UML 2 FOR DUMMIES®

by Michael Jesse Chonoles
and James A. Schardt

Wiley Publishing, Inc.

UML 2 FOR DUMMIES®

by Michael Jesse Chonoles
and James A. Schardt

Wiley Publishing, Inc.

UML 2 For Dummies®
Published by
Wiley Publishing, Inc.
909 Third Avenue
New York, NY 10022
www.wiley.com

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

Library of Congress Control Number: 2003105654

ISBN: 0-7645-2614-6

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/QS/QX/QT/IN

About the Authors

Michael Jesse Chonoles: An established system developer, educator, author, and consultant, Michael has done just about everything that you can do in software and system development — business, requirements, and software analysis; software, system, and architectural design; coding in many languages; testing and quality control — right through marketing, packing, and shrink-wrapping the software. His titles include Chief of Methodology for the Advanced Concepts Center (ACC), Software Development Practice Area Director, Consulting Analyst, Software Standard and Practices Manager, Test Director, Senior Software Engineer, several varieties of Team/Project Lead/Staff, and (his personal favorite) Wizard. At the Advanced Concepts Center, he was responsible for the content and direction of its Object-Oriented and Requirements-Gathering Curricula as well as its Software Development Practice. Together with his co-author, he constructed a software/system-development methodology, CADIT, which was an early attempt to combine agile techniques with aerospace discipline. He continues his quest to make the complicated simple, while increasing the professional rigor, quality, and productivity of his audience's working lives.

Michael has been involved in many aspects of UML, even before there was a UML. He's been an active member of the UML RTF (Revision Task Force) at OMG — and frequently writes, lectures, speaks, and suggests UML topics.

Michael has an MSE in Systems Engineering from the University of Pennsylvania and BSs in Math and Physics from MIT. He can be contacted at `michaeljessechonoles@alum.mit.edu`.

James A. Schardt: As the Chief Technologist with the Advanced Concepts Center, James provides 24 years of experience and a firm grounding in object-oriented development, data warehousing, and distributed systems. He teaches and mentors Fortune 50 companies in the U.S. and abroad. His many years of practice in object-oriented systems, database design, change management, business engineering, instructional design, systems-architecture assessment, business engineering, and team facilitation bring a wealth of experience to his assignments.

He authors papers on data warehousing and object technology and also wrote a column for *Report on Object-Oriented Analysis and Design*. James speaks at The Data Warehouse Institute's world conferences on a regular basis. He delivers a two-day presentation on collecting and structuring the requirements for enterprise data-warehouse development.

James is always looking for ways to improve the way that we develop systems and software. Clients request him by name to deliver his exceptional knowledge transfer skills, both in the classroom and as a mentor on projects. Over the years, James has managed major research and development programs, invented new systems methods, developed "intelligent" information-access systems, and provided unique insights into clients' difficult development problems.

James has an MSE in Systems Engineering from the University of Pennsylvania. He can be reached via schardt@acm.org.

Dedication

Michael dedicates this book to his wife Susann and to their son Zev, for their love, support, sacrifice, and silliness.

Jim dedicates this book to his wife Martha for her sustaining love and encouragement, and to M. R. Bawa Muhaiyaddeen as the guiding inspiration in his life.

Authors' Acknowledgments

We would like to thank all the students whom we have taught over the years for their help in shaping our ideas, and all the members of the Advanced Concepts Center, both past and present, for the chance to work with some of the best practitioners in the business of systems and software development.

Together we acknowledge the absolutely necessary help, encouragement, and moral support of our Wiley editors Terri Varveris and Kala Schrager.

Michael would like to thank a whole bunch of people who have helped him over the years, and specifically with this book: Susann Chonoles for teaching him how to write better and for help in proofreading; Zev Chonoles, for being a Test Dummy For Dummies and reading his chapters; his managers Bob DeCarli, Mike Duffy, and Barbara Zimmerman, who encouraged him even when he messed up; and his high-school buddies Joseph Newmark, Jeffrey Landsman, and Barry Salowitz, who keep on telling him what he's doing wrong. It goes without saying that he's grateful to his parents for everything.

