大学计算机教育国外著名教材系列（影印版）

PEARSON
Addison Wesley

UML DISTILLED
A BRIEF GUIDE TO THE STANDARD OBJECT MODELING LANGUAGE
THIRD EDITION

UML精粹
标准对象建模语言简明指南（第3版）

Martin Fowler 著

PEARSON Education

清华大学出版社

育国外著名教材系列（影印版）

UML Distilled

A Brief Guide to the Standard Object Modeling Language

Third Edition

UML 精粹

标准对象建模语言简明指南

（第 3 版）

Martin Fowler　著

清华大学出版社
北　京

图书在版编目(CIP)数据

UML 精粹：标准对象建模语言简明指南：第 3 版 /（美）福勒（Fowler, M.）著. —影印本. —北京：清华大学出版社，2006.3
（大学计算机教育国外著名教材系列）
ISBN 7-302-12542-2

Ⅰ. U…　Ⅱ. 福…　Ⅲ. 面向对象语言，UML－程序设计－高等学校－教材－英文　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2006）第 009651 号

出 版 者：	清华大学出版社	**地　　址：**	北京清华大学学研大厦
	http://www.tup.com.cn	**邮　　编：**	100084
	社总机：010-6277 0175	**客户服务：**	010-6277 6969

印 装 者：清华大学印刷厂
发 行 者：新华书店总店北京发行所
开　　本：185×230　**印张：**12.75
版　　次：2006 年 3 月第 1 版　2006 年 3 月第 1 次印刷
书　　号：ISBN 7-302-12542-2/TP・8022
印　　数：1~3000
定　　价：26.00 元

出 版 说 明

进入 21 世纪，世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才，谁就能在竞争中取得优势。高等教育，作为培养高素质人才的事业，必然受到高度重视。目前我国高等教育的教材更新较慢，为了加快教材的更新频率，教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始，与国外著名出版公司合作，影印出版了“大学计算机教育丛书（影印版）”等一系列引进图书，受到国内读者的欢迎和支持。跨入 21 世纪，我们本着为我国高等教育教材建设服务的初衷，在已有的基础上，进一步扩大选题内容，改变图书开本尺寸，一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材，组成本套“大学计算机教育国外著名教材系列（影印版）”，以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材，以利我们把“大学计算机教育国外著名教材系列（影印版）”做得更好，更适合高校师生的需要。

清华大学出版社

本书的评价

"*UML Distilled* remains the best introduction to UML notation. Martin's agile and pragmatic approach hits the sweet spot, and I wholeheartedly recommend it!"

—Craig Larman
Author of *Applying UML and Patterns*

"Fowler cuts through the complexity of UML to get users started quickly."

—Jim Rumbaugh
Author and originator of UML

"Martin Fowler's *UML Distilled* is an excellent way to get started with UML. In fact for most users, *UML Distilled* contains all you need to apply UML successfully. As Martin points out, UML can be used in many ways, but the most common is as a widely recognized notation for sketching designs. This book does an excellent job of distilling the essence of UML. Highly recommended."

—Steve Cook
Software Architect
Microsoft Corporation

"Short books on UML are better than long books on UML. This is still the best short book on UML. In fact, it's the best short book on many subjects."

—Alistair Cockburn
Author and President, Humans and Technology

"The book is immensely useful, readable, and—one of its great virtues—delightfully concise for the immense scope of its subject. If you only buy one book on UML, this should be it."

—Andy Carmichael
BetterSoftwareFaster, Ltd.

"If you're using UML, this book should never be out of reach."

—John Crupi
Distinguished Engineer, Sun Microsystems
Coauthor of *Core J2EE™ Patterns*

"Anyone doing UML modeling, learning UML, reading UML, or building UML tools should have this latest edition. (I own all editions.) There is lots of good, useful information; generally, just enough to be useful, but not too much to be dry. It's a must-have reference for my bookshelf!"

—Jon Kern
Modeler

"This is a great starting point for learning the fundamentals of the UML."

—Scott W. Ambler
Author of *Agile Modeling*

"An eminently sensible description of UML and its usage, with enough humor to hold one's attention. 'The swimming metaphor no longer holds water' indeed!"

—Stephen J. Mellor
Coauthor of *Executable UML*

"This is the perfect book for those who want to use the UML but aren't interested in reading thick UML reference books and research papers. Martin Fowler selects all the critical techniques needed to use the UML for design sketches, freeing the reader from complex and rarely used UML features. Readers will find no shortage of suggestions for further reading. He gives the reader advice based on experience. It's a concise and readable book covering the essential aspects of the UML and related object-oriented concepts."

