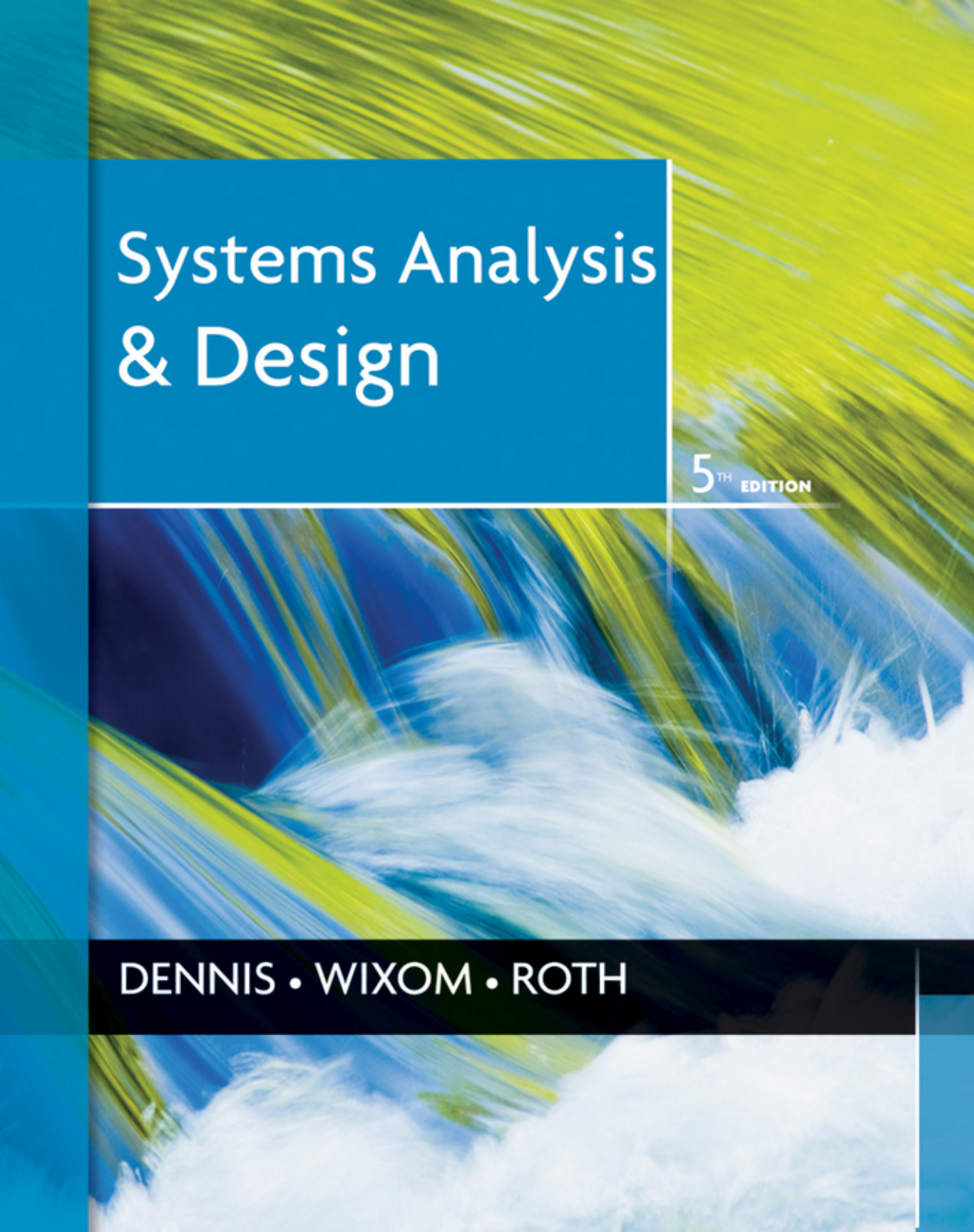
Systems Analysis
& Design
5TH EDITION
DENNIS • WIXOM • ROTH

SYSTEM ANALYSIS AND DESIGN

Fifth Edition

ALAN DENNIS

Indiana University

BARBARA HALEY WIXOM

University of Virginia

ROBERTA M. ROTH

University of Northern Iowa

WILEY

John Wiley & Sons, Inc.

http://www.wiley.com/college/dennis

CREATING THE PROJECT PLAN

Once the project is launched by being selected by the approval committee, it is time to carefully plan the project. The project manager will follow a set of project management guidelines, sometimes referred to as the project management life cycle, as he or she organizes, guides, and directs the project from inception to completion. Generally speaking, the project management phases consist of initiation, planning, execution, control, and closure.

