1.

Introduction

One of Agile Modeling's (AM) practices (discussed in Chapter 17) is *Apply Modeling Standards*, the modeling version of Extreme Programming (XP)'s *Coding Standards* (Beck 2000). Developers should agree to and follow a common set of standards and guidelines on a software project, and some of those guidelines should apply to modeling. Models depicted with a common notation and that follow effective style guidelines are easier to understand and to maintain. These models will improve communication internally within your team and externally to your partners and customers, thereby reducing the opportunities for costly misunderstandings. Modeling guidelines will also save you time by limiting the number of stylistic choices you face, allowing you to focus on your actual job – to develop software.

> A lot of the communication value in a UML diagram is still due to the layout skill of the modeler.
>
> —Paul Evitts, *A UML Pattern Language* (Evitts 2000)

When you adopt modeling standards and guidelines within your organization, your first step is to settle on a common notation. The Unified Modeling Language (UML) (Object Management Group 2004) is a good start because it defines the notation and semantics for common object-oriented models. Some projects will require more types of models than the UML describes, as I show in *The Object Primer 3/e* (Ambler 2004), but the UML will form the core of any modern modeling effort.

Your second step is to identify modeling style guidelines to help you to create consistent and clean-looking diagrams. What is the difference between a standard and a style guideline? For source code, a standard would, for example, involve naming the attributes in the format *attributeName*, whereas a style guideline would involve indenting your code within a control structure by three spaces. For models, a standard would involve using a squared rectangle to model a class on a class diagram, whereas a style would involve placing subclasses on diagrams below their superclass(es). This book describes the style guidelines that are missing from many of the UML-based methodologies that organizations have adopted, guidelines that are critical to your success in the software development game.

The third step is to enact your modeling standards and guidelines. To do this, you will need to train and mentor your staff in the modeling techniques appropriate to the projects on which they are working. You will also need to train and mentor them in your adopted guidelines, and a good start is to provide them with a copy of this book. I've been amazed at the success of *The Elements of Java Style* (Vermeulen et al. 2000) with respect to this—hundreds of organizations have adopted that book for their internal Java coding standards because they recognized that it was more cost-effective for them to buy a pocketbook for each developer than to develop their own guidelines.

1.1 Organization of This Book

This book is organized in a straightforward manner. Chapter 2 describes general diagramming principles that are applicable to all types of UML diagrams (and many non-UML diagrams for that matter). Chapter 3 describes guidelines for common UML elements such as stereotypes, notes, and frames.

Chapters 4 through 16 describe techniques pertinent to each type of UML diagram. Chapter 17 provides an overview of the values, principles, and practices of AM, with a quick reference to this popular methodology.

2. General Diagramming Guidelines

The guidelines presented in this chapter are applicable to all types of diagrams, UML or otherwise. The terms "symbols," "lines," and "labels" are used throughout:

- Symbols represent diagram elements such as class boxes, object boxes, use cases, and actors.
- Lines represent diagram elements such as associations, dependencies, and transitions between states.
- Labels represent diagram elements such as class names, association roles, and constraints.

2.1 Readability Guidelines

1. Avoid Crossing Lines

When two lines cross on a diagram, such as two associations on a UML class diagram, the potential for misreading a diagram exists.

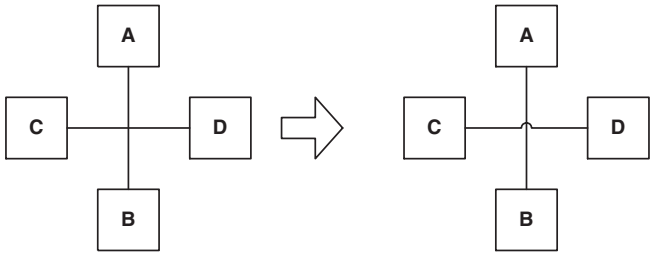

Figure 1. Depiction of crossing lines.

2. Depict Crossing Lines as a Jump

You can't always avoid crossing lines; for example, you cannot fully connect five symbols (try it and see). When you need to have two lines cross, one of them should "hop" over the other as in Figure 1.

