Modern Systems Analysis and Design

8th Edition

Joseph S. Valacich
Joey F. George

Modern Systems Analysis and Design

8th Edition

Joseph S. Valacich
University of Arizona

Joey F. George
Iowa State University

PEARSON

Boston Columbus Indianapolis New York San Francisco
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montréal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President, Business Publishing: Donna Battista
Editor-in-Chief: Stephanie Wall
Senior Sponsoring Editor: Neera Bhalla
Acquisitions Editor: Samantha Lewis
Program Manager: Emily Biberger
Editorial Assistants: Olivia Vignone, Michael Campbell
Vice President, Product Marketing: Maggie Moylan
Director of Marketing, Digital Services and Products: Jeanette Koskinas
Executive Field Marketing Manager: Adam Goldstein
Field Marketing Manager: Lenny Ann Raper
Product Marketing Assistant: Jessica Quazza
Team Lead, Program Management: Ashley Santora
Team Lead, Project Management: Jeff Holcomb
Project Manager: Ilene Kahn
Operations Specialist: Diane Peirano
Creative Director: Blair Brown
Art Director: Janet Slowik
Vice President, Director of Digital Strategy and Assessment: Paul Gentile
Manager of Learning Applications: Paul DeLuca
Full-Service Project Management and Composition: George Jacob/Integra
Interior Designer: Integra Software Services, Inc.
Cover Designer: Integra Software Services, Inc.
Cover Art: hunthomas/123rf
Printer/Binder: RR Donnelley/Roanoke
Cover Printer: Phoenix Color/Hagerstown

Library of Congress Cataloging-in-Publication Data
Hoffer, Jeffrey A.
Modern systems analysis and design/Jeffrey A. Hoffer, University of Dayton, Joey F. George, Iowa State University, Joseph S. Valacich, University of Arizona.—Eighth edition.
pages cm
Includes bibliographical references and index.
ISBN-13: 978-0-13-420492-5
ISBN-10: 0-13-420492-1
1. System design. 2. System analysis. I. George, Joey F. II. Valacich, Joseph S., 1959– III. Title.
QA76.9.S88H6197 2015
003—dc23

2015013648

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN 10: 0-13-420492-1
ISBN 13: 978-0-13-420492-5

To my mother, Mary Valacich. You are the best!

—Joe

To my mother, Loree George

—Joey

This page intentionally left blank

Brief Contents

This page intentionally left blank

Contents

Preface

DESCRIPTION

Modern Systems Analysis and Design, Eighth Edition, covers the concepts, skills, methodologies, techniques, tools, and perspectives essential for systems analysts to successfully develop information systems. The primary target audience is upper-division undergraduates in a management information systems (MIS) or computer information systems curriculum; a secondary target audience is MIS majors in MBA and MS programs. Although not explicitly written for the junior college and professional development markets, this book can also be used by these programs.

We have over 55 years of combined teaching experience in systems analysis and design and have used that experience to create this newest edition of *Modern Systems Analysis and Design*. We provide a clear presentation of the concepts, skills, and techniques that students need to become effective systems analysts who work with others to create information systems for businesses. We use the systems development life cycle (SDLC) model as an organizing tool throughout the book to provide students with a strong conceptual and systematic framework. The SDLC in this edition has five phases and a circular design.

With this text, we assume that students have taken an introductory course on computer systems and have experience designing programs in at least one programming language. We review basic system principles for those students who have not been exposed to the material on which systems development methods are based. We also assume that students have a solid background in computing literacy and a general understanding of the core elements of a business, including basic terms associated with the production, marketing, finance, and accounting functions.

NEW TO THE EIGHTH EDITION

The following features are new to the Eighth Edition:

- *New material.* To keep up with the changing environment for systems development, Chapter 12 has undergone a complete and thorough revision. While cloud computing is introduced in Chapter 2, it is given extensive coverage in the revised Chapter 12. Service-oriented architecture has been reintroduced to the book in the version of Chapter 12. Other new material includes expansions of two of the appendices to Chapter 7. The appendices on activity diagrams and on Business Process Management Notation now include additional text and figures. Throughout the book figures, tables, and related content have been updated and refreshed.
- *Updated content.* Throughout the book, the content in each chapter has been updated where appropriate. We have expanded our coverage of multiple topics in Chapter 2. Examples of updates in other chapters include revising the information on the information services (IS)/information technology job market in Chapter 1. Another example is Chapter 13, where we have updated and extended the section on information systems security. All screenshots come from current versions of leading software products. We have also made a special effort to update our reference lists, purging out-of-date material and including current references.

- *Dropped material.* In our efforts to keep the book current and to streamline it, the coverage of some things was dropped from this edition. Chapter 1 no longer includes Rapid Application Development. Chapter 12 no longer covers data warehouses or data marts. Chapter 13 no longer includes a section on Electronic Performance Support Systems.
- *Organization.* We have retained the organization of the book first introduced in the Sixth Edition. We have 14 chapters and 6 appendices. The first appendix follows Chapter 1. Four appendices follow Chapter 7, including the new one on business process modeling. The sixth appendix follows Chapter 8. This streamlined organization worked well in the Sixth and Seventh Editions, so we decided to continue with it.
- *Approach to presentation of object-oriented material.* We retain our approach to object-orientation (OO) from the last edition. Brief appendices related to the object-oriented approach continue to appear immediately after related chapters. The OO appendices appear as follows: Chapter 3 features a special OO section on IS project management. Chapter 7 now has three OO appendices: one on use cases; one on sequence diagrams; and one about activity diagrams. (The fourth appendix to Chapter 7 is about Business Process Management Notation, which is not part of UML, although it is governed by the Object Management Group (OMG).) Chapter 8 has a special section on object-oriented database design. The rationale for this organization is the same as in the past: to cleanly separate out structured and object-oriented approaches so that instructors not teaching OO can bypass it. On the other hand, instructors who want to expose their students to object-orientation can now do so with minimal effort devoted to finding the relevant OO material.
- *Updated illustrations of technology.* Screen captures have been updated throughout the text to show examples using the latest versions of programming and Internet development environments (including the latest versions of.NET, Visio, and Microsoft Office) and user interface designs. Many references to websites are provided for students to stay current with technology trends that affect the analysis and design of information systems.

Themes of *Modern Systems Analysis and Design*

1. Systems development is firmly rooted in an organizational context. The successful systems analyst requires a broad understanding of organizations, organizational culture, and organizational operations.
2. Systems development is a practical field. Coverage of current practices as well as accepted concepts and principles is essential in a textbook.
3. Systems development is a profession. Standards of practice, a sense of continuing personal development, ethics, and a respect for and collaboration with the work of others are general themes in the textbook.
4. Systems development has significantly changed with the explosive growth in databases, data-driven systems architectures, rapid development, the Internet, and Agile Methodologies. Systems development and database management can be and should be taught in a highly coordinated fashion. The text is compatible with the Hoffer, Ramesh, and Topi database text, *Modern Database Management,* Eleventh Edition, also published by Pearson. The proper linking of these two textbooks is a strategic opportunity to meet the needs of the IS academic field.

5. Success in systems analysis and design requires not only skills in methodologies and techniques, but also project management skills for managing time, resources, and risks. Thus, learning systems analysis and design requires a thorough understanding of the process as well as the techniques and deliverables of the profession.

Given these themes, this textbook emphasizes the following:

- A business, rather than a technology, perspective
- The role, responsibilities, and mind-set of the systems analyst as well as the systems project manager, rather than those of the programmer or business manager
- The methods and principles of systems development, rather than the specific tools or tool-related skills of the field

DISTINCTIVE FEATURES

The following are some of the distinctive features of *Modern Systems Analysis and Design*:

1. This book is organized in parallel to the Hoffer, Ramesh, and Topi database text, *Modern Database Management*, Twelfth Edition (2016), which will facilitate consistency of frameworks, definitions, methods, examples, and notations to better support systems analysis and design and database courses adopting both texts. Even with the strategic compatibilities between this text and *Modern Database Management*, each of these books is designed to stand alone as a market leader.
2. The grounding of systems development in the typical architecture for systems in modern organizations, including database management and web-based systems.
3. A clear linkage of all dimensions of systems description and modeling—process, decision, and data modeling—into a comprehensive and compatible set of systems analysis and design approaches. Such a broad coverage is necessary so that students understand the advanced capabilities of the many systems development methodologies and tools that are automatically generating a large percentage of code from design specifications.
4. Extensive coverage of oral and written communication skills, including systems documentation, project management, team management, and a variety of systems development and acquisition strategies (e.g., life cycle, prototyping, object orientation, Joint Application Development [JAD], systems reengineering, and Agile Methodologies).
5. Consideration of standards for the methodologies of systems analysis and the platforms on which systems are designed.
6. Discussion of systems development and implementation within the context of change management, conversion strategies, and organizational factors in systems acceptance.
7. Careful attention to human factors in systems design that emphasize usability in both character-based and graphical user interface situations.
8. Visual development products are illustrated and the current limitations technologies are highlighted.
9. The text includes a separate chapter on systems maintenance. Given the type of job many graduates first accept and the large installed base of systems, this chapter covers an important and often neglected topic in systems analysis and design texts.

PEDAGOGICAL FEATURES

The pedagogical features of *Modern Systems Analysis and Design* reinforce and apply the key content of the book.

Three Illustrative Fictional Cases

The text features three fictional cases, described below.

Pine Valley Furniture (PVF): In addition to demonstrating an electronic business-to-consumer shopping website, several other systems development activities from PVF are used to illustrate key points. PVF is introduced in Chapter 3 and revisited throughout the book. As key systems development life cycle concepts are presented, they are applied and illustrated with this descriptive case. For example, in Chapter 5 we explore how PVF plans a development project for a customer tracking system. A margin icon identifies the location of the case segments.

Hoosier Burger (HB): This second illustrative case is introduced in Chapter 7 and revisited throughout the book. HB is a fictional fast-food restaurant in Bloomington, Indiana. We use this case to illustrate how analysts would develop and implement an automated food-ordering system. A margin icon identifies the location of the case segments.

Petrie Electronics: This fictional retail electronics company is used as an extended project case at the end of 12 of the 14 chapters, beginning with Chapter 2. Designed to bring the chapter concepts to life, this case illustrates how a company initiates, plans, models, designs, and implements a customer loyalty system. Discussion questions are included to promote critical thinking and class participation. Suggested solutions to the discussion questions are provided in the Instructor's Manual.

End-of-Chapter Material

We developed an extensive selection of end-of-chapter materials that are designed to accommodate various learning and teaching styles.

- *Chapter Summary.* Reviews the major topics of the chapter and previews the connection of the current chapter with future ones.
- *Key Terms.* Designed as a self-test feature, students match each key term in the chapter with a definition.
- *Review Questions.* Test students' understanding of key concepts.
- *Problems and Exercises.* Test students' analytical skills and require them to apply key concepts.
- *Field Exercises.* Give students the opportunity to explore the practice of systems analysis and design in organizations.
- *Margin Term Definitions.* Each key term and its definition appear in the margin. Glossaries of terms and acronyms appear at the back of the book.
- *References.* References are located at the end of each chapter. The total number of references in this text amounts to over 100 books, journals, and websites that can provide students and faculty with additional coverage of topics.

USING THIS TEXT

As stated earlier, this book is intended for mainstream systems analysis and design courses. It may be used in a one-semester course on systems analysis and design or over two quarters (first in a systems analysis and then in a systems design course). Because this book text parallels *Modern Database Management*, chapters from this book and from *Modern Database Management* can be used in various sequences suitable for your curriculum. The book will be adopted typically in business schools or departments, not in computer science programs. Applied computer science or computer technology programs may also adopt the book.

The typical faculty member who will find this book most interesting is someone who

- has a practical, rather than technical or theoretical, orientation;
- has an understanding of databases and the systems that use databases; and
- uses practical projects and exercises in their courses.