—Pavel Hruby
Microsoft Business Solutions

"Like all good software developers, Fowler improves his product with each iteration. This is the only book I consider when teaching a class involving UML or if asked to recommend one that can be used to learn it."

—Charles Ashbacher
President/CEO, Charles Ashbacher Technologies

"More books should be like *UML Distilled*—concise and readable. Martin Fowler selects the parts of UML that you need, and presents them in an easy to read style. More valuable than a mere description of the modeling language, however, is the author's insight and experience in how to use this technique to communicate and document design."

—Rob Purser
Purser Consulting, LLC.

第3版前言

自古以来，最有才干的建筑师以及最杰出的设计师都确信“俭省定律”（law of parsimony）。不管它是陈述成一种悖论（“少即是多”），还是陈述成一种公案（“禅心即本心”），其智慧是永恒的。将任何事都简化到本质，使形式与功能相谐。从金字塔到悉尼歌剧院，从冯 • 诺依曼体系结构到 UNIX 及 Smalltalk，最佳建筑师与设计师们都力求遵循这一普遍的永恒原则。

认识到利用奥卡姆剃刀（Occam’s Razor）剃须的意义，当我在设计与阅读时，总在寻找一些遵循俭省定律的项目和书籍。因此，我对你现在正在阅读的这本书赞赏有加。

你可能起先对我的上述评论感到惊异。我经常接触到定义统一建模语言 UML 的冗长而难懂的规范。这些规范使工具卖主得以实现 UML 并使方法学学者得以应用 UML。七年来，我主持了一些大型国际标准化组，制定了 UML 1.1 与 UML 2.0 的规范以及其间若干次较小的修订。这段时期，UML 在表现能力与精确程度上已臻成熟，但是，由于标准化过程，也增加了不必要的复杂性。遗憾的是，标准化过程乃是因其“按委员会设计”的折衷妥协而不是因其俭省典雅而闻名于世的。

熟悉规范的晦涩难解之细微末节的 UML 专家能从 Martin 对 UML 2.0 的精粹提炼中学到什么呢？可以学到很多。一开始，Martin 就巧妙地将一种庞大复杂的语言简化成一个颇重实效的子集，后者在实践中已被证明效果不错。他摒弃了对该书的前一版增补篇幅的简易途径。由于这种语言的发展壮大，Martin 一直坚持其寻求“UML 最为有用的部分”的目标，并且所讲述的也正是这一部分。他所指的这一部分就是帮助你做 80%工作的 UML 中神话般的 20%部分。俘获并驯服这匹在逃的野兽乃是一项了不起的成就！

更令人钦佩的是，Martin 是以一种精彩动人的对话文体来实现这一目标的。通过与我们共享他的意见及趣闻轶事，他使本书成为一本有趣的读物，并且使我们联想到构建与设计系统应是既有创见又富有成效的。如果我们追求俭省公案到尽善尽美的地步，就应发现，用 UML 对项目建模就犹如在小学或中学中发现指画班与绘画班那样令人愉快。UML 应是我们创造性的一支照明杖，以及“精确指明系统蓝图以致第三方可以索价并构作系统”的一台激光器，后者乃是对任何一种真正蓝图语言的酸性检验。

因此，虽然这可能是一本小书，它却不是一本平凡的书。你可以从 Martin 的建模途径学到的东西几乎和你从他对 UML 2.0 的解释所学到的东西一样多。

在和 Martin 一道工作，以增进本次修订所阐明的 UML 2.0 的语言特征的选取与纠错上使我感到愉快。我们必须记住，所有有生命的语言（自然语言与人工语言）都必然发展或消亡。Martin 的关于新特征的选择连同你的爱好以及其他实践者的爱好，乃是 UML 修订过程中的重要部分。它们使这种语言保持生气勃勃，并帮助它通过市场中的

自然选择而发展。

在模型驱动开发成为主流以前还有很多挑战性的工作要做，但是我却因像这样一本清晰阐明 UML 建模基本原理并进行实际应用的书籍而感到鼓舞。我希望你和我一样能从中获益，并利用新的深刻了解来改进自己的软件建模实践。

Cris Kobryn
U2 伙伴的 UML 2.0 提交组主席
Telelogic 公司首席技术专家

第 1 版前言

在开始制订统一建模语言时，曾希望能产生一种表示设计的标准方式，它不仅能反映最佳工业实践，还能帮助打消软件系统建模过程的神秘。我们相信，使用标准建模语言定能鼓励更多开发人员在构作软件系统之前对系统建模。UML 的迅速广泛采用表明软件开发界人士确已深知建模的好处。

