Part One
Foundations for Systems Development

Chapter 1
The Systems Development Environment

Chapter 2
The Origins of Software

Chapter 3
Managing the Information Systems Project
Y
ou 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 cooperate, complement, and support each other within an organizational setting.

You'll also discover as you read this book 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 effectively work in whatever environment you 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 in such a way 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 Broadway Entertainment Company, Inc. (BEC). The BEC case helps demonstrate how what you learn in each chapter might fit into a practical organizational situation. The BEC case begins after Chapter 3; the remaining book chapters through Chapter 16 each have an associated BEC case. The first BEC section introduces the company and its existing information systems. This introduction to BEC provides insights into the company and its systems that will help you understand BEC more completely when we look at the requirements and design for new systems in later case sections.
The Systems Development Environment

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Define information systems analysis and design.
- Describe the different types of information systems.
- Describe the information systems development life cycle (SDLC).
- Explain Rapid Application Development (RAD) and its constituent parts: prototyping, Joint Application Design (JAD), and computer-aided software engineering (CASE) tools.
- Describe the Agile Methodologies and eXtreme Programming.
- 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 uses 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 trend is to incorporate the Internet, especially the World Wide Web, more and more into an organization’s way of doing business. Although you are probably most famil-
Information systems analysis and design: The complex organizational process whereby computer-based information systems are developed and maintained.

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

Career in information technology (IT) present a great opportunity for you to make a significant and visible impact on business. Demand for information systems professionals peaked in 2000, at the height of the Internet investment craze, and then stabilized at the beginning of 2002. Now, however, demand for skilled information technology workers appears to be growing. According to the business magazine Business 2.0 (Kaihla et al., 2006), five of the ten hottest jobs in the United States through 2014 are in the information technology sector. These top jobs, in order from most to least growth potential, are: (1) network systems and data communication analyst (58% growth), (2) computer software engineer for applications (48% growth), (3) computer software engineer for systems software (43%), (4) network and computer systems administrator (43%), and (5) database administrator (38%). These high levels of demand come despite continuing outsourcing of some IT jobs outside the United States. However, according to the Information Technology Association of America (ITAA, 2005), the U.S. economy will continue to generate net gains in the number of IT jobs despite outsourcing. One economic segment that will see a net gain in IT jobs by 2010 is the professional, consulting, and business sector. 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.

An important (but not the only) result of systems analysis and design is application software; that is, 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).

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.

In the rest of this chapter, we will examine the systems approach to analysis and design. You will learn how systems analysis and design have 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. The 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 for data storage as possible.

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 staffs. 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 PeopleSoft Inc. 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, in the first years of the new century, 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, such as ColdFusion, to speed and simplify the development of Web-based systems. Many CASE tools, such as those developed by Oracle, now 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 in the early years of the new century is the move to wireless system components. Wireless devices, such as cell phones and personal digital assistants (PDAs) (e.g., Palm Pilots or Pocket PCs), are increasingly able to 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 not only 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 an application service provider (ASP).

**TYPES OF INFORMATION SYSTEMS AND SYSTEMS DEVELOPMENT**

You might assume that several different types of information systems would be needed to satisfy all of an organization’s information systems needs. Your assumption would be correct.

There are several different types or classes of information systems. In general, these types are distinguished from each other on the basis of what the system does or by the technology used to construct the system. As a systems analyst, part of your job will be to determine which kind of system will best address the organizational problem or opportunity on which you are focusing. In addition, different classes of systems may require different development methodologies, techniques, and tools. From your prior studies and experiences with information systems, you are probably aware of at least three classes of information systems:

1. Transaction processing systems
2. Management information systems
3. Decision support systems (for individuals, groups, and executives)

In addition, many organizations recognize scientific (or technical) computing and personal productivity systems. To preview the diversity of systems development approaches, the following sections briefly highlight how systems analysis and design methods differ across the three major types of systems (Figure 1-2).