More specifically, academic programs that are trying to better relate their systems analysis and design and database courses as part of a comprehensive understanding of systems development will be especially attracted to this book.

The outline of the book generally follows the systems development life cycle, which allows for a logical progression of topics; however, it emphasizes that various approaches (e.g., prototyping and iterative development) are also used, so what appears to be a logical progression often is a more cyclic process. Part One provides an overview of systems development and previews the remainder of the book. Part One also introduces students to the many sources of software that they can draw on to build their systems and to manage projects. The remaining four parts provide thorough coverage of the five phases of a generic systems development life cycle, interspersing coverage of alternatives to the SDLC as appropriate. Some chapters may be skipped depending on the orientation of the instructor or the students' background. For example, Chapter 3 (Managing the Information Systems Project) can be skipped or quickly reviewed if students have completed a course on project management. Chapter 4 (Identifying and Selecting Systems Development Projects) can be skipped if the instructor wants to emphasize systems development once projects are identified or if there are fewer than 15 weeks available for the course. Chapters 8 (Structuring System Data Requirements) and 9 (Designing Databases) can be skipped or quickly scanned (as a refresher) if students have already had a thorough coverage of these topics in a previous database or data structures course. The sections on object orientation in Chapters 3, 7, and 8 can be skipped if faculty wish to avoid object-oriented topics. Finally, Chapter 14 (Maintaining Information Systems) can be skipped if these topics are beyond the scope of your course.

Because the material is presented within the flow of a systems development project, it is not recommended that you attempt to use the chapters out of sequence, with a few exceptions: Chapter 9 (Designing Databases) can be taught after Chapters 10 (Designing Forms and Reports) and 11 (Designing Inferfaces and Dialogues), but Chapters 10 and 11 should be taught in sequence.

THE SUPPLEMENT PACKAGE: WWW.PEARSONHIGHERED.COM/HOFFER

A comprehensive and flexible technology support package is available to enhance the teaching and learning experience. All instructor supplements are available on the text website: www.pearsonhighered.com/hoffer.

Instructor Resources

At the Instructor Resource Center, www.pearsonhighered.com/irc, instructors can easily register to gain access to a variety of instructor resources available with this text in downloadable format. If assistance is needed, our dedicated technical support team is ready to help with the media supplements that accompany this text. Visit http://247.pearsoned.com for answers to frequently asked questions and toll-free user support phone numbers.

The following supplements are available with this text:

- Instructor's Manual
- Test Bank
- TestGen® Computerized Test Bank
- PowerPoint Presentation

ACKNOWLEDGMENTS

The authors have been blessed by considerable assistance from many people on all aspects of preparation of this text and its supplements. We are, of course, responsible for what eventually appears between the covers, but the insights, corrections, contributions, and prodding of others have greatly improved our manuscript. Over the years, dozens of people have reviewed the various editions of this textbook. Their contributions have stimulated us, frequently prompting us to include new topics and innovative pedagogy. We greatly appreciate the efforts of the many faculty and practicing systems analysts who have reviewed this text.

We extend a special note of thanks to Jeremy Alexander, who was instrumental in conceptualizing and writing the PVF WebStore feature that appears in Chapters 4 through 14. The addition of this feature has helped make those chapters more modern and innovative. We would also like to thank Jeff Jenkins, of Brigham Young University, for his help with the Visual Basic screenshots in the current edition.

We also wish to thank Atish Sinha of the University of Wisconsin–Milwaukee for writing the original version of some of the object-oriented analysis and design material. Dr. Sinha, who has been teaching this topic for several years to both undergraduates and MBA students, executed a challenging assignment with creativity and cooperation.

We are also indebted to our undergraduate and MBA students, who have given us many helpful comments as they worked with drafts of this text, and our thanks go to Fred McFadden (University of Colorado, Colorado Springs), Mary Prescott (University of South Florida), Ramesh Venkataraman (Indiana University), and Heikki Topi (Bentley University) for their assistance in coordinating this text with its companion book, *Modern Database Management*, also by Pearson Education.

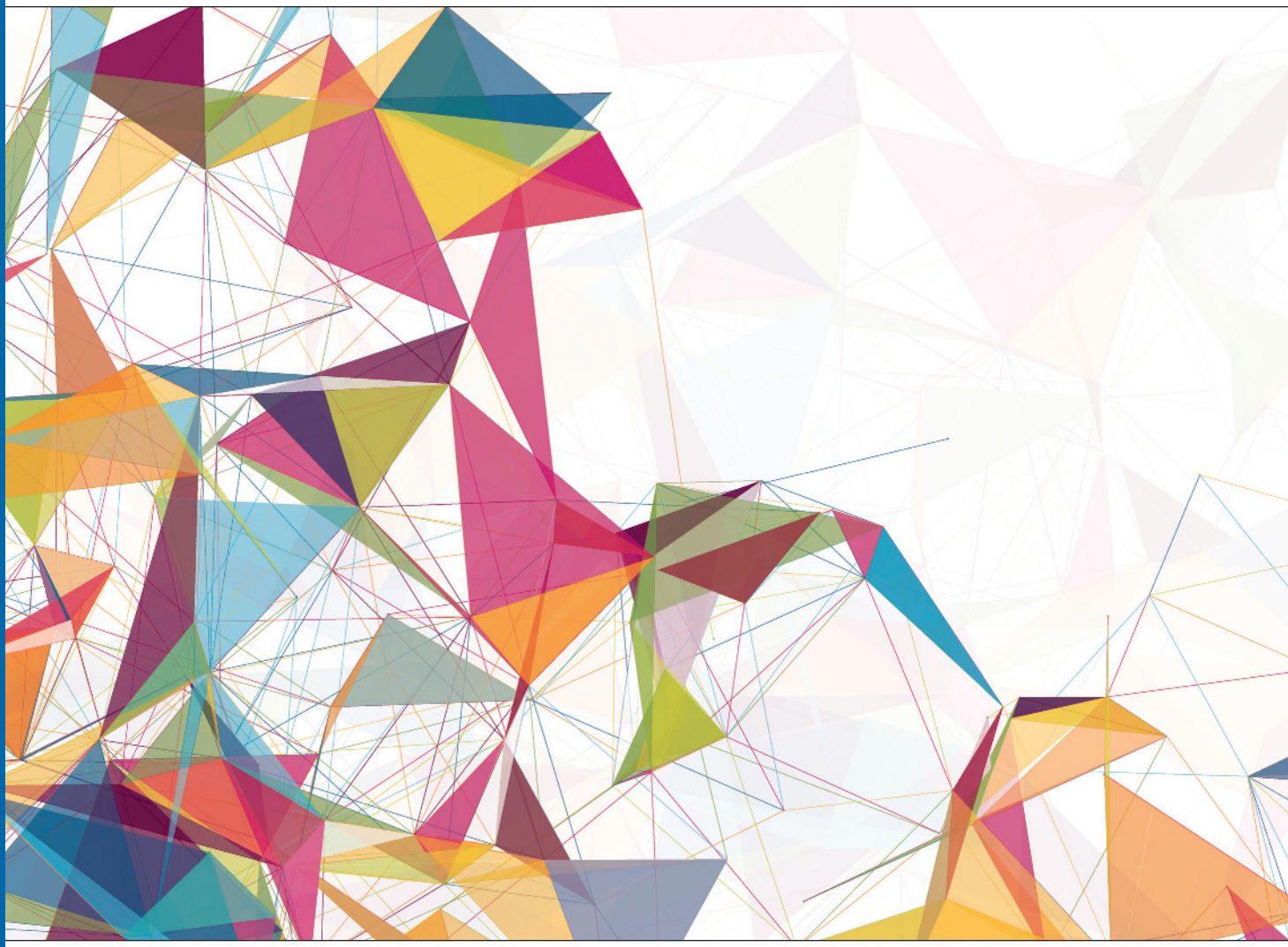
Finally, we have been fortunate to work with a large number of creative and insightful people at Pearson, who have added much to the development, format, and production of this text. We have been thoroughly impressed with their commitment to this text and to the IS education market. These people include: Nicole Sam (Acquisitions Editor), Neeraj Bhalla (Senior Sponsoring Editor), Olivia Vignone (Editorial Assistant), Ilene Kahn (Project Manager). We would also like to thank George Jacobs and the crew at Integra Software Services, Inc.

The writing of this text has involved thousands of hours of time from the authors and from all of the people listed previously. Although our names will be visibly associated with this book, we know that much of the credit goes to the individuals and organizations listed here for any success it might achieve. It is important for the reader to recognize all the individuals and organizations that have been committed to the preparation and production of this book.

Joseph S. Valacich, Tucson, Arizona
Joey F. George, Ames, Iowa

PART ONE

Foundations for Systems Development



PART ONE

Foundations for Systems Development

You are beginning a journey that will enable you to build on every aspect of your education and experience. Becoming a systems analyst is not a goal; it is a path to a rich and diverse career that will allow you to exercise and continue to develop a wide range of talents. We hope that this introductory part of the text helps open your mind to the opportunities of the systems analysis and design field and to the engaging nature of systems work.

Chapter 1 shows you what systems analysis and design is all about and how it has evolved over the past several decades. As businesses and systems have become more sophisticated and more complex, there has been an increasing emphasis on speed in systems analysis and design. Systems development began as an art, but most businesspeople soon realized this was not a tenable long-term solution to developing systems to support business processes. Systems development became more structured and more like engineering, and managers stressed the importance of planning, project management, and documentation. Now we are witnessing a reaction against excesses in all three of these areas, and the focus has shifted to agile development. The evolution of systems analysis and design and the current focus on agility are explained in Chapter 1. It is also important, however, that you remember that systems analysis and design exists within a multifaceted organizational context that involves other organizational members and external parties. Understanding systems development requires an understanding not only of each technique, tool, and method, but also of how these elements complement and support each other within an organizational setting.

As you read this book, you'll also discover that the systems analysis and design field is constantly adapting to new situations due to a strong commitment to constant improvement. Our goal in this book is to provide you with a mosaic of the skills needed to work effectively in any environment where you may find yourself, armed with the knowledge to determine the best practices for that situation and argue for them effectively.

Chapter 2 presents an introduction to the many sources from which software and software components can be obtained. Back when systems analysis and design was an art, all systems were written from scratch by in-house experts. Businesses had little choice. Now there is little excuse for in-house development, so it becomes crucial that systems analysts understand the software industry and the many different sources of software. Chapter 2 provides an initial map of the software industry landscape and explains most of the many choices available to systems analysts.

Chapter 3 addresses a fundamental characteristic of life as a systems analyst: working within the framework of projects with constrained resources. All systems-related work demands attention to deadlines, working within budgets, and coordinating the work of various people. The very nature of the systems development life cycle (SDLC) implies a systematic approach to a project, which is a group of related activities leading to a final deliverable. Projects must be planned, started, executed, and completed. The planned work of the project must be represented so that all interested parties can review and understand it. In your job as a systems analyst, you will have to work within the schedule and other project plans, and thus it is important to understand the management process controlling your work.

Finally, Part I introduces the Petrie Electronics case. The Petrie case helps demonstrate how what you learn in each chapter might fit into a practical organizational situation. The case begins after Chapter 2; the remaining book chapters through Chapter 13 each have an associated case installment. The first section introduces the company and its existing information systems. This introduction provides insights into Petrie, which will help you understand the company more completely when we look at the requirements and design for new systems in later case sections.

Chapter 1

The Systems Development Environment

Learning Objectives

After studying this chapter, you should be able to

1.1 define *information systems analysis and design*,

1.2 describe the information systems development life cycle (SDLC),

1.3 explain computer-aided software engineering (CASE) tools,

1.4 describe the Agile Methodologies and eXtreme Programming, and

1.5 explain object-oriented analysis and design and the Rational Unified Process (RUP).

