1

Introduction to the UML

Topics Covered in This Chapter

What Is the Unified Modeling Language (UML)?

The Unified Modeling Language (UML) is the standard visual modeling language used for modeling businesses, software applications, and system architectures. Although the UML is a standard of the Object Management Group (OMG—http://www.omg.org/), the UML is not just for modeling object-oriented (OO) software applications. The UML is a graphical language that was designed to be very flexible and customizable. This enables you to create many different types of models, including models for understanding business processes, workflow, sequences of queries, applications, databases, architectures, and more.

Where Did the UML Come From?

To understand the UML, it helps to know its origins. During the late 1980s and into the 1990s, many object-oriented modeling techniques were being developed to model software. Because different people developed these approaches using different visual modeling techniques and notations, the world of application modeling was becoming divided. To further complicate matters, some techniques were designed just for application modeling, and others were targeted at specific areas such as database design. Some leveraged the strengths of the others, but some remained distinct.

Three of these methodologies began to lead the pack in the marketplace. While working for General Electric, Jim Rumbaugh created the Object Modeling Technique (OMT). Ivar Jacobson developed his Object-Oriented Software Engineering method (a.k.a. the Objectory Method), primarily supporting the telecommunicationsindustry in Sweden. Grady Booch developed the self-named Booch Method. Each had their strengths and weaknesses, and each had somewhat different followings.

In the mid 1990s, Rational Software hired Jim Rumbaugh to join Grady Booch and combine their modeling methods into what became version 0.8, the first public draft of what was then called the Unified Method. In 1995, Jacobson joined Rumbaugh and Booch at Rational. Together, they developed version 0.9 of the Unified Method in 1996. Other companies joined Booch, Rumbaugh, and Jacobson as part of the UML Consortium. In 1997, they submitted version 1.0 of the Unified Method—renamed as the Unified Modeling Language, or UML—to the OMG. As an independent standards body, the OMG took over the UML development and released subsequent versions of the UML (see Figure 1-1). This year (2004), the final approval of the latest version, UML 2.0, is expected.

What makes the UML different from the independent notations we mentioned earlier is that the UML is the creation not of just Booch, Rumbaugh, and Jacobson, but also of many industry experts, software development tool companies, corporate software development organizations, and others. So began the worldwide standard modeling language of software development.

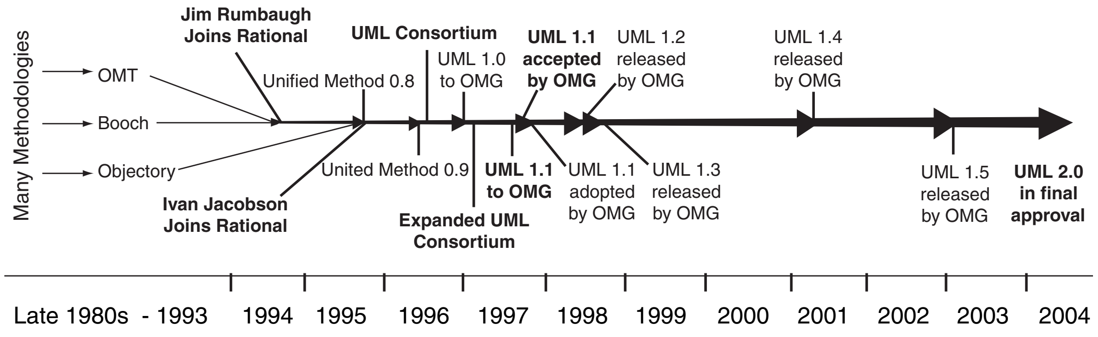

Figure 1-1 *History of the UML.*

Is the UML Proprietary?

As you decide whether to use the UML to model, one of the main things you need to consider is whether other people who join your organization will be able to understand what you have done and whether it will be communicated unambiguously. Both are good reasons for wanting to select a modeling language that is in the public domain and that is understood around the world.

As we discussed earlier in this chapter, the UML was designed because the different modeling languages that were available at the time were leading to a divergence in the ways to model. However, bringing the three major methods together wasn't quite enough. That is why Rational sought out the involvement of organizations such as IBM, Oracle, Platinum Technologies, and many others to be UML partners in the creation of the UML. They then handed development to the OMG to ensure that the UML would become a standard.