**Transaction Processing Systems**

Transaction processing systems (TPSs) automate the handling of data about business activities or transactions, which can be thought of as simple, discrete events in the life of an organization. Data about each transaction are captured, transactions are verified and accepted or rejected, and validated transactions are stored for later aggregation. Reports may be produced immediately to provide standard transaction summaries, and transactions may be moved from process to process in order to handle all aspects of the business activity.

The analysis and design of a TPS means focusing on the firm’s current procedures for processing transactions, whether those procedures are manual or automated. The focus on current procedures implies a careful tracking of data capture, flow, processing, and output. The goal of TPS development is to improve transaction processing by speeding it up, using fewer people, improving efficiency and accuracy,
integrating it with other organizational information systems, or providing information not previously available.

**Management Information Systems**

A management information system (MIS) takes the relatively raw data available through a TPS and converts them into a meaningful aggregated form that managers need to fulfill their responsibilities. Developing an MIS calls for a good understanding of what kind of information managers require and how managers use information in their jobs. Sometimes managers themselves may not know precisely what they need or how they will use information. Thus, the analyst must also develop a good understanding of the business and the TPSs that provide data for an MIS.

Management information systems often require data from several TPSs (e.g., customer order processing, raw material purchasing, and employee timekeeping). Development of an MIS can, therefore, benefit from a data orientation, in which data are considered an organization resource separate from the TPS in which they are captured. Because it is important to be able to draw on data from various subject areas, developing a comprehensive and accurate model of the data is essential in building an MIS.

**Decision Support Systems**

Decision support systems (DSSs) are designed to help organizational decision makers make decisions. Instead of providing summaries of data, as with an MIS, a DSS provides an interactive environment in which decision makers can quickly manipulate data and models of business operations. A DSS is composed of a database (which may be extracted from a TPS or an MIS), mathematical or graphical models of business processes, and a user interface (or dialogue module) that provides a way for the decision maker, usually a nontechnical manager, to communicate with the DSS. A DSS may
use both hard historical data as well as judgments (or “what if” scenarios) about alternative histories or possible futures. In many cases, the historical data come from a firm’s data warehouse. A data warehouse is a collection of integrated, subject-oriented databases designed to support the decision support function, where each unit of data is relevant to some moment in time (Bischoff, 1997). One form of a DSS, an executive information system (EIS), emphasizes the unstructured capability for senior management to explore data starting at a high level of aggregation and selectively drill down into specific areas where more detailed understandings of the business are required. In either case, a DSS is characterized by less structured and less predictable use. It is a software resource intended to support a certain scope of decision-making activities (from problem finding to choosing a course of action).

The systems analysis and design for a DSS often concentrates on the three main DSS components: database, model base, and user dialogue. As with an MIS, a data orientation is most often used for understanding user requirements. In addition, the systems analysis and design project will carefully document the mathematical rules that define interrelationships among different data. These relationships are used to predict future data or to find the best solutions to decision problems. Thus, decision logic must be carefully understood and documented. Also, because a decision maker typically interacts with a DSS, the design of easy-to-use yet thorough user dialogues and screens is important. Because a DSS often deals with situations not encountered every day or situations that can be handled in many different ways, there can be considerable uncertainty as to what a DSS should actually do. Thus, systems developers often use methods that prototype the system and iteratively and rapidly redevelop the system based on trial use. Hence, the development of a DSS often does not follow as formal a project plan as one for a TPS or an MIS, because the software deliverable is more uncertain at the beginning of the project.

**Summary of Information Systems Types**

Many of the information systems that you will build or maintain will contain aspects of each of the three major types of information systems. Thus, as a systems analyst, you will likely employ specific methodologies, techniques, and tools associated with each of the three information system types. Table 1-1 summarizes the general characteristics and development methods for each type.

Thus far, we have concentrated on the context of information systems development, looking at the different organizations where software is developed, the people involved in development efforts, and the different types of information systems that exist in organizations. Now that we have a good idea of context, we can turn to the actual process by which many information systems are developed in organizations—the systems development life cycle.

