CHAPTER 1

Introduction to Systems Analysis and Design

Chapter 1 introduces the systems development life cycle (SDLC), the fundamental four-phase model (planning, analysis, design, and implementation) common to all information systems development projects. It describes the evolution of systems development methodologies and discusses the roles and skills required for a systems analyst. The chapter then overviews the basic characteristics of object-oriented systems and the fundamentals of object-oriented systems analysis and design and closes with a description of the Unified Process and its extensions and the Unified Modeling Language.

OBJECTIVES

- Be familiar with the different roles played by and the skills of a systems analyst.
- Understand the fundamental systems development life cycle and its four phases.
- Understand the evolution of systems development methodologies.
- Be familiar with the fundamental principles of object-oriented systems analysis and design.
- Be familiar with the Unified Process, its extensions, and the Unified Modeling Language.
- Be familiar with the basic characteristics of object-oriented systems.

INTRODUCTION

The *systems development life cycle (SDLC)* is the process of understanding how an information system (IS) can support business needs by designing a 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 1996 survey by the Standish Group found that 42 percent of all corporate IS projects were abandoned before completion. A similar study conducted in 1996 by the General Accounting Office found that 53 percent of all U.S. government IS projects were abandoned. Unfortunately, many of the systems that are not abandoned are delivered to the users significantly late, cost far more than planned, and have fewer features than originally planned. For example, IAG Consulting reports that 80 percent of the projects were over time, 72 percent were over budget, and 55 percent contained less than the full functionality; Panorama Consulting Solutions reports that 54 percent of the ERP projects were over time, 56 percent were over budget, and 48 percent delivered less than 50 percent of the initial benefits; and an IBM study reports that 59 percent of the projects missed one or more of on time, within budget, and quality constraints.[1] Although we would

[1] For more information on the problem, see Capers Jones, *Patterns of Software System Failure and Success* (London: International Thompson Computer Press, 1996); and Keith Ellis, *Business Analysis Benchmark: The Impact of Business Requirements on the Success of Technology Projects* (2008). Retrieved May 2014 from IAG Consulting, www.iag.biz; H. H. Jorgensen, L. Owen, and A. Neus, *Making Change Work* (2008). Retrieved May 2014 from IBM, www.ibm. com; Panorama Consulting Solutions, *2012 ERP Report* (2012). Retrieved May 2014 from Panorama-Consulting.com.

like to promote this book as a silver bullet that will keep you from IS failures, we readily admit that a silver bullet that guarantees IS development success simply does not exist. Instead, this book provides you with several fundamental concepts and many practical techniques that you can use to improve the probability of success.

The key person in developing a system 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. Systems analysts work with a variety of people and learn how they conduct business. Specifically, they work with a team of systems analysts, programmers, and others on a common mission. Systems analysts feel the satisfaction of seeing systems that they designed and developed make a significant business impact, knowing that they contributed unique skills to make that happen.

However, the primary objective of a systems analyst is not to create a wonderful system; instead, it 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 have been abandoned because the analysts tried to build a wonderful system without clearly understanding how the system would fit with an organization's goals, current business processes, and other information systems to provide value. An investment in an information system is like any other investment. 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 that it can earn greater profits or serve its constituents more effectively.

This book introduces the fundamental skills a systems analyst needs. This pragmatic book discusses the best practices in systems development; it does not present a general survey of systems development that covers 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 to your own systems development project the ideas and concepts in the examples. This book guides you through all the steps for delivering a successful information 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.

TYPICAL SYSTEMS ANALYST ROLES AND SKILLS

It is clear during systems development that 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. Understanding what to change and 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 technology that will make up the new system, and the way both can 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 levels, and they put their analytical skills to the test regularly.

Analysts often 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 analysts). 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 they also need to manage people with whom they work and they need to 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 associated with roles performed on a project. In the early days of systems development, most organizations expected one person, the analyst, to have all 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 containing several individuals with clearly defined responsibilities. Different organizations divide the roles differently. Most IS teams include many other individuals, such as the *programmers*, who actually write the programs that make up the system, and *technical writers*, who prepare the help screens and other documentation (e.g., user manuals and systems manuals).

Business Analyst

A *business analyst* focuses on the business issues surrounding the system. These issues 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 likely has business experience and some type of professional training. He or she represents the interests of the project sponsor and the ultimate users of the system. A business analyst assists in the planning and design phases but is most active during the analysis phase.

Systems Analyst

A *systems analyst* focuses on the IS issues surrounding the system. This person develops ideas and suggestions for how information technology 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. A systems analyst likely has 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 through the project but perhaps less so during the implementation phase.

Infrastructure Analyst

An *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). An 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 probably has 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 will ultimately have to operate and support the new system once it has been installed. An infrastructure analyst works throughout the project but perhaps less so during planning and analysis phases.

Change Management Analyst

A *change management analyst* focuses on the people and management issues surrounding the system installation. The roles of this person include ensuring that the 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 should 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. A change management analyst works most actively during the implementation phase but begins laying the groundwork for change during the analysis and design phases.

Project Manager

A *project manager* is responsible for ensuring that the project is completed on time and within budget and that the system delivers all benefits 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 has significant experience in project management and has probably 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.

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 last) 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 or where the telephone jacks will be placed). Finally, the house is built following the blueprints, often with some changes directed by the customer as the house is erected.

The systems development life cycle (SDLC) has a similar set of four fundamental *phases*: planning, analysis, design, and implementation. Different projects might 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 upon *techniques* that produce *deliverables* (specific documents and files that provide understanding about the project).

For example, in applying for admission to a university, all students go through the same phases: information gathering, applying, and accepting. Each of these phases has steps; for example, information gathering includes steps such as 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).

In many projects, the SDLC phases and steps proceed in a logical path from start to finish. In other projects, the project teams move through the steps consecutively, incrementally, iteratively, or in other patterns. In this section, we describe the phases, the actions, and some of the techniques that are used to accomplish the steps at a very high level.

For now, there are two important points to understand about the SDLC. First, you should get a general sense of the phases and steps through which IS projects move 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 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 (e.g., from the marketing department and accounting department) 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 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 project, 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*, which 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. During this phase, the project team investigates any current system(s), identifies opportunities for improvement, 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*, which describe how the business will operate if the new system is developed.
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) who 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." Most organizations

continue to use the name *analysis* for this phase, however, so we use it in this book as well. Just keep in mind 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; 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* is first developed. It clarifies whether the system will be developed by the company's own programmers, whether the system 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, which describes the hardware, software, and network infrastructure to 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.

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 that it performs as designed. Because the cost of bugs can be immense, testing is one of the most critical steps in implementation. Most organizations give more time and attention to testing than to 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. 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, based on the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, whereas others have been developed by consulting firms to sell to clients. Many organizations have internal methodologies that have been honed over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

There are many ways to categorize methodologies. One way is by looking at whether they focus on business processes or the data that support the business. A *process-centered methodology* emphasizes process models as the core of the system concept. In Figure 1-1, for example, process-centered methodologies would focus first on defining the processes (e.g., assemble sandwich ingredients). *Data-centered methodologies* emphasize data models as the core of the system concept. In Figure 1-1, data-centered methodologies would focus first on defining the contents of the storage areas (e.g., refrigerator) and how the contents were organized.[2] In contrast, *object-oriented methodologies* attempt to balance the focus between process and data by incorporating both into one model. In Figure 1-1, these methodologies would focus first on defining the major elements of the system (e.g., sandwiches, lunches) and look at the processes and data involved with each element.

Another important factor in categorizing methodologies is the sequencing of the SDLC phases and the amount of time and effort devoted to each.[3] In the early days of computing, programmers did not understand the need for formal and well-planned life-cycle methodologies. They tended to move directly from a very simple planning phase right into the construction step of the implementation phase—in other words, from a very fuzzy, not-well-thought-out system request into writing code. This is the same approach that you 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 might 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 this section, we describe a set of different classes of systems development methodologies: structured design, rapid application development, object-oriented systems analysis and design, agile development, DevOps, and custom methodologies.

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. Numerous process-centered and data-centered methodologies follow the basic approach of the two structured design categories outlined next.

[2] 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; see James Martin, *Information Engineering,* vols. 1–3 (Englewood Cliffs, NJ: Prentice Hall, 1989). A widely accepted standardized non-object-oriented methodology that balances processes and data is IDEF; see FIPS 183, *Integration Definition for Function Modeling* (Federal Information Processing Standards Publications, U.S. Department of Commerce, 1993).