As a result, the UML is *not* proprietary. It is an open modeling standard designed and supported by software companies, consultants, other corporations, and governments who need and rely on this standard.

Although a standard open language is critical to protect you from being locked in by the whims of a technology vendor, having a modeling language that is flexible also is key. As technologies and businesses change, so does the way you model. The UML has conventions built into it that enable you to customize it as needed. These customized versions are created using "stereotypes." You will learn more about them in Chapter 5, "Application Modeling."

Is the UML Only for Object-Oriented Development?

We travel the world talking about modeling and the UML. When we begin to discuss using the UML for business or data modeling (both of which we cover in later chapters), one of the first questions we hear is, "How would I use the UML for that? Isn't it only for object-oriented development?" This is one of the biggest myths we run across. The myth comes from the reality that the UML was devised to satisfy the need to model object-oriented systems and to enable Component-Based Development (CBD). In an OO system, generally several components are tied together using what are called "interfaces." To understand how those different components interact, it is quite useful to build a model.

Although the UML was originally built for this cause, it also was built with other needs in mind. Grady Booch once told us that when he and his colleagues were designing the UML, they based a lot of what they did on the different database modeling techniques already being used in the industry. Similarly, one of the strengths of Jacobson's Objectory Method was its business modeling capability. So when they added elements of Jacobson's Objectory Method to the UML mix, they added business modeling to UML.

Today, you can model almost anything you want to in the UML by using its built-in extension and customization capabilities. The UML features an underlying meta-model (see the "Deep Dive" sidebar on meta-models later in this chapter) that enables the UML to be flexible enough so that you can do what you need to do with it. We have seen the UML used for modeling businesses, data, organizations, theoretical political systems, legal contracts, biological systems, languages, hardware, non-object-oriented application modeling such as COBOL, and many other modeling tasks.

Is the UML a Methodology?

Methodology:

"Methodology n.

1a. A body of practices, procedures, and rules used by those who work in a discipline or engage in an inquiry; a set of working methods: *the methodology of genetic studies; a poll marred by faulty methodology*.

b. The study or theoretical analysis of such working methods.

2. The branch of logic that deals with the general principles of the formation of knowledge.

3. *Usage Problem*…

Methodology can properly refer to the theoretical analysis of the methods appropriate to a field of study or to the body of methods and principles particular to a branch of knowledge. … In recent years, however, *methodology* has been increasingly used as a pretentious substitute for *method* in scientific and technical contexts, as in *The oil company has not yet decided on a methodology for restoring the beaches*. … But the misuse of *methodology* obscures an important conceptual distinction between the tools of scientific investigation (properly *methods*) and the principles that determine how such tools are deployed and interpreted." [AMER1]

This very typical definition of the term *methodology* explains that a methodology is much more than a language. You can see from the "usage problem" discussed in this definition how this can confuse some people who are new to the UML. The UML is a language. Object-oriented analysis and design (OOAD) is a process, governed by specific practices. Although languages, including the UML, have rules for syntax and usage, they do not have procedures (i.e., processes) or practices. A methodology must include these things as well. So, although a common language is needed in a specific discipline, language alone does not make a methodology. This is true for the UML as well. Thus, you can use the UML with various methodologies, but it is not a methodology itself.

What Is Happening Now with the UML?

As of this writing, the UML is in the final stages of approval for its latest revision, version 2.0. The OMG has been developing this version of the UML for many years. It combines the efforts of more than 100 organizations, bringing together the best practices they developed over the first few versions of the UML as well as needs they identified for the future.

Along with enhancing the UML infrastructure, adding new modeling capabilities, and enabling the easier exchange of models (i.e., between tools or systems), one of the OMG's main goals when developing UML 2.0 was to make it more extensible to accommodate present as well as future needs. For example, one long-standing need that is being addressed is the use of the UML to model embedded systems. (Unlike general-purpose systems such as desktop computers, embedded systems are special-purpose systems such as pacemakers, automotive braking systems, digital cameras, cruise missiles, mobile phones, and so forth that contain hardware and software designed to perform specific functions.) Typically, you would model embedded systems using different languages. But in the on-demand world of today, where you need to link your embedded systems with business systems in your organization, you need to understand how everything works together. This is greatly simplified if you model everything in the same language because it enables you to share information across different types of technologies and different modeling efforts. Prior to version 2.0, the UML provided some of this capability, but the additions the OMG made to the language in version 2.0 have greatly increased this capability.

