PEARSON NEW INTERNATIONAL EDITION

Object-Oriented Software Engineering Using UML, Patterns, and Java

Bernd Bruegge Allen H. Dutoit

Third Edition

ALWAYS LEARNING

PEARSON

Pearson New International Edition

Object-Oriented Software Engineering
Using UML, Patterns, and Java
Bernd Bruegge Allen H. Dutoit
Third Edition

PEARSON

- Identifying objects, their attributes and associations, takes many iterations, often with the client.
- Object identification uses many sources, including the problem statement, use case model, the glossary, and the event flows of the use cases.
- A nontrivial use case can require many sequence diagrams and several class diagrams. It is unrealistic to represent all discovered objects in a single diagram. Instead, each diagram serves a specific purpose—for example, depicting associations among entity objects, or depicting associations among participating objects in one use case.
- Key deliverables, such as the glossary, should be kept up to date as the analysis model is revised. Others, such as sequence diagrams, can be redone later if necessary. Maintaining consistency at all times, however, is unrealistic.
- There are many different ways to model the same application domain or the same system, based on the personal style and experience of the analyst. This calls for developing style guides and conventions within a project, so that all analysts can communicate effectively.

5.7 Further Readings

The classification of analysis objects into entity, boundary, and control objects has been made popular by the Objectory method [Jacobson et al., 1992]. These concepts originated from the model/view/controller (MVC) paradigm used in the Smalltalk-80 environment and also found their way into the Java Swing user interface framework [JFC, 2009].

CRC cards were introduced by Beck and Cunningham for teaching object-oriented thinking to novices and experienced developers in an OOPSLA paper entitled *A Laboratory For Teaching Object-Oriented Thinking* [Beck & Cunningham, 1989]. CRC cards are used extensively in the responsibility-driven design method from Wirfs-Brock [Wirfs-Brock et al., 1990].

Object-oriented analysis and design has evolved from many different sets of heuristics and terminologies. Modeling, like programming, is a craft, and requires much experience and willingness to make mistakes (hence the importance of client and user feedback). *Object-Oriented Modeling and Design* [Rumbaugh et al., 1991] provides an excellent guide to novices for class modeling. A more recent book, *Applying UML and Patterns* [Larman, 2005], provides a comprehensive treatment of object-oriented analysis and design, including use case modeling and reusing design patterns. For dynamic modeling with state machines, *Doing Hard Time: Using Object Oriented Programming and Software Patterns in Real Time Applications* [Douglass, 1999] provides detailed information and modeling heuristics on the topic.

5.8 Exercises

5-1 Consider a file system with a graphical user interface, such as Macintosh's Finder, Microsoft's Windows Explorer, or Linux's KDE. The following objects were identified from a use case describing how to copy a file from a floppy disk to a hard disk: `File`, `Icon`, `TrashCan`, `Folder`, `Disk`, `Pointer`. Specify which are entity objects, which are boundary objects, and which are control objects.

5-2 Assuming the same file system as before, consider a scenario consisting of selecting a `File` on a floppy, dragging it to `Folder` and releasing the mouse. Identify and define at least one control object associated with this scenario.

5-3 Arrange the objects listed in Exercises 5-1 and 5-2 horizontally on a sequence diagram, the boundary objects to the left, then the control object you identified, and finally, the entity objects. Draw the sequence of interactions resulting from dropping the file into a folder. For now, ignore the exceptional cases.

5-4 Examining the sequence diagram you produced in Exercise 5-3, identify the associations between these objects.

5-5 Identify the attributes of each object that are relevant to this scenario (copying a file from a floppy disk to a hard disk). Also consider the exception cases "There is already a file with that name in the folder" and "There is no more space on disk."

5-6 Consider the object model in Figure 5-32 (adapted from [Jackson, 1995]):

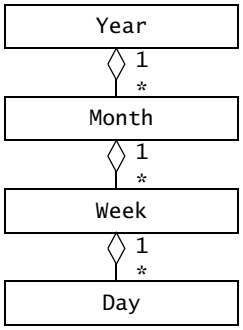

Figure 5-32 A naive model of the Gregorian calendar (UML class diagram).

Given your knowledge of the Gregorian calendar, list all the problems with this model. Modify it to correct each of them.