**DEVELOPING INFORMATION SYSTEMS AND THE SYSTEMS DEVELOPMENT LIFE CYCLE**

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 sys-

**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.
TABLE 1-1  Systems Development for Different IS Types

<table>
<thead>
<tr>
<th>IS Type</th>
<th>IS Characteristics</th>
<th>Systems Development Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction processing system</td>
<td>High-volume, data capture focus; goal is efficiency of data movement and processing and interfacing different TPSs</td>
<td>Process orientation; concern with capturing, validating, and storing data and with moving data between each required step</td>
</tr>
<tr>
<td>Management information system</td>
<td>Draws on diverse yet predictable data resources to aggregate and summarize data; may involve forecasting future data from historical trends and business knowledge</td>
<td>Data orientation; concern with understanding relationships among data so data can be accessed and summarized in a variety of ways; builds a model of data that supports a variety of uses</td>
</tr>
<tr>
<td>Decision support system</td>
<td>Provides guidance in identifying problems, finding and evaluating alternative solutions, and selecting or comparing alternatives; potentially involves groups of decision makers; often involves semi-structured problems and the need to access data at different levels of detail</td>
<td>Data and decision logic orientations; design of user dialogue; group communication may also be key, and access to unpredictable data may be necessary; nature of systems requires iterative development and almost constant updating</td>
</tr>
</tbody>
</table>

A systems development organization uses a slightly different life cycle model, with anywhere from three to almost twenty identifiable phases.

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 1-3). 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 1-4). However conceived, the systems development life cycle used in an organization is an orderly set of activities conducted and planned for each development phase.
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 (see Figure 1-5). Even if a particular methodology does not look like a cycle, and Figure 1-5 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 1-6. Notice that our model is circular. We use this SDLC as one example of a methodology but, more importantly, as a way to arrange the topics of systems analysis and design. Thus, what you learn in this book 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. In the end of the section, we summarize this discussion in a table that lists the main deliverables or outputs from each SDLC phase.

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 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 IS department, including a schedule for developing new major systems. In smaller organizations (as well as in large ones), determination of which systems to

**Planning:** The first phase of the SDLC in which an organization’s total information system needs are identified, analyzed, prioritized, and arranged.
Figure 1-5
Merrill Lynch’s development methodology (from Mearian, 2002)

- Gather requirements from end users.
- Start with high-level design work.
- Create a plan for testing the software.
- Build and test a prototype application.
- Begin major development work.

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 IS 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.

Figure 1-6
SDLC-based guide to this book

Chapters 4–5
Planning

Chapter 16
Maintenance

Analysis
Chapters 6–9

Implementation

Chapter 15

Chapters 10–14
Design
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.

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.

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 1-7 illustrates the differences between logical and physical design.

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 a 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.

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

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

**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.

**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.

**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, 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 changing business conditions. These changes are necessary to keep the system running and useful. In a sense, maintenance is not a separate phase

**Maintenance:** The final phase of the SDLC in which an information system is systematically repaired and improved.
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 mainte-
nance 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 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 resem-
bles the life cycle itself.

The SDLC is a highly linked set of phases whose products feed the activities in
subsequent phases. Table 1-2 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 is, the greater the need is for
project management. Several project management techniques have been developed
over the past 50 years, 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.

### 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 each other are well specified, and the sequenc-
ing of phases from one to the next, from beginning to end, has a compelling logic. In many 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 started with Rapid Application Development (RAD) and is seen in such current practices as the 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 RAD, 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 prototyping, Joint Application Design, and CASE tools, which make RAD, Agile Methodologies, and eXtreme Programming possible.

### 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 phys-
ich 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 delegated 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, 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 results in 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 100 times 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. Although the application of some engineering processes to software development, such as the strict waterfall SDLC approach, have been criticized (Martin, 1999; Fowler, 2003), one very influential practice successfully borrowed from engineering is called prototyping. We will discuss prototyping next, followed by an introduction to Joint Application Design (JAD) and CASE tools. Both prototyping and JAD incorporate standard parts of the typical systems analysis and design process, and CASE and related software tools to support the development process are also widely available. Prototyping, JAD, and CASE tools are all necessary components of RAD, which you will read about after you are introduced to these three topics.