What Is a Model?

Good question.

Model:

1. A miniature representation of an object.
2. A pattern on which something not yet produced will be based.
3. A design or type.
4. One serving as an example to be emulated or imitated.

Modeling:

1. To construct a plan, esp. after a pattern. [WEBS1]

You are surrounded by models every day. As you get ready for work in the morning, you turn on your television. The morning news presents the local weather map showing how rain will be moving in this weekend. You reach for your vitamins. The cap is imprinted with a diagram showing how to remove the child-proof (or should we say adult-proof) cap.

You complete your morning ritual and jump into your automobile for the always-pleasant commute to the office. You stop to pick up some breakfast. You notice that behind the counter is a laminated poster containing a series of pictures (no words) showing the fast-food staff how to assemble your breakfast sandwich. Then, back on the road you go.

Half way to the office, you hear a traffic report of an accident ahead, so you pull off the road and consult your road atlas for an alternate route. You stay alert for the overhead highway signs with the large arrows indicating the lanes of traffic and which lane turns into the alternate exit you are searching for. You arrive at the office. As you stroll through the lobby, you see the glass-enclosed 3D replica of the new corporate headquarters that is under construction. You make it to your cubicle just in time to attend a morning meeting where a benefits person is showing bar charts of how much your money can earn if you invest it in the company retirement plan.

The weather map (a model or representation of the weather), the child-proof cap (a process model of how to remove the cap), the laminated sandwich instructions (another process model), the road atlas (an abstracted model of the road), the highway sign (a directional model of the highway), the corporate headquarters replica (a physical model of the building and surrounding terrain), and the bar chart (an analytical model) all model various aspects of your world.

A model, in the sense that we will discuss in this book, is a visual way of depicting your business, its rules, the use of your systems, applications, and system architectures, and interactions within your systems. You might have seen this described as "visual modeling," a term made popular by many computer-aided software engineering (CASE) vendors over the past couple of decades. Because models don't necessarily have to be visual (they can be textual or mathematical, for example, as we saw earlier), visual modeling has come to describe exactly what it implies: models that are visual in nature, using specific graphical representations.

In the remainder of this chapter, you will learn about the value of modeling, specifically when designing software and applications. First, you will learn why models should be built and then who should be building those models. We also introduce the different types of UML models that you can build.

Why Should I Build Models?

One of the most frequent objections to the UML we have heard isn't about the UML itself; it's that some people believe that there is no value in modeling. One theme you will pick up throughout this book is that modeling for modeling's sake is of little value, but when you do it for the right reasons, modeling is quite valuable. Modeling helps you to communicate designs, quickly clarify complex problems and scenarios, and ensure that your designs come closer to reality prior to implementation. This can save you and your organization a lot of time and money, and it enables teams of people (be they teams of 2 or 2,000) to work together more effectively and ensure that they are working toward the same common goals.

Think of it this way. Would you build a house without a plan? You might not build a small-scale version of the house first, but you (or your architect and builder) certainly would have design sketches, architecture drawings, and engineering assessments in hand prior to starting construction.

From the Real World—Houses Are Similar to Software

Before embarking on my software development career, I spent several years building houses. During that time, I learned that when you use a good model (design), the house will stand through time and different weather conditions and will be amicable to changes that might be needed later, such as adding a window. If you don't use a good model but instead use a design you have in your mind, there is a good chance your final product won't be what you intended.

Designing software brings similar challenges. You need to ensure that the designs and plans are eventually realized and realized correctly so that they last architecturally through time, and you must also ensure that if you do need to make changes, they too can endure over time. Also, you need to understand what changes will break your design. When you add a window to a house, for instance, you need to know where the pipes and wires are and what types of support exist to handle the load. Likewise, when you add a component to a system, you need to make sure the system won't come crashing down as a result.