In large organizations or on large projects, the role of project manager is commonly filled by a professional specialist in project management. In smaller organizations or on smaller projects, the systems analyst may fill this role. The project manager must make a myriad of decisions regarding the project, including determining the best project methodology, developing a work plan for the project, determining a staffing plan, and establishing mechanisms to coordinate and control the project.

Project Methodology Options

As we discussed in Chapter 1, the Systems Development Life Cycle (SDLC) provides the foundation for the processes used to develop an information system. 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 they vary in terms of the progression that is followed through the phases of the SDLC. 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. Here we will review several of the predominant methodologies that have evolved over time.

Waterfall Development With *waterfall development*, analysts and users proceed sequentially from one phase to the next. (See Figure 2-2.) The key deliverables for each phase are typically voluminous (often, hundreds of pages) and are presented

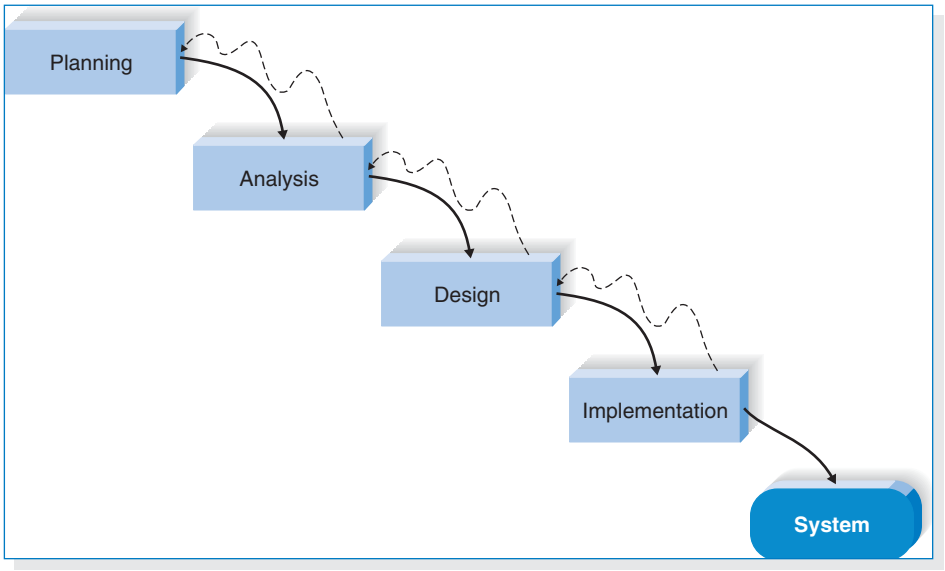

FIGURE 2-2
Waterfall Development

to the approval committee and project sponsor for approval as the project moves from phase to phase. Once the work produced in one phase is approved, the phase ends and the next phase begins. As the project progresses from phase to phase, it moves forward in the same manner as a waterfall. While it is possible to go backward through the phases (e.g., from design back to analysis), it is quite difficult. (Imagine yourself as a salmon trying to swim upstream in a waterfall).

Waterfall development methodologies have the advantages of identifying requirements long before programming begins and limiting changes to the requirements as the project proceeds. The key disadvantages are that the design must be completely specified before programming begins, a long time elapses between the completion of the system proposal in the analysis phase and the delivery of system, and testing is treated almost as an afterthought in the implementation phase. In addition, the deliverables are often a poor communication mechanism, so important requirements may be overlooked in the volumes of documentation. If the project team misses an important requirement, expensive post-implementation programming may be needed. Users may forget the original purpose of the system, since so much time has elapsed between the original idea and actual implementation. Also, in today's dynamic business environment, a system that met the 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.