Prototyping: An iterative process of systems development in which requirements are converted to a working system that is continually revised through close collaboration between an analyst and users.

Prototyping Designing and building a scaled-down but functional version of a desired system is known as prototyping. A prototype can be built with any computer language or development tool, but special prototyping tools have been developed to simplify the process. A prototype can be developed with visual development tools; with the query, screen, and report design tools of a database management system; and with CASE tools.

Using prototyping as a development technique (see Figure 1-11), the analyst works with users to determine the initial or basic requirements for the system. The analyst then quickly builds a prototype. When the prototype is completed, the users work with it and tell the analyst what they like and do not like about it. The analyst uses this feedback to improve the prototype and takes the new version back to the users. This iterative process continues until the users are relatively satisfied with what they have seen. Two key advantages of the prototyping technique are the large extent
Computer-aided software engineering (CASE) tools: Software tools that provide automated support for some portion of the systems development process.

to which prototyping involves the user in analysis and design and its ability to capture requirements in concrete, rather than verbal or abstract, form. In addition to being used as a stand-alone process, prototyping may also be used to augment the SDLC. For example, a prototype of the final system may be developed early in analysis to help the analysts identify what users want. Then the final system is developed based on the specifications of the prototype. We discuss prototyping in greater detail in Chapter 6 and use various prototyping tools in Chapters 11 and 12 to illustrate the design of the user interface and system outputs.

**CASE Tools** 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, or CASE, tools. CASE tools have been developed for internal use and for sale by several leading firms, including Oracle (Designer), Computer Associates (Advantage Gen), and IBM (Rational Rose). Figure 1-12 shows a class diagram rendered with Rational Rose.

CASE tools are used to support a wide variety of SDLC activities. CASE tools can be used to help in the project identification and selection, project initiation and planning, analysis, and design phases and/or in the implementation and maintenance phases of the SDLC. 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 difference is that CASE tools automate the repository for easier updating and consistency.

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.
Figure 1-12
A class diagram from IBM’s Rational Rose
(Source: IBM)

- 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.

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 1-3 summarizes the usage of CASE tools within the SDLC.

### Table 1-3: Examples of CASE Usage within the SDLC

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

marizes 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.

**Joint Application Design** In the late 1970s, systems development personnel at IBM developed a new process for collecting information system requirements and reviewing system designs. This process is called **Joint Application Design (JAD)**. The basic idea behind JAD is to bring structure to the requirements-determination phase of analysis and to the reviews that occur as part of design. Users, managers, and systems developers are brought together for a series of intensive, structured meetings run by a JAD session leader, who maintains the structure and adheres to the agenda. By gathering all the people directly affected by an information system into one room at the same time to work together to agree on system requirements and design details, time and organizational resources are better managed. As an added plus, group members are more likely to develop a shared understanding of what the information system is supposed to do. We will discuss JAD in more detail in Chapter 6.

**Rapid Application Development**

**Rapid Application Development (RAD)** is an approach to developing information systems that promises better and cheaper systems and more rapid deployment by having systems developers and end users work together jointly in real time to develop systems. RAD grew out of the convergence of two trends: (1) the increased speed and turbulence of doing business in the late 1980s and early 1990s and (2) the ready availability of high-powered, computer-based tools to support systems development and easy maintenance. As the conditions of doing business in a changing, competitive global environment became more turbulent, management in many organizations began to question if it made sense to wait two to three years to develop systems (in a methodical, controls-rich process) that would be obsolete upon completion.

The ready availability of increasingly powerful software tools created to support RAD also increased interest in this approach. RAD is becoming more and more a legitimate way to develop information systems. Today, the focus is increasingly on the rapid development of Web-based systems. RAD tools, and software created to support rapid development, almost all provide for the speedy creation of Web-based applications. For example, IBM has developed a suite of tools that enable the fast, easy development of e-business applications. These tools include VisualAge Generator, VisualAge for Java, WebSphere Studio, and WebSphere Application Server.