Lessons Learned

1. Design so that you can accommodate unplanned changes.
2. Create a well-documented design so that others who are new to the design can still work with it.
3. Have a visual model of an architecture to help you determine implications of change.

Watch Out

Do not model for modeling's sake. Make your models actionable so that users understand why you created the models and what you expect should result from them in the future.

Also, be warned! Watch out for "analysis paralysis." This occurs when you spend too much time analyzing a problem and don't get to the point where you are being productive.

When you focus on making your analysis or modeling actionable, you set a plan as to what you want to achieve from your analysis. You also adopt a mindset to approach modeling without rushing, making it of value, and then moving on to the next thing—be it another model, a change to the business or business process, or something else such as writing code.

Another reason you should model is simply to understand your business and its processes. You model your business processes not only to understand what the business does and how it functions but also so that you can identify how changes will affect the business. Modeling the business helps you to identify strengths and weaknesses, identify areas that need to be changed or optimized, and in some cases, simulate different business process options.

Why Should I Model with the UML?

When we explain why people should model with the UML, we like to draw a parallel to the field of electrical engineering. Electrical engineers draw a schematic for an electrical device in a standard way (see Figure 1-2), using a common visualization so that no matter where you are in the world, the schematic is always interpreted in the same way. Anyone who is trained in reading such a diagram can easily understand the circuit design and how the device will function. Even those who might not be experts but who understand the symbols can still understand what part belongs where and how they are connected.

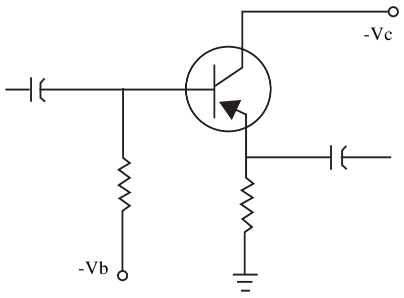

Figure 1-2 *An electrical schematic.*

Other walks of life have similar languages or notations that are specific to their disciplines. The field of music, for instance, has a standard notation (see Figure 1-3). This was a critical development in the musical field. Before this notation was developed, the only way a composer could correctly and consistently teach musicians how to play his music was in person (a conundrum that closely parallels the software industry before the development of the UML).

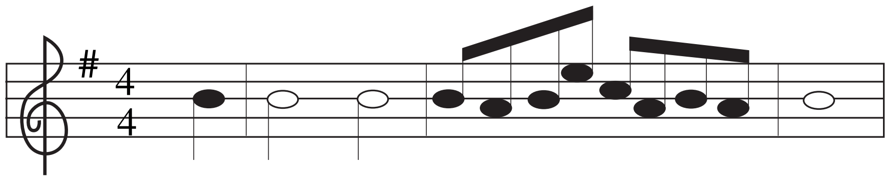

Figure 1-3 *Musical notation.*

Similarly, mathematics has its own specific notation (see Figure 1-4). In fact, the language of mathematics is often cited as the one common language that all advanced civilizations understand.

$$F(z) = [1/(2\pi)^{0.5}] \int_{-\infty}^{z} e^{-(\frac{1}{2})t^2}\, dt$$

Figure 1-4 *Mathematical notation.*

The same is true for the UML. It provides a common "language" to bring together business analysts, software developers, architects, testers, database designers, and the many other professionals who are involved in software design and development so that they can understand the business, its requirements, and how the software and architectures will be created. Although a cellist might not understand how to play the trumpet, she will understand what notes a trumpeter plays and when because she understands musical notation. Similarly, a business analyst who knows the UML can understand what a programmer is creating using the UML because the UML is a common language. With the ongoing need to think globally when building software, the ability provided by the UML to communicate globally becomes very important.

From the Real World—Not Lost in Translation

We were recently working with a large financial institution. They began to outsource much of their software development for a specific project. In an effort to keep costs low, they chose a well-respected international systems integrator to run the job, but they also wanted to maintain control of the system that was being built, so they kept their own architects on the project. Because the company developed the system in this way, rather than doing everything themselves, senior management had the comfort of knowing that the project would fulfill the specified requirements while staying within budget and time constraints

The financial institution had the people who understood their business. They had the domain expertise and the contacts within the business to gather and verify the requirements as needed. Their initial concerns

continues

From the Real World—Not Lost in Translation (*continued*)

centered on the ability to transfer information between the different organizations and across international borders. That information had to be well understood and couldn't be allowed to slow down the process. Understanding this prior to the request for proposal, the financial institution included that the contractor must use a UML-based modeling tool and follow specific quality processes as part of their contract. They found quickly that having a common language to interpret requirements and architectures enabled them to understand each other without having to translate designs and desires.