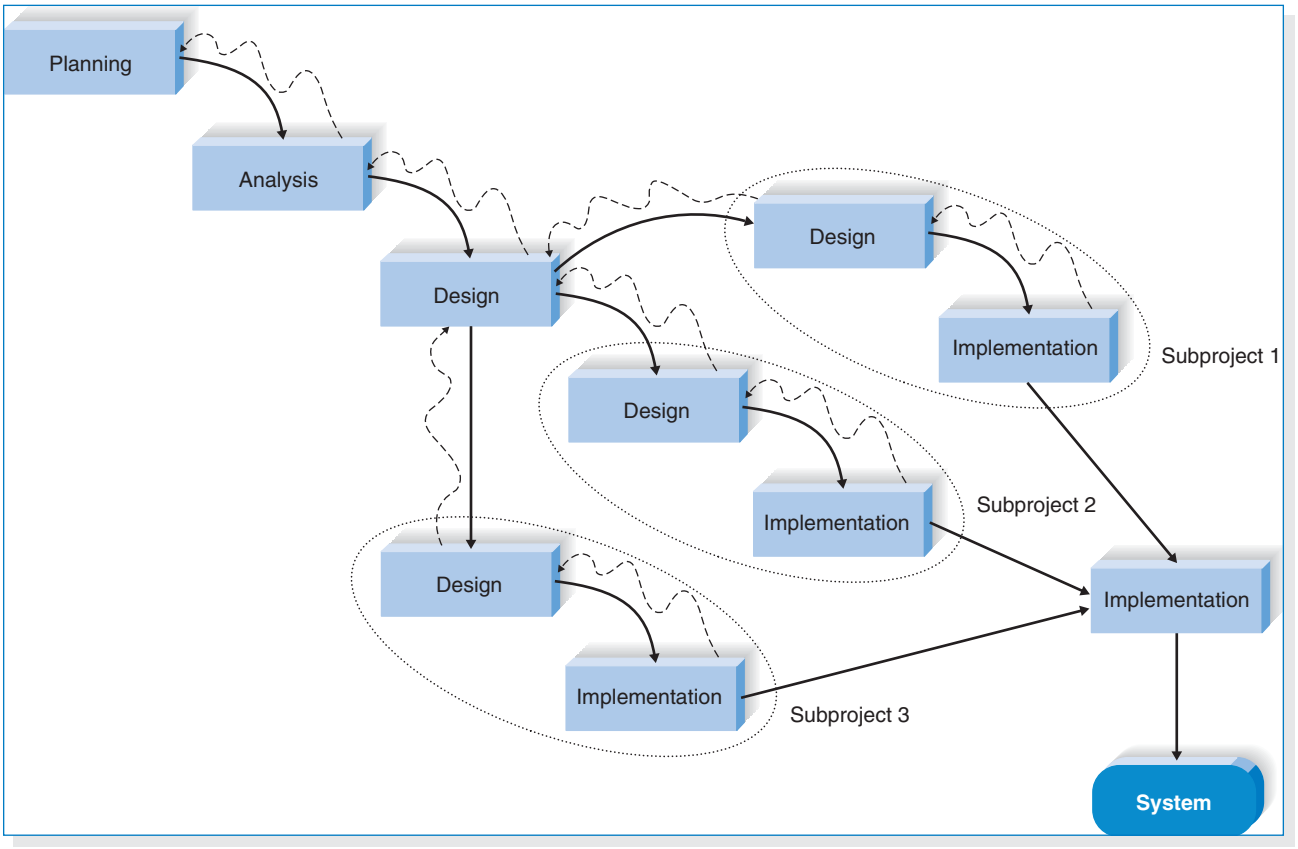

FIGURE 2-3
Parallel Development

There are two important variants of waterfall development. The ***parallel development*** methodologies evolved to address the lengthy time frame of waterfall development. As shown in Figure 2-3, 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 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.

Parallel development reduces the time required to deliver a system, so changes in the business environment are less likely to produce the need for rework. The approach still suffers from problems caused by voluminous deliverables. It also adds a new problem: If the subprojects are not completely independent, design decisions in one subproject may affect another, and at the project end, integrating the subprojects may be quite challenging.

The ***V-model*** is another variation of waterfall development that pays more explicit attention to testing. As shown in Figure 2-4, the development process proceeds down the left-hand slope of the V, defining requirements and designing system components. At the base of the V, the code is written. On the upward-sloping right side of the model, testing of components, integration testing, and, finally, acceptance testing are performed. A key concept of this model is that as requirements are specified and components designed, testing for those elements is also defined. In this manner, each level of testing is clearly linked to a part of the analysis or design phase, helping to ensure high quality and relevant testing and maximize test effectiveness.