He'd also like to acknowledge Jim Schardt for his work toward understanding UML in all its forms, and Lou Varveris for his insight, recommendations, and for access to the Popkin's System Architect tool. He's also grateful to all the members of the OMG ADTF and the UML Gurus for their technical advice, encouragement, and support over the years — especially Cris Kobryn, Jim Odell, Jim Rumbaugh, Philippe Desfray, and Bran Selic.

Jim would like to thank a number of individuals who helped him develop his knowledge and skills over the years: David Oliver for his systems perspective; Michael Kamfonas for his data-warehouse development insights; Michael Chonoles for his work toward understanding UML in all its forms; Jim Rumbaugh and Fred Eddy for their mentoring on object-oriented analysis; and Michael Blaha and William Premerlani for their guiding hand in developing database-design techniques using UML.

Publisher's Acknowledgments

We're proud of this book; please send us your comments through our online registration form located at www.dummies.com/register/.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Media Development

Project Editor: Kala Schrager

Acquisitions Editor: Theresa Varveris

Senior Copy Editor: Barry Childs-Helton

Technical Editor: Lou Varveris

Editorial Manager: Kevin Kirschner

Media Development Supervisor: Richard Graves

Editorial Assistant: Amanda Foxworth

Cartoons: Rich Tennant, www.the5thwave.com

Production

Project Coordinators: Kristie Rees, Dale White

Layout and Graphics: Seth Conley, Kelly Emkow, Carrie Foster, LeAndra Hosier, Stephanie D. Jumper, Michael Kruzil, Mary Gillot Virgin

Proofreaders: Laura Albert, Susan Moritz, Dwight Ramsey, TECHBOOKS Production Services

Indexer: TECHBOOKS Production Services

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher

Andy Cummings, Vice President and Publisher

Mary C. Corder, Editorial Director

Publishing for Consumer Dummies

Diane Graves Steele, Vice President and Publisher

Joyce Pepple, Acquisitions Director

Composition Services

Gerry Fahey, Vice President of Production Services

Debbie Stailey, Director of Composition Services

Contents at a Glance

Table of Contents

Introduction

If, like us, you're a software developer or computer professional of some sort, you probably have to deal with the stereotype that developers can't express themselves among normal humans about normal things. Unfortunately, this book may not help you with that particular challenge, but it can help improve your ability to communicate with other developers about technical matters. *UML* (Unified Modeling Language) is a graphical language that is suitable to express software or system requirements, architecture, and design. You can use UML to communicate with other developers, your clients, and increasingly, with automated tools that generate parts of your system.

If you're already familiar with UML, you know how powerful and expressive it is — but don't be surprised if you're impressed all over again by the new features of UML 2. Perhaps you found some parts of UML too complicated or the apparent benefit too obscure. Well, the UML gurus have revamped UML in many areas — making easier to express yourself exactly and clearly — and they have also added fresh capabilities for the latest software- and system-development problems that you're facing.

But because your problems are complex — and your solutions are sometimes even more complex — UML is not always simple to learn. It's a large and multifaceted language, capable of helping in all areas of development, from analysis to test as well as from database to embedded-real-time. To some, it's a bewildering array of diagrams and symbols. Sometimes it might appear to you that the UML gurus purposely make it too complicated (and with UML 2, even more so) for *the rest of us* to understand.

Bottom line: You need a practical, experience-based guide to the ins and outs of this new language. Let this book be that guide. We boiled down our experiences with UML (in many environments) and our skills as educators to focus on key UML capabilities that you need *first* to be more productive.

So, with straightforward English and concrete examples, we give you a leg up on expressing yourself and being more creative on the job. (Hey, it *could* help you get a raise — just don't expect us to help you get a date.)

