CHAPTER 2

INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH THE UNIFIED MODELING LANGUAGE, VERSION 2.0

This chapter introduces Object-Oriented Systems Analysis and Design with the Unified Modeling Language, Version 2.0. First, the chapter introduces the basic characteristics of object-oriented systems. Second, it introduces UML 2.0. Third, the chapter overviews Object-Oriented Systems Analysis and Design and describes the Unified Process. Finally, based on the Unified Process and the UML 2.0, the chapter provides a minimalist approach to Object-Oriented Systems Analysis and Design with UML 2.0.

OBJECTIVES:

- Understand the basic characteristics of object-oriented systems.
- Be familiar with the Unified Modeling Language (UML), Version 2.0.
- Be familiar with the Unified Process.
- Understand a minimalist approach to object-oriented systems analysis and design.

CHAPTER OUTLINE

INTRODUCTION

Until recent years, analysts focused on either data or business processes when developing systems. As they moved through the SDLC, they emphasized either the data for the system (data-centric approaches) or the processes that it would support (process-centric approaches). In the mid-1980s, developers were capable of building systems that could be more efficient if the analyst worked with a system's data and processes simultaneously and focused on integrating the two. As such, project teams began using an *object-oriented approach*, whereby self-contained modules called *objects* (containing both data and processes) were used as the building blocks for systems.

The ideas behind object-oriented approaches are not new. They can be traced back to the object-oriented programming languages *Simula*, created in the 1960s, and *Smalltalk*, created in the early 1970s. Until the mid-1980s, developers had to keep the data and processes separate to be capable of building systems that could run on the mainframe computers of that era. Today, due to the increase in processor power and the decrease in processor cost, object-oriented approaches are feasible. One of the major hurdles of learning object-oriented approaches to developing information systems is the volume of new terminology. In this chapter we overview the basic characteristics of an object-oriented system, provide an overview of the second version of the Unified Modeling Language (UML, 2.0), introduce basic object-oriented systems analysis and design and the Unified Process, and present a minimalist approach to object-oriented systems analysis and design with UML 2.0.

BASIC CHARACTERISTICS OF OBJECT-ORIENTED SYSTEMS

Object-oriented systems focus on capturing the structure and behavior of information systems in little modules that encompass both data and process. These little modules are known as objects. In this section of the chapter we describe the basic characteristics of object-oriented systems, which include classes, objects, methods, messages, encapsulation, information hiding, inheritance, polymorphism, and dynamic binding.

Classes and Objects

A *class* is the general template we use to define and create specific instances, or objects. Every object is associated with a class. For example, all of the objects that capture information about patients could fall into a class called Patient, because there are attributes (e.g., names, addresses, and birth dates) and methods (e.g., insert new instances, maintain information, and delete entries) that all patients share (see Figure 2-1).

An *object* is an instantiation of a class. In other words, an object is a person, place, event, or thing about which we want to capture information. If we were building an appointment system for a doctor's office, classes might include doctor, patient, and appointment. The specific patients like Jim Maloney, Mary Wilson, and Theresa Marks are considered *instances*, or objects, of the patient class.

Each object has *attributes* that describe information about the object, such as a patient's name, birth date, address, and phone number. The *state* of an object is defined by the value of its attributes and its relationships with other objects at a particular point in time. For example, a patient might have a state of "new" or "current" or "former."

Each object also has *behaviors*. The behaviors specify what the object can do. For example, an appointment object likely can schedule a new appointment, delete an appointment, and locate the next available appointment.

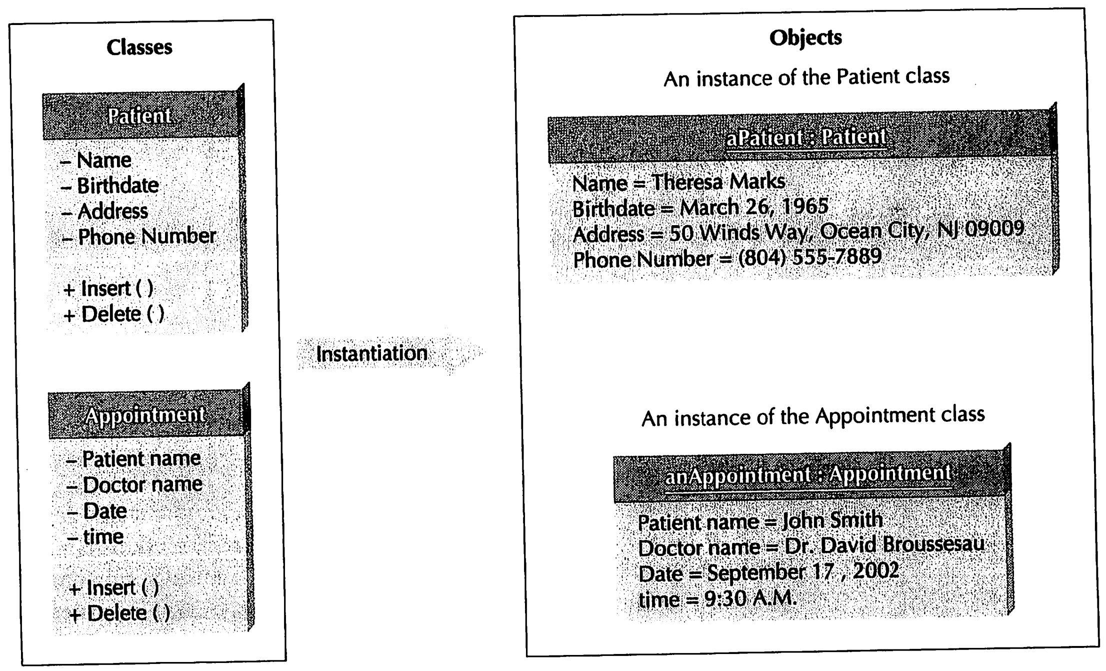
Classes
Patient
- Name
- Birthdate
- Address
- Phone Number
+ Insert ( )
+ Delete ( )
Appointment
- Patient name
- Doctor name
- Date
- time
+ Insert ( )
+ Delete ( )
Instantiation
Objects
An instance of the Patient class
aPatient : Patient
Name = Theresa Marks
Birthdate = March 26, 1965
Address = 50 Winds Way, Ocean City, NJ 09009
Phone Number = (804) 555-7889
An instance of the Appointment class
anAppointment : Appointment
Patient name = John Smith
Doctor name = Dr. David Broussesau
Date = September 17, 2002
time = 9:30 A.M.

FIGURE 2-1 Classes and Objects

One of the more confusing aspects of object-oriented systems development is the fact that in most object-oriented programming languages, both classes and instances of classes can have attributes and methods. Class attributes and methods tend to be used to model attributes (or methods) that deal with issues related to all instances of the class. For example, to create a new patient object, a message is sent to the patient class to create a new instance of itself. However, from a systems analysis and design point of view, we will focus primarily on attributes and methods of objects and not of classes.

Methods and Messages

Methods implement an object's behavior. A method is nothing more than an action that an object can perform. As such, they are analogous to a function or procedure in a traditional programming language such as C, Cobol, or Pascal. *Messages* are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another object. For example, if a patient is new to the doctor's office, the system will send an insert message to the application. The patient object will receive a message (instruction) and do what it needs to do to go about inserting the new patient into the system (execute its method). See Figure 2-2.

Encapsulation and Information Hiding

The ideas of *encapsulation* and *information hiding* are interrelated in object-oriented systems. However, neither of the terms is new. Encapsulation is simply the combination of process and data into a single entity. Traditional approaches to information systems development tend to be either process-centric (e.g., structured systems) or data-centric (e.g., information engineering). Object-oriented approaches combine process and data into holistic entities (objects).

Good Example
Car

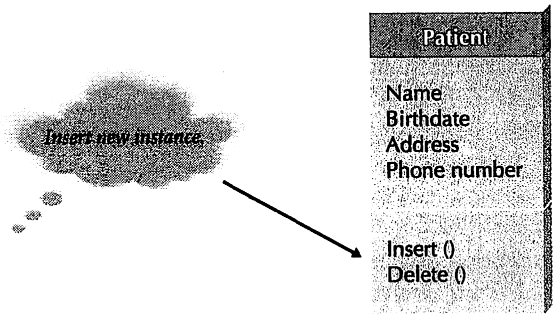
Insert new instance.
Patient
Name
Birthdate
Address
Phone number
Insert ()
Delete ()
A message is sent to the application.
The object's insert method will respond to the message and insert a new patient instance.

FIGURE 2-2
Messages and Methods

Information hiding was first promoted in structured systems development. The principle of information hiding suggests that only the information required to use a software module be published to the user of the module. Typically, this implies the information required to be passed to the module, and the information returned from the module is published. Exactly how the module implements the required functionality is not relevant. We really do not care how the object performs its functions, as long as the functions occur. In object-oriented systems, combining encapsulation with the information hiding principle suggests that the information hiding principle be applied to objects instead of merely applying it to functions or processes. As such, objects are treated like black boxes.

The fact that we can use an object by calling methods is the key to reusability because it shields the internal workings of the object from changes in the outside system, and it keeps the system from being affected when changes are made to an object. In Figure 2-2, notice how a message (insert new patient) is sent to an object yet the internal algorithms needed to respond to the message are hidden from other parts of the system. The only information that an object needs to know is the set of operations, or methods, that other objects can perform and what messages need to be sent to trigger them.