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

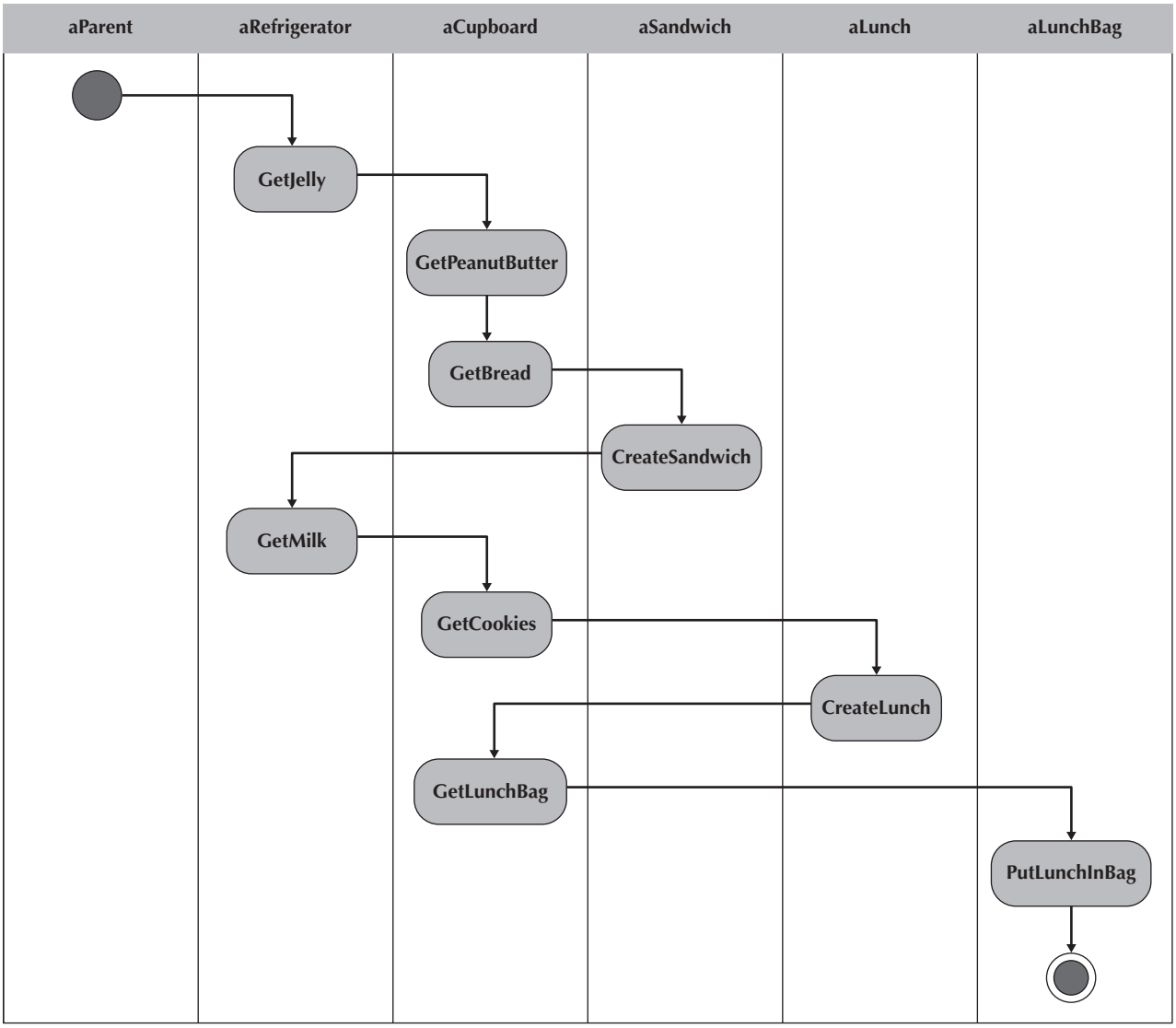

FIGURE 1-1 A Simple Behavioral Model for Making a Simple Lunch

Waterfall Development The original structured design methodology (still used today) is *waterfall development*. With waterfall development-based methodologies, the analysts and users proceed in sequence from one phase to the next. The key deliverables for each phase are typically very long (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 referred to as 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 against a waterfall).

Structured design also introduced the use of formal modeling or diagramming techniques to describe the basic business processes and the data that support them. Traditional

structured design uses one set of diagrams to represent the processes and a separate set of diagrams to represent data. Because two sets of diagrams are used, the systems analyst must decide which set to develop first and use as the core of the system: process-model diagrams or data-model diagrams.

The two key advantages of the structured design waterfall approach are that it identifies system requirements long before programming begins and it minimizes changes to the requirements as the project proceeds. The two key disadvantages are that the design must be completely specified before programming 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). 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 able to remember interior lights that come on when the doors open or to specify the right number of valves on the engine?). A system can also require significant rework because the business environment has changed from the time when the analysis phase occurred.

Parallel Development *Parallel development* methodology attempts to address the problem of long delays between the analysis phase and the delivery of the system. Instead of doing design and implementation in sequence, it performs a general design for the whole system and then divides the project into a series of distinct subprojects that can be designed and implemented in parallel. Once all subprojects are complete, the separate pieces are integrated and the system is delivered.

The primary advantage of this methodology is that it can reduce the time to deliver a system; thus, there is less chance of changes in the business environment causing rework. However, sometimes the subprojects are not completely independent; design decisions made in one subproject can affect another, and the end of the project can require significant integration efforts.

Rapid Application Development (RAD)

A second category of methodologies includes *rapid application development (RAD)*-based methodologies. These are a newer class of systems development methodologies that emerged in the 1990s. RAD-based methodologies attempt to address both weaknesses of structured design methodologies by adjusting 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 closer to what is needed.[4]

Most RAD-based methodologies recommend that analysts use special techniques and computer tools to speed up the analysis, design, and implementation phases, such as computer-aided software engineering (CASE) tools, joint application design (JAD) sessions, fourth-generation or visual programming languages that simplify and speed up programming, and code generators that automatically produce programs from design specifications. The combination of the changed SDLC phases and the use of these tools and techniques improves the speed and quality of systems development. However, there is one possible subtle problem with RAD-based methodologies: managing user expectations. Owing to the use of the tools and techniques that can improve the speed and quality of systems development, user expectations of what is possible can change dramatically. As a user better understands the information technology (IT), the systems requirements tend to expand. This was less of a problem when using methodologies that spent a lot of time thoroughly documenting requirements.

[4] One of the best RAD books is Steve McConnell, *Rapid Development* (Redmond, WA: Microsoft Press, 1996).

Phased Development A *phased development*-based methodology breaks an 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.

Once version 1 is implemented, work begins on version 2. Additional analysis is performed based on the previously identified requirements and combined with new ideas and issues that arose from the 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 the system does not perform all the functions the users need at first, it does begin to provide business value sooner than if the system were delivered after completion, as is the case with the waterfall and 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 of 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 and to manage users' expectations along the way.

Prototyping A *prototyping*-based methodology performs 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, the basics of 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 is used. This is shown to the users and the project sponsor, who provide comments. These comments are used to reanalyze, redesign, and reimplement a second prototype, which 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.

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

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 become 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 it had been driven 10,000 miles).

Throwaway Prototyping *Throwaway prototyping*-based methodologies are similar to 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 those previously discussed, and they have a very different appearance.

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. However, users might not completely understand many of the features they suggest, 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 enable users to understand the issues under consideration. For example, suppose users are not completely clear on how an order-entry system should work. 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 pretend data to ensure that they could do a full-blown program successfully.

A system developed using this type of methodology relies on several design prototypes during the analysis and design phases. Each of these 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 these methodologies and prototyping methodologies, 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 can take longer to deliver the final system as compared to prototyping-based methodologies, but this type of methodology usually produces more stable and reliable systems.

Object-Oriented Systems Analysis and Design (OOSAD)

A third category of information systems development methodologies is object-oriented analysis and design. Object-oriented approaches to developing information systems, technically speaking, can use any of the traditional methodologies. However, the object-oriented approaches are most associated with a phased development RAD. The primary difference between a traditional approach like structured design and an object-oriented approach is how a problem is decomposed. In traditional approaches, the problem-decomposition process is either process-centric or data-centric. However, processes and data are so closely related that it is difficult to pick one or the other as the primary focus. Based on this lack of congruence with the real world, new *object-oriented methodologies* have emerged that use the RAD-based sequence of SDLC phases but attempt to balance the emphasis between process and data by focusing the analysis of problems on objects that contain both data and processes.

According to the creators of the Unified Modeling Language (UML), Grady Booch, Ivar Jacobson, and James Rumbaugh,[5] any modern object-oriented approach to developing information systems must be use-case driven, architecture-centric, and iterative and incremental.

Use-Case Driven *Use-case driven* means that *use cases* are the primary modeling tools defining the behavior of the system. A use case describes how the user interacts with the system to perform some activity, such as placing an order, making a reservation, or searching for information. The use cases are used to identify and to communicate the requirements of the system to the programmers who must write the system. Use cases are inherently simple because they focus on only one business process at a time. In contrast, the process model diagrams used by

[5] Grady Booch, Ivar Jacobson, and James Rumbaugh, *The Unified Modeling Language User Guide* (Reading, MA: Addison-Wesley, 1999).

traditional structured and RAD methodologies are far more complex because they require the systems analyst and user to develop models of the entire system. With traditional methodologies, each system is decomposed into a set of subsystems, which are, in turn, decomposed into further subsystems, and so on. This goes on until no further process decomposition makes sense, and it often requires dozens of pages of interlocking diagrams. In contrast, a use case focuses on only one business process at a time, so developing models is much simpler.[6]

Architecture-Centric Any modern approach to systems analysis and design should be architecture-centric. *Architecture-centric* means that the underlying software architecture of the evolving system specification drives the specification, construction, and documentation of the system. Modern object-oriented systems analysis and design approaches should support at least three separate but interrelated architectural views of a system: functional, static, and dynamic. The *functional*, or *external, view* describes the behavior of the system from the perspective of the user. The *structural*, or *static, view* describes the system in terms of attributes, methods, classes, and relationships. The *behavioral*, or *dynamic, view* describes the behavior of the system in terms of messages passed among objects and state changes within an object.