How to Use This Book

There's a right way and a wrong way to use this book. Luckily (like its subject, UML 2), this book is remarkably versatile. If you're a traditionalist,

you can read it from cover to cover (although you'll probably stop at the index). That's a great approach if you're really new to UML. If you're familiar with earlier versions of UML, you can skip around looking for the new UML 2 stuff. You may miss our (ahem) great insights into the rest of UML, but you know why you bought the book — do what works. Using any of these techniques will get you familiar with your book so that you can count on it to help unstick you if you hit a snag with UML.

After you make friends with your book, you'll probably find yourself taking advantage of its *just-in-time* features. With just a bit of page flipping, you'll be at a section that's full of examples, tips, techniques, and warnings that will help you with your UML modeling.

There are other ways to use this book . . . and some of them are wrong ways. It's not going to work that well as a doorstop (wrong size), and it probably won't impress your date (unless you're dating a developer who's new to UML). However, it'll look great on your bookshelf — silently conveying to your boss your desire to improve — but if you never open it, you won't get the full benefit.

Some Presumptuous Assumptions

If you're reading this, we can safely assume that not only have you *already* opened the book, you're probably also a developer of software, systems, or databases, and you want to read or write UML 2 diagrams. Perhaps you're a manager or business analyst in the same boat.

We won't assume that you know any particular computer language, although knowing one will certainly help.

For the most part, we assume that you fall into one of two major categories: Either you're a modeler (with a yen to communicate requirements or how you think the world works), or you're a developer (looking to explore alternative designs or communicate your results). Either way, this book is for you.

We assume that you're capable of using a tool to draw UML diagrams — we don't care which one. If the only tool that you have your hands on is *in* your hands (as opposed to on-screen), you won't be at a disadvantage when you use this book (although your diagrams won't be quite as tidy if you're drawing with a stick on wet sand). You may even be better off doing some diagrams by hand; electronic UML tools are often expensive and may not yet be up to date with all the neat UML 2 features that we cover. If you're itching for a high-tech UML tool, take a look at Chapter 23 where we list of some of the more useful examples (in all price categories).

How This Book Is Organized

Here's your first practical hint about using UML: *Put about five to nine major elements on a diagram — no more.* Studies have shown (we've always wondered who does this type of study) that most people have a hard time comprehending more than about nine elements at a time. Likewise, when designing this book, we decided to follow our own advice and to divide the book into just seven parts.

Remember that you don't have to read this book in order. Just choose the parts and chapters that you need at the time.

Part I: UML and System Development

If you want to know what UML is (and why knowing it is useful), this is the place to go; it covers the basics of UML and how it can be used. You'll also find some common principles for communicating or developing systems with UML. These principles guided the UML gurus when they created UML; the same principles can guide you to effective use of it. Ways to apply these principles crop up throughout the book.

Part II: The Basics of Object Modeling

When you model by using UML, the basics are the things (or *objects*) that you draw and the relationships among them. You'll find information on classes, objects, associations, inheritances, and generalizations. No matter what type of development you do, understanding this part will probably be essential.

Part III: The Basics of Use-Case Modeling

Use cases (detailed real-world examples) allow you to understand and communicate the purpose of a system or its components. They are great for organizing your thoughts — and your system — when you want to get a value-added product out the door.

Part IV: The Basics of Functional Modeling

When the objects in your system get busy and you want to explain the details of their complex behavior, you'll need a technique to do so. UML supplies

several to choose from — and this part explains and compares them. You'll see several different types of interaction diagrams (such as sequence, communication, and activity) in action, and discover how to combine them to create solutions, patterns, and frameworks. If you're experienced with UML, you'll find lots of new UML 2 stuff in this part.

Part V: Dynamic Modeling

Your objects are more that just clumps of data stuck together with a few functions. The objects that you develop are more like living things; they remember the past and live their lives by changing their states in response to incoming events. In this part, you can make sure that they get a life — and that you know how to explain it. Come to this part for state charts.

