CHAPTER 1
INTRODUCTION TO SYSTEMS ANALYSIS AND DESIGN

This chapter introduces the systems development life cycle, the fundamental four-phase model (planning, analysis, design, and implementation) that is common to all information system development projects. It then examines several commonly used system development methodologies that differ in their focus and approach to each of these phases. The chapter closes with a discussion of the skills and roles needed within the project team.

OBJECTIVES

- Understand the fundamental systems development life cycle and its four phases.
- Understand several different categories of system development methodologies and how to choose among them.
- Be familiar with the different skills and roles required on the project team.

CHAPTER OUTLINE

001-027_dennis3e_01.qxd 10/7/05 10:20 AM Page 2

INTRODUCTION

The *systems development life cycle* (SDLC) is the process of understanding how an information system (IS) can support business needs, designing the system, building it, and delivering it to users. If you have taken a programming class or have programmed on your own, this probably sounds pretty simple. Unfortunately, it is not. A 2004 survey by the Standish Group found that just 28% of IT projects succeed these days. Outright failures—IT projects cancelled before completion—occur in 18% of all IT projects. Unfortunately, many of the systems that aren't abandoned are delivered to the users significantly late, cost far more than planned, and have fewer features than originally planned.

Most of us would like to think that these problems only occur to "other" people or "other" organizations, but they happen in most companies. See Figure 1-1 for a sampling of significant IT project failures. Even Microsoft has a history of failures and overdue projects (e.g., Windows 1.0, Windows 95).[1]

Although we would like to promote this book as a "silver bullet" that will keep you from experiencing failed IS projects, we must admit that a silver bullet guaranteeing IS development success does not exist.[2] Instead, this book will provide you with several fundamental concepts and many practical techniques that you can use to improve the probability of success.

The key person in the SDLC is the systems analyst who analyzes the business situation, identifies opportunities for improvements, and designs an information system to implement them. Being a systems analyst is one of the most interesting, exciting, and challenging jobs around. As a systems analyst, you will work with a variety of people and learn how they conduct business. Specifically, you will work with a team of systems analysts, programmers, and others on a common mission. You will feel the satisfaction of seeing systems that you designed and developed make a significant business impact, while knowing that your unique skills helped make that happen.

It is important to remember that the primary objective of the systems analyst is not to create a wonderful system. The primary goal is to create value for the organization, which for most companies means increasing profits (government agencies and not-for-profit organizations measure value differently). Many failed systems were abandoned because the analysts tried to build a wonderful system without clearly understanding how the system would support the organization's goals, current business processes, and other information systems to provide value. An investment in an information system is like any other investment, such as a new machine tool. The goal is not to acquire the tool, because the tool is simply a means to an end; the goal is to enable the organization to perform work better so it can earn greater profits or serve its constituents more effectively.

This book will introduce you to the fundamental skills you need to be a systems analyst. This is a pragmatic book that discusses best practices in systems

[1] For more information on the problem, see Capers Jones, *Patterns of Software System Failure and Success,* London: International Thompson Computer Press, 1996; Capers Jones, *Assessment and Control of Software Risks,* Englewood Cliffs, NJ: Yourdon Press, 1994; Julia King, "IS Reins in Runaway Projects," *Computerworld.* February 24, 1997.

[2] The idea of using the silver bullet metaphor was first described in a paper by Frederick Brooks. See Frederick P. Brooks, Jr., "No Silver Bullet—Essence and Accident in Software Engineering," *Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference,* edited by H.-J. Kugler (1986): 1069–76.

Company	Project	Outcome
Snap-On Inc.	Conversion to new order-entry system from The Baan Co.	Orders delayed, inventory miscounted after system installation in Dec. 1997. $50 million in lost sales, operating costs soar 40% during first half of 1998. Profits decline 22% compared to same period previous year.
FoxMeyer Corp.	SAP ERP System	Bungled ERP installation in 1996. FoxMeyer driven into bankruptcy. Numerous lawsuits resulted with FoxMeyer seeking over $1 billion in damages.
Greyhound Lines Inc.	"Trips" reservation and bus dispatch system	Over $6 million spent on building Trips in early 1990s. System crashed after installation in 1993 when sale prices offered on bus fares. Agents resorted to writing tickets by hand. Ridership dropped 12% in one month. $64 million loss for first half of 1994. CEO and CFO resign.
Hershey Foods Corp.	IBM-led installation and integration of SAP, Manuistics Group Inc. and Siebel Systems software	Compressed rollout of new $112 million ERP system resulted in inaccurate inventory data, shipment delays, and incomplete orders. Sales fell 12% in the quarter following implementation; $150.5 million less than prior year.
Norfolk Southern Corp.	Systems integration with merger target Consolidated Rail Corp.	Over $113 million in lost business during 1998-99 merger. More than a year of train backups, untrackable freight and crew-scheduling problems. More than $80 million extra spent to resolve problems.

Source: Top 10 Corporate Information Technology Failures, *Computerworld,* September 30, 2002.

FIGURE 1-1
Significant IT Failures in the 1990s

development; it does not present a general survey of systems development that exposes you to everything about the topic. By definition, systems analysts *do things* and challenge the current way that organizations work. To get the most out of this book, you will need to actively apply the ideas and concepts in the examples and in the "Your Turn" exercises that are presented throughout to your own systems development project. This book will guide you through all the steps for delivering a

CONCEPTS IN ACTION 1-A An Expensive False Start

A real-estate group in the federal government cosponsored a data warehouse with the IT department. A formal proposal was written by IT in which costs were estimated at $800,000, the project duration was estimated to be eight months, and the responsibility for funding was defined as the business unit's. The IT department proceeded with the project before hearing whether the proposal was ever accepted.

The project actually lasted two years because requirements gathering took nine months instead of one and a half, the planned user base grew from 200 to 2,500, and the approval process to buy technology for the project took a year. Three weeks prior to technical delivery, the IT Director canceled the project. This failed endeavor cost the organization $2.5 million.

Source: "Data Warehousing Failure: Case Studies and Findings" *The Journal of Data Warehousing,* by Hugh J. Watson et al, 4 (1), 1999, pp. 44–54.

QUESTION:
Why did this system fail? Why would a company spend money and time on a project and then cancel it? What could have been done to prevent this?

successful information systems. Also, we will illustrate how one organization (which we call CD Selections) applies the steps in one project (developing a Web-based CD sales system). By the time you finish the book, you won't be an expert analyst, but you will be ready to start building systems for real.

In this chapter, we first introduce the basic SDLC that IS projects follow. This life cycle is common to all projects, although the focus and approach to each phase of the life cycle may differ. In the next section, we discuss three fundamentally different types of methodologies (structured design, rapid application development, and agile development). Finally, we discuss one of the most challenging aspects of systems development—the depth and breadth of skills systems analysts must possess. Today, most organizations use project teams whose members bring unique but complementary skills. This chapter closes with a discussion of the key roles played by members of the systems development team.

THE SYSTEMS DEVELOPMENT LIFE CYCLE

In many ways, building an information system is similar to building a house. First, the house (or the information system) starts with a basic idea. Second, this idea is transformed into a simple drawing that is shown to the customer and refined (often through several drawings, each improving on the other) until the customer agrees that the picture depicts what he or she wants. Third, a set of blueprints is designed that presents much more detailed information about the house (e.g., the type of water faucets, where the telephone jacks will be placed). Finally, the house is built following the blueprints—and often with some changes and decisions made by the customer as the house is erected.

The SDLC has a similar set of four fundamental *phases:* planning, analysis, design, and implementation (Figure 1-2). Different projects may emphasize different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases. Each phase is itself composed of a series of *steps,* which rely on *techniques* that produce *deliverables* (specific documents and files that provide understanding about the project).

For example, when you apply for admission to a university, there are several phases that all students go through: information gathering, applying, and accepting. Each of these phases has steps: information gathering includes steps like searching for schools, requesting information, and reading brochures. Students then use techniques (e.g., Internet searching) that can be applied to steps (e.g., requesting information) to create deliverables (e.g., evaluations of different aspects of universities).

Figure 1-2 suggests that the SDLC phases and steps proceed in a logical path from start to finish. In some projects, this is true, but in many projects, the project teams move through the steps consecutively, incrementally, iteratively, or in other patterns. In this section, we describe at a very high level the phases, steps, and some of the techniques that are used to accomplish the steps. We should emphasize that, in practice, an organization may follow one of many variations on the overall SDLC.

