Chapter 1

Introduction

WHAT IS VISUAL MODELING?

VISUAL MODELING IS a way of thinking about problems using models organized around real-world ideas. Models are useful for understanding problems, communicating with everyone involved with the project (customers, domain experts, analysts, designers, etc.), modeling enterprises, preparing documentation, and designing programs and databases. Modeling promotes better understanding of requirements, cleaner designs, and more maintainable systems.

Models are abstractions that portray the essentials of a complex problem or structure by filtering out nonessential details, thus making the problem easier to understand. Abstraction is a fundamental human capability that permits us to deal with complexity. Engineers, artists, and craftsmen have built models for thousands of years to try out designs before executing them. Development of software systems should be no exception. To build complex systems, the developer must abstract different views of the system, build models using precise notations, verify that the models satisfy the requirements of the system, and gradually add detail to transform the models into an implementation.

We build models of complex systems because we cannot comprehend such systems in their entirety. There are limits to the human capacity to understand complexity. This concept may be seen in the world of architecture. If you want to build a shed in your backyard, you can just start building; if you want to build a new house, you probably need a blueprint; if you are building a skyscraper, you definitely need a blueprint. The same is true in the world of software. Staring at lines of source code or even analyzing forms in Visual Basic does little to provide the programmer with a global view of a development project. Constructing a model allows the designer to focus on the big picture of how a project's components interact, without having to get bogged down in the specific details of each component.

Increasing complexity, resulting from a highly competitive and ever-changing business environment, offers unique challenges to system developers. Models help us organize, visualize, understand,

and create complex things. They are used to help us meet the challenges of developing software today and in the future.

THE TRIANGLE FOR SUCCESS

I HAVE OFTEN used the triangle for success as shown in Figure 1-1 to explain the components needed for a successful project. You need all three facets—a notation, a process, and a tool. You can learn a notation, but if you don't know how to use it (process), you will probably fail. You may have a great process, but if you can't communicate the process (notation), you will probably fail. And lastly, if you cannot document the artifacts of your work (tool), you will probably fail.

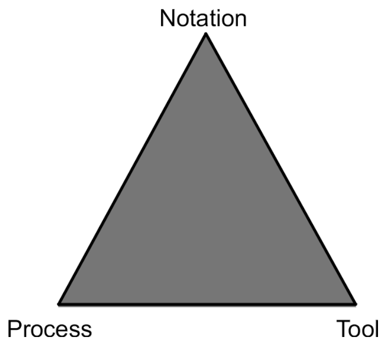

Figure 1-1 Triangle for Success

THE ROLE OF NOTATION

NOTATION PLAYS AN important part in any model—it is the glue that holds the process together. "Notation has three roles:

- It serves as the language for communicating decisions that are not obvious or cannot be inferred from the code itself.
- It provides semantics that are rich enough to capture all important strategic and tactical decisions.

- It offers a form concrete enough for humans to reason and for tools to manipulate."[1]

The Unified Modeling Language (UML) provides a very robust notation, which grows from analysis into design. Certain elements of the notation (for example, classes, associations, aggregations, inheritance) are introduced during analysis. Other elements of the notation (for example, containment implementation indicators and properties) are introduced during design.

HISTORY OF THE UML

DURING THE 1990S many different methodologies, along with their own set of notations, were introduced to the market. Three of the most popular methods were OMT (Rumbaugh), Booch, and OOSE (Jacobson). Each method had its own value and emphasis. OMT was strong in analysis and weaker in the design area. Booch 1991 was strong in design and weaker in analysis. Jacobson was strong in behavior analysis and weaker in the other areas.

As time moved on, Booch wrote his second book, which adopted a lot of the good analysis techniques advocated by Rumbaugh and Jacobson, among others. Rumbaugh published a series of articles that have become known as OMT-2 that adopted a lot of the good design techniques of Booch. The methods were beginning to converge but they still had their own unique notations. The use of different notations brought confusion to the market since one symbol meant different things to different people. For example, a filled circle was a multiplicity indicator in OMT and an aggregation symbol in Booch. You will hear the term "method wars" being used to describe this period of time—is a class a cloud or a rectangle? Which one is better?