Iterative and Incremental Modern object-oriented systems analysis and design approaches emphasize *iterative* and *incremental* development that undergoes continuous testing and refinement throughout the life of the project. This implies that the systems analysts develop their understanding of a user's problem by building up the three architectural views little by little. The systems analyst does this by working with the user to create a functional representation of the system under study. Next, the analyst attempts to build a structural representation of the evolving system. Using the structural representation of the system, the analyst distributes the functionality of the system over the evolving structure to create a behavioral representation of the evolving system. As an analyst works with the user in developing the three architectural views of the evolving system, the analyst iterates over each of and among the views. That is, as the analyst better understands the structural and *behavioral views,* the analyst uncovers missing requirements or misrepresentations in the functional view. This, in turn, can cause changes to be cascaded back through the structural and behavioral views. All three architectural views of the system are interlinked and dependent on each other (see Figure 1-2). As each increment and iteration is completed, a more-complete representation of the user's real functional requirements is uncovered.

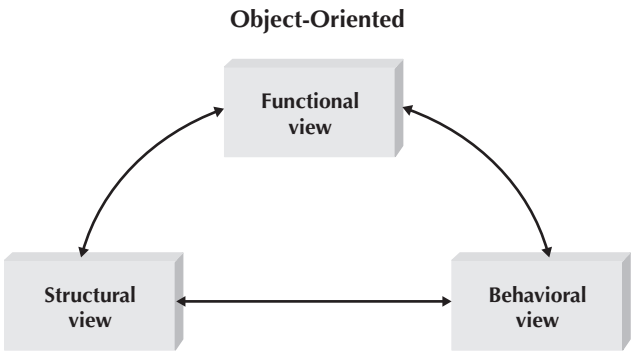

FIGURE 1-2 Iterative and Incremental Development

[6] For those of you who have experience with traditional structured analysis and design, this is one of the most unusual aspects of object-oriented analysis and design using UML. Unlike structured approaches, object-oriented approaches stress focusing on just one use case at a time and distributing that single use case over a set of communicating and collaborating objects.

Benefits of Object-Oriented Systems Analysis and Design Concepts in the object-oriented approach enable analysts to break a complex system into smaller, more-manageable modules, work on the modules individually, and easily piece the modules back together to form an information system. This modularity makes systems development easier to grasp, easier to share among members of a project team, and easier to communicate to users, who are needed to provide requirements and confirm how well the system meets the requirements throughout the systems development process. By modularizing systems development, the project team actually is creating reusable pieces that can be plugged into other systems efforts or used as starting points for other projects. Ultimately, this can save time because new projects don't have to start completely from scratch.

Agile Development[7]

A fourth category of systems development methodologies is *agile development*. All agile development methodologies are based on the agile manifesto and a set of twelve principles. The emphasis of the manifesto is to focus the developers on the working conditions of the developers, the working software, the customers, and addressing changing requirements instead of focusing on detailed systems development processes, tools, all-inclusive documentation, legal contracts, and detailed plans. These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. These methodologies are typically based only on the twelve principles of agile software. These principles include the following:

- Software is delivered early and continuously through the development process, satisfying the customer.
- Changing requirements are embraced regardless of when they occur in the development process.
- Working software is delivered frequently to the customer.
- Customers and developers work together to solve the business problem.
- Motivated individuals create solutions; provide them the tools and environment they need, and trust them to deliver.
- Face-to-face communication within the development team is the most efficient and effective method of gathering requirements.
- The primary measure of progress is working, executing software.
- Both customers and developers should work at a pace that is sustainable. That is, the level of work could be maintained indefinitely without any worker burnout.
- Agility is heightened through attention to both technical excellence and good design.
- Simplicity, the avoidance of unnecessary work, is essential.
- Self-organizing teams develop the best architectures, requirements, and designs.
- Development teams regularly reflect on how to improve their development processes.

Based on these principles, agile methodologies focus on streamlining the systems development process by eliminating much of the modeling and documentation overhead and the time spent on those tasks. Instead, projects emphasize simple, iterative application development.[8] All agile development methodologies follow a simple cycle through the traditional phases of

[7] Three good sources of information on agile development and object-oriented systems are S. W. Ambler, *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process* (New York: Wiley, 2002); C. Larman, *Agile & Iterative Development: A Manager's Guide* (Boston: Addison-Wesley, 2004); R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices* (Upper Saddle River, NJ: Prentice Hall, 2003).

[8] See Agile Alliance, www.agilealliance.com.

the systems development process. Virtually all agile methodologies are used in conjunction with object-oriented technologies.

However, agile methodologies do have critics. One of the major criticisms deals with today's business environment, where much of the actual information systems development is offshored, outsourced, and/or subcontracted. Given agile development methodologies requiring co-location of the development team, this seems to be a very unrealistic assumption. A second major criticism is that if agile development is not carefully managed, and by definition it is not, the development process can devolve into a prototyping approach that essentially becomes a "programmers gone wild" environment where programmers attempt to hack together solutions. A third major criticism, based on the lack of actual documentation created during the development of the software, raises issues regarding the auditability of the systems being created. Without sufficient documentation, neither the system nor the systems-development process can be assured. A fourth major criticism is based on whether agile approaches can deliver large mission-critical systems.

Even with these criticisms, given the potential for agile approaches to address the application backlog and to provide timely solutions to many business problems, agile approaches should be considered in some circumstances. Furthermore, many of the techniques encouraged by attending to the underlying purpose of the agile manifesto and the set of twelve agile principles are very useful in object-oriented systems development. Two of the more popular examples of agile development methodologies are extreme programming (XP) and Scrum.

Extreme Programming[9] *Extreme programming (XP)* is founded on four core values: communication, simplicity, feedback, and courage. These four values provide a foundation that 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 principle.[10] Third, developers must make incremental changes to grow the system, and they must not only accept change, they must embrace change. 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.

Testing and efficient coding practices are the core of XP. Code is tested each day and is placed into an integrative testing environment. If bugs exist, the code is backed out until it is completely free of errors.

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.

XP adherents claim many strengths associated with developing software using XP. Programmers work closely with all stakeholders, and communication among all stakeholders is improved. Continuous testing of the evolving system is encouraged. The system is developed in an evolutionary and incremental manner, which allows the requirements to evolve as the stakeholders understand the potential that the technology has in providing a solution to

[9] For more information, see K. Beck, *eXtreme Programming Explained: Embrace Change* (Reading, MA: Addison-Wesley, 2000); C. Larman, *Agile & Iterative Development: A Manager's Guide* (Boston: Addison-Wesley, 2004); M. Lippert, S. Roock, and H. Wolf, *eXtreme Programming in Action: Practical Experiences from Real World Projects* (New York: Wiley, 2002); www.extremeprogramming.org.

[10] Keep it simple, stupid.

their problem. Estimation is task driven and is performed by the programmer who will implement the solution for the task under consideration. Because all programming is done in pairs, a shared responsibility for each software component develops among the programmers. Finally, the quality of the final product increases during each iteration.

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,[11] the success of an XP development effort is doubtful. This tends to throw into doubt the whole idea of bringing outside contractors into an existing team environment using XP.[12] The chance of outsiders jelling with insiders might simply be too optimistic. XP requires a great deal of discipline, otherwise projects will become unfocused and chaotic. XP is recommended only for small groups of developers—no more than ten developers—and it is not advised for large mission-critical applications. Owing to the lack of analysis and design documentation, there is only code documentation associated with XP, so maintaining large systems built with XP may be impossible. And because mission-critical business information systems tend to exist for a long time, the utility of XP as a business information systems development methodology is in doubt. Finally, the methodology needs a lot of on-site user input, something to which many business units cannot commit.[13] However, some of the techniques associated with XP are useful in object-oriented systems development. For example, user stories, pair programming, and continuous testing are invaluable tools from which object-oriented systems development could benefit.

Scrum[14] *Scrum* is a term that is well known to rugby fans. In rugby, a scrum is used to restart a game. In a nutshell, the creators of the Scrum method believe that no matter how much you plan, as soon as the software begins to be developed, chaos breaks out and the plans go out the window.[15] The best you can do is to react to where the proverbial rugby ball squirts out. You then sprint with the ball until the next scrum. In the case of the Scrum methodology, a sprint lasts thirty working days. At the end of the sprint, a system is delivered to the customer.

Of all systems development approaches, on the surface, Scrum is the most chaotic. To control some of the innate chaos, Scrum development focuses on a few key practices. Teams are self-organized and self-directed. Unlike other approaches, Scrum teams do not have a designated team leader. Instead, teams organize themselves in a symbiotic manner and set their own goals for each sprint (iteration). Once a sprint has begun, Scrum teams do not consider any additional requirements. Any new requirements that are uncovered are placed on a backlog of requirements that still need to be addressed. At the beginning of every workday, a Scrum meeting takes place. At the end of each sprint, the team demonstrates the software to the client. Based on the results of the sprint, a new plan is begun for the next sprint.

[11] 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: Dorset/House, 1987).

[12] 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); S. W. Ambler, "Chicken Little Was Right," *Software Development* (October 2003).

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

[14] For more information, see C. Larman, *Agile & Iterative Development: A Manager's Guide* (Boston: Addison-Wesley, 2004); K. Schwaber and M. Beedle, *Agile Software Development with Scrum* (Upper Saddle River, NJ: Prentice Hall, 2001); R. Wysocki, *Effective Project Management: Traditional, Agile, Extreme*, 5th Ed. (Indianapolis, IN: Wiley Publishing, 2009).