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps that IS projects move through and some of the techniques that produce certain deliverables. Second, it is important to understand that the SDLC is a process of *gradual refinement.* The deliverables produced in the analysis phase provide a general idea of the shape of the new system. These deliverables are used as input to the design phase, which then refines

Phase	Chapter	Step	Technique	Deliverable
Planning Focus: Why build this system? How to structure the project? Primary Outputs: — System Request with Feasibility Study — Project Plan	2	Identify Opportunity	Project Identification	System Request
	2	Analyze Feasibility	Technical Feasibility Economic Feasibility Organizational Feasibility	Feasibility Study
	3	Develop Workplan	Time Estimation Timeboxing Task Identification Work Breakdown Structure PERT Chart Gantt Chart Scope Management	Project Plan — Workplan
	3	Staff Project	Project Staffing Project Charter	— Staffing Plan
	3	Control and Direct Project	CASE Repository Standards Documentation Risk Management	— Standards List — Risk Assessment
Analysis Focus: Who, what, where and when for this system? Primary Output — System Proposal	4	Develop Analysis Strategy	Business Process Automation Business Process Improvement Business Process Reengineering	System Proposal
	4	Determine Business Requirements	Interview JAD session Questionnaire Document Analysis Observation	— Requirements Definition
	5	Create Use Cases	Use Case Analysis	— Use Cases
	6	Model Processes	Data Flow Diagramming	— Process Models
	7	Model Data	Entity Relationship Modeling Normalization	— Data Model
Design Focus: How will this system work? Primary Output: — System Specification	8	Design Physical System	Design Strategy	Alternative Matrix System Specification
	9	Design Architecture	Architecture Design Hardware & Software Selection	— Architecture Report — Hardware & Software Specification
	10	Design Interface	Use Scenario Interface Structure Interface Standards Interface Prototype Interface Evaluation	— Interface Design
	11	Design Programs	Data Flow Diagramming Program Structure Chart Program Specification	— Physical Process Model — Program Design
	12	Design Databases and Files	Data Format Selection Entity Relationship Modeling Denormalization Performance Tuning Size Estimation	— Database & File Specification — Physical Data Model
Implementation Focus: Delivery and support of completed system. Primary Output: — Installed System	13	Construct System	Programming Software Testing Performance Testing	Test Plan Programs Documentation Migration Plan
	14	Install System	Conversion Strategy Selection Training	— Conversion Plan — Business Contingency Plan — Training Plan
	14	Maintain System	Support Selection System Maintenance Project Assessment	Support Plan Problem Report Change Request
	14	Post-implementation	Post-implementation Audit	Post-implementation Audit Report

FIGURE 1-2
Systems Development Life Cycle Phases

them to produce a set of deliverables that describes in much more detailed terms exactly how the system will be built. These deliverables in turn are used in the implementation phase to produce the actual system. Each phase refines and elaborates on the work done previously.

Planning

The *planning phase* is the fundamental process of understanding *why* an information system should be built and determining how the project team will go about building it. It has two steps:

1. During *project initiation*, the system's business value to the organization is identified—how will it lower costs or increase revenues? Most ideas for new systems come from outside the IS area (from the marketing department, accounting department, etc.) in the form of a system request. A *system request* presents a brief summary of a business need, and it explains how a system that supports the need will create business value. The IS department works together with the person or department that generated the request (called the *project sponsor*) to conduct a feasibility analysis. The *feasibility analysis* examines key aspects of the proposed project:
   - The technical feasibility (Can we build it?)
   - The economic feasibility (Will it provide business value?)
   - The organizational feasibility (If we build it, will it be used?)

   The system request and feasibility analysis are presented to an information systems *approval committee* (sometimes called a *steering committee*), which decides whether the project should be undertaken.
2. Once the project is approved, it enters *project management*. During project management, the *project manager* creates a *workplan*, staffs the projects, and puts techniques in place to help the project team control and direct the project through the entire SDLC. The deliverable for project management is a *project plan* that describes how the project team will go about developing the system.

Analysis

The *analysis phase* answers the questions of *who* will use the system, *what* the system will do, and *where* and *when* it will be used. See Figure 1-2. During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system. This phase has three steps:

1. An *analysis strategy* is developed to guide the project team's efforts. Such a strategy usually includes an analysis of the current system (called the *as-is system*) and its problems, and then ways to design a new system (called the *to-be system*).
2. The next step is *requirements gathering* (e.g., through interviews or questionnaires). The analysis of this information—in conjunction with input from the project sponsor and many other people—leads to the development of a concept for a new system. The system concept is then used as a basis to develop a set of business *analysis models* that describes how the business will operate if the new system were developed. The set of models typically includes models that represent the data and processes necessary to support the underlying business process.

3. The analyses, system concept, and models are combined into a document called the *system proposal,* which is presented to the project sponsor and other key decision makers (e.g., members of the approval committee) that decide whether the project should continue to move forward.

The system proposal is the initial deliverable that describes what business requirements the new system should meet. Because it is really the first step in the design of the new system, some experts argue that it is inappropriate to use the term *analysis* as the name for this phase; some argue a better name would be *analysis and initial design.* Because most organizations continue to use the name *analysis* for this phase, we will use it in this book as well. It is important to remember, however, that the deliverable from the analysis phase is both an analysis and a high-level initial design for the new system.

Design

The *design phase* decides *how* the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms, and reports that will be used; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:

1. The *design strategy* must be developed. This clarifies whether the system will be developed by the company's own programmers, whether it will be outsourced to another firm (usually a consulting firm), or whether the company will buy an existing software package.
2. This leads to the development of the basic *architecture design* for the system that describes the hardware, software, and network infrastructure that will be used. In most cases, the system will add or change the infrastructure that already exists in the organization. The *interface design* specifies how the users will move through the system (e.g., navigation methods such as menus and on-screen buttons) and the forms and reports that the system will use.
3. The *database and file specifications* are developed. These define exactly what data will be stored and where they will be stored.
4. The analyst team develops the *program design,* which defines the programs that need to be written and exactly what each program will do.

This collection of deliverables (architecture design, interface design, database and file specifications, and program design) is the *system specification* that is handed to the programming team for implementation. At the end of the design phase, the feasibility analysis and project plan are reexamined and revised, and another decision is made by the project sponsor and approval committee about whether to terminate the project or continue. See Figure 1-2.

Implementation

The final phase in the SDLC is the *implementation phase,* during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is the longest and most expensive single part of the development process. This phase has three steps:

1. System *construction* is the first step. The system is built and tested to ensure it performs as designed. Since the cost of bugs can be immense, testing is one of the most critical steps in implementation. Most organizations spend more time and attention on testing than on writing the programs in the first place.
2. The system is installed. *Installation* is the process by which the old system is turned off and the new one is turned on. It may include a direct cutover approach (in which the new system immediately replaces the old system), a parallel conversion approach (in which both the old and new systems are operated for a month or two until it is clear that there are no bugs in the new system), or a phased conversion strategy (in which the new system is installed in one part of the organization as an initial trial and then gradually installed in others). One of the most important aspects of conversion is the development of a *training plan* to teach users how to use the new system and help manage the changes caused by the new system.
3. The analyst team establishes a *support plan* for the system. This plan usually includes a formal or informal post-implementation review, as well as a systematic way for identifying major and minor changes needed for the system.

SYSTEMS DEVELOPMENT METHODOLOGIES

A *methodology* is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies and each one is unique because of its emphasis on processes versus data and the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, while others have been developed by consulting firms to sell to clients. Many organizations have their own internal methodologies that have been refined over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

All system development methodologies lead to a representation of the system concept in terms of processes and data; however, they vary in terms of whether the methodology places primary emphasis on business processes or on the data that supports the business. As an illustration, refer to the diagram shown in Figure 1-3, depicting the activities and information used in producing the payroll for an organization. The open-ended rectangles in the diagram represent data storage containers; the rounded rectangles represent activities performed; and the arrows represent activity inputs and outputs.

Process-centered methodologies focus first on defining the activities associated with the system, i.e., the processes. Process-centered methodologies utilize process models (Chapter 6) as the core of the system concept. Analysts concentrate initially on representing the system concept as a set of processes with information flowing into and out of the processes (e.g., in Figure 1-3, pay check details flow in to the Produce Pay Checks process, and pay checks are produced as output).

