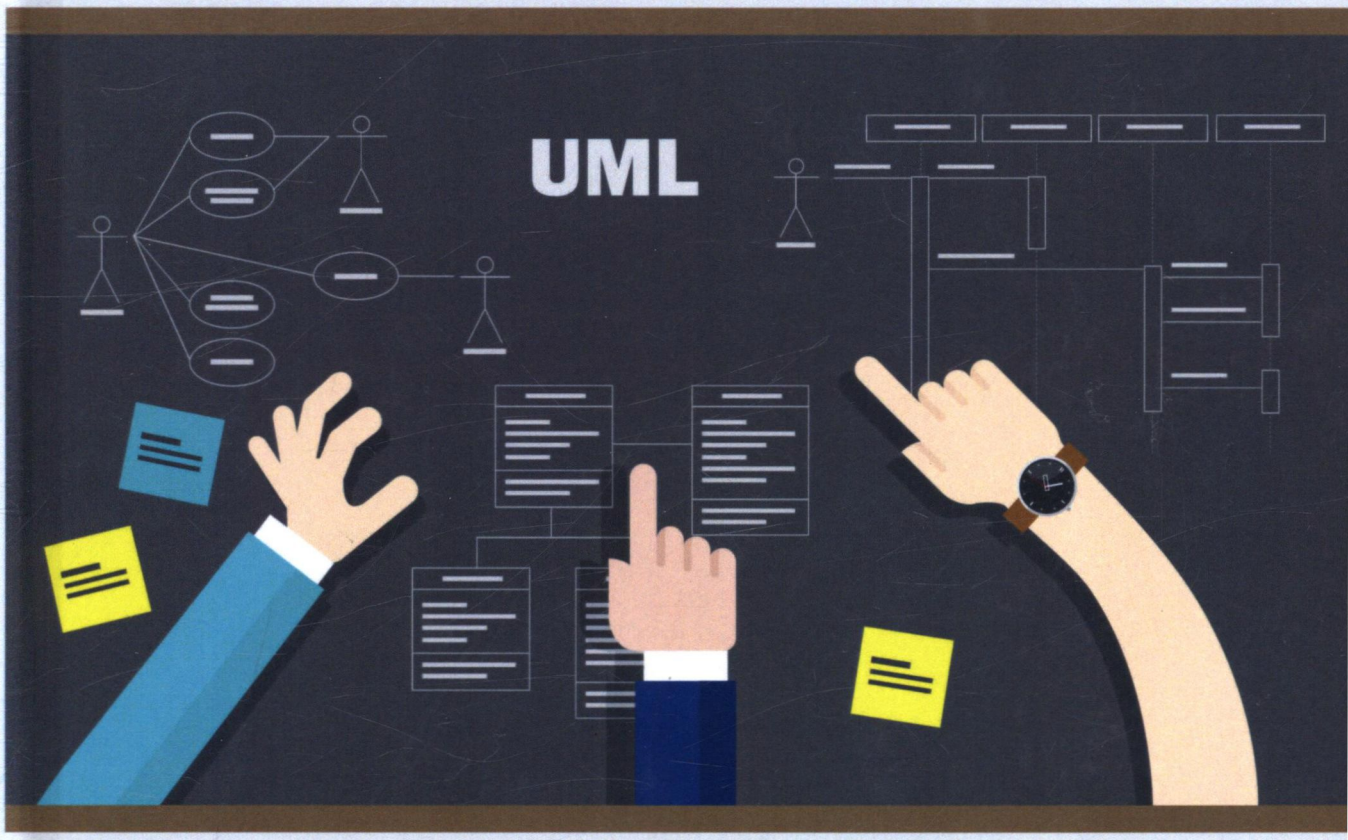
Software Engineering with UML



Bhuvan Unhelkar

CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

Software Engineering with UML uses the Object Management Group's Unified Modeling Language (UML 2.5) standard to engineer high-quality software solutions. The premise of this textbook is that communication is the key to good software engineering and that modeling forms the basis of such communications. UML-based models facilitate and enhance communication between business analysts, users, designers, architects, and testers of the system under development. The textbook covers the 14 different modeling constructs in UML 2.5.

Because the object-oriented approach to developing software introduces fundamentals for high quality software development, the topic is interwoven throughout this book in discussing the fundamentals of software engineering, applying those fundamentals in modeling, and developing software solutions. The UML is presented as three interrelated models: Model of the Problem Space (MOPS), Model of the Solution Space (MOSS), and Model of the Architectural Space (MOAS).

The textbook also provides helpful hints on how a software engineer can work in an Agile development environment and understand the wider project management aspect of producing software solutions.