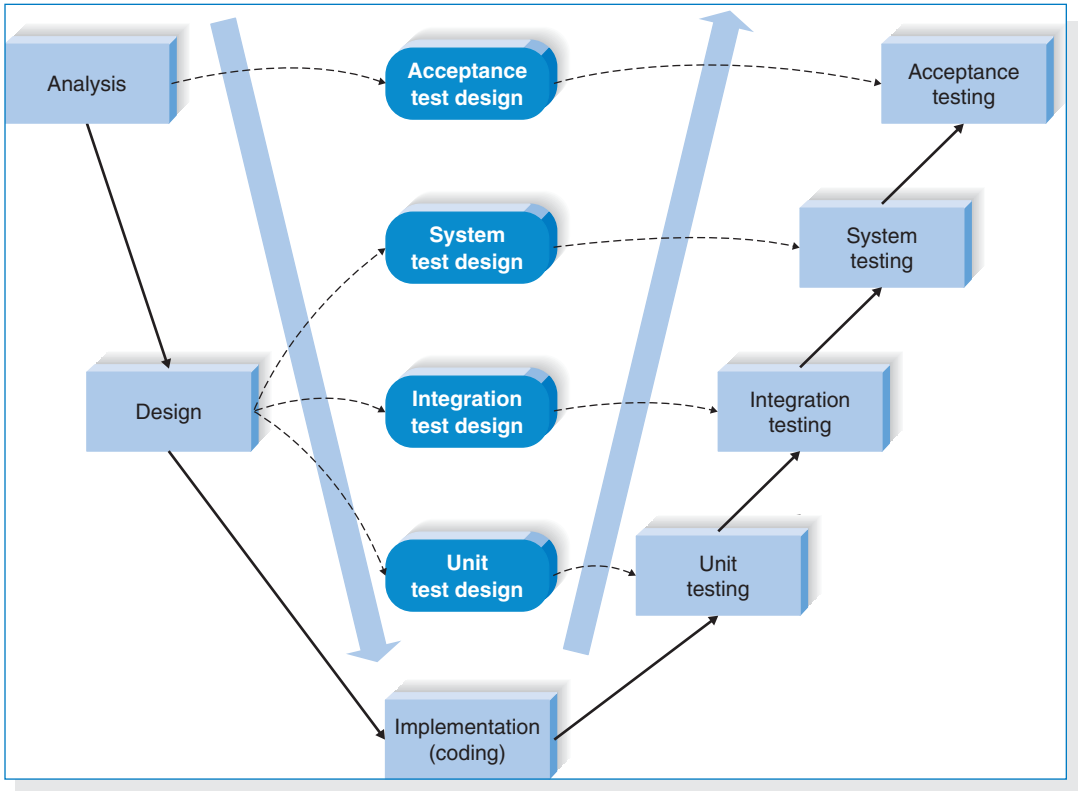

FIGURE 2-4
V-Model

The V-model is simple and straightforward and improves the overall quality of systems through its emphasis on early development of test plans. Testing focus and expertise is involved in the project earlier rather than later; plus, the testers gain knowledge of the project early. It still suffers from the rigidity of the waterfall development process, however, and is not always appropriate for the dynamic nature of the business environment.

Rapid Application Development (RAD)[5] *Rapid application development* is a collection of methodologies that emerged in response to the weaknesses of waterfall development and its variations. RAD incorporates special techniques and computer tools to speed up the analysis, design, and implementation phases in order to get some portion of the system developed quickly and into the hands of the users for evaluation and feedback. CASE (computer-aided software engineering) tools, JAD (joint application development) sessions, fourth-generation/visual programming languages (e.g., Visual Basic.NET), and code generators may all play a role in RAD. While RAD can improve the speed and quality of systems development, it may also introduce a problem in managing user expectations. As systems are developed more quickly and users gain a better understanding of information technology, user expectations may dramatically increase and system requirements may expand during the project (sometimes known as *scope creep* or *feature creep*).