Data-centered methodologies focus first on defining the contents of the data storage containers and how the contents are organized. Data-centered methodologies utilize data models (Chapter 7) as the core of the system concept. For example, analysts concentrate initially on identifying the data that must be available to produce

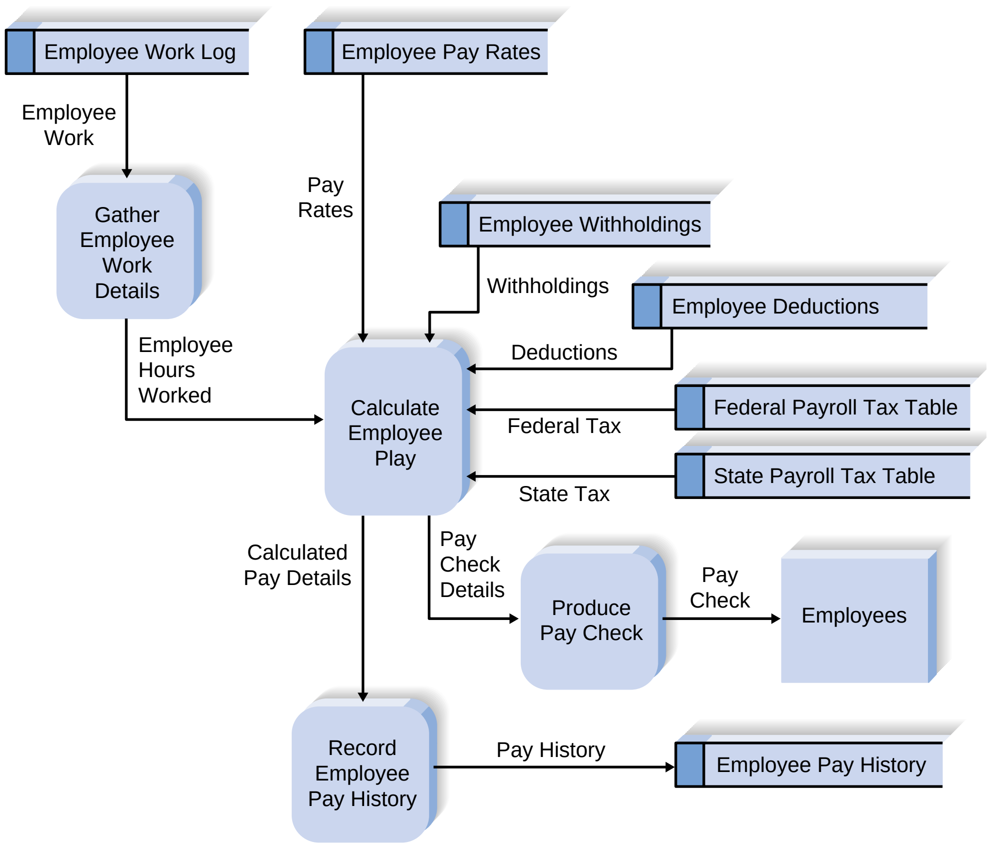

FIGURE 1-3
A Simple Model for Employee Payroll

the payroll and organizing that data into well-defined structures (e.g., employee work log, employee pay rates, payroll tax tables, employee pay history, etc.).

Object-oriented methodologies (Chapter 15) attempt to balance the focus between processes and data. Object-oriented methodologies utilize the *Unified Modeling Language (UML)* to describe the system concept as a collection of objects incorporating both data and processes.[3]

Another important factor in categorizing methodologies is the sequencing of the SDLC phases and the amount of time and effort devoted to each.[4] In the early days of computing, the need for formal and well-planned life cycle methodologies was not well understood. Programmers tended to move directly from a very simple planning phase right into the construction step of the implementation phase; in

[3] The classic modern process-centered methodology is that by Edward Yourdon, *Modern Structured Analysis,* Englewood Cliffs, NJ: Yourdon Press, 1989. An example of a data-centered methodology is information engineering by James Martin, *Information Engineering,* volumes 1–3, Englewood Cliffs, NJ: Prentice Hall, 1989. Many new object-oriented methodologies are based on the Unified Modeling Language defined in *UML Document Set,* Santa Clara, CA: Relational Software Corp., 1997. A widely accepted standardized methodology that balances processes and data is IDEF; see FIPS 183, *Integration Definition for Function Modeling.* Federal Information Processing Standards Publications, Washington, DC: U.S. Department of Commerce, 1993.

[4] A good reference for comparing systems development methodologies is Steve McConnell, *Rapid Development,* Redmond, WA: Microsoft Press, 1996.

other words, they moved directly from a very fuzzy, not-well-thought-out system request into writing code.

This is the same approach that you may sometimes use when writing programs for a programming class. It can work for small programs that require only one programmer, but if the requirements are complex or unclear, you may miss important aspects of the problem and have to start all over again, throwing away part of the program (and the time and effort spent writing it). This approach also makes teamwork difficult because members have little idea about what needs to be accomplished and how to work together to produce a final product.

In the following sections, we describe three major categories of systems development methodologies that have evolved over time: Structured Design, Rapid Application Development (RAD), and Agile Development. Each category represents a collection of methodologies that attempts to improve on previous practice, and varies in terms of the progression through the SDLC phases and the emphasis placed on each phase.

Structured Design

The first category of systems development methodologies is called *structured design*. These methodologies became dominant in the 1980s, replacing the previous ad hoc and undisciplined approach. Structured design methodologies adopt a formal step-by-step approach to the SDLC that moves logically from one phase to the next.

Structured design also introduced the use of formal modeling or diagramming techniques to describe a system's basic business processes and the data that support them. Traditional structured design uses one set of diagrams to represent the processes (process models) and a separate set of diagrams to represent data (data models). Because two sets of models are used, the systems analyst must decide which set to develop first and use as the core of the system—process models or data models. Since each type of model is important to the system, there is much debate over whether to emphasize processes before data or vice versa. Numerous process-centered and data-centered methodologies follow the basic approach of the two structured design categories outlined next.

Waterfall Development

The original structured design methodology (that is still used today) is *waterfall development*. With waterfall development-based methodologies, the analysts and users proceed sequentially from one phase to the next (see Figure 1-4). The key deliverables for each phase are typically voluminous (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins. This methodology is called waterfall development because it moves forward from phase to phase in the same manner as a waterfall. Although it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult (imagine yourself as a salmon trying to swim upstream in a waterfall as shown in Figure 1-4).

The two key advantages of waterfall development-based methodologies are that system requirements are identified long before programming begins and that changes to the requirements are minimized as the project proceeds. The two key disadvantages are that the design must be completely specified before programming

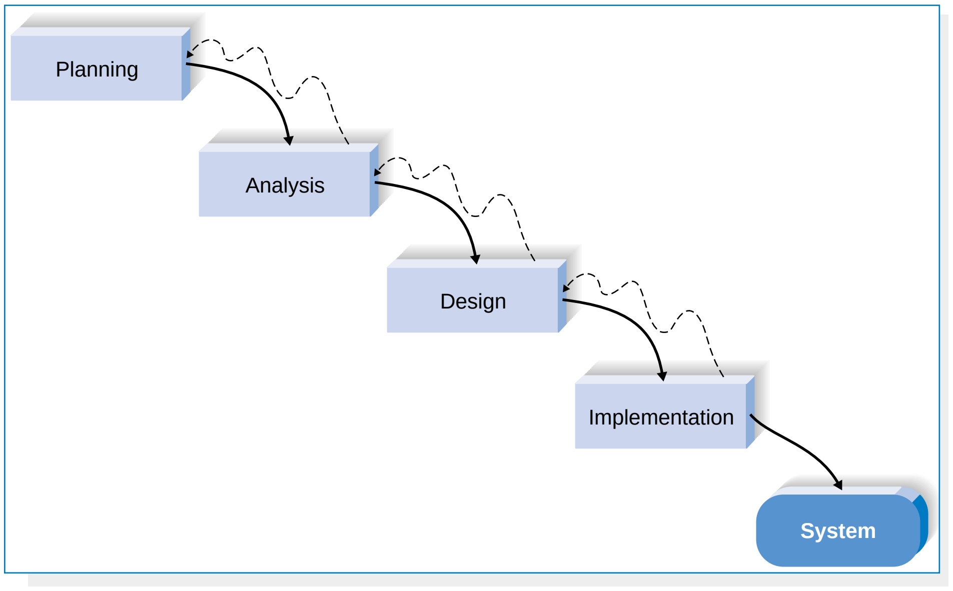

FIGURE 1-4
Waterfall Development-based Methodology

begins and that a long time elapses between the completion of the system proposal in the analysis phase and the delivery of the system (usually many months or years). The deliverables are often a poor communication mechanism, so important requirements can be overlooked in the documentation. Users rarely are prepared for their introduction to the new system, which occurs long after the initial idea for the system was introduced. If the project team misses important requirements, expensive post-implementation programming may be needed (imagine yourself trying to design a car on paper; how likely would you be to remember to include interior lights that come on when the doors open or to specify the right number of valves on the engine?).