Lessons Learned

1. Having a common way to understand what needs to be built helped this team of different organizations, languages, and countries communicate effectively to successfully deliver an application that was on time and, more importantly, that met the requirements of their end users.

What Can I Model with the UML?

The UML enables you to model many different facets of your business, from the actual business and its processes to IT functions such as database design, application architectures, hardware designs, and much more. Designing software and systems is a complicated task requiring the coordinated efforts of different groups performing various functions: capturing the needs of the business and systems, bringing software components together, constructing databases, assembling hardware to support the systems, and so on.

You can use the different types of UML diagrams (summarized later in this chapter) to create various types of models. Figure 1-5 lists these models and their usage. The models are composed of different diagram types, model elements, and linkages between model elements that enable you to trace between them so that you can understand how they relate. Different people in the organization use these models to describe different information. As we continue throughout this book, we will elaborate on these models, their use, and the roles of the people who will be using these models.

Model Type	Model Usage
Business	Business processes, workflow, organization
Requirements	Requirements capture and communication
Architecture	High-level understanding of the system being built, interaction between different software systems, communicate system designs to developers
Application	Architecture of the lower-level designs inside the system itself
Database	Design the structures of the database and how it will interact with the application(s)

Figure 1-5 *Model types and usage.*

Who Should Build Models?

Not everyone should be involved in building models, but that doesn't mean everyone can't take advantage of the models that are built. In software design and development, you should start with models that help you understand the business and end with models designed to test your applications (and repeat this process with each software iteration thereafter). A model should be a living item that continues to be updated as the business and system are updated. The model should provide understanding, communication, and direction. If it isn't updated throughout the entire software development process, it can become stale and useless, so it's imperative that your organization has a process for dealing with models and with overall model development that includes who should be creating, updating, and maintaining the models over time.

Your organization also needs to take advantage of its staff as well as external resources. An organization typically has many domain experts, often in a business analyst-type role. These people would be building and designing business models of what is in place today (as is) and where the organization is heading (as desired). These experts often must build application models at the architectural level. As such, they must understand how what you are building will interact with itself (components within itself) and with other systems inside the organization. Architects also might take on this role as well.

As an application developer, writing code is what you like to do. But writing code without designing how it will interact can be dangerous to the integrity of the system, and that is why developers should also model the code before they write it. If they are using certain tools that enable code generation from the models developed, you can automate the creation of the boring code development tasks and get down to doing the fun development work of implementing business- and technology-specific code.

As a tester, you might not get involved directly in model building, but as you will learn later in this book, understanding the models can be quite useful when you have to create your tests. If testers are following an Extreme Programming (XP) style of software development process, they might also get involved in creating the models. XP proclaims that development requirements come from test cases that are created prior to the start of coding. That approach is a bit different from other processes in which test cases are created *from* requirements and aren't considered *the* requirements. This means that when testers follow an XP process, they also are really designing the requirements and thus modeling them (textually), which ensures a better understanding by everyone else involved in the development process.

From the Real World—Flying Toward Success

A large airline company I worked with had a great way of deciding who would model and how to avoid the issue of too many people modeling. The company created teams of people from different parts of the business, including information technology (IT). You might have heard the term of "two in a box," referring to putting two people with different skills together so that they can team up and leverage each other's strengths. This company put more than two people together, but they got similar results. They teamed up to design first-level models (called domain models), in which they defined different elements such as the many different types of agents involved within the airline industry and how they interacted with the different parts of the business. More than 10 different types of agents were in their organization, including travel, ticket, gate, etc. By coming together as a team, they leveraged the different types of expertise available and ensured that all constituents involved in the process agreed to both the terminology and how each agent was involved in the business process. As the book continues, we will examine other successes seen in this organization.

What Is a Diagram?