Introduction

Information systems analysis and design is a complex, challenging, and stimulating organizational process that a team of business and systems professionals uses to develop and maintain computer-based information systems. Although advances in information technology continually give us new capabilities, the analysis and design of information systems is driven from an organizational perspective. An organization might consist of a whole enterprise, specific departments, or individual work groups. Organizations can respond to and anticipate problems and opportunities through innovative use of information technology. Information systems analysis and design is therefore an organizational improvement process. Systems are built and rebuilt for organizational benefits. Benefits result from adding value during the process of creating, producing, and supporting the organization's products and services. Thus the analysis and design of information systems is based on your understanding of the organization's objectives, structure, and processes, as well as your knowledge of how to exploit information technology for advantage.

In the current business environment, the Internet, especially the World Wide Web, has been firmly integrated into an organization's way of doing business. Although you are probably most familiar with marketing done on the web and web-based retailing sites, such as eBay or Amazon.com, the overwhelming majority of business use of the web is business-to-business applications. These applications run the gamut of everything businesses do, including transmitting orders and payments to suppliers, fulfilling orders and collecting payments from customers, maintaining business relationships, and establishing electronic marketplaces where businesses can shop online for the best deals on resources they need for assembling their products and services. Although the Internet seems to pervade business these days, it is important to remember that many of the key aspects of business—offering a product or service for sale, collecting payment, paying employees, maintaining supplier and client relationships—have not changed. Understanding the business and how it functions is still the key to successful systems development, even in the fast-paced, technology-driven environment that organizations find themselves in today.

Careers in information technology (IT) present a great opportunity for you to make a significant and visible impact on business. The demand for skilled information technology workers is growing. According to the US Bureau of Labor Statistics, the professional IT workforce will grow by more than 22 percent between 2010 and 2020 (Thibodeau, 2012). The fastest growth will come for software developers (32 percent) and database administrators (31 percent). One particular aspect of the information technology industry, cloud computing, created almost 14 million technology and related jobs between 2012 and 2015 (McDougall, 2012). Annual revenues from

Information systems analysis and design
The complex organizational process whereby computer-based information systems are developed and maintained.

cloud computing will be over $1.1 trillion (USD) starting that year. And the growth will be global, with the number of cloud computing jobs in Brazil increasing by 186 percent, the number of jobs in China and India almost doubling, and growth in cloud-related jobs increasing by 66 percent in the United States. (See more about cloud computing in Chapter 2.) With the challenges and opportunities of dealing with rapid advances in technology, it is difficult to imagine a more exciting career choice than information technology, and systems analysis and design is a big part of the IT landscape. Furthermore, analyzing and designing information systems will give you the chance to understand organizations at a depth and breadth that might take many more years to accomplish in other careers.

Application software
Computer software designed to support organizational functions or processes.

An important (but not the only) result of systems analysis and design is **application software**, software designed to support a specific organizational function or process, such as inventory management, payroll, or market analysis. In addition to application software, the total information system includes the hardware and systems software on which the application software runs, documentation and training materials, the specific job roles associated with the overall system, controls, and the people who use the software along with their work methods. Although we will address all of these various dimensions of the overall system, we will emphasize application software development—your primary responsibility as a systems analyst.

In the early years of computing, analysis and design was considered an art. Now that the need for systems and software has become so great, people in industry and academia have developed work methods that make analysis and design a disciplined process. Our goal is to help you develop the knowledge and skills needed to understand and follow such software engineering processes. Central to software engineering processes (and to this book) are various methodologies, techniques, and tools that have been developed, tested, and widely used over the years to assist people like you during systems analysis and design.

Methodologies are comprehensive, multiple-step approaches to systems development that will guide your work and influence the quality of your final product—the information system. A methodology adopted by an organization will be consistent with its general management style (e.g., an organization's orientation toward consensus management will influence its choice of systems development methodology). Most methodologies incorporate several development techniques.

Techniques are particular processes that you, as an analyst, will follow to help ensure that your work is well thought out, complete, and comprehensible to others on your project team. Techniques provide support for a wide range of tasks, including conducting thorough interviews to determine what your system should do, planning and managing the activities in a systems development project, diagramming the system's logic, and designing the reports your system will generate.

Tools are typically computer programs that make it easy to use and benefit from techniques and to faithfully follow the guidelines of the overall development methodology. To be effective, techniques and tools must both be consistent with an organization's systems development methodology. Techniques and tools must make it easy for systems developers to conduct the steps called for in the methodology. These three elements—methodologies, techniques, and tools—work together to form an organizational approach to systems analysis and design (see Figure 1-1).

Systems analyst
The organizational role most responsible for the analysis and design of information systems.

Although many people in organizations are responsible for systems analysis and design, in most organizations the **systems analyst** has the primary responsibility. When you begin your career in systems development, you will most likely begin as a systems analyst or as a programmer with some systems analysis responsibilities. The primary role of a systems analyst is to study the problems and needs of an organization in order to determine how people, methods, and information technology can best be combined to bring about improvements in the organization. A systems analyst helps system users and other business managers define their requirements for new or enhanced information services. As such, a systems analyst is an agent of change and innovation.

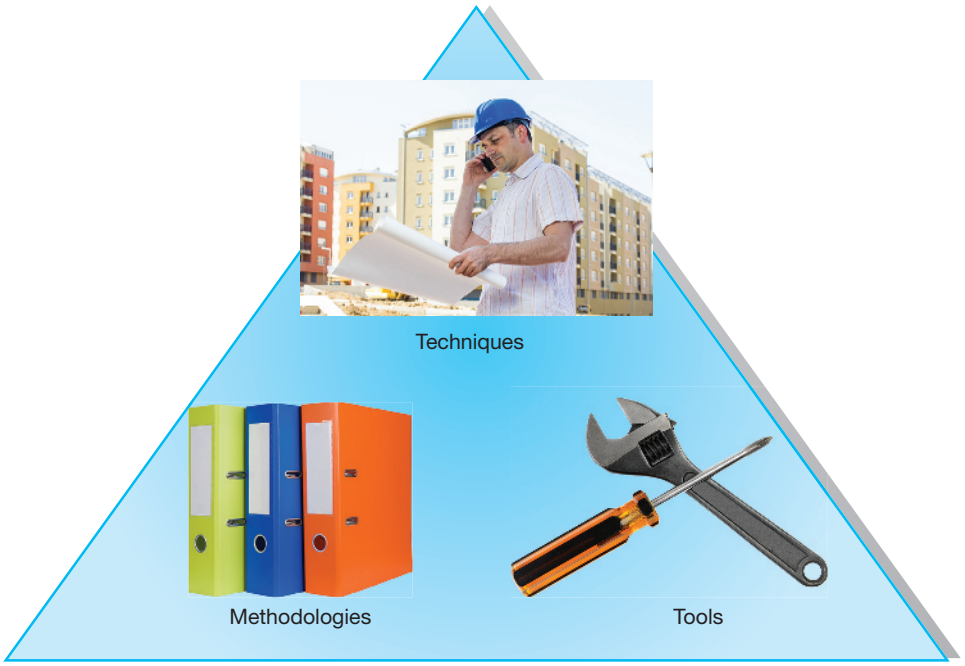


FIGURE 1-1
An organizational approach to systems analysis and design is driven by methodologies, techniques, and tools
Sources: Top: Mitarart/Fotolia; Left: Lev/Fotolia; Right: PaulPaladin/Fotolia

In the rest of this chapter, we will examine the systems approach to analysis and design. You will learn how systems analysis and design has changed over the decades as computing has become more central to business. You will learn about the systems development life cycle, which provides the basic overall structure of the systems development process and of this book. This chapter ends with a discussion of some of the methodologies, techniques, and tools created to support the systems development process.

A MODERN APPROACH TO SYSTEMS ANALYSIS AND DESIGN

The analysis and design of computer-based information systems began in the 1950s. Since then, the development environment has changed dramatically, driven by organizational needs as well as by rapid changes in the technological capabilities of computers. In the 1950s, development focused on the processes the software performed. Because computer power was a critical resource, efficiency of processing became the main goal. Computers were large, expensive, and not very reliable. Emphasis was placed on automating existing processes, such as purchasing or paying, often within single departments. All applications had to be developed in machine language or assembly language, and they had to be developed from scratch because there was no software industry. Because computers were so expensive, computer memory was also at a premium, so system developers conserved as much memory as possible for data storage.

The first procedural, or third-generation, computer programming languages did not become available until the beginning of the 1960s. Computers were still large and expensive, but the 1960s saw important breakthroughs in technology that enabled the development of smaller, faster, less expensive computers—minicomputers—and the beginnings of the software industry. Most organizations still developed their applications from scratch using their in-house development staff. Systems development was more an art than a science. This view of systems development began to change in the 1970s, however, as organizations started to realize how expensive it was to develop customized information systems for every application. Systems development came to be more

disciplined as many people worked to make it more like engineering. Early database management systems, using hierarchical and network models, helped bring discipline to the storage and retrieval of data. The development of database management systems helped shift the focus of systems development from processes first to data first.

The 1980s were marked by major breakthroughs in computing in organizations, as microcomputers became key organizational tools. The software industry expanded greatly as more and more people began to write off-the-shelf software for microcomputers. Developers began to write more and more applications in fourth-generation languages, which, unlike procedural languages, instructed a computer on what to do instead of how to do it. Computer-aided software engineering (CASE) tools were developed to make systems developers' work easier and more consistent. As computers continued to get smaller, faster, and cheaper, and as the operating systems for computers moved away from line prompt interfaces to windows- and icon-based interfaces, organizations moved to applications with more graphics. Organizations developed less software in-house and bought relatively more from software vendors. The systems developer's job went through a transition from builder to integrator.

The systems development environment of the late 1990s focused on systems integration. Developers used visual programming environments, such as PowerBuilder or Visual Basic, to design the user interfaces for systems that run on client/server platforms. The database, which may be relational or object-oriented, and which may have been developed using software from firms such as Oracle, Microsoft, or Ingres, resided on the server. In many cases, the application logic resided on the same server. Alternatively, an organization may have decided to purchase its entire enterprise-wide system from companies such as SAP AG or Oracle. Enterprise-wide systems are large, complex systems that consist of a series of independent system modules. Developers assemble systems by choosing and implementing specific modules. Starting in the middle years of the 1990s, more and more systems development efforts focused on the Internet, especially the web.

Today there is continued focus on developing systems for the Internet and for firms' intranets and extranets. As happened with traditional systems, Internet developers now rely on computer-based tools to speed and simplify the development of web-based systems. Many CASE tools directly support web application development. More and more, systems implementation involves a three-tier design, with the database on one server, the application on a second server, and client logic located on user machines. Another important development is the move to wireless system components. Wireless devices can access web-based applications from almost anywhere. Finally, the trend continues toward assembling systems from programs and components purchased off the shelf. In many cases, organizations do not develop the application in-house. They don't even run the application in-house, choosing instead to use the application on a per-use basis by accessing it through the cloud.

DEVELOPING INFORMATION SYSTEMS AND THE SYSTEMS DEVELOPMENT LIFE CYCLE

Systems development methodology
A standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems.

Systems development life cycle (SDLC)
The traditional methodology used to develop, maintain, and replace information systems.

Most organizations find it beneficial to use a standard set of steps, called a **systems development methodology**, to develop and support their information systems. Like many processes, the development of information systems often follows a life cycle. For example, a commercial product follows a life cycle in that it is created, tested, and introduced to the market. Its sales increase, peak, and decline. Finally, the product is removed from the market and replaced by something else. The **systems development life cycle (SDLC)** is a common methodology for systems development in many organizations; it features several phases that mark the progress of the systems analysis and design effort. Every textbook author and information systems development organization uses a slightly different life-cycle model, with anywhere from 3 to almost 20 identifiable phases.