Today's dynamic business world often imposes rapid environmental change on businesses. A system that meets existing environmental conditions during the analysis phase may need considerable rework to match the environment when it is implemented. This rework requires going back to the initial phase and making needed changes through each of the subsequent phases in turn.

Parallel Development The *parallel development*-based methodologies attempt to address the long time interval between the analysis phase and the delivery of the system. Instead of doing the design and implementation in sequence, a general design for the whole system is performed, then the project is divided into a series of distinct subprojects that can be designed and implemented in parallel. Once all subprojects are complete, there is a final integration of the separate pieces, and the system is delivered (Figure 1-5).

The primary advantage of these methodologies is that the schedule time required to deliver a system is shortened; thus, there is less chance of changes in the business environment causing rework. The approach still suffers from problems caused by lengthy deliverables. It also adds a new problem: sometimes the subprojects are not completely independent; design decisions made in one subproject may affect another, and the end of the project may involve significant integration challenges.

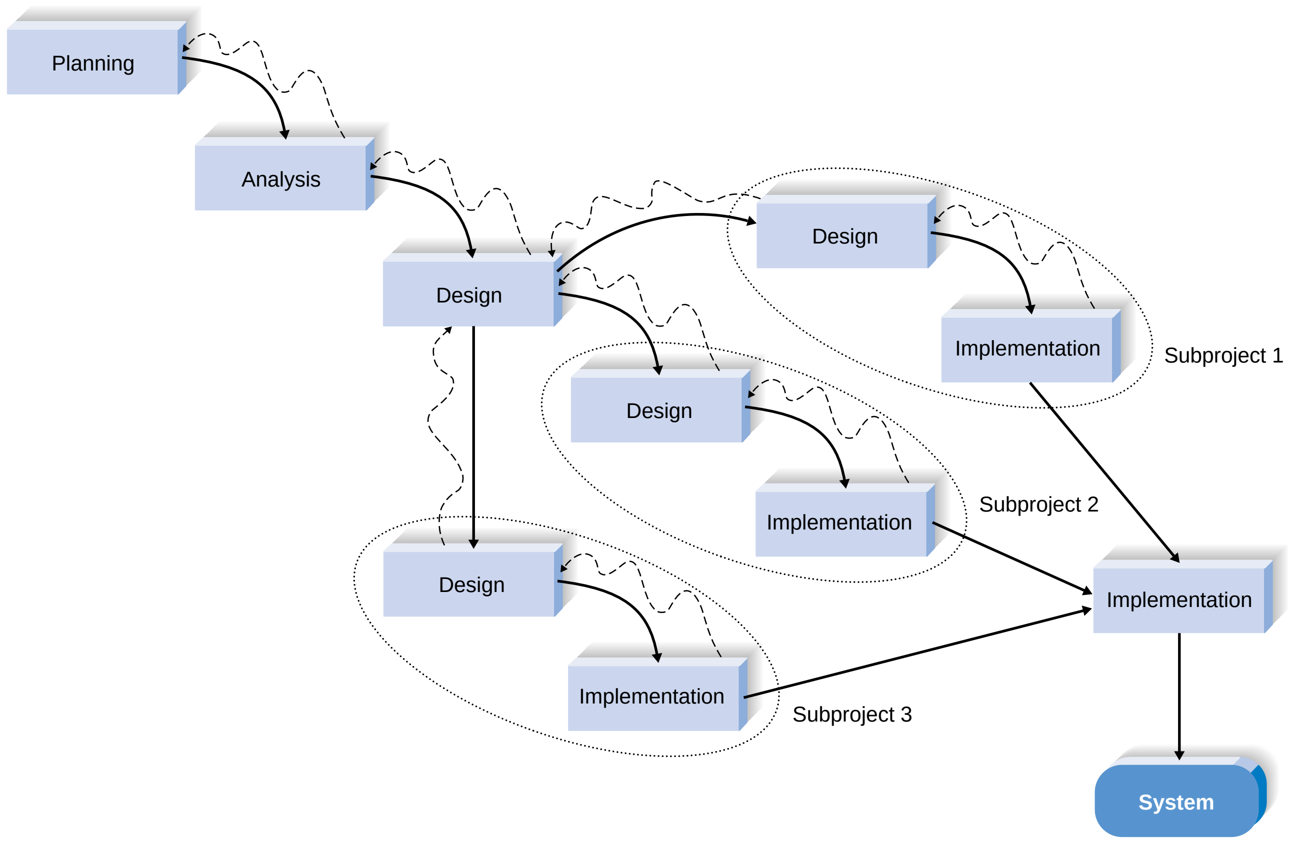

FIGURE 1-5
A Parallel Development-based Methodology

Rapid Application Development (RAD)

The second system development methodology category includes *rapid application development (RAD)*-based methodologies. These are a newer class of system development methodologies that emerged in the 1990s in response to both structured design methodology weaknesses. RAD-based methodologies adjust the SDLC phases to get some part of the system developed quickly and into the hands of the users. In this way, the users can better understand the system and suggest revisions that bring the system close to what is needed.[5] There are process-centered, data-centered, and object-oriented methodologies that follow the basic approaches of the three RAD categories described in this section.

Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as CASE (computer-aided software engineering) tools (see Chapter 3), JAD (joint application design) sessions (see Chapter 5), fourth-generation/visual programming languages that simplify and speed up programming (e.g., Visual Basic.NET), and code generators that automatically produce programs from design specifications. It is the combination of the changed SDLC phases and the use of

[5] One of the best RAD books is that by Steve McConnell, *Rapid Development*, Redmond. WA: Microsoft Press, 1996.

these tools and techniques that improves the speed and quality of systems development. One possible subtle problem with RAD-based methodologies, however, is managing user expectations. As systems are developed more rapidly and users gain a better understanding of information technology, user expectations may dramatically increase and system requirements expand during the project. This was less of a problem with structured design methodologies where the system requirements, once determined, were allowed only minimal change.

Phased Development *The phased development*-based methodologies break the overall system into a series of *versions* that are developed sequentially. The analysis phase identifies the overall system concept, and the project team, users, and system sponsor then categorize the requirements into a series of versions. The most important and fundamental requirements are bundled into the first version of the system. The analysis phase then leads into design and implementation, but only with the set of requirements identified for version 1 (see Figure 1-6).