This textbook helps software engineers appreciate the importance and the relevance of software modeling in creating high-quality software programs. Budding software engineers need to learn right from the outset that developing good solutions is a lot more than "coding." This book covers additional topics, such as user interface design, nonfunctional requirements (NFRs), quality assurance, and testing, to ensure appropriate breadth and sufficient depth that is apt for teaching-learning software engineering.

This book is based on the author's teaching, research, and practical experience in software engineering. Students and practitioners alike will find themselves building on the knowledge gained here and applying it to the intricacies of software engineering.

For instructors, Web support for this book includes:

- All presentation material including all figures and slides for each chapter
- Suggestions on tutorial sessions and roughly worked examples for the team project
- Administrative and lab requirements for the project work including suggested CASE tools
- Suggestions on assessments and marks and time distribution

Software Engineering with UML is designed to be of value to both undergraduate and postgraduate courses in software modeling through appropriate selection of chapters and corresponding emphasis on exercises and case studies. The value for practitioners lies in the example-based explanations and practical hints and tips through the discussions.

CRC Press
Taylor & Francis Group
an **informa** business
www.crcpress.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
711 Third Avenue
New York, NY 10017
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

K35643

ISBN: 978-1-138-29743-2

www.crcpress.com

Unhelkar Software Engineering with UML

Software Engineering with UML

Bhuvan Unhelkar

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

International Standard Book Number-13: 978-1-138-29743-2 (Hardback)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Software Engineering with UML

Janalee and Allen Heinemann

Who succinctly abstracts humanity in all its joys and grace:

Worth Modeling!

Foreword

A few months ago, I was talking with one of my colleagues, and our conversation meandered into modeling skills, or rather the lack thereof, in the new generation of software engineers. More specifically, my colleague lamented the lack of knowledge about the Unified Modeling Language (UML) and object-oriented (OO) design skills among developers these days. Time and again we had both run into very smart developers whose effectiveness was nowhere near what it should be mainly owing to their lack of modeling skills and experience. Worse yet, many of them were proud of it! Yikes!

To be fair, though, one of the problems seems to be the lack of emphasis on good modeling and design in software engineering curricula. Adding to this problem is the fact that so few good books have been written on UML in the past five years. The need for books and other teaching materials that explain software engineering with UML and show its application in practice cannot be overemphasized. This modeling material also needs to keep up with the changes in software engineering such as the development of services, analytics, and mobile apps. Yet, even online forums, sites, and blogs that discuss software modeling do so sparingly. Budding programmers rarely get to see good examples of models and are not able to develop their skills and techniques in applying these models in enhancing the quality and productivity of their work.

And there is a dire need for both quality and productivity to improve in practice. From controllers in street lights to aviation systems and battlefield targeting, software systems permeate every aspect of our society. These software systems need proper definitions, architectures, designs, testing, and sensible deployment. A commonly-understood, standardized modeling language is imperative. The UML defines the standard modeling artifacts when it comes to OO technology.* The use of UML has significantly enhanced the quality and acceptability of software applications by enabling formal requirements modeling, undertaking quality designs, and providing sound basis for iterative development of solutions.

In this book, Dr. Unhelkar applies UML to these critical societal functions. While the primary focus of this book is to teach UML standards, techniques, diagrams, and models, they are all couched within the fundamentals of OO software engineering. These OO fundamentals—classification, abstraction, inheritance, association, encapsulation, and polymorphism—set the tone for the use of UML in subsequent chapters. The contents within each chapter reflect the experiences of Dr. Unhelkar in teaching and practicing software engineering with the UML. The book is replete with short, simple examples that explain the fundamentals of OO software engineering; then there the book explains the use of each of the UML diagrams and their relevance in practice. It also has one of the key things that I always look for in a book—a running example of practical case study that helps make the material relevant to both students and practitioners of software

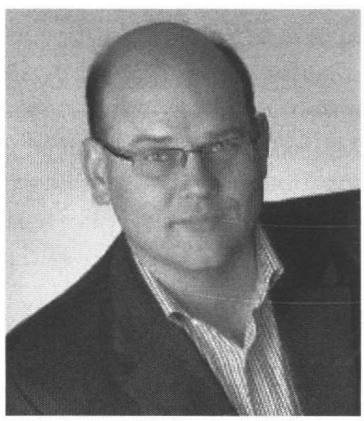
* The Object Primer 3rd Edition: Agile Modeling Driven Development with UML 2.—See more at http://www.agilemodeling.com/essays/umlDiagrams.htm

engineering. The presentation of common errors in modeling, discussion questions, team-based project work, and quizzes makes this book invaluable to readers. I think this book goes a long way toward helping rectify the situation around the lack of modeling in teaching and practice.