UML 的创建本身为一迭代与渐进过程，这和大型软件系统的建模颇为相似。最终结果成为一标种准，它建立在面向对象界很多个人和公司提出的想法和所做贡献的基础之上，并且也反映了这些想法与贡献。我们开始致力于 UML，但很多别人帮助我们臻于成功。对他(她)们的贡献特致谢意。

创建并商定一种标准建模语言本身为一重要挑战。如何教育软件开发界，并以一种既易理解又按软件开发过程的方式来介绍 UML 亦为一重要挑战。在这本容易被人误解、且更新到反映 UML 最新变动的小书中，Martin Fowler 遇到了更多的挑战。

Martin 不仅以一种清晰而友善的文体介绍了 UML 的关键方面，而且还清楚阐明了 UML 在开发过程中所起的作用。阅读中，我们享受到了 Martin 从 12 年以上的设计与建模经验中所得到的大量宝贵的建模见识和智慧。

结果是一本引导成千上万开发人员进入 UML、并促使他（她）们更热衷于以这种当前标准建模语言进一步开拓很多建模好处的书。

谨向任何一位有兴趣于首先览读 UML、并对它在开发过程中所起的关键作用有一概貌了解的建模人员或开发人员推荐此书。

Grady Booch
Ivar Jacobson
James Rumbaugh

序

在我的人生中有很多方面都是幸运的。我的一次鸿运是于 1997 年在合适的地点以合适的知识撰写了本书的第 1 版。溯及那时，面向对象建模的混乱天地刚刚开始在统一建模语言（UML）的旗帜下统一起来。从那时起，UML 已成为不只是对象的标准而已是软件图示建模的标准。我的幸运是，这本书成为一本最流行的 UML 书，销售量超过 25 万册。

好！这对我来说固然不错，但你是否应该购买这本书呢？

我要强调的是，这是一本篇幅简短的书。它不指望给出这些年来日益发展的 UML 每一方面的细节。我的意图是去发现 UML 中最为有用的部分，并且只对你讲述这一部分。虽然一本篇幅较大的书会使你了解更多的细节，但也会占用你更多的阅读时间。而时间是你对一本书的最大投资。为了保持这本书篇幅不大，我曾经花时间去选择最佳的部分，这样你就不必去自行选择了。（令人难过的是，“更小”并不总是意味着“更便宜”，产生一本高质量的技术书是有明确固定成本的。）

购买本书的理由之一是要开始学习 UML。由于这是一本小书，它会使你很快抓住 UML 的基本要点。掌握了这些内容以后，就可以通过一些篇幅较大的书（如《用户指南》［Booch, UML user］或《基准手册》［Rumbaugh, UML Reference］）来研究 UML 的更多细节。

本书亦可用作 UML 的最常用部分的方便参考书。虽然本书并未涵盖 UML 的所有部分，但比起很多别的 UML 读物，它却是一本颇为轻便、易于携带的书。

它也是一本作者有自己见地的书。我已经在对象领域工作了很长一段时间，因而，对什么工作起来得心应手、什么不然，有明确的想法。任何一本书都反映作者的意见，我也不打算隐瞒自己的看法。因此，如果你在找寻带有客观性的东西，你就要去尝试别的书。

虽然很多人曾经告诉我，本书是一本关于对象的良好导论，但这却非我的撰写本意。如果你要找寻一本面向对象设计的导论，我推荐 Craig Larman 的书［Larman］。

很多对 UML 感兴趣的人都利用工具。本书集中考虑 UML 的标准以及通常用法，并未涉及各种工具所支持的细节。虽然 UML 消解了“前 UML”表示法的巴别（Babel）塔，但在绘制 UML 图时，仍然在工具示明什么和允许什么之间有一些令人烦恼的差异。

在本书中我没有过多谈到模型驱动体系结构（MDA）。虽然很多人把这两者（指 UML 和 MDA——译注）看作一回事，但很多开发人员利用 UML 而对 MDA 却并无兴趣。如果你要多学一些 MDA，我就应该开始先用这本书使你能了解到 UML 的梗概，然后再转去阅读一本更为专门的 MDA 的书。

虽然本书的要点是 UML，我也增加了少许像 CRC 卡那样对面向对象设计有价值的别的技术材料。UML 只是你用对象取得成就所需要的一部分，而我认为，向你介绍一些别的技术，还是重要的。

在这样一本简短的书中，不可能涉及到有关 UML 如何与源代码联系的问题，特别是，尚无一种标准方式去进行这样的对应。但是，我要指出一些实现 UML 片段的常用编码技术。我的代码例子是用 Java 与 C#书写的，这是因为我发现这些语言通常是最被广泛了解的语言。不要假定我喜欢 Java 和 C#，我用 Smalltalk 写的代码太多了！

为何对 UML 操心?