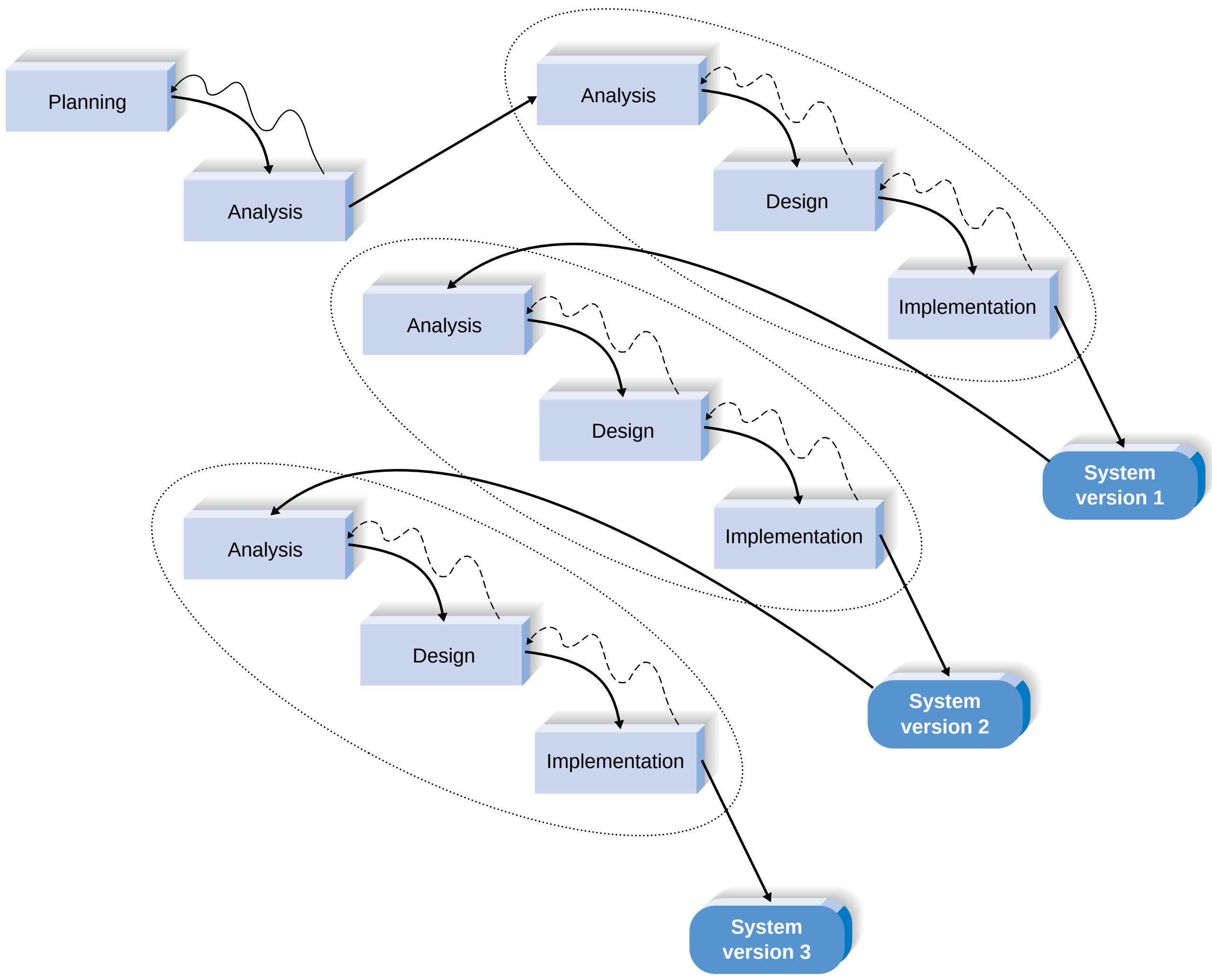

FIGURE 1-6
A Phased Development-based Methodology

Once version 1 is implemented, work begins on version 2. Additional analysis is performed on the basis of the previously identified requirements and combined with new ideas and issues that arose from users' experience with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.

Phased development-based methodologies have the advantage of quickly getting a useful system into the hands of the users. Although it does not perform all the functions the users need at first, it begins to provide business value sooner than if the system were delivered only after all requirements are completed, as is the case with the waterfall or parallel methodologies. Likewise, because users begin to work with the system sooner, they are more likely to identify important additional requirements sooner than with structured design situations.

The major drawback to phased development is that users begin to work with systems that are intentionally incomplete. It is critical to identify the most important and useful features and include them in the first version while managing users' expectations along the way.

Prototyping The *prototyping*-based methodologies perform the analysis, design, and implementation phases concurrently, and all three phases are performed repeatedly in a cycle until the system is completed. With these methodologies, a basic analysis and design are performed, and work immediately begins on a *system prototype*, a "quick-and-dirty" program that provides a minimal amount of features. The first prototype is usually the first part of the system that the user will use. This is shown to the users and the project sponsor, who provide reaction and comments. This feedback is used to reanalyze, redesign, and reimplement a second prototype that provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization. After the prototype (now called the "system") is installed, refinement occurs until it is accepted as the new system (see Figure 1-7).

The key advantage of a prototyping-based methodology is that it *very* quickly provides a system for the users to interact with, even if it is not initially ready for widespread organizational use. Prototyping reassures the users that the project team is working on the system (there are no long time intervals in which the users perceive little progress), and the approach helps to more quickly refine real require-

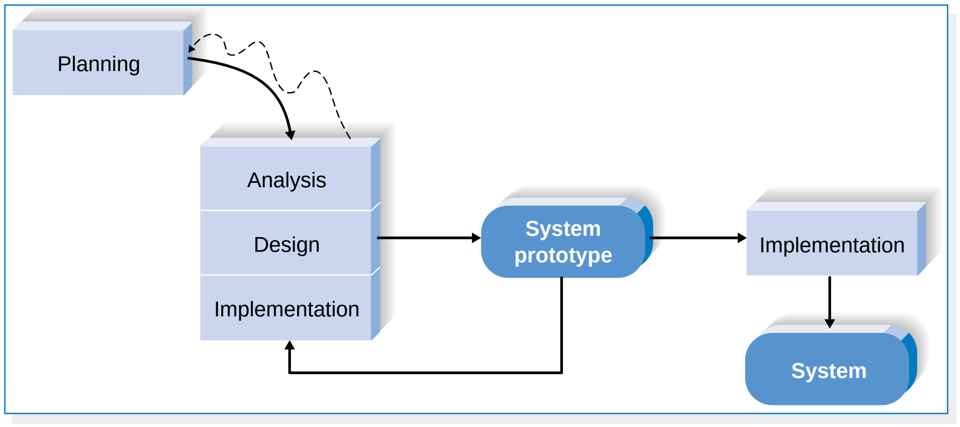

FIGURE 1-7
A Prototyping-based Methodology

ments. Rather than attempting to understand system specification materials, the users can interact with the prototype to better understand what it can and cannot do.

The major problem with prototyping is that its fast-paced system releases challenge attempts to conduct careful, methodical analysis. Often the prototype undergoes such significant changes that many initial design decisions prove to be poor ones. This can cause problems in the development of complex systems because fundamental issues and problems are not recognized until well into the development process. Imagine building a car and discovering late in the prototyping process that you have to take the whole engine out to change the oil (because no one thought about the need to change the oil until after the car had been driven 10,000 miles).

Throwaway Prototyping *Throwaway prototyping*-based methodologies are similar to the prototyping-based methodologies in that they include the development of prototypes; however, throwaway prototypes are done at a different point in the SDLC. These prototypes are used for a very different purpose than ones previously discussed, and they have a very different appearance[6] (see Figure 1-8).

The throwaway prototyping-based methodologies have a relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. Many of the features suggested by the users may not be well understood, however, and there may be challenging technical issues to be solved. Each of these issues is examined by analyzing, designing, and building a *design prototype.* A design prototype is not a working system; it is a product that represents a part of the system that needs additional refinement, and it contains only enough detail to

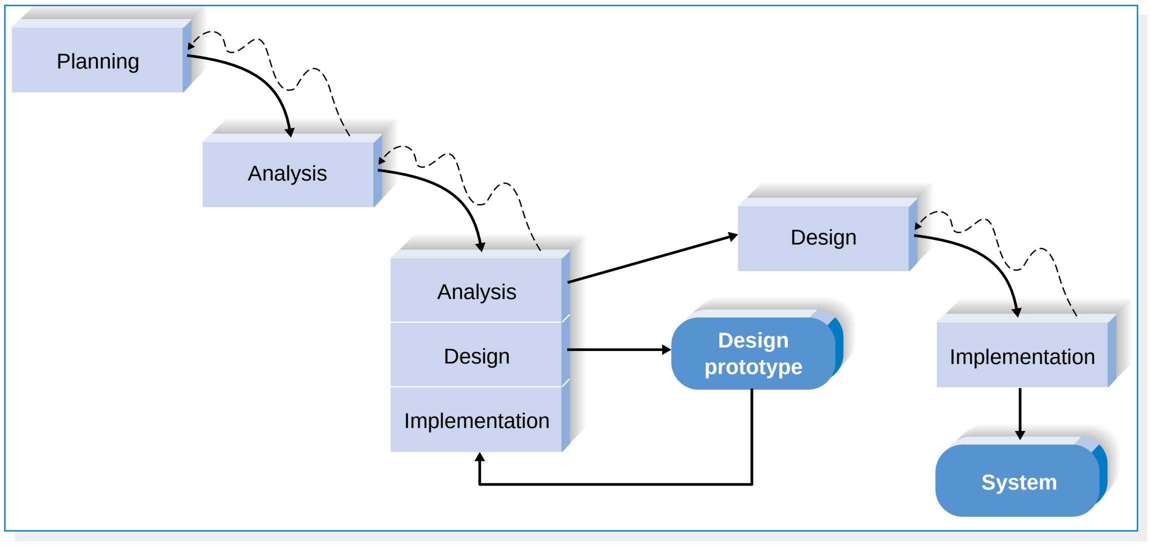

FIGURE 1-8
A Throwaway Prototyping-based Methodology

[6] Our description of the throwaway prototyping methodology is a modified version of the spiral development methodology developed by Barry Boehm, "A Spiral Model of Software Development and Enhancement," *Computer,* May 1988, 21(5):61–72.