Part VI: Modeling the System's Architecture

Whether you're an architect, programmer, or construction worker, you build complex architectures. Computer systems and software applications distribute themselves across different hardware platforms — and spread throughout the Internet. This part outlines steps that you can use to design your systems for their mission by using system plans, packaging, and subsystems.

Part VII: The Part of Tens

Everyone enjoys making lists (and daydreaming that they'll be read aloud, backward, on late-night talk shows). Here are our top-ten lists of useful tips, tools, Web sites, and diagrams. They're likely to be your top-tens, too.

Icons Used in This Book

Appropriately for a book about graphical communication (even if it *is* software-oriented), there are signposts throughout to help you find your way.

This icon identifies the really new stuff in UML 2. Not every modified feature will get this flag, but it does alert those who are familiar with UML 1.*x* that something's really different here.

Here's a simpler way of doing something that can make it easier than the typical approach. Think of it as a shortcut to better UML.

UML can be a maze — and it can be amazing. These are gentle reminders to reinforce important points.

If you see this icon but ignore it, you'll be in good company but a bad mood.

When you see this icon, you know that we thought the associated material really interesting — but every time we tell people enthusiastically about it, they fall asleep. Skip these sections if you want.

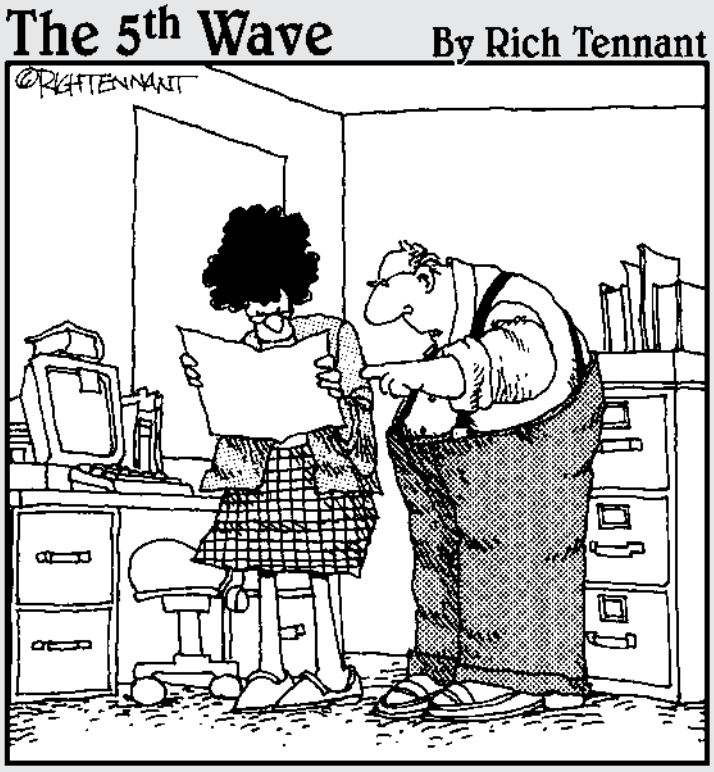
Where to Go from Here

Okay, you're now ready to explore the world of UML 2 modeling. Relax. You've got the tools that you need in your head and your hands (one of them is this book), and it's safe to explore.

So, go ahead and *express yourself* with the power of UML 2.

Part I

UML and System Development



"No, it's not a pie chart; it's just a corn chip that got scanned into the document."

In this part . . .

Building systems or software isn't that tough if you can communicate with your clients, co-workers, managers, and tools. Unfortunately, as your problems get harder and more complex, the risks that emerge from miscommunication become greater — and more severe when they do crop up.

Fortunately, there's a straightforward, visual language that you can use that will help promote more precise and more efficient communication about the nature of your system in all its aspects — software, requirements, architectures, designs, design patterns, and implementations. This language is *UML,* the Unified Modeling Language. The newest version, UML 2, has become more powerful and more useful than ever.