[15] Scrum developers are not the first to question the use of plans. One of President Eisenhower's favorite maxims was, "In preparing for battle I have always found that plans are useless, but planning is indispensable." M. Dobson, *Streetwise Project Management: How to Manage People, Processes, and Time to Achieve the Results You Need* (Avon, MA: F+W Publications, 2003), p. 43.

Scrum meetings are one of the most interesting aspects of the Scrum development process. The team members attend the meetings, but anyone can attend. However, with very few exceptions, only team members may speak. One prominent exception is management providing feedback on the business relevance of the work being performed by a specific team. In this meeting, all team members stand in a circle and report on what they accomplished during the previous day, state what they plan to do today, and describe anything that blocked progress the previous day. To enable continuous progress, any block identified is dealt with within one hour. From a Scrum point of view, it is better to make a "bad" decision about a block at this point in development than to not make a decision. Because the meetings take place each day, a bad decision can easily be undone. Larman[16] suggests that each team member should report any additional requirements that have been uncovered during the sprint and anything that the team member learned that could be useful for other team members to know.

One of the major criticisms of Scrum, as with all agile methodologies, is that it is questionable whether Scrum can scale up to develop very large mission-critical systems. A typical Scrum team size is no more than seven members. The only organizing principle put forth by Scrum followers to address this criticism is to organize a scrum of scrums. Each team meets every day, and after the team meeting takes place, a representative (not leader) of each team attends a scrum-of-scrums meeting. This continues until the progress of entire system has been determined. Depending on the number of teams involved, this approach to managing a large project is doubtful. However, as in XP and other agile development approaches, many of the ideas and techniques associated with Scrum development are useful in object-oriented systems development, such as the focus of a Scrum meeting, the evolutionary and incremental approach to identifying requirements, and the incremental and iterative approach to the development of the system.

DevOps

Another category of systems development methodologies is DevOps.[17] *DevOps* approaches incorporate ideas from both object-oriented approaches to systems development and lean manufacturing into the agile approaches. Two of the more popular approaches are DAD[18] and SAFe.[19] The primary extensions that DevOps approaches support are the inclusion of operations personnel with the development team and continuous deployment.

In earlier approaches, end users were identified as very important stakeholders who should be included in the development team. However, new technology developers realized that including end users, even though necessary, was not sufficient to deliver high quality systems. For example, with cloud technologies concerns related to deployment of the system should be addressed before the development team "throws the system over the wall" to the operations personnel. Consequently, DevOps approaches include operations personnel as members of the development team. In this way, nonfunctional requirements, such as security, can be addressed during the design of the system to be deployed.

[16] C. Larman, *Agile & Iterative Development: A Manager's Guide* (Boston: Addison-Wesley, 2004).

[17] For more information, see Gene Kim, Jez Humble, Patrick Debois, and John Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations* (Portland, OR: IT Revolution Press, 2016); Nicole Forsgren, Jex Humble, and Gene Kim, *The Science Behind DevOps, Accelerate: Building and Scaling High Performing Technology Organizations* (Portland, OR: IT Revolution Press, 2018).

[18] For more information, see Scott W. Ambler and Mark Lines, *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise* (Upper Saddle River, NJ: IBM Press, 2012).

[19] For more information, see Richard Knaster and Dean Leffingwell, *SAFe® Distilled: Applying the Scaled Agile Framework® for Lean Software and Systems Engineering* (Boston, MA: Addison-Wesley, 2017); Dean Leffingwell with Alex Yakyma, Richard Knaster, Drew Jemilo, and Inbar Oren, *SAFe® Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering* (Boston, MA: Addison-Wesley, 2017).

With both object-oriented and agile approaches, deployment was only performed after a version of the system was completed. In other words, the operations personnel took over the system and deployed it. With DevOps approaches, deployment is completed when a new chunk of software has passed a set of relevant tests. Consequently, deployment of a new version of the system is not performed all at once. Instead, the new version is deployed in pieces over time; this is also referred to as a rolling upgrade. This approach requires quite a bit of automation of the actual deployment pipeline.[20]

Custom Methodologies

This last category of systems development methodologies supports the idea of applying both agile and object-oriented ideas to the systems development methodology itself. The idea behind these approaches is to support the development team in a way that is customized to their specific systems development project. In this case, only a high-level framework is used to create the project-specific method. Two of the approaches that have made progress in this area are based on the work of Brian Henderson-Sellers[21] and of Ivar Jacobson.[22] Both of these groups allow the team to pick and choose different components of a methodology from a component repository to create a custom methodology. Care must be taken to ensure that each component is plug compatible. By that we mean the inputs into a component are compatible with the outputs of the preceding component in the methodology.

THE UNIFIED PROCESS

The Unified Process is a specific methodology that maps out when and how to use the various Unified Modeling Language (UML) techniques for object-oriented analysis and design. The primary contributors were Grady Booch, Ivar Jacobson, and James Rumbaugh. Whereas the UML provides structural support for developing the structure and behavior of an information system, the Unified Process provides the behavioral support. The Unified Process, of course, is use-case driven, architecture-centric, and iterative and incremental. Furthermore, the Unified Process is a two-dimensional systems development process described by a set of phases and workflows. The phases are inception, elaboration, construction, and transition. The workflows include business modeling, requirements, analysis, design, implementation, test, deployment, configuration and change management, project management, and environment.[23] Figure 1-3 depicts the Unified Process.

[20] A good overview of these approaches is described in Len Bass, Ingo Weber, and Liming Zhu, *DevOps: A Software Architect's Perspective* (Upper Saddle River, NJ: Pearson Education, 2015).

[21] For more information, see Ian Graham, Brian Henderson-Sellers, and Houman Younessi, *The OPEN Process Specification* (New York, NY: ACM Press, 1997); Brian Henderson-Sellers, Anthony Simons, and Houman Younessi, *The OPEN Toolbox of Techniques* (New York, NY: ACM Press, 1998); Donald G. Firesmith and Brian Henderson-Sellers, *The OPEN Process Framework* (London, Addison-Wesley, 2002); Brian Henderson-Seller, Jolita Ralyte', Par J. Agerfalk, and Matti Rossi, *Situational Method Engineering* (Berlin: Springer, 2014).

[22] For more information, see Ivar Jaconson, Pan-Wei Ng, Paul E. McMahon, Ian Spence, and Svante Lidman, *The ESSENCE of Software Engineering: Applying the SEMAT Kernel* (Upper Saddle River, NJ: Addison-Wesley, 2013); Ivar Jacobson, Harold "Bud" Lawson, Pan-Wei Ng, Paul E. McMahon, and Michael Goedicke, *The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons!* (ACM Books, 2019).

[23] The material in this section is based on Khawar Zaman Ahmed and Cary E. Umrysh, *Developing Enterprise Java Applications with J2EE and UML* (Boston, MA: Addison-Wesley, 2002); Jim Arlow and Ila Neustadt, *UML and the Unified Process: Practical Object-Oriented Analysis & Design* (Boston, MA: Addison-Wesley, 2002); Peter Eeles, Kelli Houston, and Wojtek Kozacynski, *Building J2EE Applications with the Rational Unified Process* (Boston, MA: Addison-Wesley, 2003); Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process* (Reading, MA: Addison-Wesley, 1999); Philippe Kruchten, *The Rational Unified Process: An Introduction,* 2nd Ed. (Boston, MA: Addison-Wesley, 2000); "Rational Unified Process: Best Practices for Software Development Teams," Rational Software White Paper, TP026B, Rev 11/01.

Phases

The *phases* of the Unified Process support an analyst in developing information systems in an iterative and incremental manner. The phases describe how an information system evolves through time. Depending on which development phase the evolving system is currently in, the level of activity varies over the *workflows*. The curve in Figure 1-3 associated with each workflow approximates the amount of activity that takes place during the specific phase. For example, the inception phase primarily involves the business modeling and requirements workflows, while practically ignoring the test and deployment workflows. Each phase contains a set of iterations, and each iteration uses the various workflows to create an incremental version of the evolving system. As the system evolves through the phases, it improves and becomes more complete. Each phase has objectives, a focus of activity over the workflows, and incremental deliverables. Each of the phases is described next.

Inception In many ways, the *inception phase* is very similar to the planning phase of a traditional SDLC approach. In this phase, a business case is made for the proposed system. This includes feasibility analysis that should answer questions such as the following:

Do we have the technical capability to build it (technical feasibility)?

If we build it, will it provide business value (economic feasibility)?

If we build it, will it be used by the organization (organizational feasibility)?