enable users to understand the issues under consideration. For example, suppose users are not completely clear on how an order entry system should work. The analyst team might build a series of HTML pages viewed using a Web browser to help the users visualize such a system. In this case, a series of mock-up screens *appear* to be a system, but they really do nothing. Or, suppose that the project team needs to develop a sophisticated graphics program in Java. The team could write a portion of the program with artificial data to ensure that they could create a full-blown program successfully.

A system that is developed using this type of methodology probably uses several design prototypes during the analysis and design phases. Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built. Once the issues are resolved, the project moves into design and implementation. At this point, the design prototypes are thrown away, which is an important difference between this approach and prototyping, in which the prototypes evolve into the final system.

Throwaway prototyping-based methodologies balance the benefits of well-thought-out analysis and design phases with the advantages of using prototypes to refine key issues before a system is built. It may take longer to deliver the final system as compared with prototyping-based methodologies (because the prototypes do not become the final system), but the approach usually produces more stable and reliable systems.

Agile Development

A third category of systems development methodologies is still emerging today: *Agile Development.*[7] These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. They focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead and the time spent on those tasks. Instead, projects emphasize simple, iterative application development. Examples of Agile Development methodologies include extreme programming,[8] Scrum,[9] and the Dynamic Systems Development Method (DSDM).[10] To illustrate an agile development methodology, we describe extreme programming in the next section. Typically, extreme programming is used in conjunction with object-oriented programming languages.

Extreme Programming

Extreme programming (XP)[11] is founded on four core values: communication, simplicity, feedback, and courage. These four values provide a foundation XP developers use to create any system. First, the developers must provide rapid feedback to the end users on a continuous basis. Second, XP requires developers to follow the KISS (Keep It Simple, Stupid) principle. Third, developers must make incremental changes to grow the system and they must

[7] For more information, see www.AgileAlliance.org.

[8] For more information, see www.extremeprogramming.com.

[9] For more information, see www.controlchaos.com.

[10] For more information, see www.dsdm.com.

[11] For more information, see K. Beck, *eXtreme Programming Explained: Embrace Change,* Reading, MA: Addison-Wesley, 2000, and M. Lippert, S. Roock, and H. Wolf, *eXtreme Programming in Action: Practical Experiences from Real World Projects,* New York: John Wiley & Sons, 2002.

embrace change, not merely accept it. Fourth, developers must have a quality-first mentality. XP also supports team members in developing their own skills.

Three of the key principles that XP uses to create successful systems are continuous testing, simple coding performed by pairs of developers, and close interactions with end users to build systems very quickly. After a superficial planning process, project teams perform analysis, design, and implementation phases iteratively (see Figure 1-9).

Testing and efficient coding practices are core to XP. In fact, each day code is tested and placed into an integrative testing environment. If bugs exist, the code is backed out until it is completely free of errors. XP relies heavily on *refactoring*, which is a disciplined way to restructure code to keep it simple.

An XP project begins with user stories that describe what the system needs to do. Then, programmers code in small, simple modules and test to meet those needs. Users are required to be available to clear up questions and issues as they arise. Standards are very important to minimize confusion, so XP teams use a common set of names, descriptions, and coding practices. XP projects deliver results sooner than even the RAD approaches, and they rarely get bogged down in gathering requirements for the system.

For small projects with highly motivated, cohesive, stable, and experienced teams, XP should work just fine. However, if the project is not small or the teams aren't jelled[12] then the likelihood of a successful XP project is reduced. Consequently, the use of XP in combination with outside contractors produces a highly questionable outcome, since the outside contractors may never "jell" with insiders.[13] XP requires a great deal of discipline to prevent projects from becoming unfocused and chaotic. Furthermore, it is only recommended for small groups of developers (not more than ten), and it is not advised for mission-critical applica-

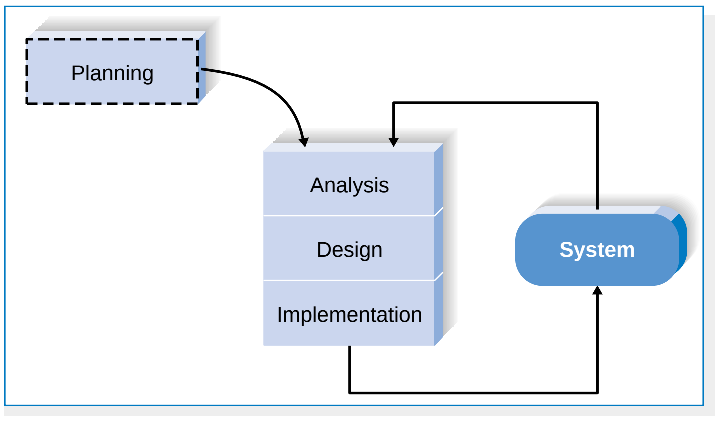

FIGURE 1-9
An Extreme Programming-based Methodology

[12] A "jelled team" is one that has low turnover, a strong sense of identity, a sense of eliteness, a feeling that they jointly own the product being developed, and enjoyment in working together. For more information regarding jelled teams, see T. DeMarco and T. Lister. *Peopleware: Productive Projects and Teams*, New York, Dorsett, House, 1987.

[13] Considering the tendency for offshore outsourcing, this is a major obstacle for XP to overcome. For more information on offshore outsourcing, see P. Thibodeau, ITAA panel debates outsourcing pros, cons, *Computerworld Morning Update,* September 25, 2003; and S.W. Ambler, "Chicken Little Was Right," *Software Development,* October 2003.

tions. Since little analysis and design documentation is produced with XP there is only code documentation; therefore, maintenance of large systems developed using XP may be impossible. Also, since mission-critical business information systems tend to exist for a long time, the utility of XP as a business information system development methodology is in doubt. Finally, the methodology requires considerable on-site user input, something that is frequently difficult to obtain.[14]

Selecting the Appropriate Development Methodology

Since there are many methodologies, the first challenge faced by analysts is to select which methodology to use. Choosing a methodology is not simple, because no one methodology is always best (if it were, we'd simply use it everywhere!). Many organizations have standards and policies to guide the choice of methodology. You will find that organizations range from having one "approved" methodology to having several methodology options to having no formal policies at all.

Figure 1-10 summarizes some important methodology selection criteria. One important item not discussed in this figure is the degree of experience of the analyst team. Many of the RAD methodologies require the use of new tools and techniques that have a significant learning curve. Often these tools and techniques increase the complexity of the project and require extra time for learning. Once they are adopted and the team becomes experienced, the tools and techniques can significantly increase the speed in which the methodology can deliver a final system.

Clarity of User Requirements When the user requirements for what the system should do are unclear, it is difficult to understand them by talking about them and explaining them with written reports. Users normally need to interact with technology to really understand what the new system can do and how to best apply it to their needs. The RAD methodologies of prototyping and throwaway prototyping are usually more appropriate when user requirements are unclear because

	Structured Methodologies		RAD Methodologies			Agile Methodologies
Ability to Develop Systems	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP
with Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent
with Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Poor
that are Complex	Good	Good	Good	Poor	Excellent	Poor
that are Reliable	Good	Good	Good	Poor	Excellent	Good
with a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent
with Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Good

FIGURE 1-10
Criteria for Selecting a Methodology

[14] Many of the observations described on the utility of XP as a development approach were based on conversations with Brian Henderson-Sellers.

they provide prototypes for users to interact with early in the SDLC. Agile development may also be appropriate if on-site user input is available.