RAD may be conducted in a variety of ways. ***Iterative development*** breaks the overall project into a series of *versions* that are developed sequentially. The most important and fundamental requirements are bundled into the first version of the system. This version is developed quickly by a mini-waterfall process, and once implemented, the users can provide valuable feedback to be incorporated into the next version of the system. (See Figure 2-5.) Iterative development gets a preliminary version of the system to the users quickly so that business value is provided. Since users are working with the system, important additional requirements may be identified and incorporated into subsequent versions. The chief disadvantage of iterative development is that users begin to work with a system that is intentionally incomplete. Users must accept that only the most critical requirements of the system will be available in the early versions and must be patient with the repeated introduction of new system versions.

System prototyping performs the analysis, design, and implementation phases concurrently in order to quickly develop a simplified version of the proposed system and give it to the users for evaluation and feedback. (See Figure 2-6). The system prototype is a "quick and dirty" version of the system and provides minimal features. Following reaction and comments from the users, the developers reanalyze, redesign, and reimplement a second prototype that corrects deficiencies and adds more features. This cycle continues until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization. System prototyping very quickly provides a system for users to evaluate and reassures users that progress is being made. The approach is very useful when users have difficulty expressing requirements for the system. A disadvantage, however, is the lack of careful, methodical analysis prior to making design and implementation decisions. System prototypes may have some fundamental design limitations that are a direct result of an inadequate understanding of the system's true requirements early in the project.