Inheritance

Inheritance, as an information systems development characteristic, was proposed in data modeling in the late 1970s and the early 1980s. The data modeling literature suggests using inheritance to identify higher-level, or more general, classes of objects. Common sets of attributes and methods can be organized into *superclasses*. Typically, classes are arranged in a hierarchy whereby the *superclasses*, or general classes, are at the top, and the *subclasses*, or specific classes, are at the bottom. In Figure 2-3, person is a superclass to the classes Doctor and Patient. Doctor, in turn, is a superclass to general practitioner and specialist. Notice how a class (e.g., doctor) can serve as a superclass and subclass concurrently. The relationship between the class and its superclass is known as the *A-Kind-Of (AKO)* relationship. For example, in Figure 2-3, a general practitioner is A-Kind-Of doctor, which is A-Kind-Of person.

Subclasses *inherit* the appropriate attributes and methods from the superclasses above them. That is, each subclass contains attributes and methods from its parent superclass. For example, Figure 2-3 shows that both doctor and patient are subclasses of person and therefore will inherit the attributes and methods of the person class. Inheritance makes it simpler to define classes. Instead of repeating the attributes and methods in the doctor and patient classes separately, the attributes and methods that are common

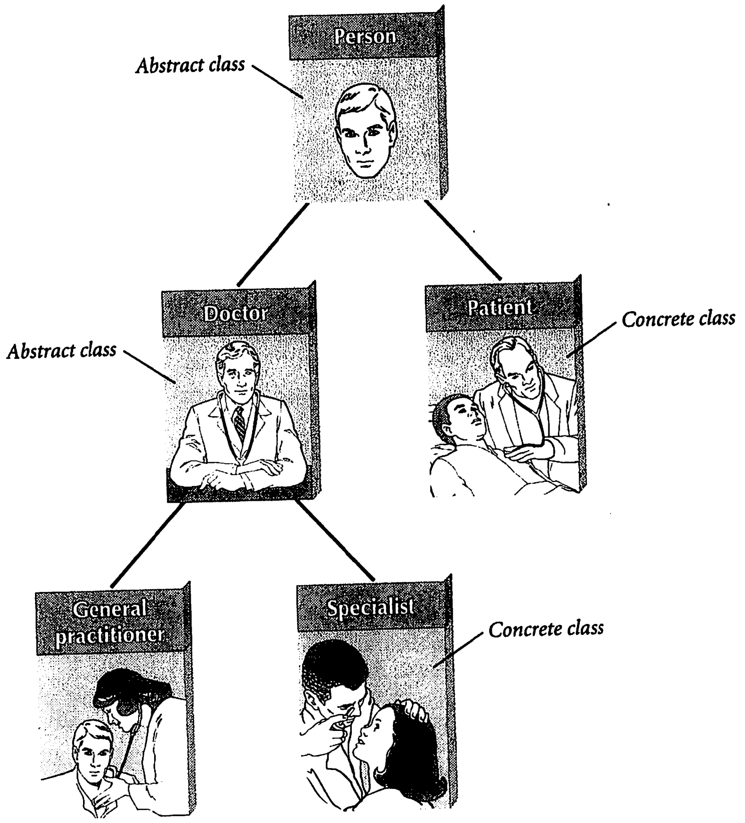

FIGURE 2-3
Class Hierarchy

to both are placed in the person class and inherited by those classes below it. Notice how much more efficient hierarchies of object classes are than the same objects without a hierarchy in Figure 2-4.

Most classes throughout a hierarchy will lead to instances; any class that has instances is called a *concrete class*. For example, if Mary Wilson and Jim Maloney were instances of the patient class, patient would be considered a concrete class. Some classes do not produce instances because they are used merely as templates for other more specific classes (especially those classes located high up in a hierarchy). The classes are referred to as *abstract classes*. Person would be an example of an abstract class. Instead of creating objects from person, we would create instances representing the more specific classes of Doctor and Patient, both types of *person* (see Figure 2-3). What kind of class is the general practitioner class? Why?

YOUR TURN

2-1 Encapsulation and Information Hiding

Come up with a set of examples of using encapsulation and information hiding in everyday life. For example, is there any information about yourself that you would not mind if everyone knew? How would someone retrieve this information? What about personal information that you would prefer to be private? How would you prevent someone from retrieving it?

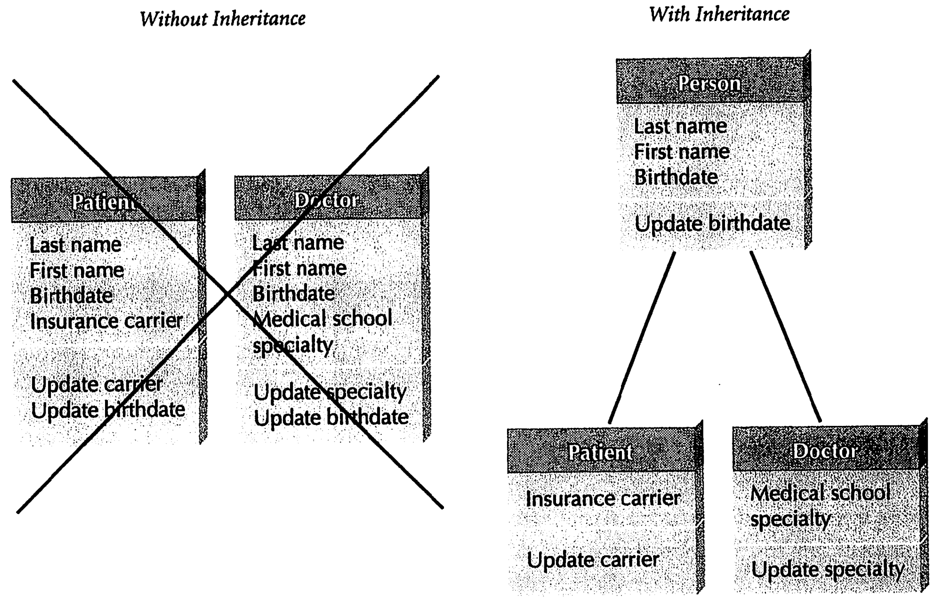

FIGURE 2-4
Inheritance

Polymorphism and Dynamic Binding

Polymorphism means that the same message can be interpreted differently by different classes of objects. For example, inserting a patient means something different than inserting an appointment. As such, different pieces of information need to be collected and stored. Luckily, we do not have to be concerned with *how* something is done when using objects. We can simply send a message to an object, and that object will be responsible for interpreting the message appropriately. For example, if we sent the message "Draw yourself" to a square object, a circle object, and a triangle object, the results would be very different, even though the message is the same. Notice in Figure 2-5 how each object responds appropriately (and differently) even though the messages are identical.

Polymorphism is made possible through *dynamic binding*. Dynamic, or late, binding is a technique that delays typing the object until run-time. As such, the specific method that is actually called is not chosen by the object-oriented system until the system is running. This is in contrast to *static binding*. In a statically bound system, the type of object would be determined at compile time. Therefore, the developer would have to choose which method should be called instead of allowing the system to do it. This is why in most traditional programming languages you find complicated decision logic based on the different types of objects in a system. For example, in a traditional programming language, instead of sending the message "Draw yourself" to the different

YOUR TURN

2-2 Inheritance

See if you can come up with at least three different classes that you might find in a typical business situation. Select one of the classes and create at least a three-level inheritance hierarchy using the class. Which of the classes are abstract, if any, and which ones are concrete?