我们使用设计图示法已经有了一段时间。对我来说，其主要价值在于交流与理解。一个好的图往往能帮助你交流设计想法，特别是在你想要避免很多细节时。图也可以帮助你理解一个软件系统或一个业务过程。作为试图明白一些事情的开发组一方，图既能帮助你理解又能帮助你把这些理解在开发组内交流。虽然设计图示法不是(至少还不是)正文编程语言的接替，它们却是一个有益的助手。

很多人相信，在将来，图示技术在软件开发中将起统率作用，我却对此比较怀疑，但是，鉴别这些表示法能做什么，不能做什么，则肯定是有用的。

在这些图示法中，UML 的重要性源于它在面向对象开发界的广泛使用与标准化。UML 已经成为不只是面向对象界内部的主要图示法，而且在非面向对象界也是一种流行的技术。

本书的结构

第 1 章是对 UML 的一个简介：UML 是什么，对不同的人它所具有的不同含义，以及它源于何处。

第 2 章谈软件过程。虽然软件过程完全独立于 UML，我认为重要的是，了解过程以便看出像 UML 那样语言的来龙去脉。特别是，重要的是了解迭代开发的作用，对面向对象界的很多人来说，迭代开发是一种基本的过程途径。

我是围绕 UML 内的各种图型来组织本书的其余部分的。第 3 章和第 4 章讨论 UML 两个最有用的部分：类图（核心）与顺序图。即使本书很小，我相信，利用我在这几章中所讨论的技术，你可以从 UML 中得到最大的好处。UML 是一头日益增长的大型野兽，但是你却不需要它的全部。

第 5 章谈论类图的一些比较次要但仍有用的部分的细节。第 6 章到第 8 章表述进一步显露系统结构（structure）的三种有用的图： 对象图、包图以及部署图。

第 9 章至第 11 章示明另外三种有用的行为（behavioral）技术： 用案、状态图（虽然正式称作状态机图，它们一般称为状态图）以及活动图。第 12 章至第 17 章非常简短，

它们描述了一些不太重要的图，对这些图我只提供了一个快速的例子与解释。

本书前后插页综述图示法最常用的部分。我常听人说，它们是本书最重要的部分。可能你会发现，在你阅读本书某些其他部分时引用它们十分方便。

第3版的变动

如果你有本书的先前几版，也许你想要知道不同之处是什么，更重要的是，是否应该购买新版。

促使我撰写并出版第3版的主要动力是UML 2的出现。UML 2增加了很多新材料，包括一些新图型。即使是熟悉的图也有不少新的图示法,例如，顺序图中的交互架构。如果你想要知道发生了什么，而又不想费力去读完规范（我肯定不会建议你这样做!），本书应该给你一个好的概述。

我已趁此机会完全改写了本书的大部分内容，从正文到例子都使之进入最新状态。我吸收了很多那些过去几年来在讲授与使用UML中所学到的东西。因此，虽然这本超薄的UML书的精神未曾触动，但很多话语都是新的。

几年来我努力工作以使本书尽可能成为内容最新的书。由于UML经历了变动，我尽最大努力与之齐头并进。本书是以UML 2草案为基础的。该草案已于2003年6月被有关委员会接受。在那次投票和更正式的投票之间不太可能出现进一步的变动，因此，我感到，对于付印我的这一修订本来说，UML 现在已经足够稳定。任何进一步更新的信息我将在我的网站（http://martinfowler.com）上公布。

致谢

多年来，很多人都居于本书成功的一方。我首先要感谢的是 Carter Shanklin 和 Kendall Scott。Carter是Addison Wesley的编辑，是他建议我写这本书的。Kendall Scott帮助我整理先前两版，仔细检查了正文和图形。他们一起在不可想象的短期内完成了难以完成的第1版的出版工作，而且保持了人们对Addison-Wesley所期待的高质量。他们还在UML看来什么都未稳定的早期保持摆脱变动的困境。