As Figure 1-13 shows, the same phases that are followed in the traditional SDLC are also followed in RAD, but the phases are shortened and combined with each other to produce a more streamlined development technique. Planning and design phases in RAD are shortened by focusing work on system function and user interface requirements at the expense of detailed business analysis and concern for system performance issues. Also, RAD usually looks at the system being developed in isolation from other systems, thus eliminating the time-consuming activities of coordinating with existing standards and systems during design and development. The emphasis in RAD is generally less on the sequence and structure of processes in the life cycle and more on doing different tasks in parallel with each other and on using prototyping extensively. Notice also that the iteration in the RAD life cycle is limited to the design and development phases. This is where the bulk of the work in a RAD approach takes place.
To succeed, RAD relies on bringing together several systems development components. RAD depends on extensive user involvement. End users are involved from the beginning of the development process, when they participate in application planning; through requirements determination, when they work with analysts in system prototyping; and then into design and implementation, when they work with system developers to validate final elements of the system’s design. Much end-user involvement takes place in the prototyping process, when users and analysts work together to design interfaces and reports for new systems. The prototyping is conducted in sessions that resemble traditional JAD sessions. The primary difference is that in RAD the prototype becomes the basis for the new system—the screens designed during prototyping become screens in the production system. This is accomplished through reliance on integrated CASE tools, which include code generators for creating a code from the designs end users and analysts create during prototyping. The code includes the interfaces as well as the application programs that use them. Alternatively, RAD may employ visual development environments instead of CASE tools with code generators, but the benefits from rapid prototyping are the same. In many cases, the basis for the production system is being built even as users are talking about the system during development workshops. In many cases, end users can get hands-on experience with the developing system before the design workshops are over. To further help speed the process, the reuse of templates, components, or previous systems described in the CASE tool repository is strongly encouraged.

**Agile Methodologies**

RAD is just one reaction to the problems with the traditional waterfall methodology for systems development. As you might imagine, many other 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 1-4). 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 the 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 devel-

**TABLE 1-4 The Agile Manifesto**

<table>
<thead>
<tr>
<th>The Manifesto for Agile Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seventeen anarchists agree:</td>
</tr>
<tr>
<td>We are uncovering better ways of developing software by doing it and helping others do it.</td>
</tr>
<tr>
<td>Through this work we have come to value:</td>
</tr>
<tr>
<td>- Individuals and interactions over processes and tools.</td>
</tr>
<tr>
<td>- Working software over comprehensive documentation.</td>
</tr>
<tr>
<td>- Customer collaboration over contract negotiation.</td>
</tr>
<tr>
<td>- Responding to change over following a plan.</td>
</tr>
<tr>
<td>That is, while we value the items on the right, we value the items on the left more.</td>
</tr>
<tr>
<td>We follow the following principles:</td>
</tr>
<tr>
<td>- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.</td>
</tr>
<tr>
<td>- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.</td>
</tr>
<tr>
<td>- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.</td>
</tr>
<tr>
<td>- Business people and developers work together daily throughout the project.</td>
</tr>
<tr>
<td>- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.</td>
</tr>
<tr>
<td>- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.</td>
</tr>
<tr>
<td>- Working software is the primary measure of progress.</td>
</tr>
<tr>
<td>- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.</td>
</tr>
<tr>
<td>- Continuous attention to technical excellence and good design enhances agility.</td>
</tr>
<tr>
<td>- Simplicity—the art of maximizing the amount of work not done—is essential.</td>
</tr>
<tr>
<td>- The best architectures, requirements, and designs emerge from self-organizing teams.</td>
</tr>
<tr>
<td>- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.</td>
</tr>
</tbody>
</table>

—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: From Fowler and Highsmith, 2001. Used by permission.)
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 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
- Customers who understand the process and will get involved

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. Thus, organizations need various approaches for developing information systems, depending on the characteristics of the system and the development team.