The end of the method wars as far as notation is concerned comes with the adoption of the Unified Modeling Language (UML). "UML is a language used to specify, visualize, and document the artifacts of an object-oriented system under development. It represents the unification of the Booch, OMT, and Objectory notations,

[1] Booch, Grady. *Object Solutions.* Redwood City, CA: Addison-Wesley, 1995.

as well as the best ideas from a number of other methodologists as shown in Figure 1-2. By unifying the notations used by these object-oriented methods, the Unified Modeling Language provides the basis for a *de facto* standard in the domain of object-oriented analysis and design founded on a wide base of user experience."[2]

The UML is an attempt to standardize the artifacts of analysis and design: semantic models, syntactic notation, and diagrams. The first public draft (version 0.8) was introduced in October 1995. Feedback from the public and Ivar Jacobson's input were included in the next two versions (0.9 in July 1996 and 0.91 in October 1996). Version 1.0 was presented to the Object Management Group (OMG) for standardization in July 1997. Additional enhancements were incorporated into the 1.1 version of UML, which was presented to the OMG in September 1997. In November 1997, the UML was adopted as the standard modeling language by the OMG. The current version of the UML is UML 1.4 and work is progressing on UML 2.0. You can find more information on the UML by visiting the OMG web site at www.omg.org.

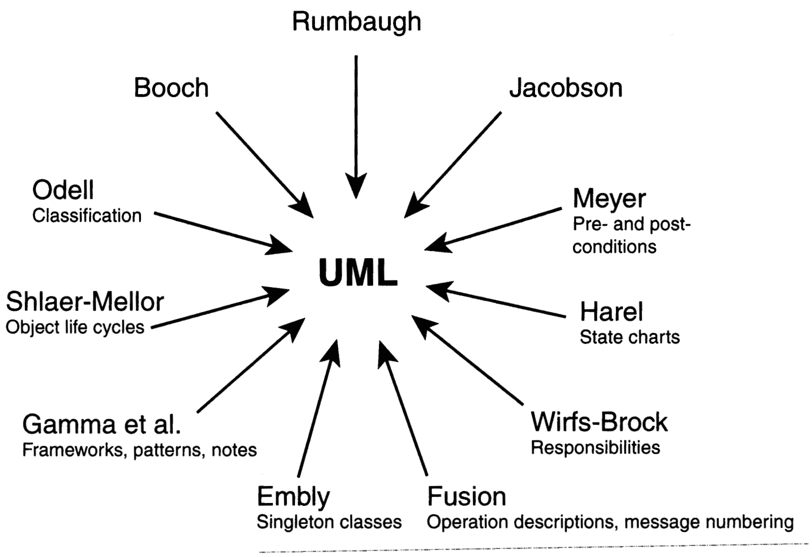

Figure 1-2 UML Inputs

[2] *The Unified Method,* Draft Edition (0.8). Rational Software Corporation, October, 1995.

THE ROLE OF PROCESS

A SUCCESSFUL DEVELOPMENT project satisfies or exceeds the customer's expectations, is developed in a timely and economical fashion, and is resilient to change and adaptation. The development life cycle must promote creativity and innovation. At the same time, the development process must be controlled and measured to ensure that the project is indeed completed. "Creativity is essential to the crafting of all well-structured object-oriented architectures, but developers allowed completely unrestrained creativity tend to never reach closure. Similarly, discipline is required when organizing the efforts of a team of developers, but too much discipline gives birth to an ugly bureaucracy that kills all attempts at innovation."[3] A well-managed iterative and incremental life cycle provides the necessary control without affecting creativity.

WHAT IS ITERATIVE AND INCREMENTAL DEVELOPMENT?