Now, you may wonder why am I writing the foreword for this book. I've been associated with UML since its inception in the mid-1990s; I wrote the first publicly published article about UML for *Object Magazine* in 1996, and my second book, *Building Object Applications That Work*, published in 1997, was the first book to cover UML. In 2002, my book *Agile Modeling* featured UML extensively, showing how to take a lightweight approach to modeling and documentation. In 2005, the *Object Primer 3rd Edition* went into even more detail about how UML and modeling in general are key aspects of enterprise-class Agile software development. So I have a fairly deep background in UML and OO software design and have written about it extensively. During my visits to Australia, I was also invited by Dr. Unhelkar to present at the Special Interest Group of the Australian Computer Society. In the mid-2000s, my focus shifted from UML and objects to software process in general, culminating in my continuous work with Mark Lines on the Disciplined Agile (DA) framework. Despite this change in focus, I remain extremely interested in developing good software models using UML standards.

I recommend this book to anyone who is serious about software engineering. The fundamental skills and knowledge about software engineering and UML outlined here will be of immense value to both students and practitioners. I take this opportunity to compliment Dr. Unhelkar on authoring this much needed yet simple and practical book on a vital topic within software engineering.

Scott Ambler
Toronto, Canada



Scott Ambler is a Senior Consulting Partner with Scott Ambler + Associates, working with organizations around the world to help them improve their software processes. Ambler is globally known for training, coaching, and mentoring in disciplined Agile and lean strategies at both the project and organizational level. Ambler is (co-)author of several books and white papers on object-oriented software development, software process, disciplined Agile delivery (DAD), Agile model-driven development (AMDD), Agile database techniques, and the Enterprise Unified Process (EUP)™. He is also a regular invitee for keynote addresses in conferences worldwide. He is a Fellow of the International Association of Software Architects and the Disciplined Agile Consortium. He was a Senior Contributing Editor with *Dr. Dobb's Journal* and occasionally writes for Cutter Consortium and IBM Developerworks. Please visit ScottAmbler.com for further details.

Preface

*Modeling saves time and energy.**

Welcome to *Software Engineering with UML.* This book acknowledges and uses the Object Management Group's Unified Modeling Language (UML 2.5) standard to engineer high-quality software solutions. In an age of ever-increasing demand on software developers, clarity of communication and conveyance of understanding are prerequisites for success. Rapidly changing technologies for development, crunching time to produce working solutions, unpredictable business and legal environments, exploding data, cross-platform testing, globally dispersed development teams, and incessant requirements dictated by highly knowledgeable users place a premium on the technical and professional skills of a software engineer.

The premises of this book are that communication is the key to good software engineering and that modeling forms the basis of such communication. UML-based models facilitate and enhance communication between business analysts, users, designers, architects, and testers of the system under development. UML version 2.5 covers 14 different modeling constructs (package, use case, activity, interaction overview, class, sequence, communication, object, state machine, component, deployment, composite structure, timing, and profile diagrams). UML diagrams are based on a robust meta-model, which also enables extensibility mechanisms (stereotypes, tags, and notes).

An object-oriented approach to developing software introduces fundamentals for high-quality software development. Therefore, the topic of object orientation is interwoven throughout this book—in discussing the fundamentals of software engineering and applying those fundamentals in modeling, and developing software solutions.

UML grew out of a need to standardize a varying sets of notations and design approaches. Today it has evolved and stabilized for use across multiple software engineering functions such as capturing and modeling requirements of the problem to be solved, designing and prototyping the software solution, and understanding the constraints and impact of the solution on the existing enterprise-level architecture. UML is presented in this book as three interrelated models: model of the problem space (MOPS), model of the solution space (MOSS), and model of the architectural space (MOAS). These models are not watertight compartments but, rather, a way of delineating the tools (diagrams) provided by the UML based on a role and its purpose within a software project. As an ISO standard, UML certainly forms an integral part of a software engineer's toolkit.

Methods (processes) for developing software solutions form an important and integral part of software engineering. This material touches key areas of software development methods. Helpful

* From a Tony Robbins seminar.

hints are provided on how a software engineer can work in an Agile development environment and also understand the wider project management aspect of producing software solutions.

In the era of mobile apps, Cloud-based services, the Internet of Things (IoT), and Big Data analytics, a skeptic might be prone to discount the value of modeling (and in particular UML). Successful software development shows that disciplined modeling remains integral to communications across multiple stakeholders involved in developing solutions. The aim in this book is to make software engineers appreciate the importance and the relevance of software modeling in creating high-quality software programs.