Familiarity with Technology When the system will use new technology with which the analysts and programmers are not familiar (e.g., the first Web development project with Java), applying the new technology early in the methodology will improve the chance of success. If the system is designed without some familiarity with the base technology, risks increase because the tools may not be capable of doing what is needed. Throwaway prototyping-based methodologies are particularly appropriate for a lack of familiarity with technology because they explicitly encourage the developers to create design prototypes for areas with high risks. Phased development-based methodologies are good as well because they create opportunities to investigate the technology in some depth before the design is complete. While one might think prototyping-based methodologies would also be appropriate, they are much less so, because the early prototypes that are built usually only scratch the surface of the new technology. Usually, it is only after several prototypes and several months that the developers discover weaknesses or problems in the new technology.

System Complexity Complex systems require careful and detailed analysis and design. Throwaway prototyping-based methodologies are particularly well suited to such detailed analysis and design, but prototyping-based methodologies are not. The traditional structured design-based methodologies can handle complex systems, but without the ability to get the system or prototypes into users' hands early on, some key issues may be overlooked. Although the phased development-based methodologies enable users to interact with the system early in the process, we have observed that project teams who follow these methodologies tend to devote less attention to the analysis of the complete problem domain than they might if they were using other methodologies.

System Reliability System reliability is usually an important factor in system development. After all, who wants an unreliable system? However, reliability is just one factor among several. For some applications reliability is truly critical (e.g., medical equipment, missile control systems), while for other applications it is merely important (e.g., games, Internet video). Throwaway prototyping-based methodologies are most appropriate when system reliability is a high priority, because they combine detailed analysis and design phases with the ability for the project team to test many different approaches through design prototypes before completing the design. Prototyping-based methodologies are generally not a good choice when reliability is critical because they lack the careful analysis and design phases that are essential for dependable systems.

Short Time Schedules Projects that have short time schedules are well suited for RAD-based methodologies because those methodologies are designed to increase the speed of development. Prototyping and phased development-based methodologies are excellent choices when timelines are short because they best enable the project team to adjust the functionality in the system on the basis of a specific delivery date. If the project schedule starts to slip, it can be readjusted by removing functionality from the version or prototype under development. Waterfall-based methodologies are the worst choice when time is at a premium because they do not allow for easy schedule changes.

Schedule Visibility One of the greatest challenges in systems development is knowing whether a project is on schedule. This is particularly true of the structured design methodologies because design and implementation occur at the end of the project. The RAD-based methodologies move many of the critical design decisions earlier in the project to help project managers recognize and address risk factors and keep expectations in check.

PROJECT TEAM SKILLS AND ROLES

As should be clear from the various phases and steps performed during the SDLC, the project team needs a variety of skills. Project members are *change agents* who identify ways to improve an organization, build an information system to support them, and train and motivate others to use the system. Leading a successful organizational change effort is one of the most difficult jobs that someone can do. Understanding what to change, how to change it, and convincing others of the need for change requires a wide range of skills. These skills can be broken down into six major categories: technical, business, analytical, interpersonal, management, and ethical.

Analysts must have the technical skills to understand the organization's existing technical environment, the new system's technology foundation, and the way in which both can be fit into an integrated technical solution. Business skills are required to understand how IT can be applied to business situations and to ensure that the IT delivers real business value. Analysts are continuous problem solvers at both the project and the organizational level, and they put their analytical skills to the test regularly.

Often, analysts need to communicate effectively one-on-one with users and business managers (who often have little experience with technology) and with programmers (who often have more technical expertise than the analyst). They must be able to give presentations to large and small groups and write reports. Not only do they need to have strong interpersonal abilities, but also they need to manage people with whom they work and they must manage the pressure and risks associated with unclear situations.

Finally, analysts must deal fairly, honestly, and ethically with other project team members, managers, and system users. Analysts often deal with confidential information or information that, if shared with others, could cause harm (e.g., dissent among employees); it is important to maintain confidence and trust with all people.

In addition to these six general skill sets, analysts require many specific skills that are associated with roles that are performed on a project. In the early days of systems development, most organizations expected one person, the analyst, to have all of the specific skills needed to conduct a systems development project. Some small organizations still expect one person to perform many roles, but because organizations and technology have become more complex, most large organizations now build project teams that contain several individuals with clearly defined responsibilities. Different organizations divide the roles differently, but Figure 1-11 presents one commonly used set of project team roles. Most IS teams include many other individuals such as the *programmers* who actually write the system's programs, *network engineers*, who focus on the design of the network, *database administrators*, who deal with optimizing the physical design of the database, and *technical writers*, who prepare user and system documentation.

Role	Responsibilities
Business analyst	Analyzing the key business aspects of the system
	Identifying how the system will provide business value
	Designing the new business processes and policies
Systems analyst	Identifying how technology can improve business processes
	Designing the new business processes
	Designing the information system
	Ensuring that the system conforms to information systems standards
Infrastructure analyst	Ensuring the system conforms to infrastructure standards
	Identifying infrastructure changes needed to support the system
Change management analyst	Developing and executing a change management plan
	Developing and executing a user training plan
Project manager	Managing the team of analysts, programmers, technical writers, and other specialists
	Developing and monitoring the project plan
	Assigning resources
	Serving as the primary point of contact for the project

FIGURE 1-11
Project Team Roles

Business Analyst

The *business analyst* focuses on the business issues surrounding the system. These include identifying the business value that the system will create, developing ideas and suggestions for how the business processes can be improved, and designing the new processes and policies in conjunction with the systems analyst. This individual will likely have business experience and some type of professional training (e.g., the business analyst for accounting systems will likely be a CPA [certified public accountant in the United States] or a CA [chartered accountant in Great Britain and Canada]). He or she represents the interests of the project sponsor and the ultimate

YOUR TURN

1-1 Selecting a Methodology

Suppose you are an analyst for the ABC Company, a large consulting firm with offices around the world. The company wants to build a new knowledge management system that can identify and track the expertise of individual consultants anywhere in the world on the basis of their education and the various consulting projects on which they have worked. Assume that this is a new idea that has never before been attempted in ABC or elsewhere. ABC has an international network, but the offices in each country may use somewhat different hardware and software. ABC management wants the system up and running within a year.

QUESTION:
What methodology would you recommend ABC Company use? Why?

users of the system. The business analyst assists in the planning and design phases but is most active in the analysis phase.

Systems Analyst

The *systems analyst* focuses on the IS issues surrounding the system. This person develops ideas and suggestions for how IT can improve business processes, designs the new business processes with help from the business analyst, designs the new information system, and ensures that all IS standards are maintained. The systems analyst likely will have significant training and experience in analysis and design, programming, and even areas of the business. He or she represents the interests of the IS department and works intensively throughout the project but perhaps less so during the implementation phase.

Infrastructure Analyst

The *infrastructure analyst* focuses on the technical issues surrounding how the system will interact with the organization's technical infrastructure (e.g., hardware, software, networks, and databases). The infrastructure analyst's tasks include ensuring that the new information system conforms to organizational standards and identifying infrastructure changes needed to support the system. This individual will likely have significant training and experience in networking, database administration, and various hardware and software products. He or she represents the interests of the organization and IS group that ultimately will have to operate and support the new system once it has been installed. The infrastructure analyst works throughout the project but perhaps less so during planning and analysis phases.

Change Management Analyst

The *change management analyst* focuses on the people and management issues surrounding the system installation. The roles of this person include ensuring that adequate documentation and support are available to users, providing user training on the new system, and developing strategies to overcome resistance to change. This individual likely will have significant training and experience in organizational behavior in general and change management in particular. He or she represents the interests of the project sponsor and users for whom the system is being designed. The change management analyst works most actively during the implementation phase but begins laying the groundwork for change during the analysis and design phases.

YOUR TURN

1-2 Being an Analyst

Suppose you decide to become an analyst after you graduate. What type of analyst would you most prefer to be? What type of courses should you take before you graduate? What type of summer job or internship should you seek?

QUESTION:
Develop a short plan that describes how you will prepare for your career as an analyst.

Project Manager

The *project manager* is responsible for ensuring that the project is completed on time and within budget and that the system delivers all benefits that were intended by the project sponsor. The role of the project manager includes managing the team members, developing the project plan, assigning resources, and being the primary point of contact when people outside the team have questions about the project. This individual likely will have significant experience in project management and likely has worked for many years as a systems analyst beforehand. He or she represents the interests of the IS department and the project sponsor. The project manager works intensely during all phases of the project.

