniques are more effective jointly than individually.

**Testing**

*Software Testing*

Testing is the process of examining a product to determine what defects it contains. To conduct a test, programmers require a well-defined standard or defect definition. Software components can be tested individually or in groups, or the entire system can be tested as a whole. Unit testing tests individual code modules or methods before they are integrated with other modules. Integration testing tests the behaviour of a group of modules or methods. A system test tests the behaviour of an entire system or independent subsystem. The three testing types are each correlated to a specific phase of the SDLC (see Figures 16-13 and 16-14).

A test case is a formal description of a starting state, one or more events to which software must respond, and the expected response or ending state. Test data is a set of starting states and events used to test a module, a group of modules, or an entire system. Because preparing test cases and test data is a time-consuming and tedious process, automated tools are often used to generate a complete set of test cases.

*Unit Testing*

Unit testing may require writing drivers (see Figure 16-16), stubs (see Figure 16-17), or both. A driver is a module that simulates the calling behaviour of a module that hasn’t yet been developed. A stub is a module that simulates the execution or behaviour of a module that hasn’t yet been developed. Drivers are primarily used in bottom-up development, and stubs are primarily used in top-down development.

*Integration Testing*

Integration testing looks for errors that cannot be detected during unit testing. Such errors might result from a number of problems, including the following:

- Interface incompatibilities
- Illegal parameter values
- Run-time exceptions
- Unexpected state interactions

Integration testing of OO software is much more complex and not as well understood as integration testing of structured software. There is no clear hierarchical structure to an OO program. An OO program consists of a set of interacting objects that can be created or destroyed during execution. Object interactions and control flow are dynamic and complex.

Additional factors that complicate OO integration testing include the following:

- Methods can be called by many other methods distributed across multiple classes.
- Classes can inherit methods and state variables.
• Calls to specific methods are dynamically determined at run time.
• Permanent object state variables can cause identical inputs or calls to be processed differently at different times.

The combination of these factors makes it difficult to determine an optimal testing order for OO software. (Note: See the “Notes for Moving to the Object-Oriented Approach” section for additional information about this topic).

A system test tests the behaviour of an entire system or independent subsystem. System testing is usually performed at the end of an iteration to identify significant issues, such as performance problems, and plan the work for the next iteration. Two common types of system tests are the following:

• Build and smoke tests
• Acceptance tests

A build and smoke test determines whether the system compiles, links, and runs without obvious malfunctions. Build and smoke tests should be performed daily. They provide rapid feedback about new problems and usually locate the problem in code that was added or modified since the last smoke and build test.

Usability Testing
A usability test determines whether a module, method, class, subsystem, or system meets user requirements. Because there are many types of functional and non-functional requirements, many types of usability tests are performed at many different times.

A performance test is a system test that determines whether a system can meet time-based performance criteria, such as response time or throughput.

An acceptance test is a system test that determines whether the system fulfils user requirements. Acceptance testing is usually very formal, and both developers and users participate. Acceptance tests are typically performed after installation and, if testing is successful, the system is then turned over to its owners.

Who Tests Software?
Many different people can perform testing processes. Programmers are generally responsible for unit testing their own code prior to integrating it with modules written by other programmers. In some organizations, programmers are assigned a testing buddy to help them test their own code. In larger projects, a separate quality assurance team usually performs integration and system testing. Using a separate team for these types of tests ensures independence and objectivity. Users normally participate in system testing by testing beta versions and by participating in acceptance testing.

Quick Quiz: Quality Assurance

1) What is quality assurance (QA)?
ANSWER: QA is the process of ensuring that an information system meets minimal quality standards

2) What are technical reviews, walkthroughs, and inspections?
ANSWER: A technical review is a formal or informal review of design or construction
details by a group of developers. Informal technical reviews by two or three people are often called walkthroughs. An inspection is a formal review of design or construction details by a group of developers, where each person plays a specific role (for example, presenter, critic, and secretary).