Diagram:

1. A plan, sketch, drawing, or outline designed to demonstrate or explain how something works or to clarify the relationship between the parts of a whole.
2. *Mathematics*. A graphic representation of an algebraic or geometric relationship.
3. A chart or graph. [DICT1]

For our purposes in this book, a *diagram* is the layout and visualization of different modeling elements as described within the UML. Each UML diagram is used for a specific purpose, typically to visualize a certain aspect of your system (summarized next). Each diagram uses specific UML symbols to achieve its purpose.

What Diagrams Are in the UML?

The UML contains two different basic diagram types: structure diagrams and behavior diagrams. Structure diagrams depict the static structure of the elements in your system. The various structure diagrams are as follows:

- **Class diagrams** are the most common diagrams used in UML modeling. They represent the static things that exist in your system, their structure, and their interrelationships. They are typically used to depict the logical and physical design of the system.
- **Component diagrams** show the organization and dependencies among a set of components. They show a system as it is implemented and how the pieces inside the system work together.
- **Object diagrams** show the relationships between a set of objects in the system. They show a snapshot of the system at a point in time.
- **Deployment diagrams** show the physical system's runtime architecture. A deployment diagram can include a description of the hardware and the software that resides on that hardware.

UML 2.0 adds the following structure diagrams:

- **Composite structure** diagrams show the internal structure of model elements.
- **Package diagrams** depict the dependencies between packages. (A package is a model element used to group together other model elements.)

Behavior diagrams depict the dynamic behavior of the elements in your system. The various behavior diagrams are as follows:

- **Activity diagrams** show the flow of activities within a system. You often would use them to describe different business processes.
- **Use case diagrams** address the business processes that the system will implement. The use cases describe ways the system will work and who will interact with it. [BOOCH1]
- **Statechart diagrams** show the state of an object and how that object transitions between states. A statechart diagram can contain states, transitions, events, and activities. A statechart diagram provides a dynamic view and is quite important when modeling event-driven behavior. For example, you could use a statechart diagram to describe a switch in a telephone routing system. That switch will change states based on different events, and you can model those events in a statechart diagram to understand how the switch behaves. In UML 2.0, these are called **state machine diagrams**.
- **Collaboration diagrams** are a type of interaction diagram, as are sequence diagrams (along with others in UML 2.0, noted next). The collaboration diagram emphasizes how objects collaborate and interact. In UML 2.0, the equivalent of the collaboration diagram is the **communication diagram**.
- **Sequence diagrams** are another type of interaction diagram. Sequence diagrams emphasize the time ordering of messages between different elements of a system.

UML 2.0 adds the following behavior diagrams:

- **Timing diagrams** are another type of interaction diagram. They depict detailed timing information and changes to state or condition information of the interacting elements.
- **Interaction overview diagrams** are high-level diagrams used to show an overview of flow of control between interaction sequences.

The UML 2.0 diagrams that are relevant to mere mortals will be discussed further in Chapter 8,"Is That All There Is?."

Although these diagrams are the defined diagrams of the UML, you may encounter additional diagrams that tool vendors can create, which are specific to their tools. You need not be concerned about the proliferation of additional UML diagrams. Few people even use all of the standard UML diagrams when modeling their systems. Some you may never use. In subsequent chapters, we will focus on those diagrams that are most important and that will most frequently be encountered by mere mortals.

What Is the Difference Between Diagrams and Models?

Although they might appear to be similar at first glance, a diagram and a model are different from each other. A model is an abstraction that contains all the elements needed to describe the intention of the thing being modeled. This can include all aspects concerning the business, systems, relationships, and more. A diagram is a specific view into what you are trying to understand in a specific context. Diagrams are just one way of looking at all or some part of that model. You can have a specific modeling element exist only once within a model, but the same element can appear on one or more diagrams.

For example, if I am modeling a navigational system for a vehicle, the system model will represent all the parts of the entire system. However, I may create specific diagrams that contain just the elements that deal with the map display, error correction, or the navigational satellite constellations.

Deep Dive—Meta-Models

A meta-model is a model of a model. The UML meta-model expresses the proper semantics and structure for UML models. A UML model is made up of many different elements. The meta-model defines the characteristics of these elements, the ways in which these elements can be related, and what such relationships mean.