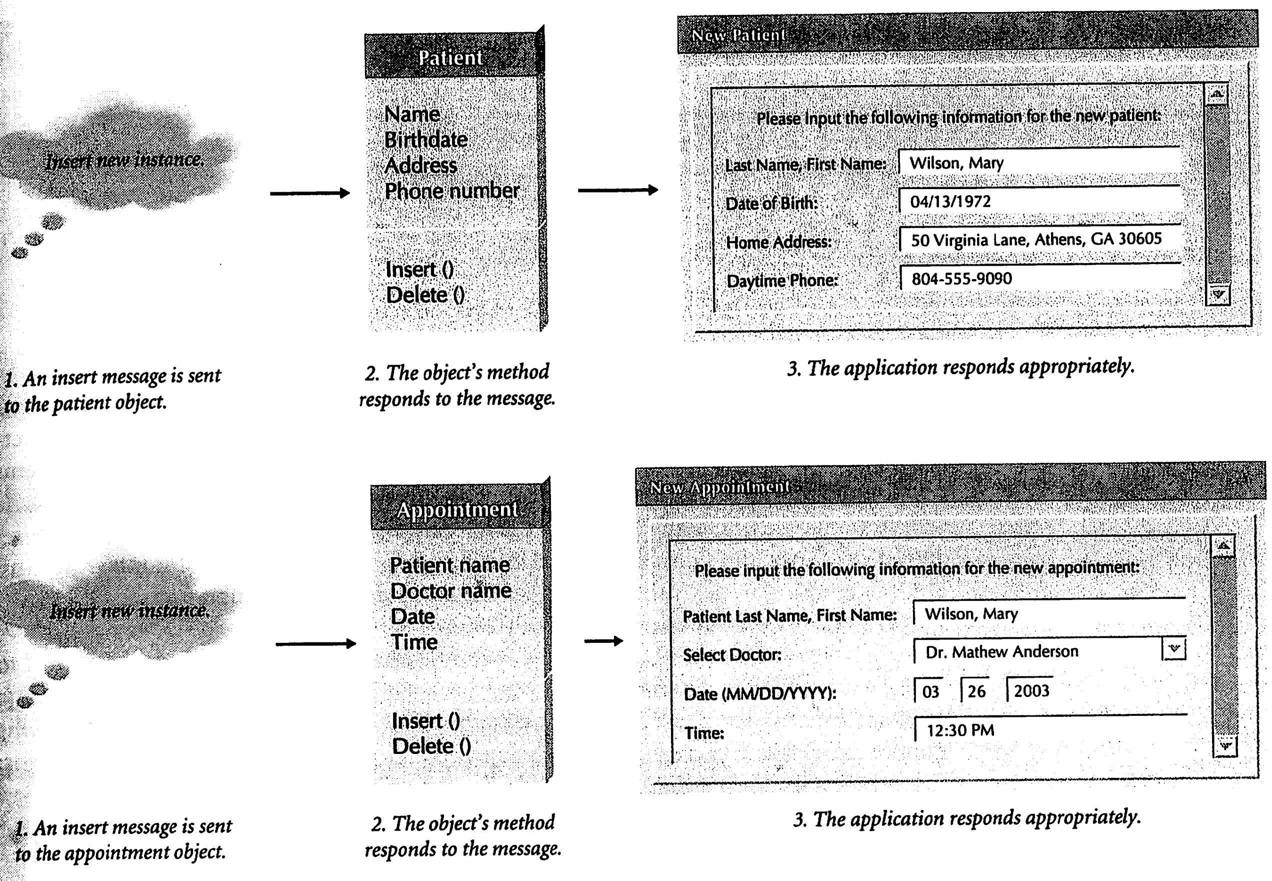
Insert new instance.
Patient
Name
Birthdate
Address
Phone number
Insert ()
Delete ()
New Patient
Please input the following information for the new patient:
Last Name, First Name:
Wilson, Mary
Date of Birth:
04/13/1972
Home Address:
50 Virginia Lane, Athens, GA 30605
Daytime Phone:
804-555-9090
1. An insert message is sent to the patient object.
2. The object's method responds to the message.
3. The application responds appropriately.
Insert new instance.
Appointment
Patient name
Doctor name
Date
Time
Insert ()
Delete ()
New Appointment
Please input the following information for the new appointment:
Patient Last Name, First Name:
Wilson, Mary
Select Doctor:
Dr. Mathew Anderson
Date (MM/DD/YYYY):
03
26
2003
Time:
12:30 PM
1. An insert message is sent to the appointment object.
2. The object's method responds to the message.
3. The application responds appropriately.

FIGURE 2-5 Polymorphism and Encapsulation

¥ || types of graphical objects in Figure 2-5, you would have to write decision logic using a case statement or a set of if statements to determine what kind of graphical object you wanted to draw, and you would have to name each draw function differently (e.g., draw-square, draw-circle, or draw-triangle). This obviously would make the system much more complicated and more difficult to understand.

THE UNIFIED MODELING LANGUAGE, VERSION 2.0

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[1]). Then in 1995, Rational Software brought three industry leaders together to create a single approach to object-oriented sys-

[1] 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, *BookTwo 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, 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).

YOUR TURN 2-3 Polymorphism and Dynamic Binding

Can you think of any way in which you use polymorphism and/or dynamic binding in your everyday life? For example, when you are told to do some task, do you always perform the task like everyone else you know does? Do you always perform the task the same way or is the method of performance depend on where you are when you perform the task?

tems 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 is to provide a common vocabulary of object-oriented terms and diagramming techniques that is 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. Over the years since, the UML has gone through multiple minor revisions. The current version of UML, Version 2.0, was accepted by the members of the OMG during their spring and summer meetings of 2003.

The Version 2.0 of the UML defines a set of fourteen diagramming techniques used to model a system. The diagrams are broken into two major groupings: one for modeling structure of a system and one for modeling behavior. The structure modeling diagrams include class, object, package, deployment, component, and composite structure diagrams. The behavior modeling diagrams include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams.[2] Figure 2-6 provides and 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 code generation and development. 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 the application of the diagrams across the entire development process makes the UML a powerful and flexible language for analysts and developers. In the remainder of this section, we provide an overview of the diagramming techniques supported by the UML. In later chapters, we provide more detail on using a subset of the UML in object-oriented systems analysis and design.

Structure Diagrams

In this section of the chapter, we introduce the static, *structure diagrams* of the UML 2.0. As mentioned above, the structure diagrams include the class, object, package, deployment, component, and composite structure diagrams. Structure diagrams provide a way for representing the data and static relationships that are in an information system. Below, we describe the basic purpose of each of the structure diagrams.

[2] The material contained in this section is based on the *Unified Modeling Language: Superstructure Version 2.0, ptc/03-08-02* (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), and 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	Illustrate the internal structure of a class, i.e., the relationships among the parts of a class.	Analysis, Design
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. Focuses 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 that takes place among a set of objects and the state changes in which they go through along a time axis.	Analysis, Design
Behavioral State Machine	Examine the behavior of one class.	Analysis, Design
Protocol State Machine	Illustrates the dependencies among the different interfaces of a class.	Analysis, Design
Use-Case	Capture business requirements for the system and to illustrate the interaction between the system and its environment.	Analysis

FIGURE 2-6 UML 2.0 Diagram Summary

Class Diagrams The primary purpose of the class diagram is to create a vocabulary that is used by both the analyst and users. *Class diagrams* typically represent the things, ideas or concepts that are contained in the application. For example, if you were building a payroll application, a class diagram would probably contain classes that represent things such as employees, checks, and the payroll register. The class diagram would also portray the relationships among classes. The actual syntax of the class diagram is presented in Chapter 7.

Object Diagrams *Object diagrams* are very similar to class diagrams. The primary difference is that an object diagram portrays objects and their relationships. The primary purpose of an object diagram is to allow an analyst to uncover additional details of a class. In some cases, instantiating a class diagram may aid a user or analyst in discovering additional relevant attributes, relationships, and/or operations, or possibly discover that some of the attributes, relationships, or operations have been misplaced. Like the class diagram, the actual syntax and use of the object diagram is presented in Chapter 7.

Package Diagrams *Package diagrams* are primarily used to group elements of the other UML diagrams together into a higher-level construct: a *package*. Package diagrams are essentially class diagrams that only show packages, instead of classes, and dependency relationships, instead of the typical relationships shown on class diagrams. For example, if we had an appointment system for a doctor's office, it may make sense to group a patient class with the patient's medical history class together to form a patient class package.

Furthermore, it could be useful to create a treatment package that contains symptoms of illnesses, illnesses, and the typical medications that are prescribed for them. We provide more details in using package diagrams in Chapters 6 through 9.

Deployment Diagrams *Deployment diagrams* are used to represent the relationships between the hardware components used in the physical infrastructure of an information system. For example, when designing a distributed information system that will use a wide area network, a deployment diagram can be used to show the communication relationships among the different nodes in the network. They also can be used to represent the software components and how they are deployed over the physical architecture or infrastructure of an information system. In this case, a deployment diagram represents the environment for the execution of the software. In Chapter 13, we describe designing the physical architecture of an information system and use an extension to the deployment diagram to represent the design.

Component Diagrams *Component diagrams* allow the designer to model physical relationships among the physical modules of code. The diagram when combined with the deployment diagram can be used to portray the physical distribution of the software modules over a network. For example, when designing client-server systems, it is useful to show which classes or packages of classes will reside on the client nodes and which ones will reside on the server. Component diagrams also can be useful in designing and developing component–based systems. Since this book focuses on object-oriented systems analysis and design, we will not discuss further the use of component diagrams.

Composite Structure Diagrams The UML 2.0 provides a new diagram for when the internal structure of a class is complex: the composite structure diagram. *Composite structure diagrams* are used to model the relationships among parts of a class. For example, when modeling a payroll register, an analyst may want a class that represents the entire report as well as classes that represent the header, footer, and detail lines of the report. In a standard class diagram, this would require the analyst to model the payroll register as four separate classes with relationships connecting them together. Instead, the composite structure diagram would contain three subclasses: header, footer, and detail lines. Composite structure diagrams also are useful when modeling the internal structure of a component for a component-based system.

Often, the composite structure diagram is a redundant modeling mechanism because its models also can be communicated using packages and package diagrams. Because of this redundancy, and because we are not covering component-based systems development, we will not discuss them further in this book.

YOUR TURN

2-4 Structure Diagrams

Why are structure diagrams considered to be static? Consider the implementation of a system for the Career Services department of your university that would support the student interview and job placement processes. Briefly describe to your primary contact in the Career Services department the kinds of information that the following structure diagrams would communicate: A) class, B) object, C) package, D) deployment, E) component, and F) composite structure. Be sure to include examples from the Career Services department so that your non-technical user will be able to understand your explanations!

Behavior Diagrams

In this section of the chapter, we introduce the dynamic, *behavior diagrams* of the UML 2.0. The behavior diagrams included in UML 2.0 are the activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams. Behavior modeling 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 the 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.