SUMMARY

The System Development Life Cycle

All system development projects follow essentially the same fundamental process called the system development life cycle (SDLC). The SDLC starts with a planning phase in which the project team identifies the business value of the system, conducts a feasibility analysis, and plans the project. The second phase is the analysis phase, in which the team develops an analysis strategy, gathers information, and builds a set of analysis models. In the next phase, the design phase, the team develops the physical design, architecture design, interface design, data base and file specifications, and program design. In the final phase, implementation, the system is built, installed, and maintained.

Systems Development Methodologies

System development methodologies are formalized approaches to implementing an SDLC. System development methodologies have evolved over several decades. Structured design methodologies, such as waterfall and parallel development, move logically from one phase to the next and are more focused on system processes (process-centric) or on system data (data-centric). They produce a solid, well-thought-out system but can overlook requirements because users must specify them early in the design process before seeing the actual system. RAD-based methodologies attempt to speed up development and make it easier for users to specify requirements by having parts of the system developed sooner either by producing different versions (phased development) or by using prototypes (prototyping, throwaway prototyping). RAD-based methodologies still tend to be either process-centric or data-centric. Agile development methodologies focus on streamlining the SDLC by eliminating many of the tasks and time associated with requirements definition and documentation. The choice of a methodology is influenced by several factors: clarity of the user requirements; familiarity with the base technology; system complexity; need for system reliability; time pressures; and need to see progress on the time schedule.

Project Team Roles and Skills

The project team needs a variety of skills. As organizational change agents, all analysts need to have general skills, such as technical, business, analytical, interpersonal, management, and ethical. However, different kinds of analysts require specific skills in addition to these. Business analysts usually have business skills that help them to understand the business issues surrounding the system, whereas systems analysts also have significant experience in analysis and design and programming. The infrastructure analyst focuses on technical issues surrounding how the system will interact with the organization's technical infrastructure, and the change management analyst focuses on people and management issues surrounding the system installation. In addition to analysts, project teams will include a project manager, programmers, network engineers, database administrators, and technical writers.

KEY TERMS

Agile development
Analysis model
Analysis phase
Analysis strategy
Approval committee
Architecture design
As-is system
Business analyst
Change agent
Change management analyst
Construction
Data model
Data-centered methodology
Database administrator
Database and file specification
Deliverable
Design phase
Design prototype
Design strategy
Extreme programming (XP)
Feasibility analysis
Gradual refinement
Implementation phase
Infrastructure analyst
Interface design
Installation
Methodology
Network engineer
Object-oriented methodology
Parallel development
Phase
Phased development
Planning phase
Process model
Process-centered methodology
Program design
Programmer
Project initiation
Project management
Project manager
Project plan
Project sponsor
Prototyping
Rapid application development (RAD)
Requirements gathering
Steering committee
Step
Structured design
Support plan
Systems analyst
System development life cycle (SDLC)
System proposal
System prototype
System request
System specification
Technical writer
Technique
Throwaway prototyping
To-be system
Training plan
Unified Modeling Language (UML)
Version
Waterfall development
Workplan

QUESTIONS

1. Compare and contrast phases, steps, techniques, and deliverables.
2. Describe the major phases in the systems development life cycle (SDLC).
3. Describe the principal steps in the planning phase. What are the major deliverables?
4. Describe the principal steps in the analysis phase. What are the major deliverables?
5. Describe the principal steps in the design phase. What are the major deliverables?
6. Describe the principal steps in the implementation phase. What are the major deliverables?
7. Describe the roles of the project sponsor and the approval committee.
8. What does *gradual refinement* mean in the context of SDLC?
9. Compare and contrast process-centered methodologies, data-centered methodologies, and object-oriented methodologies.
10. Compare and contrast structured design methodologies in general to rapid application development (RAD) methodologies in general.
11. Compare and contrast extreme programming and throwaway prototyping.

12. Describe the major elements and issues with waterfall development.
13. Describe the major elements and issues with parallel development.
14. Describe the major elements and issues with phased development.
15. Describe the major elements and issues with prototyping.
16. Describe the major elements and issues with throwaway prototyping.
17. What are the key factors in selecting a methodology?
18. What are the six general skills all project team members should have?
19. What are the major roles on a project team?
20. Compare and contrast the role of a systems analyst, business analyst, and infrastructure analyst.
21. Which phase in the SDLC is most important?

EXERCISES

A. Suppose you are a project manager using the waterfall development methodology on a large and complex project. Your manager has just read the latest article in *Computerworld* that advocates replacing the waterfall methodology with prototyping and comes to your office requesting you to switch. What do you say?
B. The basic methodologies discussed in this chapter can be combined and integrated to form new hybrid methodologies. Suppose you were to combine throwaway prototyping with the use of parallel development. What would the methodology look like? Draw a picture (similar to Figure 1-8). How would this new methodology compare to the others? Develop a new column for Figure 1-10.
C. Suppose you were an analyst working for a small company to develop an accounting system. What methodology would you use? Why?
D. Suppose you were an analyst developing a new executive information system (EIS) intended to provide key strategic information from existing corporate databases to senior executives to help in their decision making. What methodology would you use? Why?
E. Suppose you were an analyst developing a new information system to automate the sales transactions and manage inventory for each retail store in a large chain. The system would be installed at each store and exchange data with a mainframe computer at the company's head office. What methodology would you use? Why?
F. Look in the classified section of your local newspaper. What kinds of job opportunities are available for people who want analyst positions? Compare and contrast the skills that the ads ask for to the skills that we presented in this chapter.
G. Think about your ideal analyst position. Write a newspaper ad to hire someone for that position. What requirements would the job have? What skills and experience would be required? How would applicants demonstrate that they have the appropriate skills and experience?

MINICASES

1. Barbara Singleton, manager of western regional sales at the WAMAP Company, requested that the IS department develop a sales force management and tracking system that would enable her to better monitor the performance of her sales staff. Unfortunately, due to the massive backlog of work facing the IS department, her request was given a low priority. After 6 months of inaction by the IS department, Barbara decided to take matters into her own hands. Based on the advice of friends, Barbara purchased a PC and simple database software and constructed a sales force management and tracking system on her own.

Although Barbara's system has been "completed" for about 6 weeks, it still has many features that do not work correctly, and some functions are full of errors. Barbara's assistant is so mistrustful of the system that she has secretly gone back to using her old paper-based system, since it is much more reliable.

Over dinner one evening, Barbara complained to a systems analyst friend, "I don't know what went wrong with this project. It seemed pretty simple to me. Those IS guys wanted me to follow this elaborate set of steps and tasks, but I didn't think all that really applied to a PC-based system. I just thought I could build this system and tweak it around until I got what I wanted without all the fuss and bother of the methodology the IS guys were pushing. I mean, doesn't that just apply to their big, expensive systems?"

Assuming you are Barbara's systems analyst friend, how would you respond to her complaint?

2. Marcus Weber, IS project manager at ICAN Mutual Insurance Co., is reviewing the staffing arrangements for his next major project, the development of an expert system-based underwriters assistant. This new system will involve a whole new way for the underwriters to perform their tasks. The underwriters assistant system will function as sort of an underwriting supervisor, reviewing key elements of each application, checking for consistency in the underwriter's decisions, and ensuring that no critical factors have been overlooked. The goal of the new system is to improve the quality of the underwriters' decisions and to improve underwriter productivity. It is expected that the new system will substantially change the way the underwriting staff do their jobs.

Marcus is dismayed to learn that due to budget constraints, he must choose between one of two available staff members. Barry Filmore has had considerable experience and training in individual and organizational behavior. Barry has worked on several other projects in which the end users had to make significant adjustments to the new system, and Barry seems to have a knack for anticipating problems and smoothing the transition to a new work environment. Marcus had hoped to have Barry's involvement in this project.

Marcus's other potential staff member is Kim Danville. Prior to joining ICAN Mutual, Kim had considerable work experience with the expert system technologies that ICAN has chosen for this expert system project. Marcus was counting on Kim to help integrate the new expert system technology into ICAN's systems environment, and also to provide on-the-job training and insights to the other developers on this team.

Given that Marcus's budget will only permit him to add Barry or Kim to this project team, but not both, what choice do you recommend for him? Justify your answer.