3) What are the characteristics of good test cases?
ANSWER: Test cases must ensure that all instructions in a program are executed, preferably many times, with a variety of data inputs.

Data Conversion

An operational system requires a fully populated database to support ongoing processing. If an existing database will need to support a new information system, the following requirements must be met:

• The existing database schema must be modified to support new system requirements.
• New data, if any, must be added to the database to support the new system.

If a new database will be created to support a new information system, the database must be populated with data. Possible sources of data for the new database include the following:

• Files or databases of a system being replaced
• Manual records
• Files or databases of other systems in the organization
• User feedback during normal system operation

Reusing Existing Databases

Reusing an existing database is fairly common because of the difficulty and expense of creating new databases from scratch, especially when a single database often supports multiple information systems as in today’s enterprise application systems. Most database management systems allow the database structure to be modified without disrupting existing data or application programs. Typical DBMS changes include adding or modifying tables or attributes in tables.

Reloading Database Contents

Some changes to the database structure or content are too complex to be performed without damaging existing data. In that case, implementation staff must develop programs to alter data after the database has been modified. Figure 16-18 shows two possible approaches to reloading data:

• Create a new database with the required changes, and copy data from the old database to the new database.
• Extract the data from an existing database to temporary files, delete the existing data from the database, modify the database structure as required, and reload the old data from the temporary files.

Many DBMSs provide a rich set of import utilities to extract and load data from existing databases, files, or scanned documents. If DBMS import and export utilities are inadequate for data conversion, developers must construct custom conversion programs.
Creating New Databases
Adding data to a new database can be a complex and time-consuming task. Figure 16-19 shows a complex data conversion process that draws input from a variety of sources. Data is input using a mix of manual data entry, optical character recognition, conversion programs, and DBMS import and export utilities.

If manual data entry is needed to capture data for the new database, data-entry programs should be developed early in the implementation phase, and initial data entry can be structured as a user training exercise.

The effort, cost, and timing of data entry should be planned as a part of the overall project planning. This area can greatly impact the end user. Some organizations hire temporary help for manual data entry to reduce the impact on the business.

Quick Quiz: Data Conversion

1) List possible sources of data for a new database.
   ANSWER: Possible data sources include manual (paper) records, files or databases of a system being replaced, files or databases of other systems in the organization, and user feedback during normal system operation.

2) List tools and methods for adding initial data into the system database.
   ANSWER: Tools for initializing database content include normal data entry programs, optical character recognition software, custom-developed data conversion programs, and DBMS import and export utilities.

Installation

Some of the more important issues to consider when planning installation include the following:

• Cost
• Detecting and correcting errors
• Potentially disrupting the company
• Training

The most commonly used installation approaches are as follows:

• Direct installation
• Parallel installation
• Phased installation

Direct Installation
Direct installation (also called immediate cutover) installs a new system, quickly makes it operational, and immediately turns off any overlapping systems (see Figure 16-20). The primary advantages of direct installation are its simplicity and low cost. The primary disadvantage of direct installation is its risk. Because older systems are not operated in parallel, there is no backup in the event that the new system fails.
Parallel Installation

In parallel installation, the old and new systems are both operated for an extended period of time (see Figure 16-21). The primary advantage of parallel installation is a relatively low risk of system failure and the negative consequences that might result from that failure. If both systems are operated completely (that is, using all data and exercising all functions), the old system functions as a backup for the new system. The disadvantages of parallel installation are its cost and complexity. Full parallel installation may be impractical for the following reasons:

- Inputs to one system might be unusable by the other, and it might not be possible to use both types of inputs.
- The new system might use the same equipment as the old system, and there might not be sufficient capacity to operate both systems.
- Staffing levels might be insufficient to operate or manage both systems at the same time.