Jim Odell是我事业早期很多方面的导师与指路人。他还深入参与到解决持有己见的方法学学者们在技术问题和个人问题上的分歧并达成共同标准。他对本书的贡献是深邃的，而且也是难以估量的。我敢说，他对UML也同样如此。

UML 是一个标准的产物，但我对标准体却有点过敏。因此，为了了解正在进行什么，我需要有一个密探网，他们能使我了解到各种委会员的全部秘密策划的最新动态。如果没有包括Comrad Bock, Steve Cook, Cris Kobryn, Jim Odell, Guus Ramackers以及Jim Rumbaugh 等这样一些密探，我可就完了。他们全都给了我有益的指点并回答我一些无聊的问题。

Grady Booch, Ivar Jacobson 以及 Jim Rumbaugh 以“三友”（Three Amigos）而闻名。虽然几年来我曾经对他们开过一些玩笑，他们对本书却给了我很多支持与鼓励。永远不要忘记，我的刺激通常都源于珍视的赏识。

审阅人是一本书质量的关键。Carter 告诉过我，审阅人永不嫌多。本书前几版的审阅人是 Simmi Kochhar Bhargava, Grady Booch, Eric Evans, Tom Hadfield, Ivar Jacobson, Ronald E. Jeffries, Joshua Kerievsky, Helen Klein, Jim Odell, Jim Rumbaugh 以及 Vivek Salgar。

第 3 版也有一组优秀审阅人：

Conrad BockCraig Larman

Andy CarmichaelSteve Mellor

Alistair CockburnJim Odell

Steve CookAlan O’Callaghan

Luke HohmannGuus Ramackers

Pavel HrubyJim Rumbaugh

Jon KernTim Seltzer

Cris Kobryn

所有这些审阅人都花时间阅读书稿，其中每人至少发现一个令人尴尬的可笑错误。谨向他们全体致以诚挚的谢意。剩下来的任何错误完全是我的责任，发现错误后，将在网站 martin fowler.com 的图书部分勘误表页上公布。

设计并撰写 UML 规范的核心组是 Don Baisley, Morgan Bj rkander, Conrad Bock, Steve Cook, Philippe Desfray, Nathan Dykman, Anders Ek, David Frankel, Eran Gery, ystein Haugen, Sridhar Iyengar, Cris Kobryn, Birger M ller Pedersen, James Odell, Gunuar vergaard, Karin Palmkvist, Guus Ramackers, Jim Rumbaugh, Bran Selic, Thomas Weigert 以及 Larry Williams。如果没有他们，我就没有什么值得大写特写的了。

Pavel Hruby 开发了一种绝妙的 Visio 模板。我在 UML 图中用到很多该模板；你可以在 http://phruby.com 处找到相关信息。

很多人在网上与我联系，亲自提出建议和问题，并指出错误。我未能和他们所有的人取得联系，但我的感谢也是诚挚的。

我钟爱的技术书店是位于马萨诸塞州伯林顿的 SoftPro，该店的店员让我在那里度过了很多小时，浏览书架以发现人们是如何实际使用 UML 的，而且当我在那里时还为我提供上等的咖啡。

对第 3 版，组稿编辑是 Mike Hendrickson。Kim Arney Mulcahy 负责项目进度、版式设计以及绘图。Addison-Wesley 的 John Fuller 是生产编辑，而 Evelyn Pyle 与 Rebecca Rider 负责本书的文字编辑与校对。我对他们均致谢意。

在我坚持不懈写书时，Cindy 一直和我在一起。那时她在花园中耕耘以取得收益。我的父母使我有良好的教育开端，从此，其他一切都涌现而出。

Martin Fowler
Melrose, Massachusetts
http://martinfowler.com

Chapter 1

Introduction

What Is the UML?

The Unified Modeling Language (UML) is a family of graphical notations, backed by single meta-model, that help in describing and designing software systems, particularly software systems built using the object-oriented (OO) style. That's a somewhat simplified definition. In fact, the UML is a few different things to different people. This comes both from its own history and from the different views that people have about what makes an effective software engineering process. As a result, my task in much of this chapter is to set the scene for this book by explaining the different ways in which people see and use the UML.

Graphical modeling languages have been around in the software industry for a long time. The fundamental driver behind them all is that programming languages are not at a high enough level of abstraction to facilitate discussions about design.

Despite the fact that graphical modeling languages have been around for a long time, there is an enormous amount of dispute in the software industry about their role. These disputes play directly into how people perceive the role of the UML itself.