5-7 Consider the object model of Figure 5-32. Using association multiplicity only, can you modify the model such that a developer unfamiliar with the Gregorian calendar could deduce the number of days in each month? Identify additional classes if necessary.

5-8 Consider a traffic light system at a four-way crossroads (two roads intersecting at right angles). Assume the simplest algorithm for cycling through the lights (e.g., all traffic on one road is allowed to go through the crossroad, while the other traffic is stopped). Identify the states of this system and draw a state machine describing them. Remember that each individual traffic light has three states (green, yellow, and red).

5-9 From the sequence diagram Figure 2-34, draw the corresponding class diagram. Hint: Start with the participating objects in the sequence diagram.

5-10 Consider the addition of a nonfunctional requirement stipulating that the effort needed by `Advertisers` to obtain exclusive sponsorships should be minimized. Change the `AnnounceTournament` (Figure 5-23) and the `ManageAdvertisements` use case (solution of Exercise 4-12) so that the `Advertiser` can specify preferences in her profile so that exclusive sponsorships can be decided automatically by the system.

5-11 Identify and write definitions for any additional entity, boundary, and control objects participating in the `AnnounceTournament` use case that were introduced by realizing the change specified in Exercise 5-10.

5-12 Update the class diagrams of Figure 5-29 and Figure 5-31 to include the new objects you identified in Exercise 5-11.

5-13 Draw a state machine describing the behavior of the `AnnounceTournamentControl` object based on the sequence diagrams of Figures 5-26 through 5-28. Treat the sending and receiving of each notice as an event that triggers a change of state.

References

[Abbott, 1983] R. Abbott, "Program design by informal English descriptions," *Communications of the ACM*, Vol. 26, No. 11, 1983.

[Beck & Cunningham, 1989] K. Beck & W. Cunningham, "A laboratory for teaching object-oriented thinking," *OOPSLA'89 Conference Proceedings*, New Orleans, LA, Oct. 1–6, 1989.

[De Marco, 1978] T. De Marco, *Structured Analysis and System Specification*, Yourdon, New York, 1978.

[Douglass, 1999] B. P. Douglass, *Doing Hard Time: Using Object Oriented Programming and Software Patterns in Real Time Applications*, Addison-Wesley, Reading, MA, 1999.

[Jackson, 1995] M. Jackson, *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*, Addison-Wesley, Reading, MA, 1995.

[Jacobson et al., 1992] I. Jacobson, M. Christerson, P. Jonsson, & G. Overgaard, *Object-Oriented Software Engineering—A Use Case Driven Approach,* Addison-Wesley, Reading, MA, 1992.

[Jacobson et al., 1999] I. Jacobson, G. Booch, & J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, Reading, MA, 1999.

[JFC, 2009] *Java Foundation Classes*, JDK Documentation, Javasoft, 2009.

[Larman, 2005] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*, 3rd ed., Prentice Hall, Upper Saddle River, NJ, 2005.

[Rumbaugh et al., 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, & W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[Wirfs-Brock et al., 1990] R. Wirfs-Brock, B. Wilkerson, & L. Wiener, *Designing Object-Oriented Software*, Prentice Hall, Englewood Cliffs, NJ, 1990.

6

System Design: Decomposing the System

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.

—C.A.R. Hoare, in *The Emperor's Old Clothes*

System design is the transformation of an analysis model into a system design model. During system design, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software strategy, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions. The result of system design is a model that includes a subsystem decomposition and a clear description of each of these strategies.

System design is not algorithmic. Developers have to make trade-offs among many design goals that often conflict with each other. They also cannot anticipate all design issues that they will face because they do not yet have a clear picture of the solution domain. System design is decomposed into several activities, each addressing part of the overall problem of decomposing the system:

- *Identify design goals*. Developers identify and prioritize the qualities of the system that they should optimize.
- *Design the initial subsystem decomposition*. Developers decompose the system into smaller parts based on the use case and analysis models. Developers use standard architectural styles as a starting point during this activity.
- *Refine the subsystem decomposition to address the design goals*. The initial decomposition usually does not satisfy all design goals. Developers refine it until all goals are satisfied.

In this chapter, we focus on the first two activities. In the next chapter, we refine the system decomposition and provide an in-depth example with the ARENA case study.