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

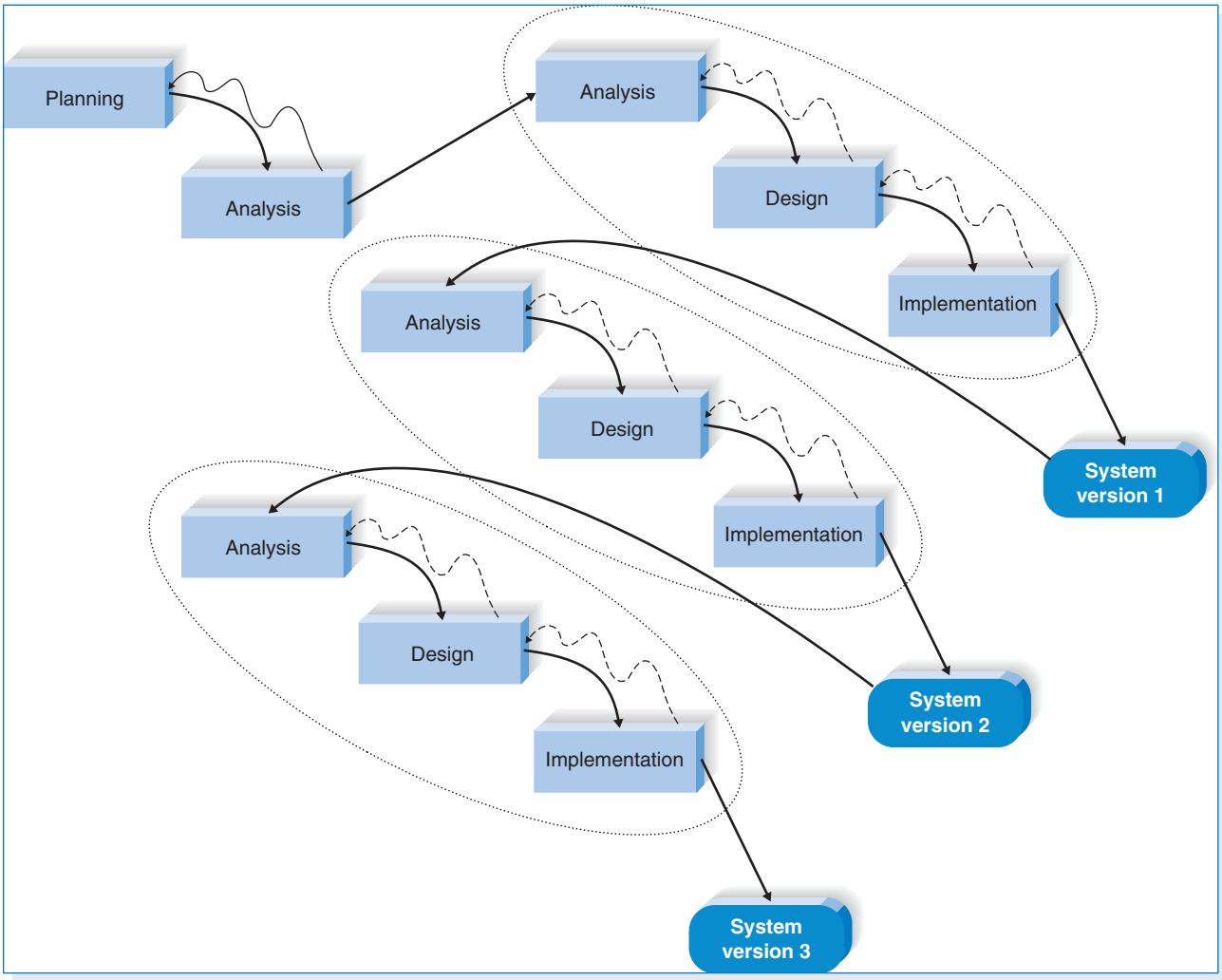

FIGURE 2-5
Iterative Development

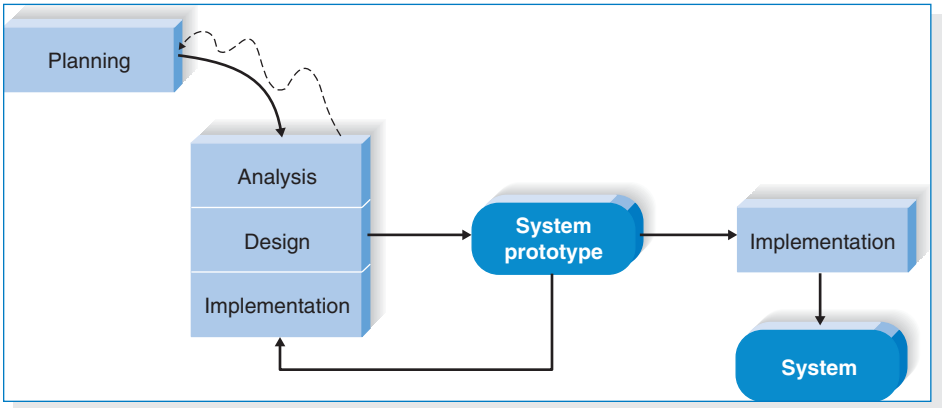

FIGURE 2-6
System Prototyping

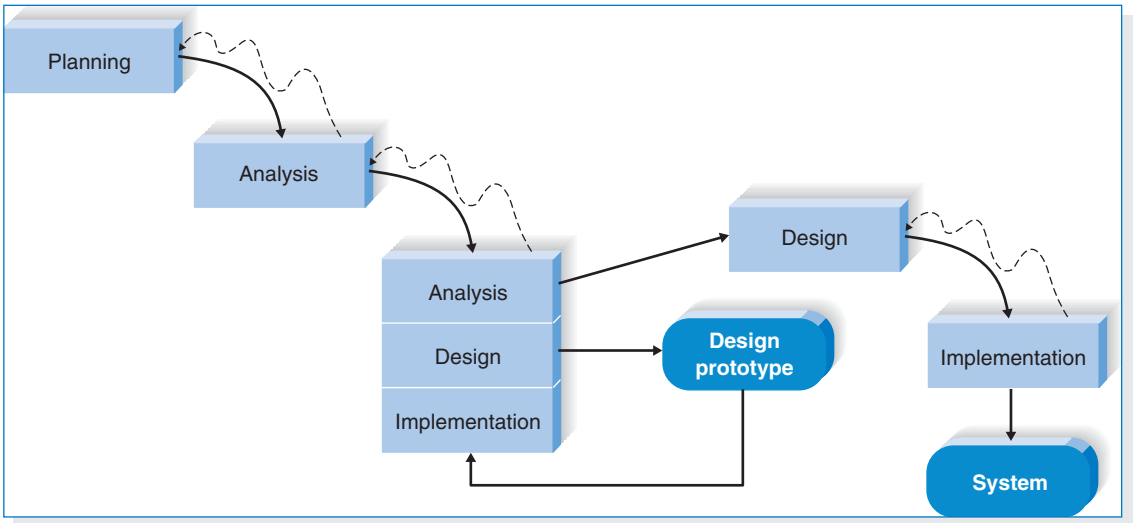

FIGURE 2-7
Throwaway Prototyping

Throwaway prototyping[6] includes the development of prototypes, but uses the prototypes primarily to explore design alternatives rather than as the actual new system (as in system prototyping). As shown in Figure 2-7, throwaway prototyping has a fairly thorough analysis phase that is used to gather requirements 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 intended to be a working system. It contains only enough detail to enable users to understand the issues under consideration.