Activity Diagrams *Activity diagrams* provide the analyst with the ability to model processes in an information system. Activity diagrams can be used to model workflows, individual use cases, or the decision logic contained within an individual method.[3] They also provide an approach to model parallel processes. Activity diagrams are further described in Chapter 6.

Interaction Diagrams *Interaction diagrams* portray the interaction among the objects of an object-oriented information system. UML 2.0 provides four different interaction diagrams: sequence, communication, interaction overview, and timing diagrams. Each of these diagrams is introduced here.

*

1. *Sequence diagrams* allow an analyst to portray the dynamic interaction among objects in an information system. Sequence diagrams are by far the most common kind of interaction diagram used in object-oriented modeling. They emphasize the time-based ordering of the activity that takes place with a set of collaborating objects. They are very useful in helping an analyst understand real-time specifications and complex use cases (see below). These diagrams can be used to describe both the logical and physical interactions among the objects. As such, they are useful in both analysis and design activities. We describe sequence diagrams in more detail in Chapter 8.

[illegible]

2. *Communication diagrams* provide an alternative view of the dynamic interaction that takes place among the objects in an object-oriented information system. Where sequence diagrams emphasize the time-based ordering of an activity, communication diagrams focus on the set of messages that are passed within a set of collaborating objects. In other words, communication diagrams depict how objects collaborate to support some aspect of the required functionality of the system. The sequence or time-based ordering of the messages is shown through a sequence numbering scheme. From a practical point of view, communication diagrams and sequence diagrams provide the same information. As such, we describe communication diagrams in more detail in Chapter 8 with the sequence diagrams.

3. *Interaction overview diagrams* help analysts understand complex use cases. They provide an overview of a process's flow of control. Interaction overview diagrams extend activity diagrams through the addition of sequence fragments from sequence diagrams. In effect, sequence fragments are treated as if they were activities in an activity diagram.

[3] For those who are familiar with traditional structured analysis and design, activity diagrams combine the ideas that underlie data flow diagrams and system flowcharts. Essentially, they are sophisticated and updated data flow diagrams. Technically speaking, activity diagrams combine process modeling ideas from many different techniques including event models, statecharts, and Petri Nets. However, UML 2.0's activity diagram has more in common with Petri Nets than the other process modeling techniques. For a good description of using Petri Nets to model business workflows see Wil van der Aalst and Kees van Hee, *Workflow Management: Models, Methods, and Systems* (Cambridge, MA: MIT Press, 2002).

The primary advantage of using interaction overview diagrams is that you can easily model alternative sequence flows. However, practically speaking, this can be accomplished using activity diagrams and use cases instead. Due to their limited use, interaction overview diagrams are not described in more detail in this book.

4. *Timing diagrams* portray the interaction between objects along a time axis. The primary purpose of the timing diagram is to show the change of state of an object in response to events over time. They tend to be very useful when developing real-time or embedded systems. However, like interaction overview diagrams, due to their limited use, we do not describe timing diagrams in more detail in this book.

State Machines In UML 2.0, there are two different *state machines*: behavior and protocol.[4] Behavior state machines are used to describe the changes that an object may go through during its lifetime. Protocol state machines portray a specified sequence of events to which an object may respond.

1. *Behavior state machines* provide a method for modeling the different states, or sets of values, that instances of a class may go through during their lifetime. For example, a patient can change over time from being a New Patient to a Current Patient to a Former Patient. Each of these "types" of patients is really a different state of the same patient. The different states are connected by events that cause the instance (patient) to transition from one state to another. We describe behavior state machines in more detail in Chapter 8.

2. *Protocol state machines* support the analyst in designing dependencies among elements of the class' interface. For example, typically speaking you must open a file or database before querying or updating it. Unlike behavior state machines, protocol state machines may be associated with component ports or class interfaces. Protocol state machines are very specialized. As such, we do not provide any more detail on them in this book.

Use Case Diagrams *Use case diagrams* allow the analyst to model the interaction of an information system and its environment. The environment of an information system includes both the end user and any external system that interacts with the information system. The primary use of the use case diagram is to provide a means to document and understand the requirements of the evolving information system. Use cases and use case diagrams are some of the most important tools that are used in object-oriented systems analysis and design. We describe use cases and use case diagrams in more detail later in this chapter and in Chapter 6.

YOUR TURN 2-5 Behavior Diagrams

Why are behavior diagrams considered to be static? Consider the implementation of a system for the Career Services department of your university that would support the student interview and job placement processes. Briefly describe to your primary contact in the Career Services department the kinds of information that the following structure diagrams would communicate: A) activity, B) sequence, C) communication, D) interaction overview, E) timing, F) behavior state machines, G) protocol state machines, and H) use case. Be sure to include examples from the Career Services department so that you non-technical user will be able to understand your explanations!

[4] UML 2.0 state machines are based on work by David Harel. See David Harel, On Visual Formalisms, *CACM*, 31 (5) (May 1988), 514–530 and David Harel, A Visual Formalism for Complex Systems, *Science of Computer Programming*, 8, (1987), 231–274.

Extension Mechanisms

As large and as complete as the UML is, it is impossible for the creators of the UML to anticipate all potential uses. Fortunately, the creators of the UML also have provided a set of extension mechanisms. These include stereotypes, tagged values, constraints, and profiles. Each of these is described next.

* **Stereotypes** A *stereotype* provides the analyst with the ability to incrementally extend the UML using the model elements already in the UML. A stereotype is shown as a text item enclosed within guillemets (<<>>) or angle brackets (<< >>). Stereotypes can be associated with any model element (e.g., class, object, use case, relationships) within any UML diagram. We will use stereotypes in Chapter 12 in conjunction with a special form of the behavior state machine: the window navigation diagram.

Tagged Values In the UML, all model elements have properties that describe them. For example, all elements have a name. There are times that it is useful to add properties to the base elements. *Tagged values* are used to add new properties to a base element. For example, if a project team was interested in tracing the authorship of each class in a class diagram, the project team could extend the class element to include an author property. It is also possible to associate tagged values with specific stereotypes. In this manner, when the analyst applies a stereotype to a model element, all of the additional tagged values associated with the stereotype also are applied. We do not describe tagged values in any more detail in this book.

Constraints *Constraints* allow the analyst to model problem domain specific semantics by placing additional restrictions on the use of model elements. Constraints are typically modeled using the Object Constraint Language (OCL).[5] We return to a discussion of constraints and object-oriented systems analysis and design in Chapter 10.

Profiles *Profiles* allow the developer to group a set of model elements that have been extended using stereotypes, tagged values, and/or constraints into a package. Profiles have been used to create modeling extensions that can address specific types of implementation platforms, such as .NET, or specific modeling domains, such as embedded systems. Profiles are simply a convenience. In this text, we do not further describe the use of profiles.

OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN

Object-oriented approaches to developing information systems, technically speaking, can use any of the traditional methodologies presented in Chapter 1 (waterfall development, parallel development, phased development, prototyping, and throwaway prototyping). However, the object-oriented approaches are most associated with a *phased development RAD* methodology. 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, when modeling real-world systems, 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

[5] For more specifics on the OCL, see Jos Warmer and Anneke Kleppe, *The Object Constraint Language: Precise Modeling with UML* (Reading, MA: Addison-Wesley, 1999) and the *UML 2.0 OCL Specification, ptc/03-10-14* (www.uml.org <http://www.uml.org/>).

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 decomposition of problems on objects that contain both data and processes. Both approaches are valid approaches to developing information systems. In this book, we focus only on object-oriented approaches.[6]

According to the creators of UML, Grady Booch, Ivar Jacobson, and James Rumbaugh,[7] any modern object-oriented approach to developing information systems must be (1) use-case driven, (2) architecture-centric, and (3) iterative and incremental.

Use-Case Driven

Use-case driven means that *use cases* are the primary modeling tool to define 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 for the system to the programmers who must write the system.

Use cases are inherently simple because they focus on only one activity at a time. In contrast, the process model diagrams used by traditional structured and RAD methodologies are far more complex because they require the system analyst and user to develop models of the entire system. With traditional methodologies, each business activity is decomposed into a set of subprocesses, which are, in turn, decomposed into further subprocesses, 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, use cases focus on only one activity at a time, so developing models is much simpler.[8] We describe use cases and use case diagrams in Chapter 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 view* describes the external behavior of the system from the perspective of the user. Use cases and use case diagrams are the primary approach used to depict the functional view. Also, in some cases, activity diagrams are used to supplement use cases. The *static view* describes the structure of the system in terms of attributes, methods, classes, and relationships. The structure diagrams portray the static view of an evolving object-oriented information system. The *dynamic view* describes the internal behavior of the system in terms of messages passed among objects and state changes within an object. The dynamic view is represented in UML by behavior diagrams.

[6] See Alan Dennis and Barbara Haley Wixom, *Systems Analysis and Design: An Applied Approach, 2nd Ed.* (New York: Wiley, 2003) for a description of the traditional approaches.