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 (2000). 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 a 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 who 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, 2000). 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**

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 it 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 that of 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...
**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.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Time</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elaboration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Construction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transition</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1-14
Phases of OOSAD-based development

### Rational Unified Process (RUP)

The phases of RUP are:

- **Inception:** Define the scope, determine the feasibility of the project, understand user requirements, and prepare a software development plan.
- **Elaboration:** Detail user requirements and develop a baseline architecture. Analysis and design activities constitute the bulk of the elaboration phase. 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.
- **Construction:** The software is actually coded, tested, and documented.
- **Transition:** The system is deployed and the users are trained and supported.

The construction phase is generally the longest and the most resource intensive, followed by the elaboration phase, which 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 1-14 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 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 RAD 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 to the earliest stages of the process. Yet Agile Methodologies, and eXtreme Programming in particular, advocate an analysis–design–code–test sequence that is itself 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 1-3. 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 1-3 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 50 years, and you also learned that there are many different kinds of information systems used in organizations, from transaction processing systems to management information systems and decision support systems. Development techniques vary with system type. 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. One of these frameworks is Rapid Application Design (RAD), which depends on prototyping; Joint Application Design (JAD); and computer-aided software engineering (CASE) tools. Another framework is 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. Throughout all of these approaches is 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. Analysis
2. Application software
3. Computer-aided software engineering (CASE) tools
4. Design
5. Implementation
6. Information systems analysis and design
7. Inheritance
8. Joint Application Design (JAD)
9. Logical design
10. Maintenance
11. Object
12. Object class
13. Object-oriented analysis and design (OOAD)
14. Physical design
15. Planning
16. Prototyping
17. Rapid Application Development (RAD)
18. Rational Unified Process (RUP)
19. Systems analyst
20. Systems development life cycle (SDLC)
21. 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.

An iterative process of systems development in which requirements are converted to a working system that is continually revised through close work between an analyst and users.

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

A structured process in which users, managers, and analysts work together for several days in a series of intensive meetings to specify or review system requirements.

Systems development methodology created to radically decrease the time needed to design and implement information systems. This methodology relies on extensive user involvement, Joint Application Design sessions, prototyping, integrated CASE tools, and code generators.

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. What is information systems analysis and design?
2. How has systems analysis and design changed over the past four decades?
3. List the different classes of information systems described in this chapter. How do they differ from each other?
4. List and explain the different phases in the SDLC.
5. List and explain some of the problems with the traditional waterfall SDLC.
6. What is prototyping?
7. What is JAD?
8. What are CASE tools?
9. Describe each major component of a comprehensive CASE system. Is any component more important than any other?
10. Describe how CASE is used to support each phase of the SDLC.
11. What is RAD?
12. Explain what is meant by Agile Methodologies.
13. What is eXtreme Programming?
14. What is object-oriented analysis and design?

Problems and Exercises

1. 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?
2. How might prototyping be used as part of the SDLC?
3. Compare Figures 1-3 and 1-4. What similarities and differences do you see?
4. Compare Figures 1-3 and 1-5. Can you match steps in Figure 1-5 with phases in Figure 1-3? How might you explain the differences between the two figures?
5. Compare Figures 1-3 and 1-13. How do they differ? How are they similar? Explain how Figure 1-13 conveys the idea of speed in development.
6. Compare Figures 1-3 and 1-10. How does Figure 1-10 illustrate some of the problems of the traditional waterfall approach that are not illustrated in Figure 1-3? How does converting Figure 1-10 into a circle (like Figure 1-3) fix these problems?
7. Explain how object-oriented analysis and design differs from the traditional approach. Why isn’t RUP (Figure 1-14) represented as a cycle? Is that good or bad? Explain your response.

**Field Exercises**

1. 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 each other. Draw a diagram that shows how these systems interact (or should interact) with each other. Are these systems well integrated?

2. 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. Surprisingly, the problems you will describe have probably occurred in one setting or another.

3. 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 developed and controlled? Who is involved in systems development, use, and control?

4. 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.

5. Go to a CASE tool vendor’s Web site 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?

6. 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.

7. 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**