Starting here, we cover the basics of UML. You find out how it may fit your situation, how and when you can use it, and what it's good for. We give you just as much background in history, terminology, and basic principles as you'll need to take advantage of UML's highly productive features.

Chapter 1

What's UML About, Alfie?

In This Chapter

- Understanding the basics of UML
- Exploring the *whys* and *whens* of UML diagrams

So you've been hearing a lot about UML, and your friends and colleagues are spending some of their time drawing pictures. And maybe you're ready to start using UML but you want to know what it's all about first. Well, it's about a lot of things, such as better communication, higher productivity, and also about drawing pretty pictures. This chapter introduces you to the basics of UML and how it can help you.

Introducing UML

The first thing you need to know is what the initials *UML* stand for. Don't laugh — lots of people get it wrong, and nothing brands you as a neophyte faster. It's not the *Universal* Modeling Language, as it doesn't intend to model everything (for example, it's not very good for modeling the stock market; otherwise we'd be rich by now). It's also not the Unified Marxist-Leninists, a Nepalese Political party (though we hope you'll never get *that* confused). It is the University of Massachusetts Lowell — but not in this context. *UML* really stands for the *Unified Modeling Language.*

Well, maybe that's not the most important thing to know. Probably just as important is that UML is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers accomplish the following tasks:

- Specification
- Visualization
- Architecture design

- Construction
- Simulation and Testing
- Documentation

UML was originally developed with the idea of promoting communication and productivity among the developers of object-oriented systems, but the readily apparent power of UML has caused it to make inroads into every type of system and software development.

Appreciating the Power of UML

UML satisfies an important need in software and system development. Modeling — especially modeling in a way that's easily understood — allows the developer to concentrate on the big picture. It helps you see and solve the most important problems now, by preventing you from getting distracted by swarms of details that are better to suppress until later. When you model, you construct an abstraction of an existing real-world system (or of the system you're envisioning), that allows you to ask questions of the model *and get good answers* — all this without the costs of developing the system first.

After you're happy with your work, you can use your models to communicate with others. You may use your models to request constructive criticism and thus improve your work, to teach others, to direct team members' work, or to garner praise and acclamation for your great ideas and pictures. Properly constructed diagrams and models are efficient communication techniques that don't suffer the ambiguity of spoken English, and don't overpower the viewer with overwhelming details.

Abstracting out the essential truth

The technique of making a model of your ideas or the world is a use of *abstraction.* For example, a map is a model of the world — it is not the world in miniature. It's a conventional abstraction that takes a bit of training or practice to recognize how it tracks reality, but you can use this abstraction easily. Similarly, each UML diagram you draw has a relationship to your reality (or your intended reality), and that relationship between model and reality is learned and conventional. And the UML abstractions were developed as conventions to be learned and used easily.

If you think of UML as a map of the world you see — or of a possible world you want — you're not far off. A closer analogy might be that of set of blueprints that show enough details of a building (in a standardized representation with

lots of specialized symbols and conventions) to convey a clear idea of what the building is supposed to be.

The abstractions of models and diagrams are also useful because they suppress or expose detail as needed. This application of *information hiding* allows you to focus on the areas you need — and hide the areas you don't. For example, you don't want to show trees and cars and people on your map, because such a map would be cumbersome and not very useful. You have to suppress some detail to use it.

You'll find the word *elide* often in texts on UML — every field has its own jargon. Rumor has it that *elide* is a favorite word of Grady Booch, one of the three methodologists responsible for the original development of UML. *Elide* literally means to omit, slur over, strike out, or eliminate. UML uses it to describe the ability of modelers (or their tools) to suppress or hide known information from a diagram to accomplish a goal (such as simplicity or repurposing).

Chapter 2 tells you more about using these concepts of *information hiding* and *abstraction* during development.