Possible modes of partial parallel operation include the following:

- Processing only a subset of input data in one of the two systems. The subset could be determined by transaction type, geography, or sampling (for example, every 10th input transaction).
- Performing only a subset of processing functions (such as updating account history but not printing monthly bills).
- Performing a combination of data and processing function subsets.

Phased Installation

In a phased installation, a new system is installed and made operational in a series of steps or phases (see Figure 16-22). Each phase can be installed directly or in parallel. The primary advantage of phased installation is reduced risk. The primary disadvantage of phased installation is increased complexity. Dividing the installation into phases creates more activities and milestones, thus making the entire process more complex. However, each individual phase contains a smaller and more manageable set of activities. If the entire system is simply too big or complex to install at one time, the reduced risks of phased installation outweigh the increased complexity inherent in managing and coordinating multiple phases. Phased installation is typically used in situations where installing the entire system at once is simply too complex or risky.

| Phased installation works best when the system architecture defines independent subsystems and the development work is planned in phases. Organizations will frequently perform a phased installation for a large, multiple-year project, such as ERP, to get a partial return on investment while the project is underway. |

Personnel Issues

Installing a new system places significant demands on personnel throughout an organization. The problem is most acute in parallel installation. Often, development and customer support personnel must be temporarily reassigned to provide sufficient manpower to operate both systems. Specialized skills in hardware and software installation and configuration are needed. Hiring temporary personnel is often a necessity. Maximum productivity is usually not achieved until months or years after the installation.
Larger organizations that are implementing multiple projects during a single fiscal year might have a specialized implementation team that works closely with project managers to plan and staff the installation phase. This approach to installation is similar to (or may be part of) the approach used with a quality assurance workgroup.

Quick Quiz: Installation

1) Describe direct installation. List its advantages and disadvantages.
   ANSWER: Direct installation installs a new system, quickly makes it operational, and immediately turns off any overlapping systems. Direct installation is easy to plan and execute, but it runs the greatest risk of interrupted service.

2) Describe parallel installation. List its advantages and disadvantages.
   ANSWER: Parallel installation operates both the old and new systems for an extended period. Parallel installation reduces the risk of interrupted service, but it is relatively complex to plan and implement.

3) Describe phased installation. List its advantages and disadvantages.
   ANSWER: In phased installation, a new system is installed and made operational in a series of steps or phases. Phased installation breaks up the complexity of planning and execution into manageable pieces, but the total planning and execution effort is greater than it is with the other installation methods.

Documentation

Documentation can be loosely classified into two types:

- System documentation – Describes system functions, architecture, and construction details.
- User documentation – Describes how to interact with and maintain the system.

Both types of documentation can be distributed in electronic, hyperlinked, or multimedia formats.

System Documentation

Analysis models, design models, and system source code comprise the majority of system documentation (see Figure 16-23). Source code is the most frequently used documentation because altering source code is the only realistic method of making small changes to a system’s behaviour. Design models must be consulted to evaluate significant changes in systems design or function. Analysis models must be consulted to evaluate significant changes in user requirements.

System documentation must be actively managed to be current and effective. Failure to maintain system documentation compromises the value of an information system by making it difficult or impossible to maintain. Large organizations with many information systems usually assign this function to one person or organization (for example, a system librarian).

System documentation mirrors the system itself. That is, any information contained within system documentation can also be obtained by directly examining the system. As changes
are made to the system, its documentation must also be updated, however. If
documentation is not updated, it is inconsistent with the system and useless to future
designers and maintenance programmers. Making documentation an integral part of the
installed system minimizes or eliminates inconsistency because updates to the system
automatically update the documentation. Some tools—in particular, CASE tools and
reverse-engineering tools—can simplify documentation and help ensure its accuracy.

Using the object-oriented approach, new developers in an organization can understand the
system by studying the UML models instead of viewing the source code. After they
understand the system design, they can work with the source code.