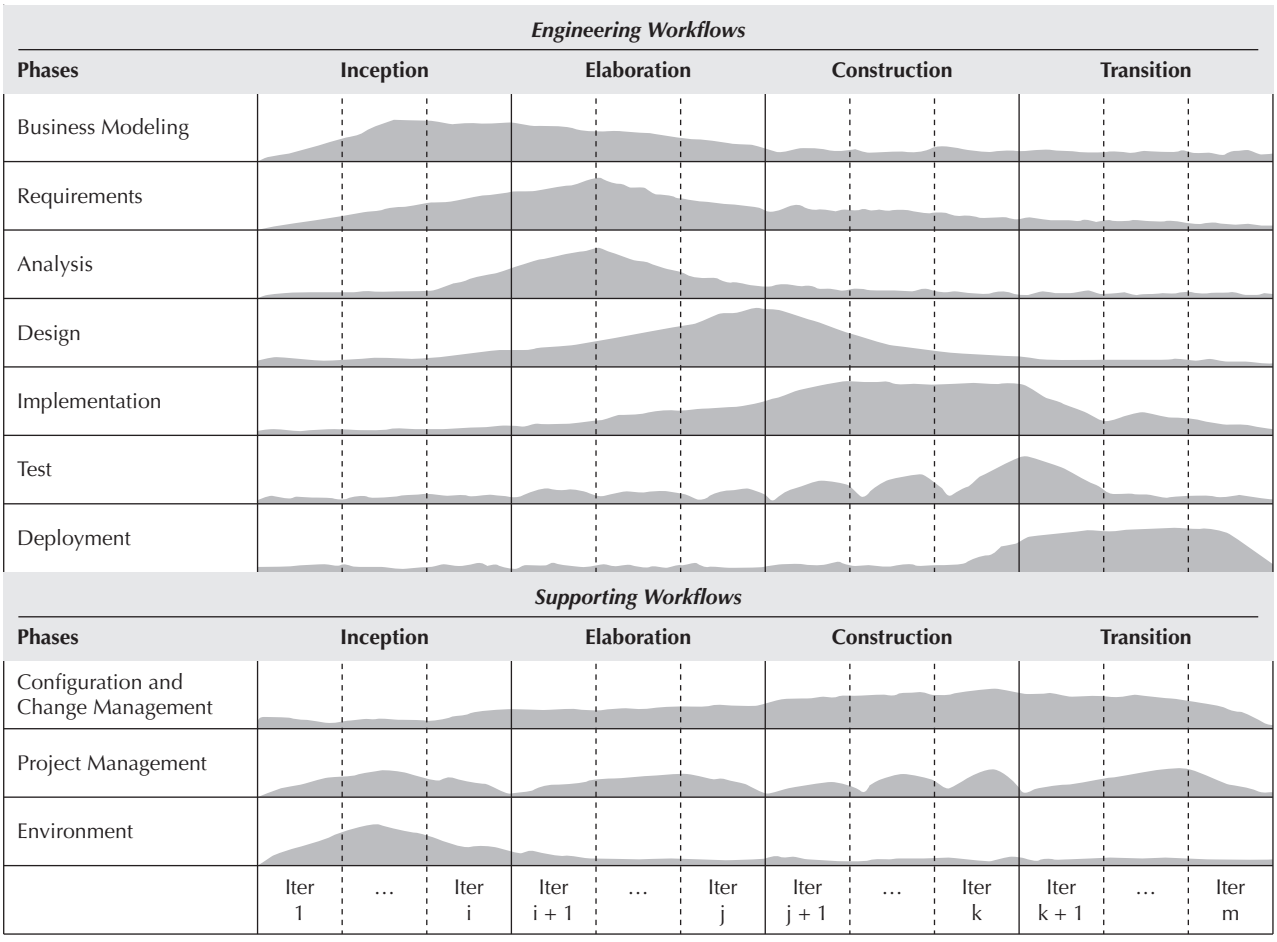

FIGURE 1-3 The Unified Process

To answer these questions, the development team performs work related primarily to the business modeling, requirements, and analysis workflows. In some cases, depending on the technical difficulties that could be encountered during the development of the system, a throwaway prototype is developed. This implies that the design, implementation, and test workflows could also be involved. The project management and environment supporting workflows are very relevant to this phase. The primary deliverables from the inception phase are a vision document that sets the scope of the project; identifies the primary requirements and constraints; sets up an initial project plan; and describes the feasibility of and risks associated with the project, the adoption of the necessary environment to develop the system, and some aspects of the problem domain classes being implemented and tested.

Elaboration When we typically think about object-oriented systems analysis and design, the activities related to the *elaboration phase* of the Unified Process are the most relevant. The *analysis* and *design workflows* are the primary focus during this phase. The elaboration phase continues with developing the vision document, including finalizing the business case, revising the risk assessment, and completing a project plan in sufficient detail to allow the stakeholders to be able to agree with constructing the actual final system. It deals with gathering the requirements, building the UML structural and behavioral models of the problem domain, and detailing how the problem domain models fit into the evolving system architecture. Developers are involved with all but the *deployment engineering workflow* in this phase. As the developers iterate over the workflows, the importance of addressing configuration and change management becomes apparent. Also, the development tools acquired during the inception phase become critical to the success of the project during this phase.[24] The primary deliverables of this phase include the UML structure and behavior diagrams and an executable of a baseline version of the evolving information system. The baseline version serves as the foundation for all later iterations. By providing a solid foundation at this point, the developers have a basis for completing the system in the construction and transition phases.

Construction The *construction phase* focuses heavily on programming the evolving information system. This phase is primarily concerned with the *implementation workflow*. However, the *requirements workflow* and the *analysis* and *design* workflows also are involved with this phase. It is during this phase that missing requirements are identified and the analysis and design models are finally completed. Typically, there are iterations of the workflows during this phase, and during the last iteration, the deployment workflow kicks into high gear. The *configuration and change management workflow*, with its version-control activities, becomes extremely important during the construction phase. At times, an iteration has to be rolled back. Without good version controls, rolling back to a previous version (incremental implementation) of the system is nearly impossible. The primary deliverable of this phase is an implementation of the system that can be released for beta and acceptance testing.

Transition Like the construction phase, the *transition phase* addresses aspects typically associated with the implementation phase of a traditional SDLC approach. Its primary focus is on the testing and deployment workflows. Essentially, the business modeling, requirements, and analysis workflows should have been completed in earlier iterations of the evolving information system. Furthermore, the testing workflow should have been executed during the earlier phases of the evolving system. Depending on the results from the testing workflow, some redesign and programming activities on the design and implementation workflows could be

[24] With UML comprising fifteen different, related diagramming techniques, keeping the diagrams coordinated and the different versions of the evolving system synchronized is typically beyond the capabilities of a mere mortal systems developer. These tools typically include project management and CASE tools. We describe the use of these tools in Chapter 2.

necessary, but they should be minimal at this point. From a managerial perspective, the project management, configuration and change management, and environment are involved. Some of the activities that take place are beta and acceptance testing, fine-tuning the design and implementation, user training, and rolling out the final product onto a production platform. Obviously, the primary deliverable is the actual executable information system. The other deliverables include user manuals, a plan to support the users, and a plan for upgrading the information system in the future.

Workflows

The workflows describe the tasks or activities that a developer performs to evolve an information system over time. The workflows of the Unified Process are grouped into two broad categories: engineering and supporting.

Engineering Workflows Engineering workflows include business-modeling, requirements, analysis, design, implementation, test, and deployment workflows. The engineering workflows deal with the activities that produce the technical product (i.e., the information system).

Business Modeling Workflow The *business modeling workflow* uncovers problems and identifies potential projects within a user organization. This workflow aids management in understanding the scope of the projects that can improve the efficiency and effectiveness of a user organization. The primary purpose of business modeling is to ensure that both developer and user organizations understand where and how the to-be-developed information system fits into the business processes of the user organization. This workflow is primarily executed during the inception phase to ensure that we develop information systems that make business sense. The activities that take place on this workflow are most closely associated with the planning phase of the traditional SDLC; however, requirements gathering, and use-case and business process modeling techniques also help us to understand the business situation.

Requirements Workflow In the Unified Process, the *requirements workflow* includes eliciting both functional and nonfunctional requirements. Typically, requirements are gathered from project stakeholders, such as end users, managers within the end user organization, and even customers. The requirements workflow is used the most during the inception and elaboration phases. The identified requirements are very helpful for developing the vision document and the use cases used throughout the development process. Additional requirements tend to be discovered throughout the development process. In fact, only the transition phase tends to have few, if any, additional requirements identified.

Analysis Workflow The *analysis workflow* primarily addresses the creation of an analysis model of the problem domain. In the Unified Process, the analyst begins designing the architecture associated with the problem domain; using the UML, the analyst creates structural and behavior diagrams that depict a description of the problem domain classes and their interactions. The primary purpose of the analysis workflow is to ensure that both the developer and user organizations understand the underlying problem and its domain without overanalyzing. If they are not careful, analysts can create *analysis paralysis*, which occurs when the project becomes so bogged down with analysis that the system is never actually designed or implemented. A second purpose of the analysis workflow is to identify useful reusable classes for class libraries. By reusing predefined classes, the analyst can avoid reinventing the wheel when creating the structural and behavior diagrams. The analysis workflow is predominantly associated with the elaboration phase, but like the requirements workflow, it is possible that additional analysis will be required throughout the development process.

Design Workflow The design workflow transitions the analysis model into a form that can be used to implement the system: the *design model*. Whereas the analysis workflow concentrates on understanding the problem domain, the design workflow focuses on developing a solution that will execute in a specific environment. Basically, the design workflow simply enhances the description of the evolving system by adding classes that address the environment of the system to the evolving analysis model. The design workflow uses activities such as detailed problem domain class design, optimization of the evolving information system, database design, user-interface design, and physical architecture design. The design workflow is associated primarily with the elaboration and construction phases of the Unified Process.