Budding software engineers need to learn from the outset that developing good solutions involves a lot more than "coding." While programming is a necessity in the field of software, it is *not* sufficient. For example, user interface design, nonfunctional requirements (NFRs), quality assurance, and testing are crucial topics in software engineering that are more or less beyond the UML. This book covers these additional topics to ensure the appropriate breadth and sufficient depth that are necessary for teaching and learning software engineering.

This book is based on the author's teaching, researching, and experiencing the nitty-gritty and nuances in the field of software engineering. Students and practitioners alike will find themselves building on the knowledge gained here and applying it to the intricacies of software engineering. The book is designed to be of value to both undergraduate and postgraduate courses in software modeling through appropriate selection of chapters and corresponding emphasis on exercises and case studies. The value for practitioners is embedded in the example-based explanations and practical hints and tips through the discussions.

Audience

The primary audiences of this book are:

- *Students (undergrad)*: These are the basic- to intermediate-level readers learning software engineering at an undergraduate level. These readers are keen to understand the basics of software engineering followed by a standard way to model requirements and create design solutions using UML.
- *Students (postgrad)*: These are readers looking for greater details on building an overall holistic software solution. These readers go deeper into the architectural and design aspects of solutions, and they are also keen to understand the process and management aspects of software projects. The impact of advanced concepts (e.g., reuse, granularity, patterns) on software solutions is also of interest to these readers.
- *Business analysts/requirements modelers*: These readers are learning to capture and model requirements using UML standards (notably use cases, activity diagrams, and business-level basic class diagrams). These people work primarily to develop the model of the problem space (MOPS).
- *Quality professionals*: These are the quality analysts and testers aiming to improve their work in enhancing the quality of a solution by inspecting the models, undertaking walk-throughs, and verifying and validating the models. These readers also need the UML to understand and communicate with the users, designers, and architects of the software solutions.
- *Teachers*: These include professors who are keen to pitch the right material at the right level. Teachers will find this book an excellent text for a typical one-semester subject (unit) totally supported by presentation material and case studies (available on the publisher's website).

- *Trainers*: Trainers conducting a 2- to 3-day industrial course in software modeling or business analysis will find that this book supports their training efforts. The book's value to trainers lies in the succinct organization of chapters with the opportunity to choose the chapters depending on the audience and time provided to conduct trainings. The team project case study enables experiential learning in industrial training courses.
- *Consultants/practitioners*: These are readers who will find the practical content and a running case study through the chapters to be of immense value.

Assumed Knowledge

This book assumes a general (introductory) understanding of software development (for example, what is a software system and what is meant by analysis and design?). Students can develop this understanding through any programming- or database-related course or by reading and absorbing the basics of analysis and designs. Practitioners easily gain this understanding through their experience. Such an introductory understanding of software development makes it easier and quicker to grasp the concepts of software engineering, object orientation, and UML-based modeling discussed in this book.

Contents

This book is divided into 21 chapters, each reflecting a topic of discussion relevant to a 90-minute industrial training session or a 2-hour lecture. Assuming an introduction, a concluding lecture, and a midterm test, this book covers a teaching period of approximately 14 weeks for an undergrad course in software modeling (or program design). An alternative selection of chapters and greater emphasis on the team project case study result in material for a graduate course in software engineering.

At the end of each chapter, readers will find discussion questions (which can be treated like exercises). It is highly recommended that these discussion questions are completed immediately after the lecture or reading of the chapter. The discussion questions are designed to help students consolidate the concepts discussed in the chapter. Each chapter also has the steps outlined for a case study. The case study must be performed on a team to enable students to appreciate the challenges and advantages in using UML in real-life software projects. Three to four students are expected to participate in this team project. The team project work is performed during the tutorials in labs outside the lecture times. The team project requires the use of a UML-based CASE tool (for example, StarUML or Visio™).

Pedagogy

This book is written for the purpose of teaching and learning UML within the context of object orientation. This book is relevant for both undergraduate and graduate students. The examples in the book are derived from the author's practical industrial experience, yet the teaching experience ensures the book will fall short with respect to academic rigor and authenticity. The book is a combination of the author's experience in various practical consulting roles—including business analysis, project management, system design, quality assurance, and testing—combined with years of teaching and coordinating UML courses at both undergraduate and graduate levels across universities in Australia, the USA, China, and India.