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

[8] For those of you that have experience with traditional structured analysis and design, this will be one of the most unusual aspects of object-oriented analysis and design using UML. Structured approaches emphasize the decomposition of the complete business process into subprocesses and sub-subprocesses. Object-oriented approaches stress focusing on just one use case activity at a time and distributing that single use case over a set of communicating and collaborating objects. Therefore, use case modeling may seem initially unsettling or counter-intuitive, but in the long run this single focus does make analysis and design simpler.

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. Each iteration of the system brings it closer and closer to real user needs.

The Unified Process

The Unified Process is a specific methodology that maps out when and how to use the various UML techniques for object-oriented analysis and design. The primary contributors were Grady Booch, Ivar Jacobsen, and James Rumbaugh of Rational. 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.

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, project management, configuration and change management, and environment. In the remainder of this section, we describe the phases and workflows of the Unified Process.[9] Figure 2-7 depicts the Unified Process.

Unified Process

Four phases

Plan ① inception

② elaboration

③ construction

④ implementation

workflows

business modeling

requirements

analysis

design

implementation

test

deployment

proj mgmt

configuration

change mgmt

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 will vary over the workflows. The curves, in Figure 2-7, 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 information 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 as follows.

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)

[9] 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, 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); Phillipe Krutchten, *The Rational Unified Process: An Introduction, 2nd Ed.* (Boston, MA: Addison-Wesley, 2000).

Planning analysis design implementation

Engineering workflows

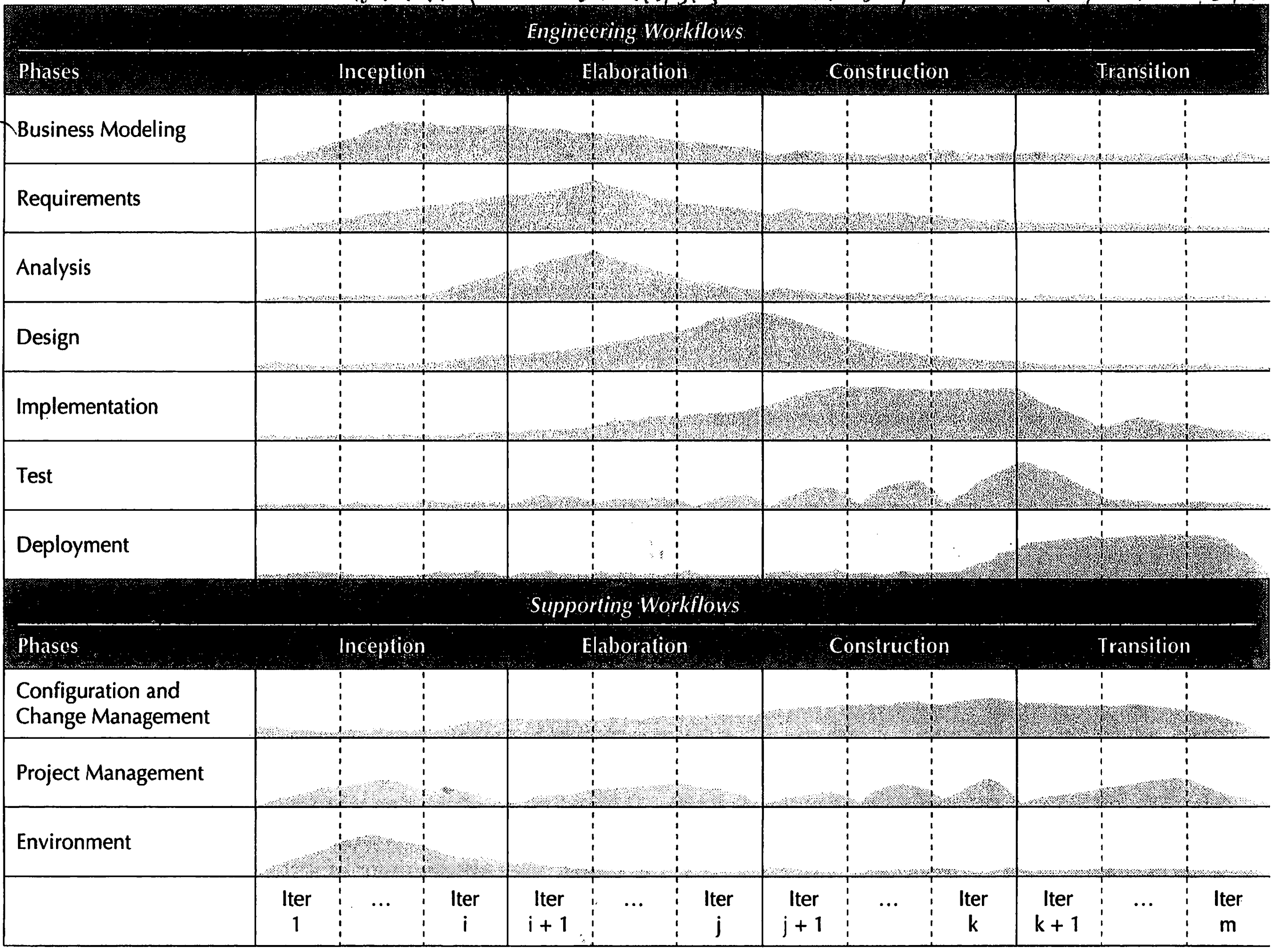

FIGURE 2-7 The Unified Process

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 throw-away prototype is developed. This implies that the design, implementation, and test workflows also could be involved. The project management and environment supporting workflows are very relevant to this phase. The primary deliverables from the inception phase are: (1) 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, and (2) the adoption of the necessary environment to develop the system.

analysis & design

Elaboration When one typically thinks 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 the 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.[10] The primary deliverables of this phase include (1) the UML structure and behavior diagrams and (2) 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 in time, the developers can begin to grow the system toward its completion in the construction and transition phases.

build

Construction The *construction phase*, as expected by its name, is heavily focused on programming the evolving information system. As such, it is primarily concerned with the implementation workflow. However, the requirements, analysis, and design workflows also are involved with this phase. It is during this phase that missing requirements are uncovered, 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 may have 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.

implement/ conversion

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. Depending on the results from the testing workflow, it is possible that some redesign and programming activities on the design and implementation workflows could be necessary, but they should be minimal at this point in time. 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 the actual rolling out of 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. We describe each of the workflows as follows.

Engineering Workflows The *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). Next, we describe each engineering workflow.

producing the technical product

[10] With UML being comprised of fourteen 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 (Computer Aided Software Engineering) tools. We describe the use of these tools in Chapter 3.

- uncovering business problems & getting an understanding of what it takes to solve them

Business Modeling. 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 primarily is 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 are used to understand the business situation.

determining user requirements

Requirements. 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. There are many different ways in which to capture requirements, including interviews, observation techniques, joint application development, document analysis, and questionnaires (see Chapter 5). As you should expect, the requirements workflow is utilized the most during the inception and elaboration phases. The identified requirements are very useful in developing the vision document and the use cases used throughout the development process. It should be stressed that 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.

- understanding the problem
- identify reusable classes

Analysis. The *analysis workflow* predominantly addresses creating an *analysis model* of the problem domain. In the Unified Process, the analyst begins designing the architecture associated with the problem domain, and using the UML, the analyst creates structural and behavioral 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 behavioral 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.* The *design workflow* transitions the analysis model into a form that can be used to implement the system: the *design model.* Where the analysis workflow concentrated 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 evolving information system description by adding classes that address the environment of the information system to the evolving analysis model. As such, the design workflow addresses activities, such as user interface design, database design, physical architecture design, detailed problem domain class design, and the optimization of the evolving information system. The design workflow primarily is associated with the elaboration and construction phases of the Unified Process.
- *Implementation.* 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, testing of the new

classes and their interactions with the incorporated reusable classes must occur. 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 primarily is associated with the elaboration and construction phases.

- *Test.* The primary purpose of the *test workflow* is to increase the quality of the evolving system. As such, 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 are involved during the elaboration and construction phases, while 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 test should be performed.

- *Deployment.* 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 information system into a user organization, the developers may have to convert the current data, interface the new software with the existing software, and provide end user training on the use of the new system.

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

- *Project management.* While the other workflows associated with the Unified Process technically are 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 due to the complexity of the two-dimensional development model of the Unified Process (workflows and phases). This workflow's activities include risk identification and management, scope management, 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.* 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 that includes, for example, diagrams, source code, and executables. During the development process, these artifacts are modified. The amount of work, and hence dollars, that goes into the development of the artifacts is substantial. As such, 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, since 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.* 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 computer aided software engineering tool that supports the development of an object-oriented information system via the UML could be required. Other tools necessary would include programming environments, project management tools, and configuration management tools. The environment workflow includes acquiring and installing these tools. Even though this workflow can be active during all of the phases of the Unified Process, it should primarily be involved with the inception phase.

A MINIMALIST APPROACH TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML 2.0

The UML is an object-oriented modeling language used to describe information systems. It provides a common vocabulary of object-oriented terms and a set of diagramming techniques that are rich enough to model any systems development project from analysis through implementation. Although the UML defines a large set of diagramming techniques, this book focuses on a smaller set of the most commonly used techniques. It should be stressed that UML is nothing more than a notation. Although unlikely, it is possible to develop an information system using a traditional approach with UML. The UML does not dictate any formal approach to developing information systems, but its iterative nature is best-suited to RAD-based approaches such as phased development (see Figure 1-4). A popular RAD-based approach that uses the UML is the Unified Process.