Implementation Workflow The primary purpose of the implementation workflow is to create an executable solution based on the design model (i.e., programming). This includes not only writing new classes but also incorporating reusable classes from executable class libraries into the evolving solution. As with any programming activity, the new classes and their interactions with the incorporated reusable classes must be tested. Finally, in the case of multiple groups performing the implementation of the information system, the implementers also must integrate the separate, individually tested modules to create an executable version of the system. The implementation workflow is associated primarily with the elaboration and construction phases.

Testing Workflow The primary purpose of the *testing workflow* is to increase the quality of the evolving system. Testing goes beyond the simple unit testing associated with the implementation workflow. In this case, testing also includes testing the integration of all modules used to implement the system, user acceptance testing, and the actual alpha testing of the software. Practically speaking, testing should go on throughout the development of the system; testing of the analysis and design models occurs during the elaboration and construction phases, whereas implementation testing is performed primarily during the construction and, to some degree, transition phases. Basically, at the end of each iteration during the development of the information system, some type of testing should be performed.

Deployment Workflow The deployment workflow is most associated with the transition phase of the Unified Process. The *deployment workflow* includes activities such as software packaging, distribution, installation, and beta testing. When actually deploying the new system into a user organization, the developers might have to convert the current data, interface the new software with the existing software, and train the end user to use the new system.

Supporting Workflows The supporting workflows include the project management, configuration and change management, and environment workflows. The supporting workflows focus on the managerial aspects of information systems development.

Project Management Workflow Whereas the other workflows associated with the Unified Process are technically active during all four phases, the *project management workflow* is the only truly cross-phase workflow. The development process supports incremental and iterative development, so information systems tend to grow or evolve over time. At the end of each iteration, a new incremental version of the system is ready for delivery. The project management workflow is quite important owing to the complexity of the two-dimensional development model of the Unified Process (workflows and phases). This workflow's activities include identifying and managing risks, managing scope, estimating the time to complete each iteration and the entire project, estimating the cost of the individual iteration and the whole project, and tracking the progress being made toward the final version of the evolving information system.

Configuration and Change Management Workflow The primary purpose of the configuration and change management workflow is to keep track of the state of the evolving system. In a nutshell, the evolving information system comprises a set of artifacts (e.g., diagrams, source code, and executables). During the development process, these artifacts are modified. A substantial amount of work—and, hence, money—is involved in developing the artifacts. The artifacts themselves should be handled as any expensive asset would be handled—access controls must be put into place to safeguard the artifacts from being stolen or destroyed. Furthermore, because the artifacts are modified on a regular, if not continuous, basis, good version control mechanisms should be established. Finally, a good deal of project management information needs to be captured (e.g., author, time, and location of each modification). The configuration and change management workflow is associated mostly with the construction and transition phases.

Environment Workflow During the development of an information system, the development team needs to use different tools and processes. The *environment workflow* addresses these needs. For example, a CASE tool that supports the development of an object-oriented information system via the UML could be required. Other tools necessary include programming environments, project management tools, and configuration management tools. The environment workflow involves acquiring and installing these tools. Even though this workflow can be active during all of the phases of the Unified Process, it should be involved primarily with the inception phase.

Extensions to the Unified Process

As large and as complex as the Unified Process is, many authors have pointed out a set of critical weaknesses. First, the Unified Process does not address staffing, budgeting, or contract management issues. These activities were explicitly left out of the Unified Process. Second, the Unified Process does not address issues relating to maintenance, operations, or support of the product once it has been delivered. Thus, it is not a complete software process; it is only a development process. Third, the Unified Process does not address cross- or inter-project issues. Considering the importance of reuse in object-oriented systems development and the fact that in many organizations employees work on many different projects at the same time, leaving out inter-project issues is a major omission.

To address these omissions, Ambler and Constantine suggest adding a production phase and two workflows: the operations and support workflow and the infrastructure management workflow (see Figure 1-4).[25] In addition to these new workflows, the test, deployment, and environment workflows are modified, and the project management and the configuration and change management workflows are extended into the production phase. These extensions are based on alternative object-oriented software processes: the OPEN process (Object-oriented Process, Environment, and Notation) and the Object-Oriented Software Process.[26]

[25] S. W. Ambler and L. L. Constantine, *The Unified Process Inception Phase: Best Practices in Implementing the UP* (Lawrence, KS: CMP Books, 2000); S. W. Ambler and L. L. Constantine, *The Unified Process Elaboration Phase: Best Practices in Implementing the UP* (Lawrence, KS: CMP Books, 2000); S. W. Ambler and L. L. Constantine, *The Unified Process Construction Phase: Best Practices in Implementing the UP* (Lawrence, KS: CMP Books, 2000); S. W. Ambler and L. L. Constantine, *The Unified Process Transition and Production Phases: Best Practices in Implementing the UP* (Lawrence, KS: CMP Books, 2002).

[26] S. W. Ambler, *Process Patterns—Building Large-Scale Systems Using Object Technology* (Cambridge, UK: SIGS Books/Cambridge University Press, 1998); S. W. Ambler, *More Process Patterns—Delivering Large-Scale Systems Using Object Technology* (Cambridge, UK: SIGS Books/Cambridge University Press, 1999); I. Graham, B. Henderson-Sellers, and H. Younessi, *The OPEN Process Specification* (Harlow, UK: Addison-Wesley, 1997); B. Henderson-Sellers and B. Unhelkar, *OPEN Modeling with UML* (Harlow, UK: Addison-Wesley, 2000).

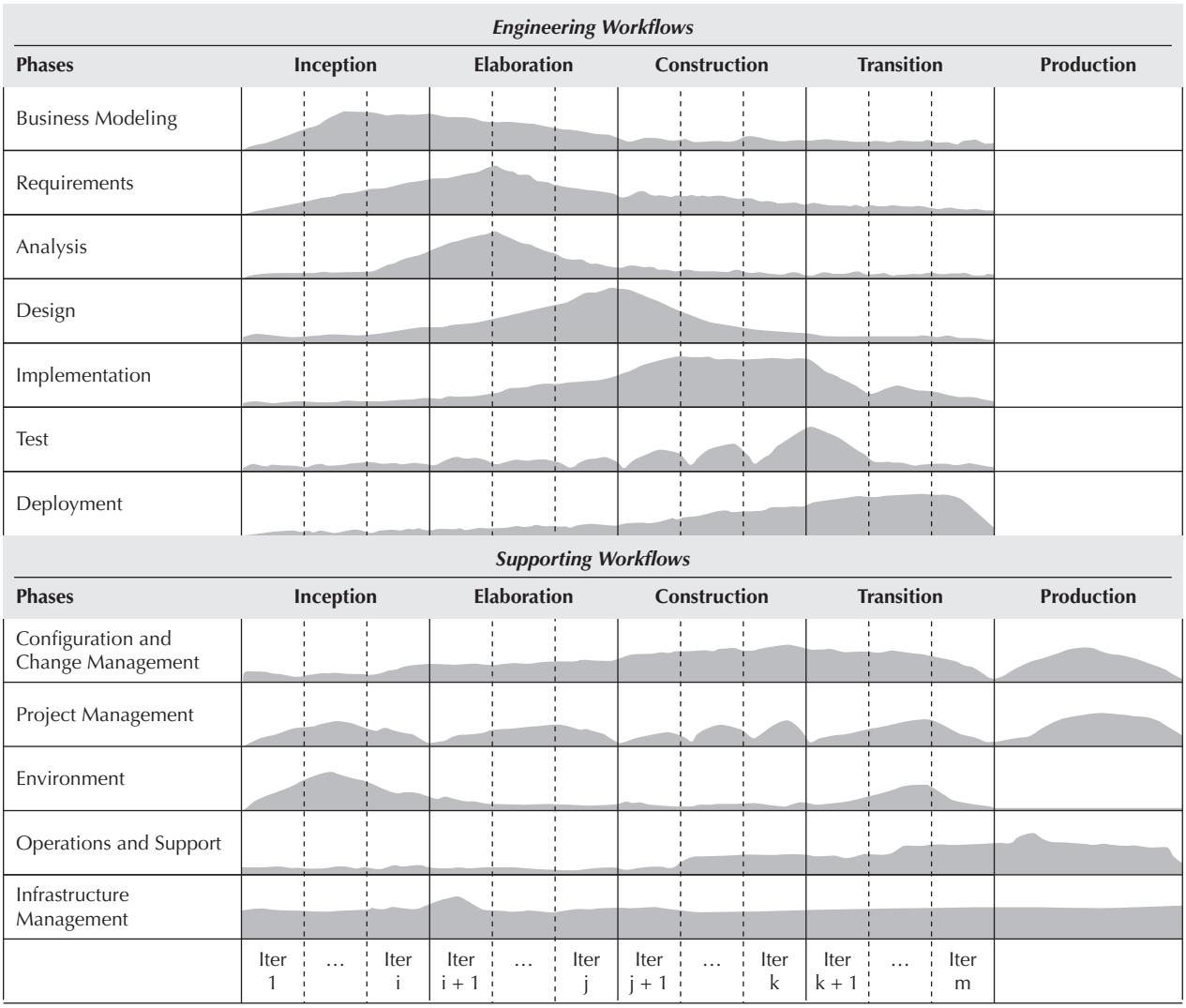

FIGURE 1-4 The Enhanced Unified Process

Production Phase The *production phase* is concerned primarily with issues related to the software product after it has been successfully deployed. This phase focuses on issues related to updating, maintaining, and operating the software. Unlike the previous phases, there are no iterations or incremental deliverables. If a new release of the software is to be developed, then the developers must begin a new run through the first four phases. Based on the activities that take place during this phase, no engineering workflows are relevant. The supporting workflows that are active during this phase include the configuration and change management workflow, the project management workflow, the new operations and support workflow, and the infrastructure management workflow.

