The Unified Modeling Language

Objectives ................................................................. 66
Pre-Test Questions ..................................................... 66
Introduction .............................................................. 67
UML Notation ............................................................ 67
Aggregation ............................................................... 68
Generalization .......................................................... 69
Delegation ................................................................. 70
Models ..................................................................... 70
Views ...................................................................... 73
Summary ................................................................. 74
Post-Test Questions .................................................... 74
OBJECTIVES

At the completion of this chapter, you will be able to:

- Describe the history of the Unified Modeling Language (UML).
- Describe the purpose of a modeling language.
- Describe the purpose of models.
- Identify UML views.

PRE-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. What is Unified Modeling Language?

The answer is

2. What is a UML view?

The answer is
The Unified Modeling Language, UML, is a method used to specify all aspects of a software system. UML prescribes syntax for describing the structures of classes, components, programs, and software systems. UML also describes the interactions between these items. While the Unified Process relies heavily on UML, UML was developed separately from it. UML was designed to be used with any iterative software development process.

The object-oriented movement began with the introduction of Smalltalk in the early 1980s. By the late 1980s, work had begun on developing methods and notations to describe object-oriented software systems. By the early 1990s, a wide variety of object-oriented development methods existed, each with its own notation and modeling language. While working at Rational Software Corp., developers Booch, Jacobson and Rumbaugh created UML in an attempt to unify the notations each had created earlier. In November 1997, the Object Management Group (OMG) adopted UML as a standard.

UML Notation

UML is used to model a system as a set of static structures and dynamic behaviors. Static structures are the classes and components that comprise a system. Booch notation was introduced earlier in the course and is used to specify class types and class relationships. UML has its own notation, UML class diagrams that specify the static structures of a software system.

The dynamic behavior of a system describes the steps in completing a task. The dynamic behavior can be viewed from two levels: the life cycle of a single object from instantiation to release, and the interaction of objects within a system. UML statechart diagrams are state diagrams of a single object. Sequence diagrams and collaboration diagrams describe the interactions between objects.
AGGREGATION

Aggregation is used in UML to describe part-of relationships. This includes relationships where the different components are related so that they are aggregated into a separate whole. As an example, a personal computer contains a processor, RAM and a hard disk. A diagram that provides an explanation of aggregation within the PC is below:

![Figure 5-1: Aggregation Representation](image)

As shown, the RAM, hard disk and processor are all parts of a personal computer. These parts aggregate to create a personal computer. The multiplicity indicators show that there must be at least one component and there can be an unspecified number of components per personal computer.

Having a variety of aggregates that make up one object can also be called composition. Composition describes relationships between aggregates, and it also describes what happens when the removal or addition of a certain component takes place. Will this lead to the deletion or addition of other components? This is called cascading. Cascading occurs when adding or removing elements of a composition affects other elements. For instance, if there is a composition that includes employees, payroll and individual expense accounts, then the removal of an employee will also lead to the removal of that employee's payroll entry and the employee's expense account.
Another kind of relationship that is modeled within UML is the generalized relationship. These include a general class of something and a specific class that resides within the general class. In the object-oriented paradigm, this is known as inheritance. These are commonly referred to as supertypes and subtypes. The supertype will include a generalized description, an abstract, of a category of objects. The subtype must conform to all rules and actions that can be performed on the supertype. However, the subtype can have rules and perform actions that the supertype cannot. One example that can be used is an animal. In the following listing, animals are broken down into mammals, which is further broken into whales. The subtype of whales then becomes a supertype for the subtype of blue whales.

1. Blue whales are a type of whale.
2. Whales are a type of mammal.
3. Mammals are a type of animal.

When using generalization, it is important to make sure that all subtypes contain all attributes of the supertype. A good test for the concept of generalization is the ability of any subtype to be inserted where a supertype is used. If the program would run without any other changes, then the generalization worked. This process in object-oriented terms would be polymorphism. Generalization and polymorphism work because, as stated, all subtypes inherit the attributes of the supertype. However, inserting a supertype into code where a subtype is used will lead to problems because there should be attributes of the subtype that are not available for the supertype. If the subtype does not include any attributes not found in the supertype, then the subtype is unnecessary. In other words, there is no need for the subtype if it is the same as the supertype.

Using the example from above, the method "breathe" is written for the animal supertype. If the breathe method is called on the subtype whale, the whale should be able to perform the action. However, the method "swim" that is written for the whale subtype cannot be called on the supertype of animal because not all animals can swim.
DELEGATION

Another way to describe relationships is by delegation. Delegation is the ability of an object to respond to a message that it receives and to issue a message to an object. Consider a business model. When an employee finishes a project she will turn this project in to a manager. That is, the employee object will send a message to the manager object. The manager will then be able to take the message of the finished project and respond by issuing another project to the employee. The employee can then take this new message and begin work on the new project.

The delegation ability of the manager is important because it allows the manager to respond to the employee. An extension of this example would be if the manager turned the finished project over to a quality assurance department. The quality assurance department would be able to respond to the project by beginning the process of editing and testing the project. This would start a new process of delegation between the manager and the quality assurance department.