For example, suppose that users are not completely clear on how an order entry system should work. The analyst team might build a series of HTML pages to be viewed on 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 by this type of methodology probably requires 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 system prototyping, in which the prototypes evolve into the final system.

Throwaway prototyping balances 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 compared with

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

system prototyping (because the prototypes do not become the final system), but the approach usually produces more stable and reliable systems.

Agile Development *Agile development*[7] is a group of programming-centric methodologies that focus on streamlining the SDLC. Much of the modeling and documentation overhead is eliminated; instead, face-to-face communication is preferred. A project emphasizes simple, iterative application development in which every iteration is a complete software project, including planning, requirements analysis, design, coding, testing, and documentation. (See Figure 2.8). Cycles are kept short (one to four weeks), and the development team focuses on adapting to the current business environment. There are several popular approaches to agile development, including extreme programming (XP)[8], Scrum[9], and dynamic systems development method (DSDM).[10] Here, we briefly describe extreme programming.

Extreme programming[11] emphasizes customer satisfaction and teamwork. Communication, simplicity, feedback, and courage are core values. Developers communicate with customers and fellow programmers. Designs are kept simple and clean. Early and frequent testing provides feedback, and developers are able to courageously respond to changing requirements and technology. Project teams are kept small.

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

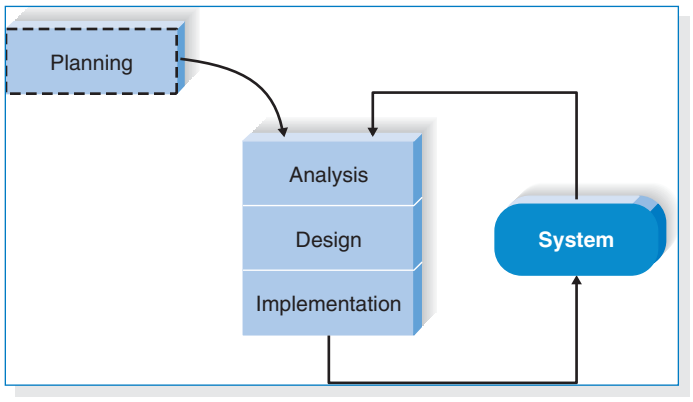

FIGURE 2-8
Extreme Programming

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

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

[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.

CONCEPTS IN ACTION

2-E Agile Development at Travelers

Travelers Insurance Company of Hartford, Connecticut has adopted agile development methodologies. The insurance field can be competitive, and Travelers wanted to have the shortest "time to implement" in the field. Travelers set up development teams of six people—two systems analysts, two representatives from the user group (such as claim services), a project manager, and a clerical support person. In the agile approach, the users are physically assigned to the development team for the project. While at first it might seem that the users are just sitting around drinking coffee and not doing their regular jobs, that is not the case. The rapport that is developed within the team allows for instant communication. The interaction is very deep and profound. The resulting software product is delivered quickly—and, generally, with all the features and nuances that the users wanted.

Questions:

1. Could this be done differently, such as through JAD sessions or having the users review the program on a weekly basis, rather than taking the users away from their real jobs to work on development?
2. What mind-set does an analyst need to work on such an approach?

jelled,[12] then the likelihood of success for the 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 recommended only for small groups of developers (not more than 10), and it is not advised for mission-critical applications. 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]

Agile versus Waterfall-Based Methodologies Agile development approaches have existed for over a decade. Agile development practices were created in part because of dissatisfaction with the sequential, inflexible structure of waterfall-based approaches. Presently, agile development has made inroads into software development organizations, and studies show an even split between agile and waterfall users.[15] Many organizations are experimenting with agile even while continuing to employ traditional waterfall approaches (see Concepts in Action 2F).

[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.

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

[15] Jan Stafford, "Agile and waterfall neck and neck as business side fails to engage." SearchSoftwareQuality.com, March 16, 2009.