The UML is a relatively open standard, controlled by the Object Management Group (OMG), an open consortium of companies. The OMG was formed to build standards that supported interoperability, specifically the interoperability of object-oriented systems. The OMG is perhaps best known for the CORBA (Common Object Request Broker Architecture) standards.

The UML was born out of the unification of the many object-oriented graphical modeling languages that thrived in the late 1980s and early 1990s. Since its appearance in 1997, it has relegated that particular tower of Babel to history. That's a service I, and many other developers, am deeply thankful for.

Ways of Using the UML

At the heart of the role of the UML in software development are the different ways in which people want to use it, differences that carry over from other graphical modeling languages. These differences lead to long and difficult arguments about how the UML should be used.

To untangle this, Steve Mellor and I independently came up with a characterization of the three modes in which people use the UML: sketch, blueprint, and programming language. By far the most common of the three, at least to my biased eye, is **UML as sketch**. In this usage, developers use the UML to help communicate some aspects of a system. As with blueprints, you can use sketches in a forward-engineering or reverse-engineering direction. **Forward engineering** draws a UML diagram before you write code, while **reverse engineering** builds a UML diagram from existing code in order to help understand it.

The essence of sketching is selectivity. With forward sketching, you rough out some issues in code you are about to write, usually discussing them with a group of people on your team. Your aim is to use the sketches to help communicate ideas and alternatives about what you're about to do. You don't talk about all the code you are going to work on, only important issues that you want to run past your colleagues first or sections of the design that you want to visualize before you begin programming. Sessions like this can be very short: a 10-minute session to discuss a few hours of programming or a day to discuss a 2-week iteration.

With reverse engineering, you use sketches to explain how some part of a system works. You don't show every class, simply those that are interesting and worth talking about before you dig into the code.

Because sketching is pretty informal and dynamic, you need to do it quickly and collaboratively, so a common medium is a whiteboard. Sketches are also useful in documents, in which case the focus is communication rather than completeness. The tools used for sketching are lightweight drawing tools, and often people aren't too particular about keeping to every strict rule of the UML. Most UML diagrams shown in books, such as my other books, are sketches. Their emphasis is on selective communication rather than complete specification.

In contrast, **UML as blueprint** is about completeness. In forward engineering, the idea is that blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up. That design should be sufficiently complete in that all design decisions are laid out, and the programmer should be able to follow it as a pretty straightforward activity that requires little thought. The designer may be the same person as the programmer, but usually the designer is a more senior developer who designs for a team of programmers. The inspiration for this approach is other forms of engineering in which professional engineers create engineering drawings that are handed over to construction companies to build.

Blueprinting may be used for all details, or a designer may draw blueprints to a particular area. A common approach is for a designer to develop blueprint-level models as far as interfaces of subsystems but then let developers work out the details of implementing those details.

In reverse engineering, blueprints aim to convey detailed information about the code either in paper documents or as an interactive graphical browser. The blueprints can show every detail about a class in a graphical form that's easier for developers to understand.

Blueprints require much more sophisticated tools than sketches do in order to handle the details required for the task. Specialized CASE (computer-aided software engineering) tools fall into this category, although the term CASE has become a dirty word, and vendors try to avoid it now. Forward-engineering tools support diagram drawing and back it up with a repository to hold the information. Reverse-engineering tools read source code and interpret from it into the repository and generate diagrams. Tools that can do both forward and reverse engineering like this are referred to as **round-trip** tools.

Some tools use the source code itself as the repository and use diagrams as a graphic viewport on the code. These tools tie much more closely into programming and often integrate directly with programming editors. I like to think of these as **tripless** tools.

The line between blueprints and sketches is somewhat blurry, but the distinction, I think, rests on the fact that sketches are deliberately incomplete, highlighting important information, while blueprints intend to be comprehensive, often with the aim of reducing programming to a simple and fairly mechanical activity. In a sound bite, I'd say that sketches are explorative, while blueprints are definitive.

As you do more and more in the UML and the programming gets increasingly mechanical, it becomes obvious that the programming should be automated. Indeed, many CASE tools do some form of code generation, which automates building a significant part of a system. Eventually, however, you reach the point at which all the system can be specified in the UML, and you reach **UML as programming language.** In this environment, developers draw UML diagrams that are compiled directly to executable code, and the UML becomes the source code. Obviously, this usage of UML demands particularly sophisticated tooling. (Also, the notions of forward and reverse engineering don't make any sense for this mode, as the UML and source code are the same thing.)