IN AN ITERATIVE and incremental life cycle (Figure 1-3), development proceeds as a series of iterations that evolve into the final system. Each iteration consists of one or more of the following process components: business modeling, requirements, analysis, design, implementation, test, and deployment. The developers do not assume that all requirements are known at the beginning of the life cycle; indeed change is anticipated throughout all phases.

This type of life cycle is a risk-mitigating process. Technical risks are assessed and prioritized early in the life cycle and are revised during the development of each iteration. Risks are attached to each iteration so that successful completion of the iteration alleviates the risks attached to it. The releases are scheduled to ensure that the highest risks are tackled first. Building the system in this fashion exposes and mitigates the risks of the system early in the

[3] Booch, Grady. *Object Solutions.* Redwood City, CA: Addison-Wesley, 1995.

life cycle. The result of this life cycle approach is less risk coupled with minimal investment.[4]

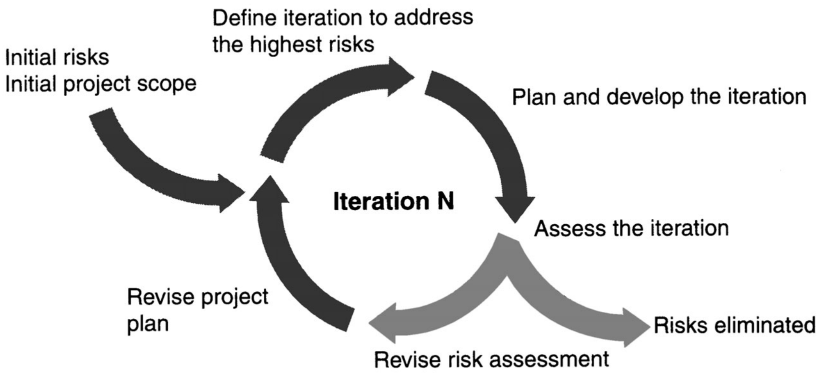

Figure 1-3 Iterative and Incremental Development

THE RATIONAL UNIFIED PROCESS

CONTROL FOR AN iterative and incremental life cycle is supported by employing the Rational Unified Process—an extensive set of guidelines that address the technical and organizational aspects of software development focusing on requirements analysis and design.

The Rational Unified Process is structured along two dimensions:

- Time—division of the life cycle into phases and iterations
- Process components—production of a specific set of artifacts with well-defined activities

Both dimensions must be taken into account for a project to succeed.

[4] More information on the application of an iterative and incremental approach to software development may be found in the article "A Rational Development Process" by Philippe Kruchten, *CrossTalk,* 9(7), July 1996, pp. 11–16. This paper is also available on the Rational web site: http://www.rational.com.

Structuring a project along the time dimension involves the adoption of the following time-based phases:

- Inception—specifying the project vision
- Elaboration—planning the necessary activities and required resources; specifying the features and designing the architecture
- Construction—building the product as a series of incremental iterations
- Transition—supplying the product to the user community (manufacturing, delivering, and training)

Structuring the project along the process component dimension includes the following activities:

- Business Modeling—the identification of desired system capabilities and user needs
- Requirements—a narration of the system vision along with a set of functional and nonfunctional requirements
- Analysis and Design—a description of how the system will be realized in the implementation phase
- Implementation—the production of the code that will result in an executable system
- Test—the verification of the entire system
- Deployment—the delivery of the system and user training to the customer

Figure 1-4 shows how the process components are applied to each time-based phase.