Benefits of Object-Oriented Systems Analysis and Design

So far we have described several major concepts that permeate the object-oriented approach, in general, and the UML 2.0 and Unified Process, in particular, but you may be wondering how these concepts affect the performance of a project team. The answer is simple. 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 system 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 SDLC.

By modularizing system 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.

Finally, many people argue that "object-think" is a much more realistic way to think about the real world. Users typically do not think in terms of data or process; instead, they see their business as a collection of logical units that contain both—so communicating in terms of objects improves the interaction between the user and the analyst or developer. Figure 2-8 summarizes the major concepts of the object-oriented approach and how each concept contributes to the benefits.

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,

Concept	Supports	Leads to
Classes, objects, methods, and messages	▨ A more realistic way for people to think about their business ▨ Highly cohesive units that contain both data and processes	▨ Better communication between user and analyst or developer ▨ Reusable objects ▨ Benefits from having a highly cohesive system (see cohesion in Chapter 13)
Encapsulation and information hiding	▨ Loosely coupled units	▨ Reusable objects ▨ Fewer ripple effects from changes within an object or in the system itself ▨ Benefits from having a loosely coupled system design (see coupling in Chapter 13)
Inheritance	▨ Allows us to use classes as standard templates from which other classes can be built	▨ Less redundancy ▨ Faster creation of new classes ▨ Standards and consistency within and across development efforts ▨ Ease in supporting exceptions
Polymorphism and Dynamic Binding	▨ Minimal messaging that is interpreted by objects themselves	▨ Simpler programming of events ▨ Ease in replacing or changing objects in a system ▨ Fewer ripple effects from changes within an object or in the system itself
Use-case driven and use cases	▨ Allows users and analysts to focus on how a user will interact with the system to perform a single activity	▨ Better understanding and gathering of user needs ▨ Better communication between user and analyst
Architecture centric and functional, static, and dynamic views	▨ Viewing the evolving system from multiple points of view	▨ Better understanding and modeling of user needs ▨ More complete depiction of information system
Iterative and incremental development	▨ Continuous testing and refinement of the evolving system	▨ Meeting real needs of users ▨ Higher quality systems

FIGURE 2-8 Benefits of the Object Approach

operations, or support of the product once it has been delivered. As such, 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 the addition of a Production phase and two workflows: the Operations and Support workflow and the Infrastructure Management workflow (see Figure 2-9).[11] In addition to these new workflows, the test, deployment and environment workflows are modified, and the project management and configuration and change management workflows are extended into the production phase. These extensions are based on alternative object-oriented software

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

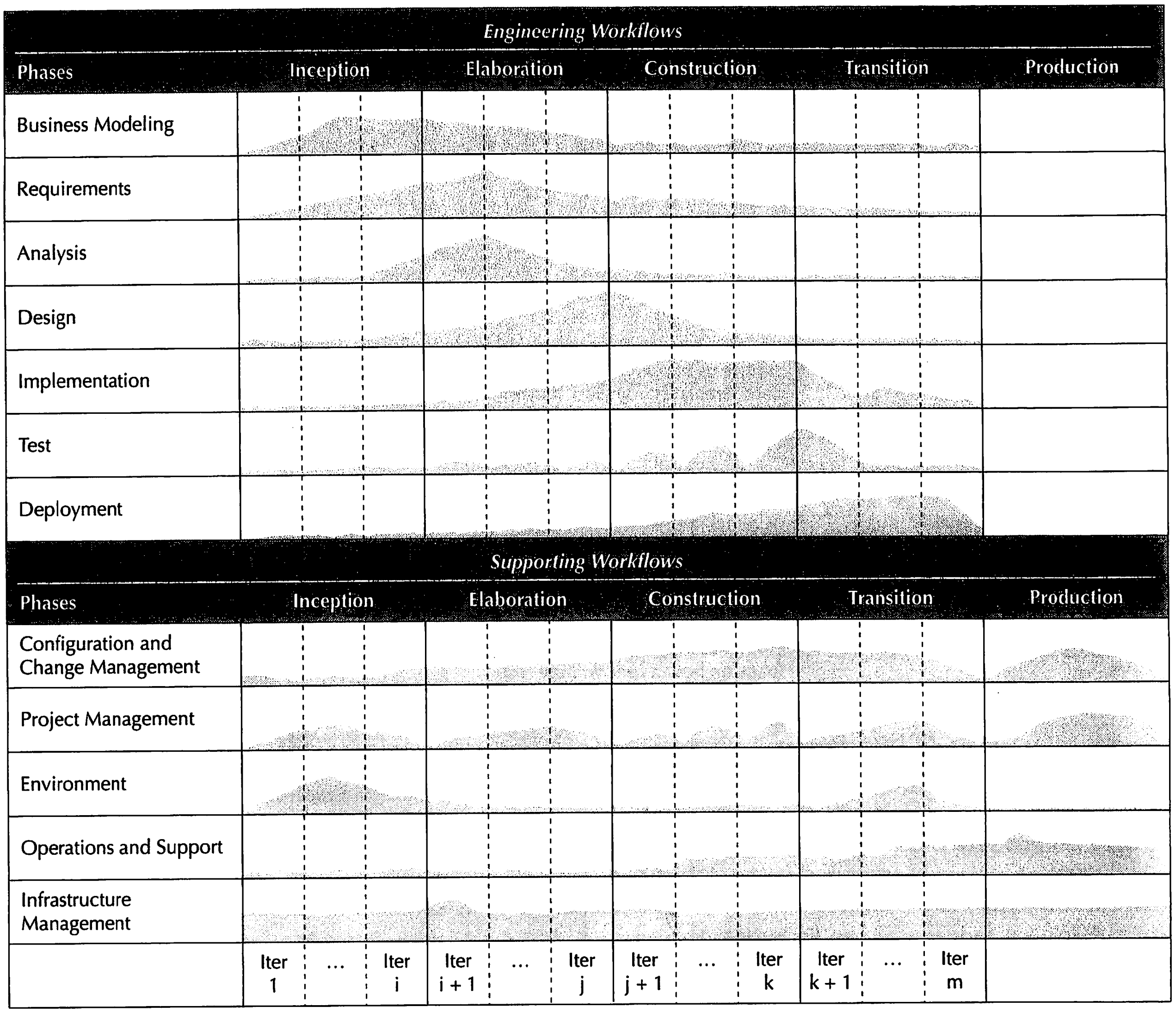

FIGURE 2-9 The Enhanced Unified Process

processes: the OPEN process and the Object-Oriented Software Process.[12] The new phase, new workflows, and the modifications and extensions to the existing workflows are described next.

post implementation

Production Phase The *production phase* is concerned primarily with issues related to the software product after it has been successfully deployed. As you should expect, the phase focuses on issues related to updating, maintaining, and operating the software. Unlike the previous phases, there are no iterations or incremental deliverables. If

[12] 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).

a new release of the software is to be developed, then the developers must begin a new run through the first four phases again. 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 and increases in level of activity 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. As such, as a developer, your work may have just begun.

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 like 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. Since the activities on this workflow tend to affect many projects and the Unified Process only focuses 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. Each of the enhancements is described next.

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

Deployment In most corporations today, legacy systems exist, and these systems have databases associated with them that must be converted to interact with the new systems. Due to the complexity of deploying new systems, the conversion requires significant planning. As such, 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 The environment workflow needed 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 Even though this 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. As such, we suggest extending project management to include these activities. Furthermore, 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 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 these changes have been identified and their impact understood, the developers can schedule the changes to be made and deployed with future releases.

The Minimalist Object-Oriented Systems Analysis and Design Approach

As we stated previously, object-oriented systems analysis and design (OOSAD) approaches are based on a phased-development RAD approach. However, because of the iteration across the functional, static, and dynamic views of the evolving information system, an actual object-oriented development process tends to be more complex than typical phased-development RAD approaches. For example, the two-dimensional model of the Unified Process and the identified extensions is much more complex than a typical phased-development RAD approach (compare Figures 1-4 and 2-9). From a learning perspective, the complexity of the enhanced Unified Process makes putting it into practice in the classroom practically impossible. As such, we have followed a minimalist style[13] of presenting a generic approach to OOSAD. The minimalist OOSAD (MOOSAD) approach that we present in this section is based on the Unified Process as extended by the processes associated with the OPEN Process and the Object-Oriented Software Process approaches to object-oriented systems development. We also have included concepts from XP[14] to help in controlling the complexity of the development process. Finally, due to the size and complexity of the UML, we only use a minimal set of the UML with our minimalist approach.[15]

Figure 2-10 portrays our modified phased-development RAD-based approach. The solid lines in Figure 2-10 represent information flows from one step in our approach to

[13] John M. Carrol, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill* (Cambridge, MA: MIT Press, 1990).

[14] For more information, see K. Beck, *eXtreme Programming Explained: Embrace Change* (Reading, MA: Addison-Wesley, 2000), M. Lippert, S. Roock, and H. Wolf, *eXtreme Programming in Action: Practical Experiences from Real World Projects* (New York: Wiley, 2002), or online at www.extremeprogramming.com. Also, see discussion in Chapter 1.