Usefulness in Developing Systems	Waterfall	Parallel	V-Model	Iterative	System Prototyping	Throwaway Prototyping	Agile Development
with unclear user requirements	Poor	Poor	Poor	Good	Excellent	Excellent	Excellent
with unfamiliar technology	Poor	Poor	Poor	Good	Poor	Excellent	Poor
that are complex	Good	Good	Good	Good	Poor	Excellent	Poor
that are reliable	Good	Good	Excellent	Good	Poor	Excellent	Good
with short time schedule	Poor	Good	Poor	Excellent	Excellent	Good	Excellent
with schedule visibility	Poor	Poor	Poor	Excellent	Excellent	Good	Good

FIGURE 2-9
Criteria for Selecting a Methodology

In fact, suggesting that an organization must be "all agile" or "all waterfall" is a false choice. Many software developers are actively seeking to integrate the best elements of both waterfall and agile into their software development practices. Hybrid agile-waterfall approaches are evolving. The process of developing information systems is never static. Most IS departments and project managers recognize that the choice of the "best" development methodology depends on project characteristics, as we discuss in the next section.

Selecting the Appropriate Development Methodology

As the previous section shows, there are many methodologies. The first challenge faced by project managers 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 2-9 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 and agile development 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. System prototyping and throwaway prototyping are usually more appropriate when user requirements are unclear, because 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 Ajax), 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 is particularly appropriate for situations where there is a lack of familiarity with technology, because it explicitly encourages the developers to create design prototypes for areas with high risks. Iterative development is good as well, because opportunities are created to investigate the technology in some depth before the design is complete. While one might think that system prototyping would also be appropriate, it is much less so because the early prototypes that are built usually only scratch the surface of the new technology. Typically, 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 is particularly well suited to such detailed analysis and design, but system prototyping is not. The waterfall 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 iterative development 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). The V-model is useful when reliability is important, due to its emphasis on testing. Throwaway prototyping is most appropriate when system reliability is a high priority, because detailed analysis and design phases are combined with the ability for the project team to test many different approaches through design prototypes before completing the design. System prototyping is generally not a good choice when reliability is critical, due to the lack of careful analysis and design phases that are essential to dependable systems.

YOUR TURN

2-3 Selecting a Methodology

Suppose that 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 that ABC Company use? Why?

CONCEPTS IN ACTION

2-F Where Agile Works and Doesn't Work

British Airways experienced problems in software development despite a willing and capable development team. Mike Croucher, brought in as chief software engineer, recommended a move to agile development after studying BA's development process. The movement to agile was carefully conducted, recognizing that agile represents a huge cultural shift for the developers. BA development team members who were amenable to and suitable for agile methods were trained as agile mentors and coaches to help ease the transition.

Converting to agile methods enabled BA to substantially shorten the time requirements of certain projects. In some cases, a project that might have taken nine months following a traditional methodology was completed in eight weeks. Only about 25% of the organization has changed to agile, however. BA recognized a continuing role for the waterfall methodology in certain areas of the organization and does not intend to force-fit agile everywhere. At BA, agile is used when the user base is requiring speed, flexibility, and customer-oriented design. Agile is ideal when an area requiring small functionality can be developed and deployed earlier, according to Croucher.

Source: Mondelo, Daniel J. "Where agile development works and where it doesn't: A user story." SearchSoftwareQuality.com. February 24, 2010.

Short Time Schedules Projects that have short time schedules are well suited for RAD methodologies because those methodologies are designed to increase the speed of development. Iterative development and system prototyping are excellent choices when time lines 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 removal of the 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 waterfall-based methodologies because design and implementation occur at the end of the project. The RAD methodologies move many of the critical design decisions to a position earlier in the project to help project managers recognize and address risk factors and keep expectations in check.

Estimating the Project Time Frame

As the previous section illustrated, some development methodologies have evolved in an attempt to accelerate the project through the SDLC as rapidly as possible while still producing a quality system. Regardless of whether time is a critical issue on a project or not, the project manager will have to develop a preliminary estimate of the amount of time the project will take. *Estimation*[16] is the process of assigning projected values for time and effort.

[16] Good books for further reading on software estimation are T. Capers Jones, *Estimation Software Costs,* New York: McGraw Hill, 1989; Coombs, *IT Project Estimation: A Practical Guide to the Costing of Software,* Cambridge University Press, 2003; and Steve McConnell, *Software Estimation: Demystifying the Black Art,* Microsoft Press, 2006.