Selecting a point of view

UML modeling also supports multiple views of the same system. Just as you can have a political map, a relief map, a road map, and a utility map of the same area to use for different purposes — or different types of architectural diagrams and blueprints to emphasize different aspects of what you're building — you can have many different types of UML diagrams, each of which is a different view that shows different aspects of your system.

UML also allows you to construct a diagram for a specialized view by limiting the diagram elements for a particular purpose at a particular time. For example, you can develop a *class diagram* — the elements of which are relevant things and their relationships to one another — to capture the analysis of the problem that you have to solve, to capture the design of your solution, or to capture the details of your implementation. Depending on your purpose, the relevant things chosen to be diagram elements would vary. During analysis, the elements that you include would be logical concepts from the problem and real world; during design, they would include elements of the design and architectural solution; and during implementation, they would primarily be software classes.

A *use case diagram* normally concentrates on showing the purposes of the system (use cases) and the users (actors). We call a use case diagram that has its individual use cases elided (hidden) a *context diagram,* because it shows the system in its environment (context) of surrounding systems and actors.

Choosing the Appropriate UML Diagram

UML has many diagrams — more, in fact, than you'll probably need to know. There are at least 13 official diagrams (actually the sum varies every time we count it) and several semiofficial diagrams. Confusion can emerge because UML usually allows you to place elements from one diagram on another if the situation warrants. And the same diagram form, when used for a different purpose, could be considered a different diagram.

In Figure 1-1, we've constructed a UML class diagram that sums up all the major types of UML diagrams (along with their relationships), using the principle of *generalization*, which entails organizing items by similarities to keep the diagram compact. (See Chapter 2 for more information on generalization.)

In Figure 1-1, the triangular arrows point from one diagram type to a more general (or more abstract) diagram type. The lower diagram type is a *kind-of* or *sort-of* the higher diagram type. Thus a `Class Diagram` is a kind of `Structural Diagram`, which is a kind of `Diagram`. The diagram also uses a dashed arrow to indicate a dependency — some diagrams reuse the features of others and depend on their definition. For example, the `Interaction Overview Diagram` depends on (or is derived from) the `Activity Diagram` for much of its notation. To get a line on how you might use UML diagrams, check out the summary in Table 1-1.

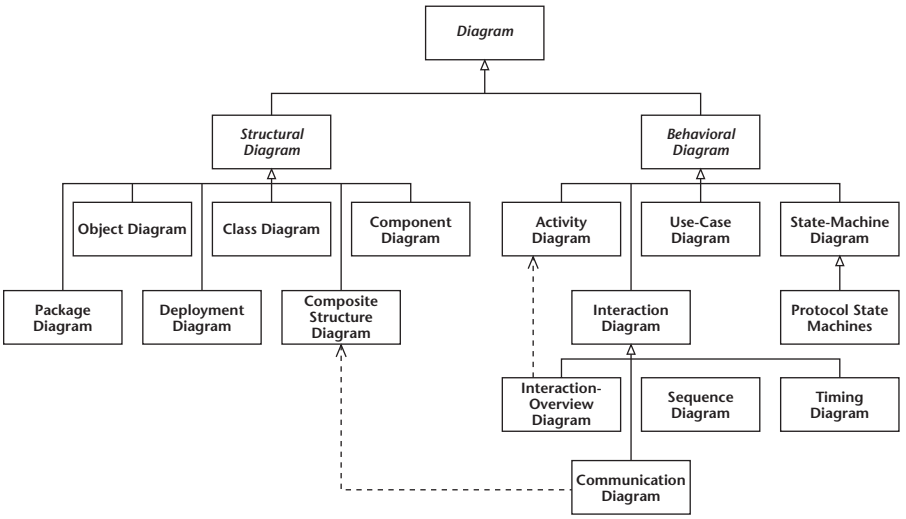


Figure 1-1: A class diagram of UML diagrams.