[15] In many places, the UML is not sufficient for our purposes. For example, the UML does not have any useful diagrams to design user interfaces. In cases where the UML is not sufficient, we will use extensions. However, our overall intention is to minimize both the size and complexity of the UML for learning purposes. For more information on the UML see www.uml.org.

another step. The dashed lines represent feedback from a later step to an earlier one. For example, plans flow from the planning step into the requirements determination and use case development step, and feedback from the requirements determination and use case development step flows back to the planning step.

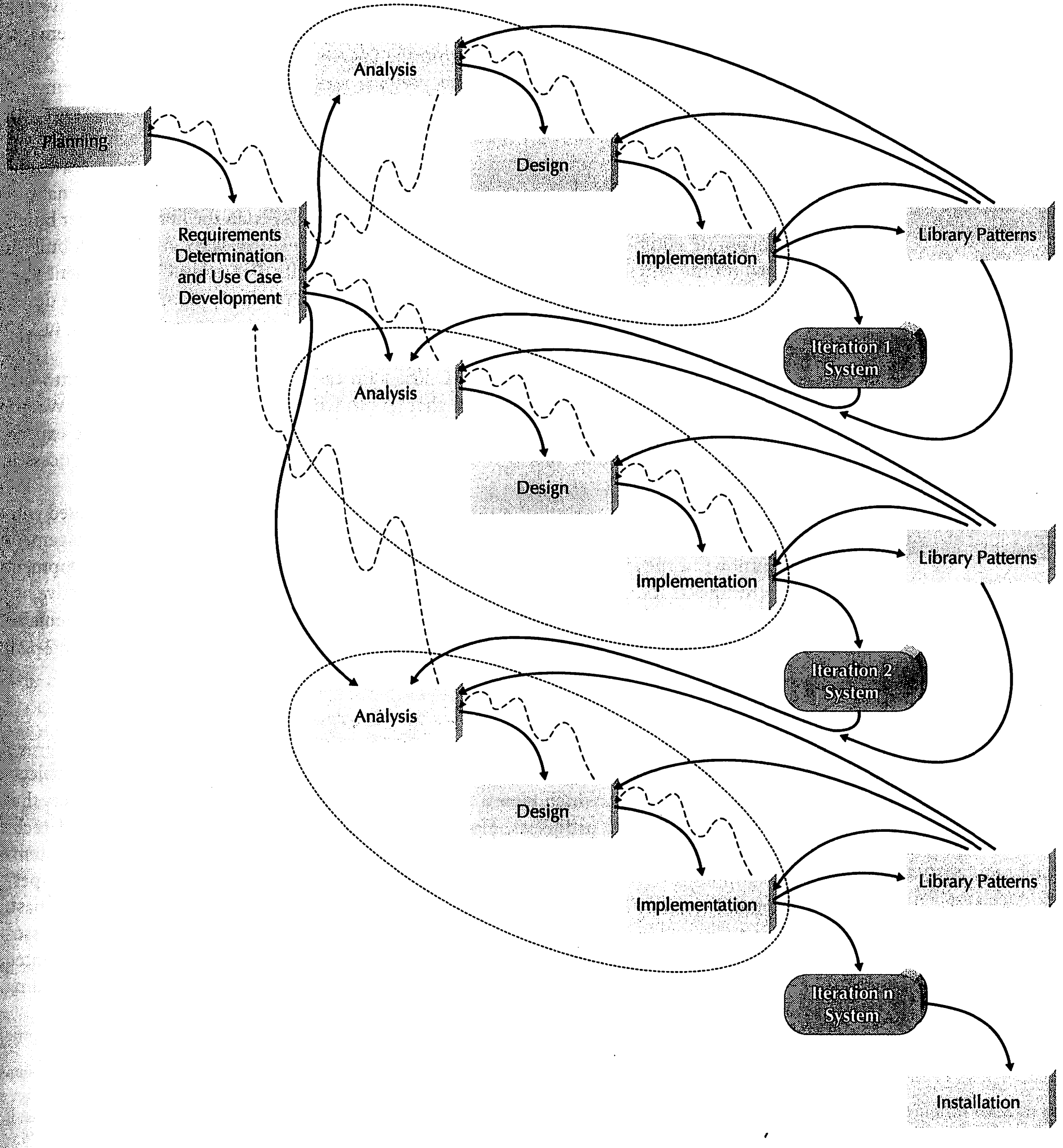

FIGURE 2-10 The Minimalist Object-Oriented Systems Analysis and Design (MOOSAD) Approach

The first step of the approach is *planning*. If you will recall, planning deals with understanding *why* an information system should be built, and determining how the project team will go about building it. The second step of the approach is *requirements determination* and *use case development*. As mentioned above, use cases identify and communicate the high-level requirements for the system (i.e., they provide a functional or external behavioral view of the system). Use cases and the use case model drive the remaining steps in our approach (i.e., all of the information required from the remaining steps in our approach is derived from the use cases and the use case model).

Next, the developers of the system perform a *build*. Each build makes incremental progress toward delivering the entire system. The first build is based on the use cases with the highest priority. Builds are performed until the system is complete. Each build comprises an analysis, design, and implementation step. Each build provides feedback to the requirements determination and use case development step and delivers a functional system to the next build and a set of *patterns* that can be included in the library.[16] Later builds incorporate additional lower-priority and newly identified use cases. Again, each build is based on the remaining use cases with the highest priority. For project management purposes, a build utilizes the idea of *timeboxing*.[17] Typically, in object-oriented systems development, the time frame for each timebox varies from one to two weeks to one to two months, depending on the size and complexity of the project.

Figure 2-11 maps the enhanced Unified Process's phases and workflows into our minimalist approach and the relevant chapters in which the material is covered. We use our minimalist OOSAD approach in this textbook only to simplify the learning experience. You should realize that a two-dimensional model of the development process is more realistic in practice.

As Figure 2-11 shows, in this text we are focused primarily in the area associated with the elaboration phase and the requirements, analysis, design, and project management workflows of the extended Unified Process. 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 remainder of the book is organized around the steps in our MOOSAD approach (see Figure 2-12).

SUMMARY

Today, the most exciting change to systems analysis and design is the move to object-oriented techniques, which view a system as a collection of self-contained objects that have both data and processes. However, the ideas underlying object-oriented techniques are simply ideas that have either evolved from traditional approaches to systems development or they are old ideas that have now become practical due to the cost-performance ratios of modern computer hardware in comparison to the ratios of the past. Today, the cost of developing modern software is composed primarily of the cost associated with the developers themselves and not the computers. As such, object-oriented approaches to developing information systems hold out much promise in controlling these costs.

[16] A pattern is a useful group of collaborating classes that provide a solution to a commonly occurring problem. We describe the use of patterns in more detail in Chapter 7.

[17] Timeboxing sets a fixed deadline for a project and delivers the system by that deadline no matter what, even if the functionality needs to be reduced. Timeboxing ensures that project teams don't get hung up on the final "finishing touches" that can drag out indefinitely. Timeboxing is covered in more detail in Chapter 4.

Enhanced UP Phases	MOOSAD Steps	Chapters
Inception	Planning Requirements Determination & Use Case Development	3–6
Elaboration	Requirements Determination & Use Case Development Analysis Design	5–13
Construction	Implementation	11, 14
Transition	Implementation Installation	14, 15
Production	Installation	15

Enhanced UP Engineering Workflows	MOOSAD Steps	Chapters
Business Modeling	Planning	3, 5–7
Requirements	Requirements Determination & Use Case Development	5–7, 12
Analysis	Analysis	6–8
Design	Design	9–13
Implementation	Implementation	11, 14
Test	Implementation	14
Deployment	Installation	15

Enhanced UP Supporting Workflows	MOOSAD Steps	Chapters
Project Management	Across Steps	3, 4, 6, 15
Configuration and Change Management	Across Steps	3, 15
Environment	Across Steps	4
Operations and Support	Installation	15
Infrastructure Management	Across Steps	4

FIGURE 2-11
The Enhanced Unified Process and the MOOSAD Approach

Basic Characteristics of Object-Oriented Systems

An object is a person, place, or thing about which we want to capture information. Each object has attributes that describe information about it and behaviors, which are described by methods that specify what an object can do. Objects are grouped into classes, which are collections of objects that share common attributes and methods. The classes can be arranged in a hierarchical fashion in which low-level classes, or subclasses, inherit attributes and methods from superclasses above them to reduce the redundancy in development. Objects communicate with each other by sending messages, which trigger methods. Polymorphism allows a message to be interpreted differently by different kinds of objects. This form of communication allows us to treat objects like black boxes and ignore the inner workings of the objects. The idea of concealing an object's inner processes and data from the outside is known as encapsulation. The benefit of these object concepts is a modular, reusable development process.