User Documentation
Topics covered in user documentation include:

- Software start-up and shutdown
- Keystroke, mouse, or command sequences to perform specific functions
- Program functions required to implement business procedures
- Common errors and ways to correct them

The quality of user documentation often suffers because it is prepared at the last minute.
Many organizations assume that knowledge of how to use the system can be passed
directly from one user to another, but this is a haphazard way to protect such an important
asset. Developing good documentation requires different skills than developing programs.
Unfortunately, programmers are often assigned to this task that should be done only by
technical writers.

Knowledge of how to use a system is as much an asset to the organization as the system
itself. High-quality user documentation helps to safeguard that investment.

User documentation is an important part of the software package. When deciding which
software to purchase, many users judge the quality of the software on the quality and
completeness of the documentation. Because high-quality user documentation can reduce
an organization’s user support costs, organizations should hire technical writers for this
part of the software development project.

Larger organizations might have a specialized desktop publishing team that works closely
with project managers to plan user documentation requirements and hire the appropriate
staff. Doing so improves the quality of the documentation and frees IT implementation
resources.

Quick Quiz: Documentation

What are the differences between documentation for end users and system operators?

ANSWER: Operator documentation covers topics, such as system start-up and shutdown,
backup and recovery, and software installation and upgrade that are not typically covered
in end-user documentation. System operators are usually more highly trained than end
users, so their documentation often has fewer “bells and whistles.”
Training and User Support

Benefits of good user training include the following:

- User support requirements are reduced.
- Users attain higher productivity levels more quickly.
- Error rates are reduced.

There are two classes of users—end users and system operators—who must be considered for documentation, training, and support. In general, end users use the system frequently and for extended periods of time, and system operators interact with the system infrequently and usually for short periods. End users solve a particular business problem with the system or implement specific business procedures. System operators are usually computer professionals with limited knowledge of the business processes that the system supports. End-user computer skill levels vary widely, whereas system operators typically have higher and more uniform skill levels. Also, the number of end users is generally much larger than the number of system operators.

End users usually require face-to-face training, oftentimes in group settings. Documentation, training, and support for end users must be detailed and must address the specific business context in which the system is being used. System operators can often train themselves if they have access to quality documentation. Their training and documentation do not need to emphasize the business context because those procedures are mostly irrelevant to their normal duties.

Training is more efficient if documentation is complete, and the system is relatively stable. However, organizations usually want users to “hit the ground running” after a new system is made operational, so training must often begin before documentation is complete and user interfaces are finalized.

User documentation, training, and support for end users could be handled by the same team and managed as a separate project.

Ongoing Training and User Support

User support can be provided by a number of methods, including the following:

- Online documentation and troubleshooting
- Resident experts
- A help desk
- Technical support

Resident experts are end users who have sufficient skills to train and support other end users. A help desk is a formal role or organization that provides user support for multiple systems. Technical support staff can assist end users, but their skills are more appropriate in situations in which a user discovered a bug or the complexity of a user’s problem exceeds the capability of other support personnel.

Quick Quiz: Training and User Support

What are the benefits of good user training?
ANSWER: Good user training reduces user support requirements, allows users to attain higher productivity levels more quickly, and reduces error rates.

**Maintenance and System Enhancement**

Maintenance involves modifying a software product after delivery or installation to correct faults, improve performance or other attributes, or adapt the product to a changed environment. The term covers virtually everything that happens to a system after installation, except total replacement or abandonment. Maintenance costs exceed development costs for most systems.

Maintenance activities include the following:

- Tracking modification requests and error reports.
- Implementing changes.
- Monitoring system performance and improving performance or increasing capacity.
- Upgrading hardware and system software.
- Updating documentation to reflect maintenance changes.

**Submitting Change Requests and Error Reports**

Changes to an operating system are normally initiated by a formal change request (see Figure 16-26). An IS manager or committee evaluates formal change requests. If the request is approved, appropriate resources are committed to implement the change.