What does this mean in plain English? Let's say you want to model various types of people. All people have some common characteristics (e.g., height, weight, eye color). But people also have characteristics and behaviors that are specific to each of them. A construction worker, a dancer, and an engineer are all types of people, but they are different from each other in some ways. You can model this in the UML because the structure of the UML meta-model allows you to model relationships between specific things (construction worker, dancer, engineer) and a general thing (a person)—this is called a "generalization" relationship. In other words, the UML meta-model sets the rules for how you can model.

The UML meta-model is also the foundation for UML's extensibility. Using the definitions of UML elements in the meta-model, you can create new UML modeling elements. You can add additional properties to the new elements. This allows you to give the new element additional characteristics and behaviors for your specific needs, while it still remains compliant with the structure and semantics of the original element that it was based upon. In this way, users can customize the UML to their specific needs.

Most technical languages, including Structured Query Language (SQL, the relational database language) and Business Process Modeling Language (BPML), have a meta-model. Different tool vendors may alter the standard meta-model to fit their tool or to differentiate it from their competition. In this way, tool vendors take advantage of the standards but also differentiate themselves from each other through these extensions.

Terms

Unified Modeling Language
Methodology
Object Modeling Technique
Objectory method
Object-oriented development
Object Management Group
Booch method
Stereotype

Model
Meta-models
Class diagram
Component diagram
Object diagram
Statechart diagram
Analysis paralysis
Diagram
Activity diagram
Collaboration diagram
Deployment diagram
Sequence diagram
Use case diagram

Summary

In this chapter, we introduced you to the UML. You learned where it came from and how it became the standard modeling language for software development. We dispelled some myths about the UML, particularly concerning object-oriented development and methodology. You learned that the UML does not restrict you in these areas.

We then moved to a brief discussion of why modeling is important, and you learned how the UML provides a common language, just as the standard notations of other domains such as engineering and mathematics do.

We continued this chapter with a discussion of what is currently happening with the UML. There you found that the UML continues to evolve through the effort of hundreds of organizations that are driving improved support for systems and software development. Then, we provided an overview of the value of modeling as it relates to software development, while showing similarities to other types of modeling such as an architectural design of a building.

Moving toward the focus of this book, we began our discussion of the UML and what it is used for. You learned that the UML is a standard language that you can use to communicate software and system designs to keep the entire team on the same page. Although a limited number of team members might build the models, everyone involved in the software development process can use the models. Architects use models to communicate intended architectural designs, customers review business models to ensure that the modelers understand their business needs, developers review models to build the right software as designed by architects and other developers, project managers use models to understand what is being built and to manage schedules, and so forth. Testers can leverage the models to support building test cases to understand how the software is to be used, and to communicate back to the developers on things they see that aren't correct.

Toward the end of this chapter, you began to understand how the different types of elements live within the UML, including the diagram and model types, and what they are used for at a high level. Models consist of multiple diagrams, and the diagrams are a visualization of elements and how they interact with other elements. Throughout this chapter, we stressed the value of using the visualization to ensure that the team works together and shares information wherever possible to manage the success of a project.

Review Questions

1. What does the acronym "UML" stand for?
2. Who controls the UML standard?
3. True or False: The UML is a proprietary standard.
4. What type of systems can you model with the UML?
5. True or False: You can use the UML only for object-oriented development.
6. What methodology do you use when you use the UML?
7. Name three benefits to modeling with the UML.
8. Does a model have to be visual?
9. What is analysis paralysis?
10. True or False: UML models are of value to even small projects of one or two developers.
11. Name two ways to model a business.
12. What is the most commonly used UML diagram?
13. What UML diagram do you use to model workflow?

14. What diagram type do business analysts most commonly use to identify high-level business processes?

You can find the answers to these questions in Appendix B.

[AMER1] *The American Heritage® Dictionary of the English Language, Fourth Edition*. Boston: Houghton Mifflin Company, 2000.

[BOOCH1] Booch, Grady, Rumbaugh, James, and Jacobson, Ivar. *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley Longman, Inc., 1999.

[DICT1] http://dictionary.reference.com/.

[WEBS1] *Webster's II New Riverside Desk Dictionary*. 1988.