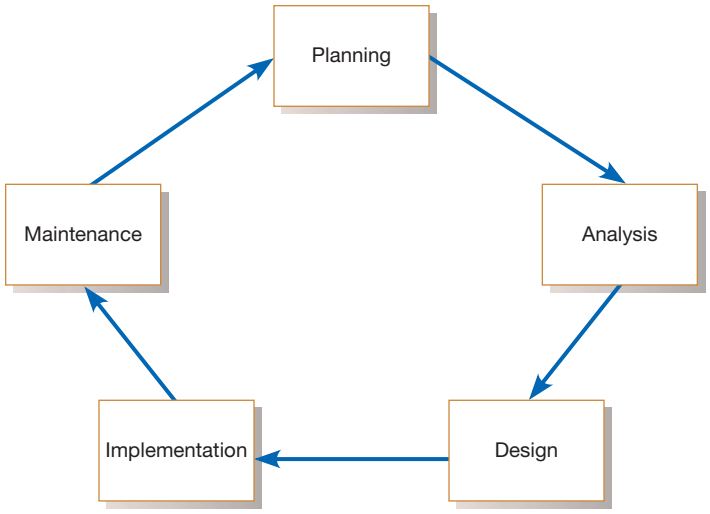


FIGURE 1-2
Systems development life cycle

The life cycle can be thought of as a circular process in which the end of the useful life of one system leads to the beginning of another project that will develop a new version or replace an existing system altogether (see Figure 1-2). At first glance, the life cycle appears to be a sequentially ordered set of phases, but it is not. The specific steps and their sequence are meant to be adapted as required for a project, consistent with management approaches. For example, in any given SDLC phase, the project can return to an earlier phase if necessary. Similarly, if a commercial product does not perform well just after its introduction, it may be temporarily removed from the market and improved before being reintroduced. In the SDLC, it is also possible to complete some activities in one phase in parallel with some activities of another phase. Sometimes the life cycle is iterative; that is, phases are repeated as required until an acceptable system is found. Some people consider the life cycle to be a spiral, in which we constantly cycle through the phases at different levels of detail (see Figure 1-3). However conceived, the systems development life cycle used in an organization is an orderly set of activities conducted and planned for each development project. The skills required of a systems analyst apply to all life-cycle models. Software is the most obvious end product of the life cycle; other essential outputs include documentation about the system and how it was developed, as well as training for users.

Every medium to large corporation and every custom software producer will have its own specific life cycle or systems development methodology in place

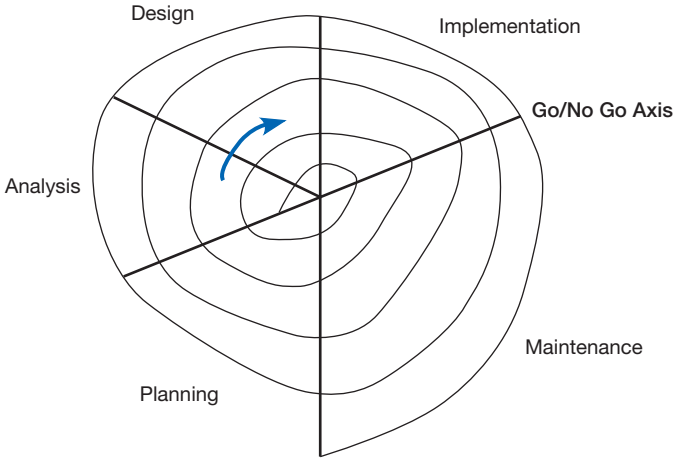


FIGURE 1-3
Evolutionary model

FIGURE 1-4
U.S. Department of Justice's systems development life cycle

(*Source:* Diagram based on *http://www.justice.gov/archive/jmd/irm/lifecycle/ch1.htm#para1.2*)

Top to bottom: haveseen/Shutterstock; Kruwt/Fotolia; Bedrin/Shutterstock; Pressmaster/Shutterstock; pilotl39/Fotolia; Sozaijiten; Elnur/Fotolia; rtguest/Shutterstock; michaeljung/Shutterstock; AleksaStudio/Shutterstock

(see Figure 1-4). Even if a particular methodology does not look like a cycle, and Figure 1-4 does not, you will probably discover that many of the SDLC steps are performed and SDLC techniques and tools are used. Learning about systems analysis and design from the life-cycle approach will serve you well no matter which systems development methodology you use.

When you begin your first job, you will likely spend several weeks or months learning your organization's SDLC and its associated methodologies, techniques, and tools. In order to make this book as general as possible, we follow a rather generic life-cycle model, as described in more detail in Figure 1-5. Notice that our model is circular. We use this SDLC as one example of a methodology but, more important, as a way to arrange the topics of systems analysis and design. Thus, what you learn in this book, you can apply to almost any life cycle you might follow. As we describe this SDLC throughout the book, you will see that each phase has specific outcomes and deliverables that feed important information to other phases. At the end of each phase, a systems development project reaches a milestone and, as deliverables are produced, they are often reviewed by parties outside the project team. In the rest of this section, we provide a brief overview of each SDLC phase. At the end of the section, we summarize this discussion in a table that lists the main deliverables or outputs from each SDLC phase.

Planning
The first phase of the SDLC in which an organization's total information system needs are identified, analyzed, prioritized, and arranged.

The first phase in the SDLC is **planning**. In this phase, someone identifies the need for a new or enhanced system. In larger organizations, this recognition may be part of a corporate and systems planning process. Information needs of the organization as a whole are examined, and projects to meet these needs are proactively identified. The organization's information system needs may result from requests to deal with problems in current procedures, from the desire to perform additional

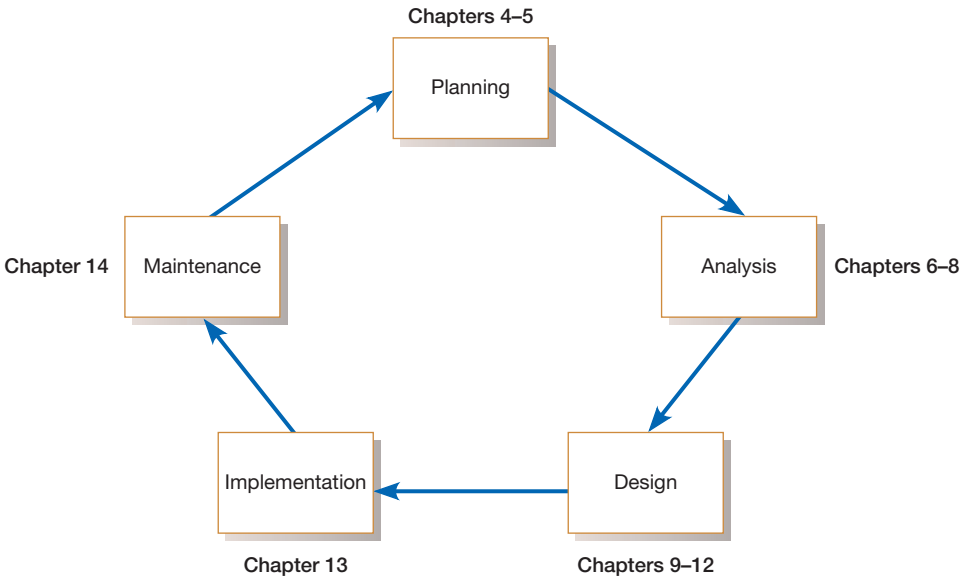


FIGURE 1-5
SDLC-based guide to this book

tasks, or from the realization that information technology could be used to capitalize on an existing opportunity. These needs can then be prioritized and translated into a plan for the information systems department, including a schedule for developing new major systems. In smaller organizations (as well as in large ones), determination of which systems to develop may be affected by ad hoc user requests submitted as the need for new or enhanced systems arises, as well as from a formalized information planning process. In either case, during project identification and selection, an organization determines whether resources should be devoted to the development or enhancement of each information system under consideration. The outcome of the project identification and selection process is a determination of which systems development projects should be undertaken by the organization, at least in terms of an initial study.

Two additional major activities are also performed during the planning phase: the formal, yet still preliminary, investigation of the system problem or opportunity at hand and the presentation of reasons why the system should or should not be developed by the organization. A critical step at this point is determining the scope of the proposed system. The project leader and initial team of systems analysts also produce a specific plan for the proposed project the team will follow using the remaining SDLC steps. This baseline project plan customizes the standardized SDLC and specifies the time and resources needed for its execution. The formal definition of a project is based on the likelihood that the organization's information systems department is able to develop a system that will solve the problem or exploit the opportunity and determine whether the costs of developing the system outweigh the benefits it could provide. The final presentation of the business case for proceeding with the subsequent project phases is usually made by the project leader and other team members to someone in management or to a special management committee with the job of deciding which projects the organization will undertake.

Analysis
The second phase of the SDLC in which system requirements are studied and structured.

The second phase in the SDLC is **analysis**. During this phase, the analyst thoroughly studies the organization's current procedures and the information systems used to perform organizational tasks. Analysis has two subphases. The first is requirements determination. In this subphase, analysts work with users to determine what the users want from a proposed system. The requirements determination process usually involves a careful study of any current systems, manual and computerized, that might be replaced or enhanced as part of the project. In the second part of analysis, analysts study the requirements and structure them according to their

interrelationships and eliminate any redundancies. The output of the analysis phase is a description of (but not a detailed design for) the alternative solution recommended by the analysis team. Once the recommendation is accepted by those with funding authority, the analysts can begin to make plans to acquire any hardware and system software necessary to build or operate the system as proposed.

Design
The third phase of the SDLC in which the description of the recommended solution is converted into logical and then physical system specifications.

The third phase in the SDLC is **design**. During design, analysts convert the description of the recommended alternative solution into logical and then physical system specifications. The analysts must design all aspects of the system, from input and output screens to reports, databases, and computer processes. The analysts must then provide the physical specifics of the system they have designed, either as a model or as detailed documentation, to guide those who will build the new system. That part of the design process that is independent of any specific hardware or software platform is referred to as **logical design**. Theoretically, the system could be implemented on any hardware and systems software. The idea is to make sure that the system functions as intended. Logical design concentrates on the business aspects of the system and tends to be oriented to a high level of specificity.

Logical design
The part of the design phase of the SDLC in which all functional features of the system chosen for development in analysis are described independently of any computer platform.

Once the overall high-level design of the system is worked out, the analysts begin turning logical specifications into physical ones. This process is referred to as **physical design**. As part of physical design, analysts design the various parts of the system to perform the physical operations necessary to facilitate data capture, processing, and information output. This can be done in many ways, from creating a working model of the system to be implemented to writing detailed specifications describing all the different parts of the system and how they should be built. In many cases, the working model becomes the basis for the actual system to be used. During physical design, the analyst team must determine many of the physical details necessary to build the final system, from the programming language the system will be written in, to the database system that will store the data, to the hardware platform on which the system will run. Often the choices of language, database, and platform are already decided by the organization or by the client, and at this point these information technologies must be taken into account in the physical design of the system. The final product of the design phase is the physical system specifications in a form ready to be turned over to programmers and other system builders for construction. Figure 1-6 illustrates the differences between logical and physical design.

Physical design
The part of the design phase of the SDLC in which the logical specifications of the system from logical design are transformed into technology-specific details from which all programming and system construction can be accomplished.

The fourth phase in the SDLC is **implementation**. The physical system specifications, whether in the form of a detailed model or as detailed written specifications, are turned over to programmers as the first part of the implementation phase. During implementation, analysts turn system specifications into a working system that is tested and then put into use. Implementation includes coding, testing, and installation. During coding, programmers write the programs that make up the system. Sometimes the code is generated by the same system used to build the detailed model of the system. During testing, programmers and analysts test individual programs and the entire system in order to find and correct errors. During installation, the new system becomes part of the daily activities of the organization. Application software is installed, or loaded, on existing or new hardware, and users are introduced to the new system and trained. Testing and installation should be planned for as early as the project initiation and planning phase; both testing and installation require extensive analysis in order to develop exactly the right approach.