Software and hardware errors can be reported using a standard change request form, but many organizations prefer to use a separate problem-tracking and management system because they need to address these problems in a timely fashion.

**Implementing a Change**

Change implementation follows a miniature version of the system development life cycle. Changes must be planned, analysis and design documentation must be developed or updated, source code must be modified and tested, and the modified system components must be installed and tested. Implementation activities are often performed on a copy of the operational system, called a test system. The test system becomes the operational system only after complete and successful testing.

**Upgrading Computing Infrastructure**

Computer hardware, system software, and networks must be periodically upgraded for many reasons, including the following:

- Software maintenance releases
- Software version upgrades
- Declining system performance

System software updates are risky. Application software that worked well with an older software version might fail when that software is updated. Many organizations avoid system software upgrades, or they test the upgrades thoroughly to reduce the risk of operational failures.

Declining performance usually results from an increase in software demands that overwhelm the configuration or capability of system. Performance can sometimes be
improved by reconfiguring existing resources. Many IS staff members, however, lack the skills to do this in-house and must rely on the expertise of contract personnel or consultants.

Quick Quiz: Maintenance and System Enhancement

1) What is software maintenance?
ANSWER: Software maintenance involves modifying a software product after delivery or installation to correct faults, improve performance or other attributes, or adapt the product to a changed environment. The term covers virtually everything that happens to a system after installation, except total replacement or abandonment. Maintenance costs exceed development costs for most systems.

2) List three typical change control procedures.
ANSWER: Changes to an operating system are normally initiated by a formal change request. An IS manager or committee evaluates formal change requests. If the request is approved, appropriate resources are committed to implement the change.

Solutions to End of Chapter Material

Review Questions

1. List and briefly describe the three basic approaches to program development order. What are the advantages and disadvantages of each?
   Input, Process, Output (IPO) development order develops input programs or modules first, processing programs or modules next, and output programs or modules last. IPO development order's advantages include simplified testing and early testing of user interfaces. The primary disadvantage of IPO development order is late implementation of outputs, which may delay discovery of errors in processing modules or programs.

   Top-down development order implements program modules in their order of invocation (procedure or function calling), which corresponds to the top-to-bottom order of modules as they are drawn on a structure chart. The primary advantage of top-down development is that there is always a working (buildable and testable) version of the program. The primary disadvantage is that there are relatively few top-level modules, which may lead to inefficient use of programmers early in the project.

   Bottom-up development order implements program modules in the reverse of their invocation order, which corresponds to the bottom-to-top order of modules as they are drawn on a structure chart. The primary advantage of bottom-up development order is that a large number of programmers can work in parallel from the start of implementation. The primary disadvantage is that a large number of driver programs must be written to simulate the behaviour of (as yet) unwritten calling top-level modules.

2. Describe three approaches to organizing programming teams. For what types of projects or development activities is each approach best suited?
A cooperating peer team includes members of roughly equal skill and experience with overlapping areas of specialization. Such a team is best suited to projects requiring experimentation and creative problem solving.

A chief developer team has a single leader who makes all of the important decisions. Such a team is best suited to projects with well-defined objectives and a clear completion path.

A collaborative specialist team has members with wide variation in and minimal overlap of skills and experience. Such a team is best suited to projects that require diagnosis, experimentation, creative and integrative problem solving, and where the project/problem spans a wide range of technology.

3. **What is a source code control system? Why is such a system necessary when multiple programmers build a program or system?**

A source code control system (SCCS) is an automated tool for tracking source code files and controlling changes to those files. SCCSs prevent multiple programmers from making inconsistent or conflicting changes to source files. An SCCS ensures that a source code file cannot be modified by more than one programmer at a time.

4. **Define the terms alpha version, beta version, and production version. Are there well-defined criteria for deciding when an alpha version becomes a beta version or a beta version becomes a production version?**

An alpha version is a system that is incomplete but ready for some level of rigorous testing. A beta version is a system that is stable enough to be tested by end users. A production version is a system that is formally distributed to users or made operational.