Chapter	Step	Sample Techniques	Deliverable
3	Identifying Business Value	System Request	System Request
	Analyze Feasibility	Technical Feasibility, Economic Feasibility, Organizational Feasibility	Feasibility Study
4	Develop Workplan Time Estimation	Task Identification	Workplan
	Staff the Project Creating a Project Charter	Creating a Staffing Plan Project Charter	Staffing Plan
	Control and Direct Project Track Tasks Coordinate Project Manage Scope Mitigate Risk	Refine Estimates PERT/CPM CASE Tool Standards List Risk Assessment	GANTT Chart
5	Requirements Determination	Improvement Identification Techniques Interviews JAD Questionnaires	Information
6	Functional Modeling	Activity Diagram Use Cases Use Case Diagram Use Case Point Estimation	Functional Model Use Case Points
7	Structural Modeling	CRC Cards Class Diagram Object Diagram	Structural Models
8	Behavioral Modeling	Sequence Diagram Communication Diagram Behavioral State Machine	Dynamic Models
9	Moving on to Design	Factoring Partitions and Layers Package Diagrams Custom Development Package Development Outsourcing	Factored Models Design Strategy

FIGURE 2-12 Textbook Overview (*Continues*)

Unified Modeling Language

The Unified Modeling Language, or UML, is a standard set of diagramming techniques that provide a graphical representation that is rich enough to model any systems development project from analysis through implementation. Today most object-oriented systems analysis and design approaches use the UML to depict an evolving system. The UML uses a set of different diagrams to portray the various views of the evolving system. The diagrams are grouped into two broad classifications: structure and behavior. The structure diagrams include class, object, package, deployment, component, and composite structure diagrams. The behavior diagrams

Chapter	Step	Sample Techniques	Deliverable
10	Class and Method Design	Reuse Factoring Design Optimization Constraints Method Specification Activity Diagram	Restructured Models Class Design Contracts Method Design
11	Data Management Layer Design	Selecting an Object Persistence Format Mapping Problem Domain Objects to Object Persistence Formats Optimizing Object Persistence Designing Data Access and Manipulation Classes	Object Persistence Design Data Access and Manipulation Design
12	Human Computer Interaction Layer Design	Windows Navigation Diagram Real Use Cases Input Design Output Design	Interface Design
13	Physical Architecture Layer Design	Hardware Design System Software Design Network Design	Physical Architecture Design Infrastructure Design
14	Construction	Software Testing	Test Plan Documentation Completed System
15	Installation	Direct Conversion Parallel Conversion Phased Conversion	Conversion Plan Training Plan
	Operations and Support	Support Strategy Post-Implementation Review	Support Plan

FIGURE 2-12 Textbook Overview (*Continued*)

include activity, sequence, communication, interaction overview, timing, behavior state machine, protocol state machine, and use case diagrams.

Object-Oriented Systems Analysis and Design

Object-oriented systems analysis and design (OOSAD) are most associated with a phased development RAD-based methodology where the time spent in each phase is very short. OOSAD uses a use-case driven, architecture centric, iterative, and incremental information systems development approach. It supports three different views of the evolving system: functional, static, and dynamic. OOSAD allows the analyst to decompose complex problems into smaller, more manageable components using a commonly accepted set of notations. Also, many people believe that users do not think in terms of data or processes, but instead think in terms of a collection of collaborating objects. As such, object-oriented systems analysis and design allows the analyst to interact with the user using objects from the user's environment instead of a set of separate processes and data.

YOUR TURN

2-6 OO Systems Analysis and Design Methodology

Review Figures 2-7, 2-9, 2-10, and 2-11. Based on your understanding of the UP, the EUP, and MOOSAD, suggest a set of steps for an alternative object-oriented systems development method. Be sure that the steps will be capable of delivering an executable and maintainable system.

One of the most popular approaches to object-oriented systems analysis and design is the Unified Process. The Unified Process is a two-dimensional systems development process described with a set of phases and workflows. The phases consist of the inception, elaboration, construction, and transition phases. The workflows are organized into two subcategories: engineering and supporting. The engineering workflows include business modeling, requirements, analysis, design, implementation, test, and deployment workflows, while the supporting workflows comprise the project management, configuration and change management, and the environment workflows. Depending on which development phase the evolving system is currently in, the level of activity will vary over the workflows.

A Minimalist Approach to Object-Oriented Systems Analysis and Design with UML

The Minimalist OOSAD (MOOSAD) approach is based on the processes described in the Unified Process, the Open Process, the Object-Oriented Software Process, and XP approaches to object-oriented systems development. It uses a modified phased-delivery RAD approach to its life cycle. It is also use-case driven, architecture centric, and iterative and incremental. It supports three different generic views of the evolving system: functional, static, and dynamic. Furthermore, it utilizes timeboxes to manage the creation of builds of the system. Finally, it uses UML 2.0 as its graphical notation to support the structural and behavioral modeling of the system.

KEY TERMS

A-Kind-Of (AKO)
Abstract classes
Activity diagram
Analysis model
Analysis paralysis
Analysis workflow
Architecture centric
Attribute
Behavior
Behavior diagrams
Behavior state machine
Build
Business modeling workflow
Class
Class diagram
Communication diagram
Component diagram
Composite structure diagram
Concrete classes
Configuration and change management workflow
Constraints
Construction phase
Deployment diagram
Deployment workflow
Design model
Design workflow
Dynamic binding
Dynamic view
Elaboration phase
Encapsulation
Engineering workflows
Environment workflow
Functional view
Inception phase
Incremental
Infrastructure management workflow
Implementation workflow
Information hiding
Inherit
Inheritance
Instance
Interaction diagram
Interaction overview diagram
Iterative
Message
Method
Object
Object diagram
Object management group (OMG)
Object-oriented approach
Object-oriented methodologies
Operations and support workflow
Package

Package diagram
Pattern
Phased development RAD
Phases
Planning
Polymorphism
Production phase
Profiles
Project management workflow
Protocol state machine
Requirements determination
Requirements workflow
Sequence diagram
Simula
Smalltalk
State
State machine
Static binding
Static view
Stereotypes
Structure diagrams
Subclass
Superclass
Supporting workflows
Tagged Values
Test workflow
Timeboxing
Timing diagram
Transition phase
Unified Modeling Language (UML)
Use case
Use case development
Use case diagram
Use-case driven
Workflows

QUESTIONS

1. Describe the major elements and issues with an object-oriented approach to developing information systems.
2. What is the difference between classes and objects?
3. What are methods and messages?
4. Why are encapsulation and information hiding important characteristics of object-oriented systems?
5. What is meant by polymorphism when applied to object-oriented systems?
6. Compare and contrast dynamic and static binding.
7. What is the Unified Modeling Language?
8. Who is the Object Management Group?
9. What is the primary purpose of structure diagrams?
10. For what are behavior diagrams used?
11. What is a use case?
12. What is meant by use-case driven?
13. Why is it important for an OOSAD approach to be architecture centric?
14. What does it mean for an OOSAD approach to be incremental and iterative?
15. What are the phases and workflows of the Unified Process?
16. Compare the phases of the Unified Process with the phases of the waterfall model described in Chapter 1.
17. What are the benefits of an object-oriented approach to systems analysis and design?
18. Compare and contrast the typical phased delivery RAD SDLC with the modified-phased delivery RAD SDLC associated with the OOSAD approach described in this chapter.
19. What are the steps or phases of the minimalist OOSAD approach described in this chapter?
20. What are the different views supported by the minimalist OOSAD approach described in this chapter?
21. What diagrams and models support the different views identified in the previous question?
22. What is a build?
23. What is a pattern?
24. How do the Unified Process's phases and workflows map into the steps of the minimalist OOSAD approach described in this chapter?

EXERCISES

A. Investigate the Unified Modeling Language on the Web. Write a paragraph news brief describing the current state of the UML. (*Hint:* A good Web site to begin with is www.uml.org.)

B. Investigate Rational's 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. (*Hint:* A good Web site to begin with is www.rational.com.)

C. 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. (*Hint:* A good Web site to begin with is www.omg.org.)

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

E. Suppose you are a project manager that 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 your office requesting you to switch. What do you say?

F. Suppose you were 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 traditional approaches described in Chapter 1? Why?

G. Suppose you were an analyst developing a new information system to automate the sales transactions and manage inventory for each retail store in a large chain. The system would be installed at each store and exchange data with a mainframe computer at the company's head office. Would you use the Unified Process to develop the system or would you prefer one of the traditional approaches described in Chapter 1? Why?

MINICASES

1. Joe Brown, the president of Roanoke Manufacturing, requested that Jack Jones, the MIS department manager, to 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 (www.ibm.com) now supports Java and Lynix on their mainframes. Furthermore, Jack learned that IBM owns Rational (www.rational.com), the creator of the UML and the Unified Precess. As such, they 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 Lynix machines on his current mainframe as a way to support the movement 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 Lynix as a basis for developing and deploying the new system on Roanoke Manufacturing's current mainframe.

2. While recently attending a software development conference, Susan Brown, a systems analyst at Staunton Consulting, was exposed to object-oriented approaches to developing software. Based on what she learned, she feels that object-oriented approaches are a must for her firm to survive in the consulting business in the future. the advantages of using object-oriented approaches seem to vastly out weigh the benefits of remaining with the software development methods used by Staunton Consulting. However, even though she has 15 years of software development experience, she feels somewhat overwhelmed with the complexity of using UML 2.0, with its 14 diagrams, and the Enhanced Unified Process.

   Assuming that you are a close friend that has been using your own firm's UML 1.3 based object-oriented systems development method for the past 5 years, how would you help Susan overcome her concern with the complexity of UML 2.0 and the Enhanced Unified Process? Furthermore, it would be helpful if you helped her create a short checklist of characteristics of software development projects that she could use to base her recommendation for her firm to switch to an object-oriented systems development approach that used UML 2.0.