Implementation
The fourth phase of the SDLC, in which the information system is coded, tested, installed, and supported in the organization.

Implementation activities also include initial user support such as the finalization of documentation, training programs, and ongoing user assistance. Note that documentation and training programs are finalized during implementation; documentation is produced throughout the life cycle, and training (and education) occurs from the inception of a project. Implementation can continue for as long as the system exists, because ongoing user support is also part of implementation. Despite the best efforts of analysts, managers, and programmers, however,

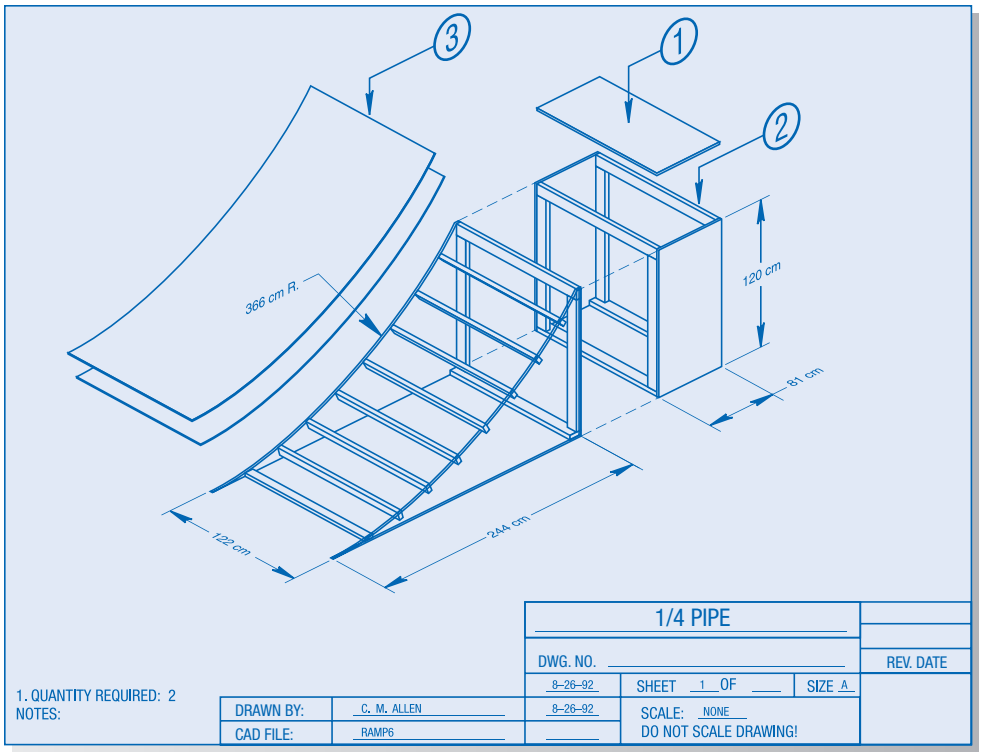
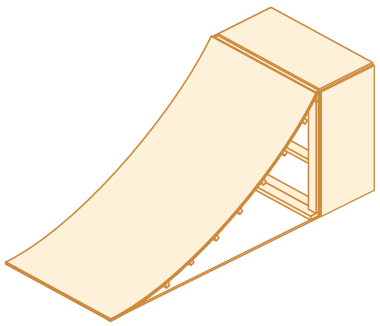


FIGURE 1-6
Difference between logical design and physical design
(a) A skateboard ramp blueprint (logical design)
(*Sources: www.tumyeto.com/tydu/skatebrd/organizations/plans/14pipe.jpg; www.tumyeto.com/tydu/skatebrd/organizations/plans/iuscblue.html.* Accessed September 16, 1999. Reprinted by permission of the International Association of Skateboard Companies.)



(b) A skateboard ramp (physical design)

installation is not always a simple process. Many well-designed systems have failed because the installation process was faulty. Even a well-designed system can fail if implementation is not well managed. Because the project team usually manages implementation, we stress implementation issues throughout this book.

The fifth and final phase in the SDLC is **maintenance**. When a system (including its training, documentation, and support) is operating in an organization, users sometimes find problems with how it works and often think of better ways to perform its functions. Also, the organization's needs with respect to the system change over time. In maintenance, programmers make the changes that users ask for and modify the system to reflect evolving business conditions. These changes are necessary to keep the system running and useful. In a sense, maintenance is not a separate phase but a repetition of the other life cycle phases required to study and implement the needed changes. One might think of maintenance as an overlay on the life cycle rather than as a separate phase. The amount of time and effort devoted to maintenance depends a great deal on the performance of the previous phases of the life cycle. There inevitably comes a time, however, when an information system is no longer performing as desired, when maintenance costs become prohibitive, or when an organization's needs have changed substantially. Such problems indicate that it is time to begin designing the system's replacement, thereby completing the loop

Maintenance
The final phase of the SDLC, in which an information system is systematically repaired and improved.

TABLE 1-1 Products of SDLC Phases

Phase	Products, Outputs, or Deliverables
Planning	Priorities for systems and projects; an architecture for data, networks, and selection hardware, and information systems management are the result of associated systems
	Detailed steps, or work plan, for project
	Specification of system scope and planning and high-level system requirements or features
	Assignment of team members and other resources
	System justification or business case
Analysis	Description of current system and where problems or opportunities exist, with a general recommendation on how to fix, enhance, or replace current system
	Explanation of alternative systems and justification for chosen alternative
Design	Functional, detailed specifications of all system elements (data, processes, inputs, and outputs)
	Technical, detailed specifications of all system elements (programs, files, network, system software, etc.)
	Acquisition plan for new technology
Implementation	Code, documentation, training procedures, and support capabilities
Maintenance	New versions or releases of software with associated updates to documentation, training, and support

and starting the life cycle over again. Often the distinction between major maintenance and new development is not clear, which is another reason maintenance often resembles the life cycle itself.

The SDLC is a highly linked set of phases whose products feed the activities in subsequent phases. Table 1-1 summarizes the outputs or products of each phase based on the in-text descriptions. The chapters on the SDLC phases will elaborate on the products of each phase as well as on how the products are developed.

Throughout the SDLC, the systems development project itself must be carefully planned and managed. The larger the systems project, the greater the need for project management. Several project management techniques have been developed over the past decades, and many have been made more useful through automation. Chapter 3 contains a more detailed treatment of project planning and management techniques. Next, we will discuss some of the criticisms of the SDLC and present alternatives developed to address those criticisms. First, however, we will introduce you to a specialized SDLC that focuses on security during development.

A SPECIALIZED SYSTEMS DEVELOPMENT LIFE CYCLE

Although the basic SDLC provides an overview of the systems development process, the concept of the SDLC can also be applied to very specific aspects of the process. As mentioned previously, the maintenance phase can be described in terms of the SDLC. Another example of a specialized SDLC is Microsoft's Security Development Lifecycle (SDL) (see http://www.microsoft.com/security/sdl/default.aspx for details). The Security Development Lifecycle is depicted in Figure 1-7. First note how the five basic phases of the development life cycle (in green) are not exactly the same as the five phases of the SDLC we will use in this book. Three of the five phases are almost identical to the phases in our SDLC. The Microsoft SDL starts with "requirements," which is similar to "analysis"; this is followed by the design phase, which is followed by implementation. Our life cycle starts with planning and ends with maintenance. Both of these phases are peculiar to systems development in an organizational context, where

FIGURE 1-7
Microsoft's Security Development Lifecycle (SDL)
(*Source: http://www.microsoft.com/security/sdl/default.aspx.* Used by permission.)

the information systems that are procured are used inside the organization. Careful planning is required to determine which systems will be developed for an organization. Each system developed is an investment, and if the organization invests in a particular system, it cannot invest in some alternative system or in something else, such as a new store. Investment funds are limited, after all. Maintenance is also peculiar to an organizational context. Once systems go into general use, the organization needs to earn as much of a return on those investments as it can, so it is important that the systems run as long as possible. Companies such as Microsoft, which develop systems for others to use, do not need to worry about internal planning and maintenance in their product life cycles. Instead, as they have limited control over systems once they have been sold, they are concerned about selling a mature and reliable product. Therefore, they have two phases after implementation: verification and release. Verification involves quality assurance for products before they are released. Release involves all of the activities related to making the product available for general use. Next, note the two parts of the SDL that precede and follow the main development phases: training (in blue) and response (in orange). Two things make this particular SDLC specialized to security issues: the two unique phases that begin and end the life cycle (training and response), and the particular security activities associated with each phase in the development life cycle.

Training in Microsoft's SDL refers to the training each member of a development team receives about security basics and trends in security. The idea behind the training—indeed, the idea behind a specialized security development life cycle—is to have security become part of the development process from the beginning and not suddenly appear at the end of the SDLC. By training team members about security and how it can be addressed throughout the life cycle, security measures can be built into the system throughout its development. The response at the end of the SDL refers to a response plan developed during the release phase. If there is a security threat to a particular product, then the previously developed response plan is executed. The security-related activities that take place throughout the development life cycle vary by phase. Listing and explaining each activity is beyond the scope of this chapter, but we can provide some examples. One specialized activity performed during the requirements phase is a separate analysis of requirements related to both security and privacy. During design, developers can model threats to a system and consider how those threats differ with different design options. During implementation, project team members can conduct static analyses of source code, looking for security threats. During verification, they can conduct dynamic analyses. As part of the release phase, team members develop the incident response plan, mentioned previously, and they conduct a final security review. By adhering to a specialized life cycle devoted to security, project team members can ensure not only that security is addressed, but that it is addressed in a planned, systematic manner.

THE HEART OF THE SYSTEMS DEVELOPMENT PROCESS

The SDLC provides a convenient way to think about the processes involved in systems development and the organization of this book. The different phases are clearly defined, their relationships to one another are well specified, and the sequencing of phases from one to the next, from beginning to end, has a compelling logic. In many

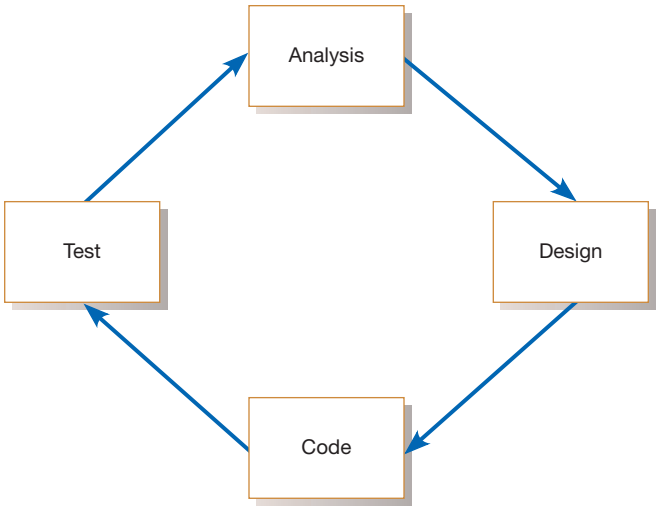


FIGURE 1-8
Analysis–design–code–test loop

ways, though, the SDLC is fiction. Although almost all systems development projects adhere to some type of life cycle, the exact location of activities and the specific sequencing of steps can vary greatly from one project to the next. Current practice combines the activities traditionally thought of as belonging to analysis, design, and implementation into a single process. Instead of systems requirements being produced in analysis, systems specifications being created in design, and coding and testing being done at the beginning of implementation, current practice combines all of these activities into a single analysis–design–code–test process (Figure 1-8). These activities are the heart of systems development, as we suggest in Figure 1-9. This combination of activities is typical of current practices in Agile Methodologies. A well-known instance of one of the Agile Methodologies is eXtreme Programming, although there are other variations. We will introduce you to Agile Methodologies and eXtreme Programming, but first it is important that you learn about the problems with the traditional SDLC. You will read about these problems next. Then you will read about CASE tools, Agile Methodologies, and eXtreme Programming.