Because users test beta versions by using them to do real work, an alpha version must meet the following criteria to become a beta version: the software must be nearly complete, and it must not be susceptible to severe crashes or other productivity-thwarting behaviours.

A beta version usually becomes a production version after successful acceptance testing.

5. **List and briefly describe implementation phase QA activities other than software testing. What is the effect on software testing of not performing non-testing QA activities?**

QA activities other than testing include technical reviews and inspections. A technical review is a formal or informal review of design or construction details by a group of developers. An inspection is a formal technical review in which each participant plays a specific role (e.g., presenter, critic, and secretary).

Software testing becomes more complex and expensive if technical reviews and inspections are skipped because a larger number of errors find their way into source code.

6. **What are the characteristics of good test cases?**

Test cases must ensure that all instructions in a program are executed, preferably many times, with a variety of data inputs.
7. Define the terms *acceptance test, integration test, system test, and unit test*. In what order are these tests normally performed? Who performs (or evaluates the results of) each type of test?

An acceptance test is a system test performed to determine whether the system fulfils user requirements.

An integration test is a test of the behaviour of a group of modules or methods.

A system test is a test of the behaviour of an entire system or independent subsystem.

A unit test is a test of an individual module or method before it is integrated with other modules or methods.

An acceptance test is normally performed by or under the direction of users. The programmer or his/her testing buddy normally performs unit testing. Integration and system tests may be performed by programmers or members of a separate quality assurance group.

8. What is a driver? What is a stub? With what type of test is each most closely associated? With what development order is each most likely to be used?

A driver is a module that simulates the calling behaviour of a module that hasn’t yet been developed. Drivers are most closely associated with unit testing and are most likely to be used with bottom-up development order.

A stub is a module that simulates the execution or behaviour of a module that hasn’t yet been developed. Stubs are most closely associated with unit testing and are most likely to be used with top-down development order.

9. List possible sources of data used to initialize a new system database. Briefly describe the tools and methods used to load initial data into the database.

Possible data sources include manual (paper) records, files or databases of a system being replaced, files or databases of other systems in the organization, and manual data entry during training or normal system operation. Tools for initializing database content include normal data entry programs, optical character recognition software, custom-developed data conversion programs, and DBMS import and export utilities.

10. Briefly describe direct, parallel, and phased installation. What are the advantages and disadvantages of each installation approach?

Direct installation installs a new system, quickly makes it operational, and immediately turns off any overlapping systems. Direct installation is the simplest to plan and execute, but runs the greatest risk of interrupted service.

Parallel installation operates both the old and new systems for an extended period. Parallel installation is relatively complex to plan and implement, but it reduces the risk of interrupted service.

Phased installation installs a new system and makes it operational in a series of steps or phases. Phased installation breaks up the complexity of planning and
execution into manageable pieces, but the total planning and execution effort is greater than with other methods.

11. **Why are additional personnel generally required during the later stages of system implementation?**
Many labour-intensive activities must be performed during installation, testing, and switchover to a new system. Any form of parallel or phased installation increases the number of these activities.

12. **What are the differences between documentation for end users and system operators?**
Operator documentation covers topics such as system start-up and shutdown, backup and recovery, and software installation and upgrade that are not typically covered in end user documentation. System operators are usually more highly trained than end users so their documentation often has fewer 'bells and whistles'.

13. **How or why is system documentation redundant with the system itself? What are the practical implications of this redundancy?**
System documentation describes the architecture and function of a system but that information can also be extracted from the system source code (although with considerably less efficiency). System documentation must be updated every time the system itself is updated to prevent inconsistency.

14. **List the types of documentation needed to support maintenance activities. Which documentation types are needed most frequently? Which are needed least frequently?**
Documentation needed to support system maintenance includes all of the outputs of the analysis, design, and implementation phases. Implementation phase outputs (i.e., source code) are needed most frequently and analysis phase outputs are needed least frequently.