3. Avoid Diagonal or Curved Lines

Straight lines, drawn either vertically or horizontally, are easier for your eyes to follow than diagonal or curved lines. A good approach is to place symbols on diagrams as if they are centered on the grid point of a graph, a built-in feature of many computer-aided system-engineering (CASE) tools. This makes it easier to connect your symbols by only using horizontal and vertical lines. Note how three lines are improved in Figure 2 when this approach is taken. Also note how the line between *A* and *C* has been depicted in "step fashion" as a line with vertical and horizontal segments.

4. Apply Consistently Sized Symbols

The larger a symbol appears, the more important it seems to be. In the first version of the diagram in Figure 2, the *A* symbol is larger than the others, drawing attention to it. If that isn't the effect that you want, then strive to make your symbols of uniform size. Because the size of some symbols is

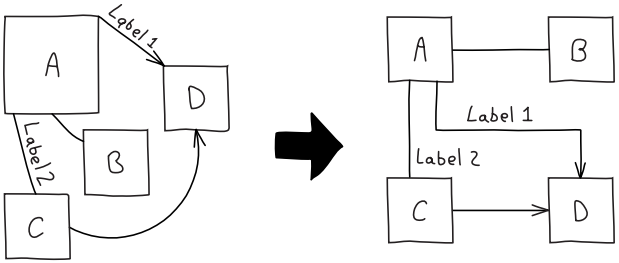

Figure 2. Improving the attractiveness of a diagram.

determined by their contents—for example, a class will vary in size based on its attributes and operations—this rule is not universally applicable. Ideally you should only deviate if you want to accentuate an aspect of your diagram (Koning, Dormann, and Van Vliet 2002).

5. Attach Lines to the Middle of Bubbles

As you can see in Figure 2, the *Label 1* line between *A* and *D* is much more readable in the updated version of the diagram.

6. Align Labels Horizontally

In Figure 2 the two labels are easier to read in the second version of the diagram. Notice how *Label 2* is horizontal even though the line it is associated with is vertical.

7. Arrange Symbols Symmetrically

Figure 3 presents a UML activity diagram (Chapter 10) depicting a high-level approach to enterprise modeling. Organizing the symbols and lines in a symmetrical manner makes the diagram easier to understand. A clear pattern will make a diagram easier to read.

8. Don't Indicate "Exploding" Bubbles

The rake symbol in the upper right corner of each activity in Figure 3 is the UML way to indicate that they "explode" to

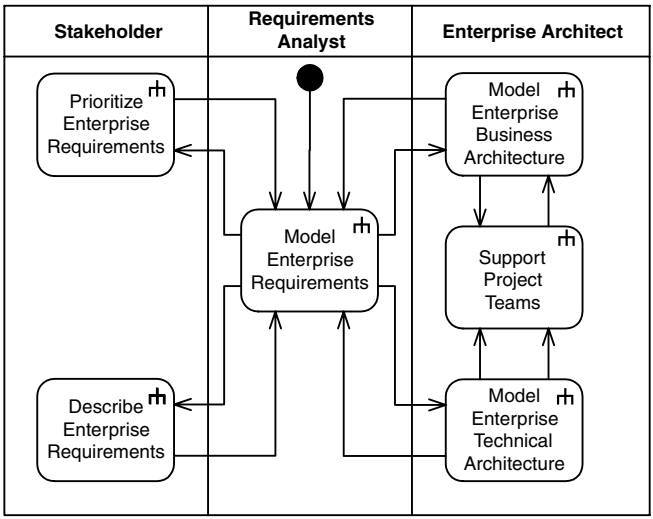

Figure 3. UML activity diagram for a software process.

another diagram showing a greater level of detail. Although this seems like a good idea, the reality is that people using a CASE tool know enough to double click on it, or whatever strategy the tool implements, to get more detail. The rake isn't adding any extra value.

9. Minimize the Number of Bubble Types

Koning, Dormann, and Van Vliet (2002) recommend that you have six or fewer bubbles on a diagram; any more risks overwhelming the user of the model.