Each activity of the process component dimension typically is applied to each phase of the time-based dimension. However, the degree to which a particular process component is applied is dependent upon the phase of development. For example, you may decide to do a proof of concept prototype during the Inception Phase, and thus, you will be doing more than just capturing requirements (you will be doing the analysis, design, implementation, and test needed to complete the prototype). The majority of the analysis process

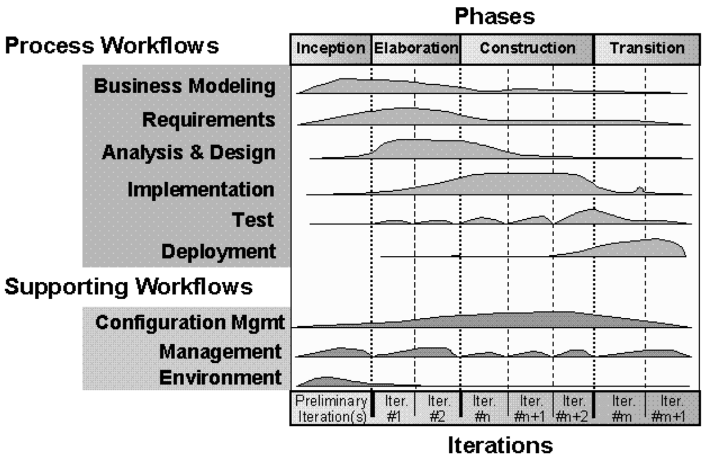

Figure 1-4 The Development Process

component occurs during the Elaboration Phase. However, it is also advisable to complete the first few iterations of the system during this phase. These first few iterations typically are used to validate the analysis decisions made for the architecture of the system. Hence, you are doing more than just analyzing the problem. During the Construction Phase of development, the system is completed as a series of iterations. As with any type of development structure, things always crop up as the system is built; thus, you are still doing some analysis.

The diagram is meant to be a guideline for the life cycle of your project. The main point is if you are still trying to figure out what you are supposed to be building as you are writing the code, you are probably in trouble. You should also note that testing is applied throughout the iteration process—you do not wait until all the code is done to see if it all works together!

This book uses a simplified version of the Rational Unified Process, which concentrates on the use of the UML to capture and document the decisions made during the Inception and Elaboration phases of development. The last few chapters lightly cover construction of the system. Although testing is a very integral part of system development, it is beyond the scope of this book.

THE RATIONAL ROSE TOOL

ANY SOFTWARE DEVELOPMENT method is best supported by a tool. When I first started OO modeling, my tool was paper and a pencil, which left a lot to be desired. There are many tools on the market today—everything from simple drawing tools to sophisticated object modeling tools. This book makes use of the tool Rational Rose. At every step, there is a description of how to use Rational Rose to complete the step.

The Rational Rose product family is designed to provide the software developer with a complete set of visual modeling tools for development of robust, efficient solutions to real business needs in the client/server, distributed enterprise, and real-time systems environments. Rational Rose products share a common universal standard, making modeling accessible to nonprogrammers wanting to model business processes as well as to programmers modeling applications logic. An evaluation version of the Rational Rose tool may be obtained at the Rational Software Corporation website at www.rational.com.

SUMMARY

VISUAL MODELING IS a way of thinking about problems using models organized around real-world ideas. Models are useful for understanding problems, communication, modeling enterprises, preparing documentation, and designing programs and databases. Modeling promotes better understanding of requirements, cleaner designs, and more maintainable systems. Notation plays an important part in any model—it is the glue that holds the process together. The Unified Modeling Language (UML) provides a very robust notation, which grows from analysis into design.

A successful development project satisfies or exceeds the customer's expectations, is developed in a timely and economical fashion, and is resilient to change and adaptation. The development life cycle must promote creativity and innovation. A well-managed iterative and incremental life cycle provides the necessary control without affecting creativity. In an iterative and incremental development life cycle, development proceeds as a series of iterations that evolve into the final system. Each iteration consists of one or more of the following process components: business modeling, requirements, analysis, design, implementation, test, and deployment.

Control for an iterative and incremental life cycle is provided in the Rational Unified Process—an extensive set of guidelines that address the technical and organizational aspects of software development, focusing on requirements analysis and design. This book uses a simplified version of the Rational Unified Process.

The Rational Rose product family is designed to provide the software developer with a complete set of visual modeling tools for development of robust, efficient solutions to real business needs in the client/server, distributed enterprise, and real-time systems environments.