15. **How do training activities differ between end users and system operators?**
End user training emphasizes hands on system use to achieve specific business objectives. Training in the business context is as important as training in system specifics. Relatively large numbers of end users allow one-to-many (classroom) training sessions and train-the-trainer approaches. Variation in skills and experience necessitate multiple training methods and materials. System operator training generally omits the business context. The relatively small number of operators generally precludes classroom-based approaches. Relatively high skill and experience levels allow greater use of self-study as a training method.

16. **How does implementing a maintenance change differ from a new system development project? How are they similar?**
Differences depend on the magnitude of the change. All but the simplest and greatest changes follow the normal SDLC phases, but the scope of activity is restricted to the change area. Significant changes require larger analysis and design scope due to the greater potential for ripple and interaction effects with other portions of the system. At the extreme, a complete overhaul of the system calls for an approach equivalent to that of developing a new system.
17. Why might system software upgrades not be installed? What are the costs of not installing them?
There is always a risk that application software will malfunction or break after upgrading system software. Thus, there is a strong motivation to avoid system software upgrades. However, not upgrading runs the risk of not fixing critical but undiscovered bugs and makes newer system software features unavailable to existing applications.

Case Studies

Case Study: Rethinking Rocky Mountain Outfitters

1. Describe the risks associated with planning the new CSS implementation and announcing the availability of Web ordering to customers. Remember that the new CSS will replace the current telephone and mail-order systems in addition to handling Web orders. How conservative should RMO be with respect to testing, installation, and customer announcements? What is the cost of being too conservative?

The risks associated with rolling out the new CSS and its Web ordering component are similar to those in the HudsonBanc case. But for RMO, failure of the system will result in a permanent loss of revenue, not just delayed cash flow, due to lost orders and angry customers that may never return.

Conservative planning is a must given the timing of the expected system conversion and the large percentage of sales during the holiday shopping season. Announcing the Web ordering system should probably be delayed until the October 31 or December 10 catalogues, especially since the lead-time for preparing and distributing the catalogues is several weeks. Also, note that the Web ordering portion of the system relies on the newest technology – exactly the technology with which RMO IS staff is least familiar.

There are several possible costs of being too conservative. First, the old system may not be able to handle the transaction volume (this presumes that the new system was developed to handle larger volumes than the existing systems). Second, some sales may be lost due to customers who view telephone and mail-based orders as too inconvenient compared to Web-based orders. Lastly, delaying the system rollout may incur additional operating costs if the old systems are more expensive to operate than the new system. If delaying cutover requires running both systems in parallel then operating costs will probably double during the busiest part of the year.

2. What fallback strategies should be developed, if any? What should the 'drop dead' date be for deciding whether to use the new CSS to process holiday orders?
Given the magnitude of the risks, it would be unwise to shut down any of the existing systems before the holiday sales season is over. Even if the new system is ready and operating in time for the holiday rush the old system should be kept in a 'ready' state in case problems are encountered. This may require complete parallel operations to maintain up-to-date data in the old systems. Or, it may be possible to direct update data in the old system on a nightly basis to keep it in sync with the
The drop dead date for deciding whether to rely primarily on the new system should be before the start of the holiday sales season – probably between October 15 and November 15, depending the exact timing of the holiday rush.

3. Develop an installation plan and schedule. Justify your approach(es) and your timetable based on your previous risk analysis.

The new system should be brought on-line one subsystem (mail, telephone, and Web) at a time. The subsystem with the lowest expected transaction volume should be brought on-line first to minimize the impact of any problems that are encountered. This approach will also allow gradual testing of 'back end' functions such as inventory and shipping. One possible schedule is:

November 1 – start processing mail orders in parallel with the old system
November 15 – start processing telephone orders in parallel with the old system
November 30 – start processing Web orders

Parallel processing and data entry on the old system for each order type should not stop for at least 2 weeks to allow sufficient time for problems to surface

4. Analyze the training requirements and develop a training plan and schedule. How can training, data conversion, and testing activities be overlapped or combined? What about training and support for customers using the Web ordering system?