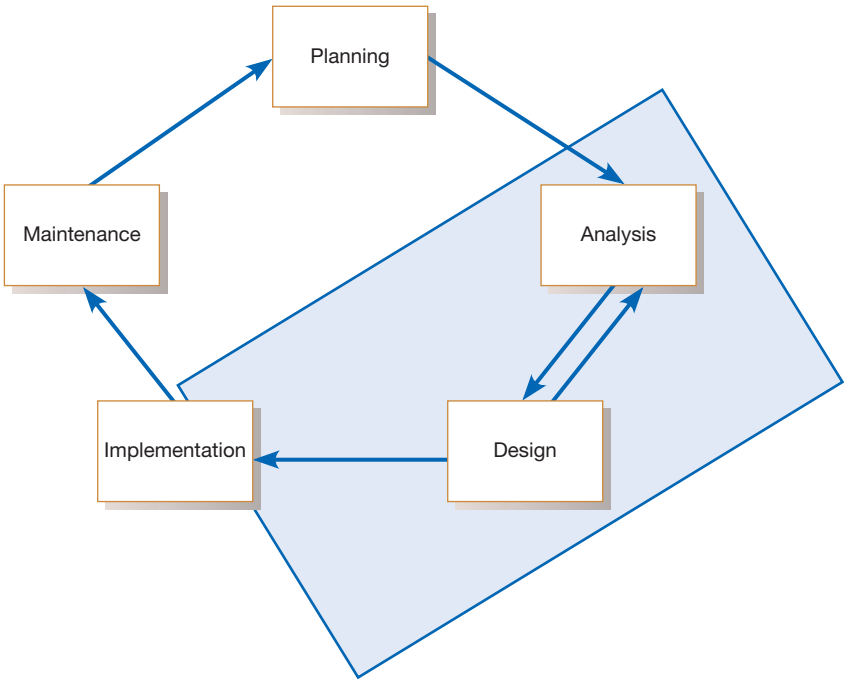


FIGURE 1-9
Heart of systems development

The Traditional Waterfall SDLC

There are several criticisms of the traditional life-cycle approach to systems development; one relates to the way the life cycle is organized. To better understand these criticisms, it is best to see the form in which the life cycle has traditionally been portrayed, the so-called waterfall (Figure 1-10). Note how the flow of the project begins in the planning phase and from there runs "downhill" to each subsequent phase, just like a stream that runs off a cliff. Although the original developer of the waterfall model, W. W. Royce, called for feedback between phases in the waterfall, this feedback came to be ignored in implementation (Martin, 1999). It became too tempting to ignore the need for feedback and to treat each phase as complete unto itself, never to be revisited once finished.

Traditionally, one phase ended and another began once a milestone had been reached. The milestone usually took the form of some deliverable or prespecified output from the phase. For example, the design deliverable is the set of detailed physical design specifications. Once the milestone had been reached and the new phase initiated, it became difficult to go back. Even though business conditions continued to change during the development process and analysts were pressured by users and others to alter the design to match changing conditions, it was necessary for the analysts to freeze the design at a particular point and go forward. The enormous amount of effort and time necessary to implement a specific design meant that it would be very expensive to make changes in a system once it was developed. The traditional waterfall life cycle, then, had the property of locking users into requirements that had been previously determined, even though those requirements might have changed.

Yet another criticism of the traditional waterfall SDLC is that the role of system users or customers was narrowly defined (Kay, 2002). User roles were often relegated to the requirements determination or analysis phases of the project, where it was assumed that all of the requirements could be specified in advance. Such an assumption, coupled with limited user involvement, reinforced the tendency of the waterfall model to lock in requirements too early, even after business conditions had changed.

In addition, under the traditional waterfall approach, nebulous and intangible processes such as analysis and design are given hard-and-fast dates for completion,

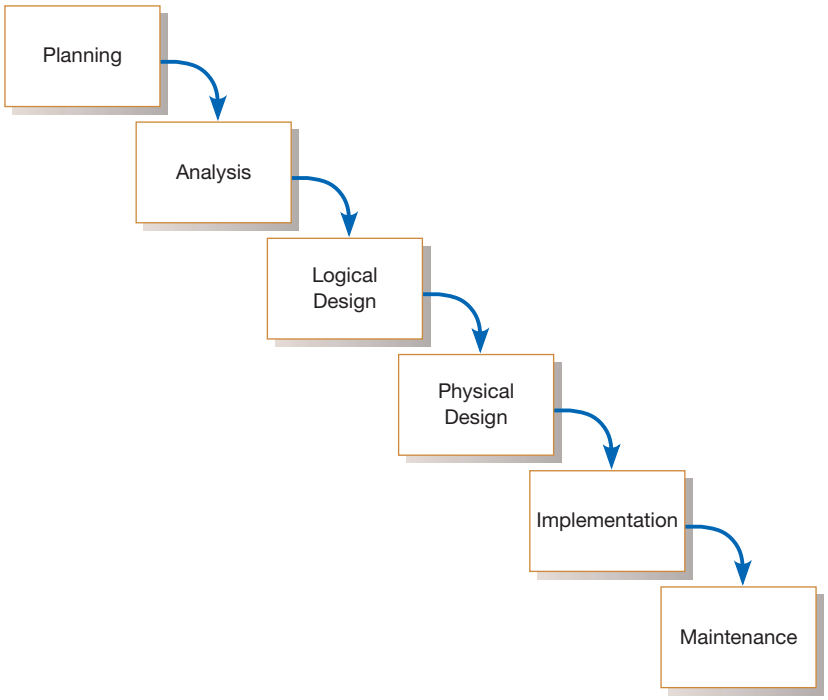


FIGURE 1-10
Traditional waterfall SDLC

and success is overwhelmingly measured by whether those dates are met. The focus on milestone deadlines, instead of on obtaining and interpreting feedback from the development process, leads to too little focus on doing good analysis and design. The focus on deadlines leads to systems that do not match users' needs and that require extensive maintenance, unnecessarily increasing development costs. Finding and fixing a software problem after the delivery of the system is often far more expensive than finding and fixing it during analysis and design (Griss, 2003). The result of focusing on deadlines rather than on good practice is unnecessary rework and maintenance effort, both of which are expensive. According to some estimates, maintenance costs account for 40 to 70 percent of systems development costs (Dorfman and Thayer, 1997). Given these problems, people working in systems development began to look for better ways to conduct systems analysis and design.

DIFFERENT APPROACHES TO IMPROVING DEVELOPMENT

In the continuing effort to improve the systems analysis and design process, several different approaches have been developed. We will describe the more important approaches in more detail in later chapters. Attempts to make systems development less of an art and more of a science are usually referred to as *systems engineering* or *software engineering*. As the names indicate, rigorous engineering techniques have been applied to systems development. One manifestation of the engineering approach is CASE tools, which you will read about next.

CASE Tools

Computer-aided software engineering (CASE) tools
Software tools that provide automated support for some portion of the systems development process.

Other efforts to improve the systems development process have taken advantage of the benefits offered by computing technology itself. The result has been the creation and fairly widespread use of **computer-aided software engineering (CASE) tools**. CASE tools have been developed for internal use and for sale by several leading firms, but the best known is the series of Rational tools made by IBM.

CASE tools are used to support a wide variety of SDLC activities. CASE tools can be used to help in multiple phases of the SDLC: project identification and selection, project initiation and planning, analysis, design, and implementation and maintenance. An integrated and standard database called a *repository* is the common method for providing product and tool integration, and has been a key factor in enabling CASE to more easily manage larger, more complex projects and to seamlessly integrate data across various tools and products. The idea of a central repository of information about a project is not new—the manual form of such a repository is called a *project dictionary* or *workbook*.

The general types of CASE tools are listed below:

- Diagramming tools enable system process, data, and control structures to be represented graphically.
- Computer display and report generators help prototype how systems "look and feel." Display (or form) and report generators make it easier for the systems analyst to identify data requirements and relationships.
- Analysis tools automatically check for incomplete, inconsistent, or incorrect specifications in diagrams, forms, and reports.
- A central repository enables the integrated storage of specifications, diagrams, reports, and project management information.
- Documentation generators produce technical and user documentation in standard formats.
- Code generators enable the automatic generation of program and database definition code directly from the design documents, diagrams, forms, and reports.

TABLE 1-2 Examples of CASE Usage within the SDLC

SDLC Phase	Key Activities	CASE Tool Usage
Project identification and selection	Display and structure high-level organizational information	Diagramming and matrix tools to create and structure information
Project initiation and planning	Develop project scope and feasibility	Repository and documentation generators to develop project plans
Analysis	Determine and structure system requirements	Diagramming to create process, logic, and data models
Logical and physical design	Create new system designs	Form and report generators to prototype designs; analysis and documentation generators to define specifications
Implementation	Translate designs into an information system	Code generators and analysis, form and report generators to develop system; documentation generators to develop system and user documentation
Maintenance	Evolve information system	All tools are used (repeat life cycle)

CASE helps programmers and analysts do their jobs more efficiently and more effectively by automating routine tasks. However, many organizations that use CASE tools do not use them to support all phases of the SDLC. Some organizations may extensively use the diagramming features but not the code generators. Table 1-2 summarizes how CASE is commonly used within each SDLC phase. There are a variety of reasons why organizations choose to adopt CASE partially or not use it at all. These reasons range from a lack of vision for applying CASE to all aspects of the SDLC, to the belief that CASE technology will fail to meet an organization's unique system development needs. In some organizations, CASE has been extremely successful, whereas in others it has not.

AGILE METHODOLOGIES

Many approaches to systems analysis and design have been developed over the years. In February 2001, many of the proponents of these alternative approaches met in Utah and reached a consensus on several of the underlying principles their various approaches contained. This consensus turned into a document they called "The Agile Manifesto" (Table 1-3). According to Fowler (2003), the Agile Methodologies share three key principles: (1) a focus on adaptive rather than predictive methodologies, (2) a focus on people rather than roles, and (3) a focus on self-adaptive processes.

The Agile Methodologies group argues that software development methodologies adapted from engineering generally do not fit with real-world software development (Fowler, 2003). In engineering disciplines, such as civil engineering, requirements tend to be well understood. Once the creative and difficult work of design is completed, construction becomes very predictable. In addition, construction may account for as much as 90 percent of the total project effort. For software, on the other hand, requirements are rarely well understood, and they change continually during the lifetime of the project. Construction may account for as little as 15 percent of the total project effort, with design constituting as much as 50 percent. Applying techniques that work well for predictable, stable projects, such as bridge building, tend not to work well for fluid, design-heavy projects such as writing software, say the Agile Methodology proponents. What is needed are methodologies that embrace change and that are able to deal with a lack of predictability. One mechanism for dealing with a lack of predictability, which all Agile Methodologies share, is iterative development (Martin, 1999). Iterative development focuses on the frequent production of working versions of a system that have a subset of the total number of required features. Iterative development provides feedback to customers and developers alike.

The Agile Methodologies' focus on people is an emphasis on individuals rather than on the roles that people perform (Fowler, 2003). The roles that people fill, of

TABLE 1-3 The Agile Manifesto

The Manifesto for Agile Software Development

Seventeen anarchists agree:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions* over processes and tools.
- *Working software* over comprehensive documentation.
- *Customer collaboration* over contract negotiation.
- *Responding to change* over following a plan.