Operations and Support Workflow The *operations and support workflow*, as you might guess, addresses issues related to supporting the current version of the software and operating the software on a daily basis. Activities include creating plans for the operation and support of the software product once it has been deployed, creating training and user documentation,

putting into place necessary backup procedures, monitoring and optimizing the performance of the software, and performing corrective maintenance on the software. This workflow becomes active during the construction phase; its level of activity increases throughout the transition and, finally, the production phase. The workflow finally drops off when the current version of the software is replaced by a new version. Many developers are under the false impression that once the software has been delivered to the customer, their work is finished. In most cases, the work of supporting the software product is much more costly and time consuming than the original development. At that point, the developer's work may have just begun. When considering the newer DevOps approaches, it is obvious that operations personnel should be actively involved with this workflow.

Infrastructure Management Workflow The *infrastructure management workflow's* primary purpose is to support the development of the infrastructure necessary to develop object-oriented systems. Activities such as development and modification of libraries, standards, and enterprise models are very important. When the development and maintenance of a problem-domain architecture model goes beyond the scope of a single project and reuse is going to occur, the infrastructure management workflow is essential. Another very important set of cross-project activities is the improvement of the software development process. Because the activities on this workflow tend to affect many projects and the Unified Process focuses only on a specific project, the Unified Process tends to ignore these activities (i.e., they are simply beyond the scope and purpose of the Unified Process).

Existing Workflow Modifications and Extensions In addition to the workflows that were added to address deficiencies contained in the Unified Process, existing workflows had to be modified and/or extended into the production phase. These workflows include the test, deployment, environment, project management, and configuration and change management workflows.

Test Workflow For high-quality information systems to be developed, testing should be done on every deliverable, including those created during the inception phase. Otherwise, less than high-quality systems will be delivered to the customer.

Deployment Workflow Legacy systems exist in most corporations today, and these systems have databases associated with them that must be converted to interact with the new systems. Owing to the complexity of deploying new systems, the conversion requires significant planning. Therefore, the activities on the deployment workflow need to begin in the inception phase instead of waiting until the end of the construction phase, as suggested by the Unified Process.

Environment Workflow The environment workflow needs to be modified to include activities related to setting up the operations and production environment. The actual work performed is similar to the work related to setting up the development environment that was performed during the inception phase. In this case, the additional work is performed during the transition phase.

Project Management Workflow Even though the project management workflow does not include staffing the project, managing the contracts among the customers and vendors, and managing the project's budget, these activities are crucial to the success of any software

development project. We suggest extending project management to include these activities. This workflow should additionally occur in the production phase to address issues such as training, staff management, and client relationship management.

Configuration and Change Management Workflow The configuration and change management workflow is extended into the new production phase. Activities performed during the production phase include identifying potential improvements to the operational system and assessing the potential impact of the proposed changes. Once developers have identified these changes and understood their impact, they can schedule the changes to be made and deployed with future releases.

Figure 1-5 shows the chapters in which the Enhanced Unified Process phases and workflows are covered. Given the offshore outsourcing and automation of information technology,[27] in this textbook, we focus primarily on the elaboration phase and the business modeling, requirements, analysis, design, and project management workflows of the Enhanced Unified Process. However, as Figure 1-5 shows, the other phases and workflows are covered.

Enhanced UP Phases	Chapters
Inception	2–4
Elaboration	3–11
Construction	8, 12
Transition	12–13
Production	13
Enhanced UP Engineering Workflows	**Chapters**
Business Modeling	2–5
Requirements	3–5, 10
Analysis	3–7
Design	7–11
Implementation	9, 12
Test	4–7, 12
Deployment	13
Enhanced UP Supporting Workflows	**Chapters**
Project Management	2, 13
Configuration and Change Management	2, 13
Environment	2
Operations and Support	13
Infrastructure Management	2

FIGURE 1-5 The Enhanced Unified Process and the Textbook Organization

[27] See Thomas L. Friedman, *The World Is Flat: A Brief History of the Twenty-First Century, Updated and Expanded Edition* (New York: Farrar, Straus, and Giroux, 2006); Daniel H. Pink, *A Whole New Mind: Why Right-Brainers Will Rule the Future* (New York: Riverhead Books, 2006).

In many object-oriented systems development environments today, code generation is supported. Thus, from a business perspective, we believe the activities associated with these workflows are the most important.

THE UNIFIED MODELING LANGUAGE

Until 1995, object concepts were popular but implemented in many different ways by different developers. Each developer had his or her own methodology and notation (e.g., Booch, Coad, Moses, OMT, OOSE, and SOMA).[28] Then in 1995, Rational Software brought three industry leaders together to create a single approach to object-oriented systems development. Grady Booch, Ivar Jacobson, and James Rumbaugh worked with others to create a standard set of diagramming techniques known as the *Unified Modeling Language (UML)*. The objective of UML was to provide a common vocabulary of object-oriented terms and diagramming techniques rich enough to model any systems development project from analysis through implementation. In November 1997, the *Object Management Group (OMG)* formally accepted UML as the standard for all object developers. During the following years, the UML has gone through multiple minor revisions. The current version of UML is Version 2.5.

Version 2.5 of the UML defines a set of fifteen diagramming techniques used to model a system. The diagrams are broken into two major groupings: one for modeling the structure of a system and one for modeling behavior. *Structure diagrams* provide a way to represent the data and static relationships in an information system. The structure diagrams include class, object, package, deployment, component, composite structure, and profile diagrams. *Behavior diagrams* provide the analyst with a way to depict the dynamic relationships among the instances or objects that represent the business information system. They also allow modeling of the dynamic behavior of individual objects throughout their lifetime. The behavior diagrams support the analyst in modeling the functional requirements of an evolving information system. The behavior modeling diagrams include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use-case diagrams.[29] Figure 1-6 provides an overview of these diagrams.

Depending on where in the development process the system is, different diagrams play a more important role. In some cases, the same diagramming technique is used throughout the development process. In that case, the diagrams start off very conceptual and abstract. As the system is developed, the diagrams evolve to include details that ultimately lead to

[28] See Grady Booch, *Object-Oriented Analysis and Design with Applications,* 2nd Ed. (Redwood City, CA: Benjamin/Cummings, 1994); Peter Coad and Edward Yourdon, *Object-Oriented Analysis,* 2nd Ed. (Englewood Cliffs, NJ: Yourdon Press, 1991); Peter Coad and Edward Yourdon, *Object-Oriented Design* (Englewood Cliffs, NJ: Yourdon Press, 1991); Brian Henderson-Sellers and Julian Edwards, *Book Two of Object-Oriented Knowledge: The Working Object* (Sydney, Australia: Prentice Hall, 1994); James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design* (Englewood Cliffs, NJ: Prentice Hall, 1991); Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Approach* (Wokingham, England: Addison-Wesley, 1992); Ian Graham, *Migrating to Object Technology* (Wokingham, England: Addison-Wesley, 1994).

[29] The material contained in this section is based on the *OMG Unified Modeling Language Version 2.5.1, December 2017* (www.uml.org). Additional useful references include Michael Jesse Chonoles and James A. Schardt, *UML 2 for Dummies* (Indianapolis, IN: Wiley, 2003); Hans-Erik Eriksson, Magnus Penker, Brian Lyons, and David Fado, *UML 2 Toolkit* (Indianapolis, IN: Wiley, 2004); Kendall Scott, *Fast Track UML 2.0* (Berkeley, CA: Apress, 2004). For a complete description of all diagrams, see www.uml.org.

Diagram Name	Used to. . .	Primary Phase
Structure Diagrams		
Class	Illustrate the relationships between classes modeled in the system	Analysis, Design
Object	Illustrate the relationships between objects modeled in the system; used when actual instances of the classes will better communicate the model	Analysis, Design
Package	Group other UML elements together to form higher-level constructs	Analysis, Design, Implementation
Deployment	Show the physical architecture of the system; can also be used to show software components being deployed onto the physical architecture	Physical Design, Implementation
Component	Illustrate the physical relationships among the software components	Physical Design, Implementation
Composite Structure Design	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class	Analysis, Design
Profile	Used to develop extensions to the UML itself	None
Behavioral Diagrams		
Activity	Illustrate business workflows independent of classes, the flow of activities in a use case, or detailed design of a method	Analysis, Design
Sequence	Model the behavior of objects within a use case; focuses on the time-based ordering of an activity	Analysis, Design
Communication	Model the behavior of objects within a use case; focus on the communication among a set of collaborating objects of an activity	Analysis, Design
Interaction Overview	Illustrate an overview of the flow of control of a process	Analysis, Design
Timing	Illustrate the interaction among a set of objects and the state changes they go through along a time axis	Analysis, Design
Behavioral State Machine	Examine the behavior of one class	Analysis, Design
Protocol State Machine	Illustrate the dependencies among the different interfaces of a class	Analysis, Design
Use-Case	Capture business requirements for the system and illustrate the interaction between the system and its environment	Analysis

FIGURE 1-6 UML 2.5 Diagram Summary

generating and developing code. In other words, the diagrams move from documenting the requirements to laying out the design. Overall, the consistent notation, integration among the diagramming techniques, and application of the diagrams across the entire development process make the UML a powerful and flexible language for analysts and developers. Later chapters provide more detail on using a subset of the UML in object-oriented systems analysis and design. In particular, these chapters describe activity, use-case, class, sequence, package, and deployment diagrams and the behavior state machines. We also introduce an optional UML diagram, the windows navigation diagram, that is an extension to the behavioral state machine that is used to design user navigation through an information system's user interfaces.