Training to process mail and telephone orders should begin as soon as those portions of the new system have passed integration testing. Sufficient training time (for example, two weeks) should be allowed to decrease error rates and increase efficiency levels before allowing personnel to process 'live' orders. Test data could be taken from orders processed by the old system, which would allow testing personnel to compare the processing results.

For the Web orders, RMO should train staff to handle customer help requests by telephone, email, or Internet chat. It might be possible to combine training for telephone order personnel and telephone-based help for Web-orders. Doing so would allow customers to call the same number for both purposes and it would simplify taking an order by telephone if the customer becomes too frustrated with the Web-based interface.

Key Terms

- **Acceptance test** – A system test performed to determine whether the system fulfils user requirements.
- **Alpha version** – A system that is incomplete but ready for some level of rigorous testing.
- **Beta version** – A system that is stable enough to be tested by end users.
- **Bottom-up development** – A development order that implements modules at the bottom of a structure chart first.
- **Build and smoke test** – A system test that is performed daily.
- **Chief developer team** – A team with a single leader who makes all important
decisions.

- **Collaborative specialist team** – A team with members who have wide variation in and minimal overlap of skills and experience.
- **Cooperating peer team** – A team with members of roughly equal skill and experience with overlapping areas of specialization.
- **Direct installation, or immediate cutover** – An installation method that installs a new system, quickly makes it operational, and immediately turns off any overlapping systems.
- **Driver** – A module, developed for unit testing that simulates the calling behaviour of a module that hasn’t yet been developed.
- **Input, process, output (IPO) development** – A development order that implements input modules first, process modules next, and output modules last.
- **Inspection** – A formal review of design or construction details by a group of developers, where each person plays a specific role.
- **Integration test** – A test of the behaviour of a group of modules or methods.
- **Maintenance release** – A system update that provides bug fixes and small changes to existing features.
- **Parallel installation** – An installation method that operates both the old and new systems for an extended time period.
- **Performance test** – A system test that determines whether a system can meet time-based performance criteria.
- **Phased installation** – An installation method that installs a new system and makes it operational in a series of steps or phases.
- **Production system** – The version of the system used from day to day.
- **Production version, release version, or production release** – A system that is formally distributed to users or made operational.
- **Quality assurance (QA)** – The process of ensuring that an information system meets minimal quality standards.
- **Response time** – The desired or maximum allowable time limit for software response to a query or update.
- **Software maintenance** – Modification of a software product after delivery to correct faults, improve performance or other attributes, or adapt the product to a changed environment.
- **Source code control system (SCCS)** – An automated tool for tracking source code files and controlling changes to those files.
- **Stud** – A module, developed for testing, that simulates the execution or behaviour of a module that hasn’t yet been developed.
- **System documentation** – Descriptions of system functions, architecture, and construction details, as used by maintenance personnel and future developers.
- **System test** – A test of the behaviour of an entire system or independent subsystem.
- **Technical review** – A formal or informal review of design or construction details by a group of developers.
- **Test case** – A formal description of a starting state, one or more events to which the software must respond, and the expected response or ending state.
- **Test data** – A set of starting states and events used to test a module, group of modules, or entire system.
- **Test system** – A copy of the production system that is modified to test changes.
- **Testing buddy** – A programmer assigned to test code written by another programmer.
- **Throughput** – The desired or minimum number of queries and transactions that must
be processed per minute or hour.

- **Top-down development** – A development order that implements modules at the top of a structure chart first.
- **Unit testing, or module testing** – Testing of individual code modules or methods before they are integrated with other modules.
- **User documentation** – Descriptions of how to interact with and maintain the system, as used by end users and system operators.