That is, while we value the items on the right, we value the items on the left more. We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Businesspeople and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

—Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (*www.agileAlliance.org*)

(*Source: http://agilemanifesto.org/* © 2001, the above authors. This declaration may be freely copied in any form, but only in its entirety through this notice.)

systems analyst or tester or manager, are not as important as the individuals who fill those roles. Fowler argues that the application of engineering principles to systems development has resulted in a view of people as interchangeable units instead of a view of people as talented individuals, each bringing something unique to the development team.

The Agile Methodologies promote a self-adaptive software development process. As software is developed, the process used to develop it should be refined and improved. Development teams can do this through a review process, often associated with the completion of iterations. The implication is that, as processes are adapted, one would not expect to find a single monolithic methodology within a given corporation or enterprise. Instead, one would find many variations of the methodology, each of which reflects the particular talents and experience of the team using it.

Agile Methodologies are not for every project. Fowler (2003) recommends an agile or adaptive process if your project involves

- unpredictable or dynamic requirements,
- responsible and motivated developers, and
- customers who understand the process and will get involved.

TABLE 1-4 Five Critical Factors That Distinguish Agile and Traditional Approaches to Systems Development

Factor	Agile Methods	Traditional Methods
Size	Well matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to products that are not critical.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce experts. Risky to use non-agile people.	Needs a critical mass of scarce experts during project definition but can work with fewer later in the project, unless the environment is highly dynamic.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thriving on chaos).	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thriving on order).

(*Source:* Boehm, Barry; Turner, Richard, *Balancing Agility and Discipline: A Guide for the Perplexed,* 1st Ed., © 2004. Reprinted and electronically reproduced by permission of Pearson Education, Inc. New York, NY.)

A more engineering-oriented, predictable process may be called for if the development team exceeds 100 people or if the project is operating under a fixed-price or fixed-scope contract. In fact, whether a systems development project is organized in terms of Agile or more traditional methodologies depends on many different considerations. If a project is considered to be high risk and highly complex, and has a development team made up of hundreds of people, then more traditional methods will apply. Less risky, smaller, and simpler development efforts lend themselves more to Agile methods. Other determining factors include organizational practice and standards, and the extent to which different parts of the system will be contracted out to others for development. Obviously, the larger the proportion of the system that will be outsourced, the more detailed the design specifications will need to be so that subcontractors can understand what is needed. Although not universally agreed upon, the key differences between these development approaches are listed in Table 1-4, which is based on work by Boehm and Turner (2004). These differences can be used to help determine which development approach would work best for a particular project.

Many different individual methodologies come under the umbrella of Agile Methodologies. Fowler (2003) lists the Crystal family of methodologies, Adaptive Software Development, Scrum, Feature Driven Development, and others as Agile Methodologies. Perhaps the best known of these methodologies, however, is eXtreme Programming, discussed next.

eXtreme Programming

eXtreme Programming is an approach to software development put together by Beck and Andres (2004). It is distinguished by its short cycles, incremental planning approach, focus on automated tests written by programmers and customers to monitor the development process, and reliance on an evolutionary approach to development that lasts throughout the lifetime of the system. Key emphases of eXtreme Programming

are its use of two-person programming teams, described later, and having a customer on-site during the development process. The relevant parts of eXtreme Programming that relate to design specifications are (1) how planning, analysis, design, and construction are all fused into a single phase of activity; and (2) its unique way of capturing and presenting system requirements and design specifications. With eXtreme Programming, all phases of the life cycle converge into a series of activities based on the basic processes of coding, testing, listening, and designing.

Under this approach, coding and testing are intimately related parts of the same process. The programmers who write the code also develop the tests. The emphasis is on testing those things that can break or go wrong, not on testing everything. Code is tested very soon after it is written. The overall philosophy behind eXtreme Programming is that the code will be integrated into the system it is being developed for and tested within a few hours after it has been written. If all the tests run successfully, then development proceeds. If not, the code is reworked until the tests are successful.

Another part of eXtreme Programming that makes the code-and-test process work more smoothly is the practice of pair programming. All coding and testing is done by two people working together to write code and develop tests. Beck says that pair programming is not one person typing while the other one watches; rather, the two programmers work together on the problem they are trying to solve, exchanging information and insight and sharing skills. Compared to traditional coding practices, the advantages of pair programming include: (1) more (and better) communication among developers, (2) higher levels of productivity, (3) higher-quality code, and (4) reinforcement of the other practices in eXtreme Programming, such as the code-and-test discipline (Beck & Andres, 2004). Although the eXtreme Programming process has its advantages, just as with any other approach to systems development, it is not for everyone and is not applicable to every project.

OBJECT-ORIENTED ANALYSIS AND DESIGN

Object-oriented analysis and design (OOAD)
Systems development methodologies and techniques based on objects rather than data or processes.

Object
A structure that encapsulates (or packages) attributes and methods that operate on those attributes. An object is an abstraction of a real-world thing in which data and processes are placed together to model the structure and behavior of the real-world object.

Inheritance
The property that occurs when entity types or object classes are arranged in a hierarchy and each entity type or object class assumes the attributes and methods of its ancestors, that is, those higher up in the hierarchy. Inheritance allows new but related classes to be derived from existing classes.

Object class
A logical grouping of objects that have the same (or similar) attributes and behaviors (methods).

There is no question that **object-oriented analysis and design (OOAD)** is becoming more and more popular (we elaborate on this approach later throughout the book). OOAD is often called the third approach to systems development, after the process-oriented and data-oriented approaches. The object-oriented approach combines data and processes (called *methods*) into single entities called **objects**. Objects usually correspond to the real things an information system deals with, such as customers, suppliers, contracts, and rental agreements. Putting data and processes together in one place recognizes the fact that there are a limited number of operations for any given data structure, and the object-oriented approach makes sense even though typical systems development keeps data and processes independent of each other. The goal of OOAD is to make systems elements more reusable, thus improving system quality and the productivity of systems analysis and design.

Another key idea behind object orientation is **inheritance**. Objects are organized into **object classes**, which are groups of objects sharing structural and behavioral characteristics. Inheritance allows the creation of new classes that share some of the characteristics of existing classes. For example, from a class of objects called "person," you can use inheritance to define another class of objects called "customer." Objects of the class "customer" would share certain characteristics with objects of the class "person": They would both have names, addresses, phone numbers, and so on. Because "person" is the more general class and "customer" is more specific, every customer is a person but not every person is a customer.

As you might expect, a computer programming language is required that can create and manipulate objects and classes of objects in order to create object-oriented information systems. Several object-oriented programming languages have been created (e.g., C++, Eiffel, and Java). In fact, object-oriented languages were developed

first, and object-oriented analysis and design techniques followed. Because OOAD is still relatively new, there is little consensus or standardization among the many OOAD techniques available. In general, the primary task of object-oriented analysis is identifying objects and defining their structure and behavior and their relationships. The primary tasks of object-oriented design are modeling the details of the objects' behavior and communication with other objects so that system requirements are met, and reexamining and redefining objects to better take advantage of inheritance and other benefits of object orientation.

The object-oriented approach to systems development shares the iterative development approach of the Agile Methodologies. Some say that the current focus on agility in systems development is nothing more than the mainstream acceptance of object-oriented approaches that have been germinating for years, so this similarity should come as no surprise (Fowler, 2003). One of the most popular realizations of the iterative approach for object-oriented development is the **Rational Unified Process (RUP)**, which is based on an iterative, incremental approach to systems development. RUP has four phases: inception, elaboration, construction, and transition (see Figure 1-11).

Rational Unified Process (RUP)
An object-oriented systems development methodology. RUP establishes four phases of development: inception, elaboration, construction, and transition. Each phase is organized into a number of separate iterations.

In the inception phase, analysts define the scope, determine the feasibility of the project, understand user requirements, and prepare a software development plan. In the elaboration phase, analysts detail user requirements and develop a baseline architecture. Analysis and design activities constitute the bulk of the elaboration phase. In the construction phase, the software is actually coded, tested, and documented. In the transition phase, the system is deployed, and the users are trained and supported. As is evident from Figure 1-11, the construction phase is generally the longest and the most resource intensive. The elaboration phase is also long, but less resource intensive. The transition phase is resource intensive but short. The inception phase is short and the least resource intensive. The areas of the rectangles in Figure 1-11 provide an estimate of the overall resources allocated to each phase.

Each phase can be further divided into iterations. The software is developed incrementally as a series of iterations. The inception phase will generally entail a single iteration. The scope and feasibility of the project is determined at this

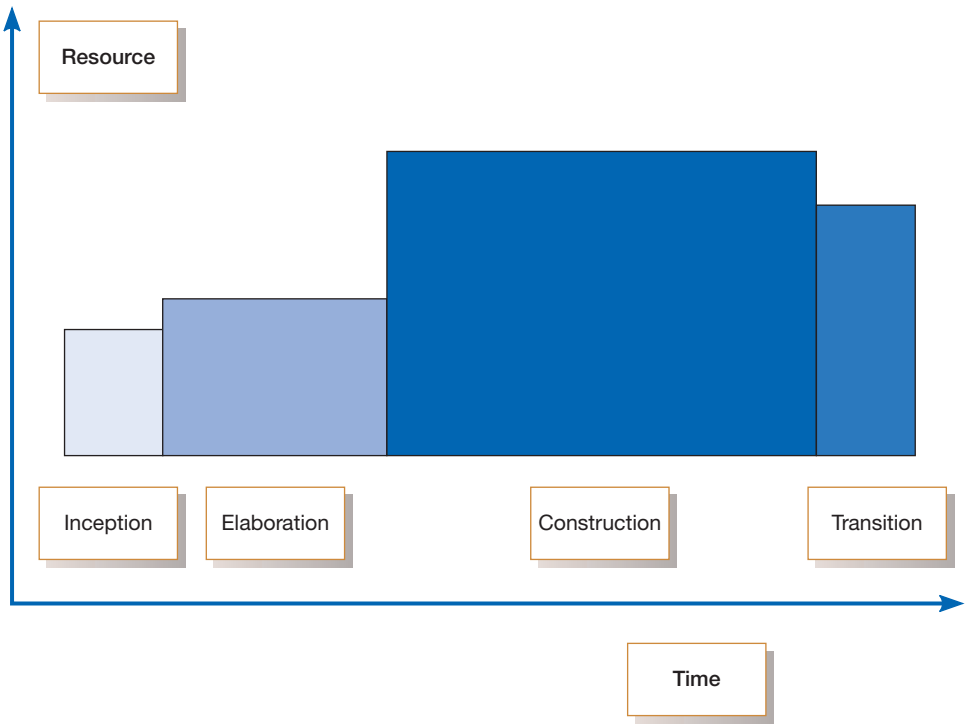


FIGURE 1-11
Phases of OOAD-based development

stage. The elaboration phase may have one or two iterations and is generally considered the most critical of the four phases (Kruchten, 2000). The elaboration phase is mainly about systems analysis and design, although other activities are also involved. At the end of the elaboration phase, the architecture of the project should have been developed. The architecture includes a vision of the product, an executable demonstration of the critical pieces, a detailed glossary and a preliminary user manual, a detailed construction plan, and a revised estimate of planned expenditures.

Although the construction phase mainly involves coding, which is accomplished in several iterations, revised user requirements could require analysis and design. The components are developed or purchased and used in the code. As each executable is completed, it is tested and integrated. At the end of the construction phase, a beta version of the project is released that should have operational capabilities. The transition phase entails correcting problems, beta testing, user training, and conversion of the product. The transition phase is complete when the project objectives meet the acceptance criteria. Once acceptance criteria have been met, the product can then be released for distribution.