MODELS

A model is a simplified representation of a complex idea or element. In the same way that a blueprint is a model of a building, UML is a model of a software system or program. The blueprint allows an architect to convey his or her vision to customers and contractors, and UML serves the same purpose within the software domain. It allows a developer to convey his vision to a programming team and management.

UML is a modeling language, and many different UML models can be used to describe the elements of a software system. These elements include system requirements, user interactions, system processing, system design, and system testing. UML is complete enough to model most situations encountered in software development, but it is extensible enough to handle unforeseen modeling issues.

The following examples of modeling levels will describe a trip to a grocery store, beginning with the realization that a house is out of food and ending with the actual purchasing of groceries.
Modeling levels

Various modeling levels are appropriate during the development of a software system. The level of detail included in each model is modified as development progresses from its initial stages through analysis, design, implementation, and testing.

High-level models

High-level models are created during the early stages of development. These models provide a framework for ideas and often diverge in many directions. High-level models are incomplete and will be heavily revised. Their purpose is to simply explore possible solutions.

If the problem is a lack of food, possible solutions include growing a garden, driving to a grocery store, or even stealing food from a neighbor. Some of these solutions are obviously more feasible than others. The point of high-level modeling is to brainstorm in as many directions as possible and to include all possibilities.

Abstract models

Abstract models are developed toward the end of analysis and the beginning of the design phase. These models describe the system architecture and provide an overview of the eventual system design. The solution revealed by the high-level model is elaborated into a more refined model. However, the design is still incomplete, since much of the implementation detail is omitted from abstract models. The goal is to verify the overall system design before designing individual components.

In this phase of modeling, a concise solution has been chosen, and the design of the solution can begin. Certain checks are used to ensure that the solution will work. Assume that, from the possible solutions to the food problem, making a trip to the grocery store was selected. The specifications that would have to be met include:

1. Making sure there is transportation
2. Knowing what to buy
3. Having money to buy the food
Full specification models

During design, full specification models are created to completely describe a software system. Full specification models also describe the process of implementing a system, including the following details:

1. All interfaces to a system
2. All classes and components
3. All interactions between classes and components
4. All business logic

When this stage of modeling is reached, all aspects of the design are required. Continuing the grocery store example, this modeling level would specify:

1. What time they would go
2. Who would be driving to the store
3. Which grocery store would be used
4. What food would be purchased
5. How to pay for the food

Additional models can be developed to serve as partial descriptions or to provide examples of the system in use. A partial model defines a subsystem or some portion of a software system's functionality. The partial model is used to develop reusable components that can be combined with other partial models to create complete systems. An exemplar model describes a specific case of the software system in use. While exemplar models cannot be used to fully describe the general-use software system, examples can increase understanding of a complex system.
UML includes a variety of diagrams that are used in the analysis and design of a system. UML is divided into eight views. Each view is a set of diagrams that depict some aspect of a system. These eight views are grouped into three major areas: structural, dynamic, and model management.

The structural view describes the static structure of a software system, including the classes, the components, and their relationships. The dynamic view describes the implementation of business logic in the design of individual classes and the interaction of classes and components. The model management view describes the relationships between UML models. In the preceding section, you learned that partial models are used to describe subsystems. The model management view describes the relationships between subsystems. The specifics of each of the UML views and diagrams will be discussed throughout this course. Table 5-1 lists the eight UML views and the diagrams used to develop each view.

Table 5-1: UML Views

<table>
<thead>
<tr>
<th>Major Area</th>
<th>View</th>
<th>Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural</strong></td>
<td>Static view</td>
<td>Class diagram</td>
</tr>
<tr>
<td></td>
<td>Use-case view</td>
<td>Use-case diagram</td>
</tr>
<tr>
<td></td>
<td>Implementation view</td>
<td>Component diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deployment diagram</td>
</tr>
<tr>
<td></td>
<td>Deployment view</td>
<td>Component diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deployment diagram</td>
</tr>
<tr>
<td><strong>Dynamic</strong></td>
<td>State machine view</td>
<td>State chart diagram</td>
</tr>
<tr>
<td></td>
<td>Activity view</td>
<td>Activity diagram</td>
</tr>
<tr>
<td></td>
<td>Interaction view</td>
<td>Sequence diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Collaboration diagram</td>
</tr>
<tr>
<td><strong>Model management</strong></td>
<td>Model management view</td>
<td>Class diagram</td>
</tr>
</tbody>
</table>
SUMMARY

Booch, Jacobson, and Rumbaugh developed the Unified Modeling Language (UML) to provide a complete language for specifying object-oriented software systems. UML was adopted as an official standard by the OMG in 1997. UML models both the static structures and the dynamic behaviors of a software system. Various modeling levels are used depending on the level of detail required by each stage in the development process. UML is divided into eight views, which are grouped into three major areas: structural, dynamic, and model management.

POST-TEST QUESTIONS

The answers to these questions are in Appendix A at the end of this manual.

1. What are the three UML modeling levels?

2. What are the eight UML views?