10. Include White Space in Diagrams

White space is the empty areas between modeling elements on your diagrams. In the first version of Figure 2 the symbols are crowding each other, whereas in the second version, the symbols are spread out from one another, thus improving the

readability of the diagram. Observe that in the second version there is adequate space to add labels to the lines.

11. Organize Diagrams Left to Right, Top to Bottom

In Western cultures, people read left to right and top to bottom and therefore this is how they will read your diagrams. If there is a starting point for reading the diagram, such as the initial state of a UML state chart diagram or the beginning of the flow of logic on a UML sequence diagram, then place it toward the top left corner of your diagram and continue appropriately from there.

12. Avoid Many Close Lines

Several lines close together are hard to follow.

13. Provide a Notation Legend

If you're not sure that all of the users of a model understand the notation that you're using, provide them with a legend that overviews it. A good legend indicates the notational symbols used, the name of the symbol, and a description of its usage. Figure 4 provides an example for robustness diagrams (Jacobson, Christerson, Jonsson, and Overgaard 1992; Rosenberg and Scott 1999), a modification of UML communication diagrams (Chapter 8).

2.2 Simplicity Guidelines

14. Show Only What You Have to Show

Diagrams showing too many details are difficult to read because they are too information-dense. One of the practices of Agile Modeling (Chapter 17) is to *Depict Models Simply*: to include only critical information on your diagrams and to exclude anything extraneous. A simple model that shows the key features that you are trying to depict—perhaps a UML

Actor	Actors represent people, organizations, or other systems which interact with our system. Examples: Student, Financial Institution, and Payment Processor.

Process/ Controller Class	Process/controller classes implement logic which crosses several business entities. Examples: Print a student transcript and Drop student from a seminar.

Domain Class	Domain classes implement fundamental business entities. Examples: Student, Seminar, Course.

Interface Class	Interface classes enable actors to interact with our system, other via a user interface or a system interface. Examples: Student registration screen and Payment Processing System Interface.

Association	Whenever an actor interacts with a class, or two classes interact, there is an association between them.

Figure 4. A legend for robustness diagrams.

class diagram depicting the primary responsibilities of classes and the relationships between them—often proves to be sufficient. Yes, you could model all of the scaffolding code that you will need to implement, but what value would that add? Very little.

15. Prefer Well-Known Notation over Esoteric Notation

Diagrams that include esoteric notation, instead of just the 20 percent "kernel notation" that does 80 percent of the job, can be difficult to read. Of course, what is well known in one organization may not be so well known in another, and so, you may want to consider supplying people with a brief summary of the notation that you're using.

16. Reorganize Large Diagrams into Several Smaller Ones

It is often better to have several diagrams showing various degrees of detail than one complex diagram that shows

everything. A good rule of thumb is that a diagram shouldn't have more than nine symbols on it, based on the 7 ± 2 rule (Miller 1957), because there is a limit on the amount of information that someone can deal with at once. "Wallpaper" diagrams, particularly enterprise data models or enterprise object models, may look interesting but they're too information-dense to be effective. When you are reorganizing a large diagram into several smaller ones, you may choose to introduce a high-level UML package diagram (Chapter 6).

17. Prefer Single-Page Diagrams

To reduce complexity, a diagram should be printable on a single sheet of paper to help reduce its scope as well as to prevent wasted time cutting and taping several pages together. Be aware that you will reduce the usability of a diagram if you need to reduce the font too much or crowd the symbols and lines.

18. Focus on Content First, Appearance Second

There is always the danger of adding hours onto your CASE tool modeling efforts by rearranging the layout of your symbols and lines to improve the diagram's readability. The best approach is to focus on the content of a diagram at first and only try to get it looking good in a rough sort of way—it doesn't have to be perfect while you're working on it. Once you're satisfied that your diagram is accurate enough, and that you want to keep it, then invest the appropriate time to make it look good. An advantage of this approach is that you don't invest significant effort improving diagrams that you eventually discard.

19. Apply Consistent, Readable Fonts

Consistent, easy-to-read fonts improve the readability of your diagrams. Good ideas include fonts in the Courier, Arial,