OUR APPROACH TO SYSTEMS DEVELOPMENT

Much of the criticism of the SDLC has been based on abuses of the life cycle perspective, both real and imagined. One of the criticisms, one based in reality, is that reliance on the life-cycle approach forced intangible and dynamic processes, such as analysis and design, into timed phases that were doomed to fail (Martin, 1999). Developing software is not like building a bridge, and the same types of engineering processes cannot always be applied (Fowler, 2003), even though viewing software development as a science rather than an art has no doubt resulted in vast improvements in the process and the resulting products. Another criticism with its basis in fact is that life cycle reliance has resulted in massive amounts of process and documentation, much of which seems to exist for its own sake. Too much process and documentation does slow down development, hence the streamlining that underlies the Agile Methodologies and the admonition from Agile developers that source code is enough documentation. A criticism of the SDLC that is based more on fiction is that all versions of the SDLC are waterfall-like, with no feedback between steps. Another false criticism is that a life-cycle approach necessarily limits the involvement of users in the earliest stages of the process. Yet Agile Methodologies, and eXtreme Programming in particular, advocate an analysis–design–code–test sequence that is a cycle (Figure 1-8), and users can be and are involved in every step of this cycle; thus, cycles in and of themselves do not necessarily limit user involvement.

Whether the criticisms have been based on fact or not, however, it is true that the traditional SDLC waterfall approach has problems, and we applaud the changes taking place in the systems development community. These changes are allowing problems with traditional approaches to be fixed, and without a doubt the result is better software produced more expertly and more quickly.

However, despite the criticisms of a life-cycle approach to systems analysis and design, the view of systems analysis and design taking place in a cycle continues to be pervasive, and, we think, true as well. There are many types of cycles, from the waterfall to the analysis–design–code–test cycle, and they all capture the iterative nature of systems development. The waterfall approach may be losing its relevance, but the cycle in Figure 1-8 is gaining in popularity, and the analysis–design–code–test cycle is embedded in a larger organizational cycle. Although we typically use the terms *systems analysis and design* and *systems development* interchangeably, perhaps it is better to think about systems analysis and design as being the cycle in Figure 1-8 and systems development as being the larger cycle in Figure 1-2. The analysis–design–code–test cycle largely ignores the organizational planning that precedes it and the organizational installation and

systems maintenance that follow, yet they are all important aspects of the larger systems development effort. And to us, the best, clearest way to think about both efforts is in terms of cycles.

Therefore, in this book you will see Figure 1-2 at the beginning of almost every chapter. We will use our SDLC as an organizing principle in this book, with activities and processes arranged according to whether they fit under the category of planning, analysis, design, implementation, or maintenance. To some extent, we will artificially separate activities and processes so that each one can be individually studied and understood. Once individual components are clearly understood, it is easier to see how they fit with other components, and eventually it becomes easy to see the whole. Just as we may artificially separate activities and processes, we may also construct artificial boundaries between phases of the SDLC. Our imposition of boundaries should never be interpreted as hard-and-fast divisions. In practice, as we have seen with the Agile Methodologies and in the introduction to OOAD, phases and parts of phases may be combined for speed, understanding, and efficiency. Our intent is to introduce the pieces in a logical manner, so that you can understand all the pieces and how to assemble them in the best way for your systems development purposes. Yet the overall structure of the cycle, of iteration, remains throughout. Think of the cycle as an organizing and guiding principle.

SUMMARY

This chapter introduced you to information systems analysis and design, the complex organizational process whereby computer-based information systems are developed and maintained. You read about how systems analysis and design in organizations has changed over the past several decades. You also learned about the basic framework that guides systems analysis and design—the systems development life cycle (SDLC), with its five major phases: planning, analysis, design, implementation, and maintenance. The SDLC life cycle has had its share of criticism, which you read about, and other frameworks have been developed to address the life cycle's problems. These approaches include computer-aided software engineering (CASE) tools and the Agile Methodologies, the most famous of which is eXtreme Programming. You were also briefly introduced to object-oriented analysis and design, an approach that is becoming more and more popular. All of these approaches share the underlying idea of iteration, as manifested in the systems development life cycle and the analysis–design–code–test cycle of the Agile Methodologies.

KEY TERMS

1.1 Analysis
1.2 Application software
1.3 Computer-aided software engineering (CASE) tools
1.4 Design
1.5 Implementation
1.6 Information systems analysis and design
1.7 Inheritance
1.8 Logical design
1.9 Maintenance
1.10 Object
1.11 Object class
1.12 Object-oriented analysis and design (OOAD)
1.13 Physical design
1.14 Planning
1.15 Rational Unified Process (RUP)
1.16 Systems analyst
1.17 Systems development life cycle (SDLC)
1.18 Systems development methodology

Match each of the key terms above with the definition that best fits it.

____ The complex organizational process whereby computer-based information systems are developed and maintained.

____ Computer software designed to support organizational functions or processes.

____ The organizational role most responsible for the analysis and design of information systems.

____ A standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems.

____ The traditional methodology used to develop, maintain, and replace information systems.

____ The first phase of the SDLC, in which an organization's total information system needs are identified, analyzed, prioritized, and arranged.

____ The second phase of the SDLC, in which system requirements are studied and structured.

____ The third phase of the SDLC, in which the description of the recommended solution is converted into logical and then physical system specifications.

____ The part of the design phase of the SDLC in which all functional features of the system chosen for development are described independently of any computer platform.

____ The part of the design phase of the SDLC in which the logical specifications of the system from logical design are transformed into technology-specific details from which all programming and system construction can be accomplished.

____ The fourth phase of the SDLC, in which the information system is coded, tested, installed, and supported in the organization.

____ The final phase of the SDLC, in which an information system is systematically repaired and improved.

____ Software tools that provide automated support for some portion of the systems development process.

____ Systems development methodologies and techniques based on objects rather than data or processes.

____ A structure that encapsulates (or packages) attributes and the methods that operate on those attributes. It is an abstraction of a real-world thing in which data and processes are placed together to model the structure and behavior of the real-world object.

____ The property that occurs when entity types or object classes are arranged in a hierarchy and each entity type or object class assumes the attributes and methods of its ancestors—that is, those higher up in the hierarchy. The property allows new but related classes to be derived from existing classes.

____ A logical grouping of objects that have the same (or similar) attributes and behaviors (methods).

____ An object-oriented systems development methodology. This methodology establishes four phases of development, each of which is organized into a number of separate iterations: inception, elaboration, construction, and transition.

REVIEW QUESTIONS

1.19 What is information systems analysis and design?

1.20 How has systems analysis and design changed over the past four decades?

1.21 List and explain the different phases in the SDLC.

1.22 List and explain some of the problems with the traditional waterfall SDLC.

1.23 What are CASE tools?

1.24 Describe each major component of a comprehensive CASE system. Is any component more important than any other?

1.25 Describe how CASE is used to support each phase of the SDLC.

1.26 Explain what is meant by Agile Methodologies.

1.27 What is eXtreme Programming?

1.28 When would you use Agile Methodologies versus an engineering-based approach to development?

1.29 What is object-oriented analysis and design?

PROBLEMS AND EXERCISES

1.30 Why is it important to use systems analysis and design methodologies when building a system? Why not just build the system in whatever way appears to be "quick and easy"? What value is provided by using an "engineering" approach?

1.31 Compare Figures 1-2 and 1-3. What similarities and differences do you see?

1.32 Compare Figures 1-2 and 1-4. Can you match steps in Figure 1-4 with phases in Figure 1-2? How might you explain the differences between the two figures?

1.33 Compare Figures 1-2 and 1-10. How does Figure 1-10 illustrate some of the problems of the traditional waterfall approach that are not illustrated in Figure 1-2? How does converting Figure 1-10 into a circle (like Figure 1-2) fix these problems?

1.34 Explain how object-oriented analysis and design differs from the traditional approach. Why isn't RUP (Figure 1-11) represented as a cycle? Is that good or bad? Explain your response.

FIELD EXERCISES

1.35 Choose an organization that you interact with regularly and list as many different "systems" (computer-based or not) as you can that are used to process transactions, provide information to managers and executives, help managers and executives make decisions, aid group decision making, capture knowledge and provide expertise, help design products and/or facilities, and assist people in communicating with one another. Draw a diagram that shows how these systems interact (or should interact) with one another. Are these systems well integrated?

1.36 Imagine an information system built without using a systems analysis and design methodology and without any thinking about the SDLC. Use your imagination and describe any and all problems that might occur, even if they seem a bit extreme or absurd. (The problems you will describe have probably occurred in one setting or another.)

1.37 Choose a relatively small organization that is just beginning to use information systems. What types of systems are being used? For what purposes? To what extent are these systems integrated with one another? With systems outside the organization? How are these systems developed and controlled? Who is involved in systems development, use, and control?

1.38 Interview information systems professionals who use CASE tools and find out how they use the tools throughout the SDLC process. Ask them what advantages and disadvantages they see in using the tools that they do.

1.39 Go to a CASE tool vendor's website and determine the product's price, functionality, and advantages. Try to find information related to any future plans for the product. If changes are planned, what changes and/or enhancements are planned for future versions? Why are these changes being made?

1.40 Use the web to find out more about the Agile Methodologies. Write a report on what the movement toward agility means for the future of systems analysis and design.

1.41 You may want to keep a personal journal of ideas and observations about systems analysis and design while you are studying this book. Use this journal to record comments you hear, summaries of news stories or professional articles you read, original ideas or hypotheses you create, and questions that require further analysis. Keep your eyes and ears open for anything related to systems analysis and design. Your instructor may ask you to turn in a copy of your journal from time to time in order to provide feedback and reactions. The journal is an unstructured set of personal notes that will supplement your class notes and can stimulate you to think beyond the topics covered within the time limitations of most courses.

REFERENCES

Beck, K., and C. Andres. 2004. *eXtreme Programming eXplained.* Upper Saddle River, NJ: Addison-Wesley.

Boehm, B., and R. Turner. 2004. *Balancing Agility and Discipline.* Boston: Addison-Wesley.

Dorfman, M., and R. M. Thayer (eds). 1997. *Software Engineering.* Los Alamitos, CA: IEEE Computer Society Press.

Fowler, M. 2003. "The New Methodologies." December. Available at http://martinfowler.com/articles/newMethodology.html. Accessed February 3, 2009.

Fowler, M., and J. Highsmith. 2001. "The Agile Manifesto." Available at www.ddj.com/architect/184414755. Accessed March 19, 2009.

Griss, M. 2003. "Ranking IT Productivity Improvement Strategies." Available at http://martin.griss.com/pub/WPGRISS01.pdf. Accessed February 3, 2009.

Kay, R. 2002. "QuickStudy: System Development Life Cycle." *Computerworld,* May 14. Available at www.computerworld.com. Accessed February 3, 2009.

Kruchten, P. 2000. "From Waterfall to Iterative Lifecycle—A Tough Transition for Project Managers." Rational Software White Paper: TP-173 5/00. Available at www.ibm.com/developerworks/rational. Accessed February 3, 2009.

Martin, R. C. 1999. "Iterative and Incremental Development I." Available at http://www.objectmentor.com/resources/articles/IIDI.pdf. Accessed October 12, 2012.

McDougall, P. 2012. "Cloud Will Create 14 Million Jobs, Study Says." *InformationWeek,* March 5. Available at http://www.informationweek.com/news/windows/microsoft_news/232601993. Accessed March 13, 2012.

Mearian, L. 2002. "Merrill Lynch Unit Puts Software Development Process to the Test." *Computerworld,* October 14. Available at www.computerworld.com. Accessed February 3, 2009.

Thibodeau, P. 2012. "IT jobs will grow 22% through 2020, says U.S." Computerworld, March 29. Available at http://www.computerworld.com/s/article/print/9225673/IT_jobs_will_grow_22_through_2020_says_U.S.?taxonomyName=Management+and+Careers&taxonomyId=14. Accessed March 3, 2012.