APPLYING THE CONCEPTS AT PATTERSON SUPERSTORE

This course will introduce many new concepts regarding object-oriented analysis and design. To make these concepts more relevant and understandable, we will apply the concepts, introduced in each chapter, to a fictitious company called Patterson Superstore.

Patterson is a retail chain established in Pittsburgh, PA, in 1985. Currently, Patterson uses a mobile application to facilitate prescription order, notification, and auto refill services. This service is widely used by Patterson's client base, and Patterson has leveraged this mobile app to gain an advantage over less technically advanced competitors.

Clients now want to use this technology to access health clinic services. The Vice President of Pharmacy Services, Max Ross, would like to use this opportunity to position Patterson as a leader in the use of technology use for clinic access. The system that he envisions will enable real-time communication with medical personnel (audio, video, and text), mobile appointment scheduling, telehealth assessment, and diagnosis of minor problems through video house calls. Throughout the book, we will revisit Patterson Superstore to see how the concepts introduced in each chapter affect this project.

You can find the rest of the case at www.wiley.com/go/dennis/systemanalysis6e/casestudy.

CHAPTER REVIEW

After reading and studying this chapter, you should be able to:

- ☐ Explain the different roles played by a systems analyst in the process of developing information systems.
- ☐ Describe the four primary phases of the Systems Development Life Cycle (SDLC).
- ☐ Explain the evolution of systems development methodologies from process-centric to data-centric to RAD-based methodologies.
- ☐ Discuss the three basic characteristics of all object-oriented systems analysis and design approach: use-case driven, architecture-centric, and iterative and incremental development.
- ☐ Describe the Unified Process.
- ☐ List and categorize, as to their primary purpose, the different diagrams associated with the Unified Modeling Language (UML).

KEY TERMS

Agile development
Analysis model
Analysis paralysis
Analysis phase
Analysis strategy
Analysis workflow
Approval committee
Architecture-centric
Architecture design
As-is system
Behavior diagram
Behavioral view
Business analyst
Business modeling workflow
Change agent
Change management analyst
Configuration and change management workflow
Construction
Construction phase
Database and file specification
Data-centered methodology
Deliverable
Deployment workflow
Design model
Design phase
Design prototype
Design strategy
Design workflow
DevOps
Dynamic view
Elaboration phase
Engineering workflow
Environment workflow
External view
Extreme programming (XP)
Feasibility analysis
Functional view
Gradual refinement
Implementation phase
Implementation workflow
Inception phase
Incremental
Infrastructure analyst
Infrastructure management workflow
Interface design
Iterative

Methodology
Object Management Group (OMG)
Object-oriented methodology
Operations and support workflow
Parallel development
Phased development
Phases
Planning phase
Process-centered methodology
Production phase
Program design
Programmer
Project management
Project management workflow
Project manager
Project plan
Project sponsor
Prototyping
Rapid application development (RAD)
Requirements gathering
Requirements workflow
Scrum
Static view
Structural view
Structure diagram
Structured design
Support plan
System proposal
System prototype
System request
System specification
Systems analyst
Systems development life cycle (SDLC)
Technical writer
Testing workflow
Throwaway prototyping
Training plan
Transition phase
Unified Modeling Language (UML)
Use case
Use-case driven
Version
Waterfall development
Workflows
Workplan

QUESTIONS

1. What are the major roles played by a systems analyst on a project team?
2. Compare and contrast the role of a systems analyst, a business analyst, and an infrastructure analyst.
3. Compare and contrast phases, steps, techniques, and deliverables.
4. Describe the major phases in the SDLC.
5. Which phase in the SDLC is most important? Why?
6. Describe the principal steps in the planning phase. What are the major deliverables?
7. Describe the principal steps in the analysis phase. What are the major deliverables?
8. Describe the principal steps in the design phase. What are the major deliverables?
9. Describe the principal steps in the implementation phase. What are the major deliverables?
10. What are the roles of a project sponsor and the approval committee?
11. What does *gradual refinement* mean in the context of SDLC?
12. Compare and contrast process-centered methodologies with data-centered methodologies.
13. Compare and contrast structured design-based methodologies in general to RAD-based methodologies in general.
14. Describe the major elements in and issues with waterfall development.
15. Describe the major elements in and issues with parallel development.
16. Describe the major elements in and issues with phased development.
17. Describe the major elements in and issues with prototyping.
18. Describe the major elements in and issues with throwaway prototyping.
19. What is a use case?
20. What is meant by use-case driven?
21. Why is it important for an OOSAD approach to be architecture-centric?
22. What does it mean for an OOSAD approach to be incremental and iterative?
23. Describe the major elements and issues with an object-oriented approach to developing information systems.
24. Describe the major elements in and issues with XP.
25. Compare and contrast extreme programming and throwaway prototyping.
26. Describe the major elements in and issues with Scrum.
27. What are the primary differences between a DevOps methodology and an agile or object-oriented methodology?
28. What are the phases and workflows of the Unified Process?
29. Compare the phases of the Unified Process with the phases of the waterfall model.
30. Who is the Object Management Group?
31. What is the Unified Modeling Language?
32. What is the primary purpose of structure diagrams? Give some examples of structure diagrams.
33. For what are behavior diagrams used? Give some examples of behavior diagrams.

EXERCISES

A. Suppose you are a project manager using a waterfall development-based methodology on a large and complex project. Your manager has just read the latest article in *Computerworld* that advocates replacing this methodology with prototyping and comes to you requesting that you switch. What would you say?

B. The basic types of 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 waterfall development. What would the methodology look like? How would this new methodology compare to the others?

C. Look on the Web for different kinds of job opportunities that 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.

D. Think about your ideal analyst position. Write an ad to hire someone for that position. What requirements would the job have? What skills and experience would be required? How would an applicant be able to demonstrate having the appropriate skills and experience?

E. Using your favorite Web search engine, find alternative descriptions of the basic characteristics of object-oriented systems.

F. Investigate IBM's Rational Unified Process (RUP) on the Web. RUP is a commercial version that extends aspects of the Unified Process. Write a brief memo describing how it is related to the Unified Process as described in this chapter.

G. Suppose you are a project manager who typically has been using a waterfall development-based methodology on a large and complex project. Your manager has just read the latest article in *Computerworld* that advocates replacing this methodology with the Unified Process and comes to you requesting you to switch. What do you say?

H. Suppose you are an analyst working for a small company to develop an accounting system. Would you use the Unified Process to develop the system, or would you prefer one of the other approaches? Why?

I. Suppose you are 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. Would you use the Unified Process to develop the system, or would you prefer one of the other approaches? Why?

J. Suppose you are an analyst working for a small company to develop an accounting system. What type of methodology would you use? Why?

K. Suppose you are an analyst developing a new executive information system intended to provide key strategic information from existing corporate databases to senior executives to help in their decision making. What type of methodology would you use? Why?

L. Investigate the Unified Modeling Language on the Web. Write a paragraph news brief describing the current state of the UML.

M. Investigate the Object Management Group (OMG) on the Web. Write a report describing the purpose of the OMG and what it is involved with besides the UML.

N. Using the Web, find a set of CASE tools that support the UML. A couple of examples include Poseidon, Rational Rose, and Visual Paradigm. Find at least two more. Write a short report describing how well they support the UML, and make a recommendation as to which one you believe would be the best for a project team to use in developing an object-oriented information system using the UML.

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, owing to the massive backlog of work facing the IS department, her request was given a low priority. After six months of inaction by the IS department, Barbara decided to take matters into her own hands. Based on the advice of friends, Barbara purchased simple database software and constructed a sales force management and tracking system on her own.

Although Barbara's system has been "completed" for about six 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, because 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 underwriter's assistant. This new system will involve a whole new way for the underwriters to perform their tasks. The underwriter's 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 underwriters' 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 because of 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.

3. Joe Brown, the president of Roanoke Manufacturing, requested that Jack Jones, the MIS department manager, investigate the viability of selling their products over the Web. Currently, the MIS department is still using an IBM mainframe as their primary deployment environment. As a first step, Jack contacted his friends at IBM to see if they had any suggestions as to how Roanoke Manufacturing could move toward supporting sales in an electronic commerce environment while keeping their mainframe as their main system. His friends explained that IBM now supports Java and Linux on their mainframes. Jack has also learned that IBM owns Rational, the creator of the UML and the Unified Process. Jack's friends suggested that Jack investigate using object-oriented systems as a basis for developing the new system. They also suggested that using the Rational Unified Process (RUP), Java, and virtual Linux machines on his current mainframe as a way to support the move toward a distributed electronic commerce system would protect his current investment in his legacy systems while allowing the new system to be developed in a more modern manner. Even though Jack's IBM friends were very persuasive, Jack is still a little wary about moving his operation from a structured systems approach to this new object-oriented approach. Assuming that you are one of Jack's IBM friends, how would you convince him to move toward using an object-oriented systems development method, such as RUP, and using Java and Linux as a basis for developing and deploying the new system on Roanoke Manufacturing's current mainframe?