Web Support

Suggested structure and formats for presentations of this material, typical assessments with timings and marks, as well as administrative requirements for this subject are available on the CRC Press/Taylor & Francis Group website (https://www.crcpress.com/9781138297432). Web support for this book includes:

- All presentation materials including all figures and slides for each chapter
- Suggestions on tutorial sessions and roughly worked examples for the team project
- Administrative and lab requirements for the project work including suggested CASE tools
- Suggestions on assessments and marks and time distribution
- Appendix A: Case Study Problem Statements for Team Projects with additional case studies not included in this book.
- Appendix B: Mid-Term
- Appendix C: Final Exam

Critiques

Readers are invited to submit criticism of this work. It would be an honor to receive genuine criticism and comments on this material that, I am sure, will not only enrich my own knowledge and understanding of the topics discussed in this book but also add to the general wealth of modeling knowledge available to the ICT community. Therefore, I extend a *thank you* in advance to all potential critics of this work.

Bhuvan Unhelkar

www.unhelkar.com

Glossary of Acronyms

BA	Business Analyst (not to be confused with business architect)
BDFAB	Big Data Framework for Agile Business
BDM	Business Domain Model—represented by class diagram at a high level (i.e., not containing technical details)
BO	Business Objective (basis for software projects)
BPM	Business Process Model—representing workflow or business processes
BPMN	Business Process Model and Notation
CAMS	Composite Agile Method and Strategy
CBT	Computer-Based Training (for users before deploying a system; automated in many cases)
CC	Cloud Computing (anything on the Cloud—includes computing, storage, analytics, platform, and infrastructure)
CMM	Capability Maturity Model—provides basis for measuring and comparing process maturities of various organizations and projects; initiative of Software Engineering Institute at Carnegie Mellon University
CMMi	Capability Maturity Model integration
CMS	Content Management System—dealing primarily with the contents of a website and its management
CRM	Customer Relationship Management—a comprehensive system including interfaces, processes, and databases to handle all aspects of customer-related processes (from identifying, marketing, and selling through to support and retirement)
CWM	Common Warehouse Metamodel
DAD	Disciplined Agile Development
DE	Domain Expert—in a particular domain or industry like banking, airlines, or hospitals
DM	Data Modelers—focusing on creating models to represent databases in backend
EA	Enterprise Architecture—brings together various (primarily technical) aspects of an enterprise/organization
ERP	Enterprise Resource Planning—typically representing large and complex software systems that include all functions of an organization (e.g., SAP, PeopleSoft, Oracle)
GUI	Graphic User Interface—also known as screens or forms
HMS	Hospital Management System—a case study used in this text to demonstrate practical application of software engineering with UML
ICT	Information and Communication Technology
ID	Interface Designer—specialist in designing various types of interfaces including, but not limited to, graphics.

IIoT	Industrial Internet of Things
IIP	Iterative, Incremental, Parallel—software development life cycle ideally suited for OO development
IOD	Interaction Overview Diagram—part of UML providing high-level overview of interaction diagrams
IoE	Internet of Everything—a more generic term to include IoT
IoT	Internet of Things—represents daily use devices that are connected to the base and with each other through the Internet
ISAM	Indexed Sequential Access Method—a method to access data through indexes
IT	Information Technology—increasingly being referred to as ICT
MDA	Model-Driven Architecture (OMG initiative)
Metamodel	Model of a Model that dictates the rules for the creation of modeling mechanisms like the UML
MOAS	Model Of Architecture Space—created primarily by the system architect in the background space using UML notations and diagrams
MOF	Meta-Object Facility—owned by OMG and forms basis for the creation of new methods
MOPS	Model Of Problem Space—created primarily by business analyst in problem space using UML notations and diagrams
MoSCoW	Must–Should–Could–Won't (four categories/priorities in terms of requirements of a software system)
MOSS	Model Of Solution Space—created primarily by system designer in solution space using UML notations and diagrams
NFR	NonFunctional Requirement—also known as operational requirement
NFRS	NonFunctional Requirement Specifications—also nonfunctional requirements
OMG	Object Management Group—responsible for unification of modeling notations resulting in UML
OO	Object Oriented—earlier considered only as a programming technique, OO now permeates all aspects of software engineering
PIoT	Personal Internet of Things
PM	Project Manager
QA	Quality Assurance
QC	Quality Control
QM	Quality Management
RM	Requirements Modeler
SA	System Architect
SD	System Designer
SDLC	Software Development Life Cycle
SEP	Software Engineering Process (also software process)
SMD	State machine Diagram—also known as state chart or state diagram
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration
UML	Unified Modeling Language
V&V	Verification and Validation
WS	Web Services
XML	eXtensible Markup Language