建模带来竞争优势

软件方法（上）

业务建模和需求（第2版）

潘加宇 著

清华大学出版社

软件方法（上）

业务建模和需求（第2版）

潘加宇 著

清华大学出版社
北 京

图书在版编目(CIP)数据

软件方法. 上，业务建模和需求 / 潘加宇 著. — 2版. —北京：清华大学出版社，2018
ISBN 978-7-302-49782-0

Ⅰ. ①软… Ⅱ. ①潘… Ⅲ. ①软件设计方法学 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2018)第 037254 号

责任编辑：陈 莉 高 岫
封面设计：周晓亮
版式设计：方加青
责任校对：曹 阳
责任印制：王静怡

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座 邮 编：100084
社 总 机：010-62770175 邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市国英印务有限公司
经 销：全国新华书店
开 本：170mm×240mm **印 张：**18.5 **字 数：**285 千字
版 次：2013 年 9 月第 1 版 2018 年 3 月第 2 版 **印 次：**2018 年 3 月第 1 次印刷
印 数：1 ～ 3500
定 价：58.00 元

产品编号：076417-01

一个人的软件方法

在学校待了很久，学的正是现在大热的模式识别、机器学习，用现在流行的术语就是人工智能、大数据。筋疲力尽地毕业之后，确定自己不是搞研究的料，也不想做“本行”，于是想改个行，把十多年的兴趣“兵棋推演”当事业做。当年的兵棋玩友做了一个微型公司，给军队做软硬件技术服务，我投奔他开发兵棋推演系统。

在学校期间，我用C++写工业机器人操作控制语言（一种领域特定语言）的语法解析器；用MATLAB和R写图像目标归类、语音分类算法；用Python写集成了指纹、图像、录音的身份特征数据采集工具；还部署算法到MPI集群，移植算法到GPU……自认为自己水平够高了，什么样的复杂算法我没有见过，我和它们谈笑风生……

有了这些底气，再加上自以为对“兵棋推演”的业务场景非常熟悉，觉得应该很容易就写出来。但事与愿违，干了一段时间，突然发现事情并不简单，根本出不来成型的系统。我泄了气，打算捏着鼻子回到正在大热的本行，但公司极力挽留，让我不胜惶恐。

这时，我通过潘老师的文章第一次了解到他的建模思想。文章里说

的“唱曲的名家，唱到极快之处，吐字依然干净利落”正是我想达到的目标，于是我购买了《软件方法》，此时是2016年4月。

正好当时公司正在给军校做一个小系统，由于双方对需求工作流的技能没有认识，对软件的责任边界也很模糊，导致已经反复折腾了两年时间。看完《软件方法》（上）之后，我开始有意识地运用书里的技能。

做需求启发时，我发现确实如书中所说：涉众往往会做而不会定义；把不同类型的涉众放在一起访谈时，只会剩下在场军衔最高那个人的意见……

此外，通过刻意关注涉众利益，出现了神奇的效果。和甲方的沟通效率和变更质量明显提升，改一个中一个，即使隔一段时间演示的时候涉众已经忘记了自己之前的想法，只要从当初记录下的涉众利益角度稍微一提，涉众就会立刻想起来，而且很容易认账，“对对对”成了口头语，对软件的功能越来越满意。

有了这个小系统从原地打转到圆满解决的牛刀小试，2016年底，一个真正的兵棋推演系统项目来了。我从寻找老大、揣摩愿景、业务建模、系统用例，到需求规约、分析模型、设计开发，学的基本都用上了。整个2017年上半年，就是在这样边学边干的紧张状态里度过的。想着这次要是再砸了就真没退路了。好在最终扛下来了，对甲方，对公司，对自己，都算是有了个交代。

由于项目涉密，具体项目细节略去不再谈。现在回顾一下，最大的挑战是，在“敌人炮火最猛烈的时候”，我始终记着《软件方法》里潘老师对核心域的描述，“只有一个领域（核心域）的知识是系统能在市场上生存的理由”。即使别人认识不到，我也必须强迫自己不断思考，找准核心域，然后像狗看骨头一样，死死地守住它的边界。

从核心域透镜的角度看问题，对军事领域很多听起来唬人的噱头有了自己的独立看法。谁应该说“信火一体”（信息与火力），谁应该说“察打分离”（侦察与打击），一个合成营要指挥下面15个不同兵种的连，责

任分配合理吗？

在离开校园两年后，我也算终于在北京这个已经感觉陌生的城市站住脚了，而此时离学习软件方法，才仅仅过去一年。

许庆晗

工学博士，兵棋开发者

2017年10月

我心更明白，你的爱将与我同在。

《掌声响起》；词：陈桂芬，曲：陈进兴，唱：凤飞飞；1974

致　谢

列出一份名单确实很为难。我只能说：只要我们之前打过交道，即使您的名字不在名单上，我也是一直记得您的。向以下各位表示深深的感谢。

2002年至今，选择了UMLChina需求和设计技能服务的所有机构和个人：。到2017年10月，系统中记载的人数是34 704人，其中来自京东的有1377人，在所有机构中位居第一，其他机构恕不一一列举。

选择本书第一版的高校老师（按姓氏）：毕文杰、曹顺良、陈志雨、侯宗浩、刘钢、赵毓芳、邹盛荣、车战斌、贾晓辉、李勇军、刘东良。

所有为本书纠错的读者（按纠错时间先后）：Cliff Peng、Tiger Gm、绿豆稀饭、Casper张、吴俊峰、Bright Zhang、商雪飞、李云、杨建媛、李勇、Leo Huang、张攀、张艳梅、gs0987、Timothy Yeh、深蓝二号、毛灵、陈小青、钟正权、吴刚、许珊、半导体、穆明明、王涛、赵卫、孙晓晔、黄明哲、辛恩平、杜宁军、徐波、徐天保、张云贵、Lyla、汤晓冬、

魏光裕、贾顶忠、叶青、涂文军、石碧川、李秀涛、黄保光、周进刚、吴佰钊、秦嘉斐、钱辰宇、刘京城、郁跃、张志坚、林炳炎、杨杰、李小平、齐世昌、刘琦、李洪洲、郭建国、杨金翠、冼浦楠、黄树成、熊德荣、许庆晗、成文华、熊飞、张立杰、杨明、贾晓辉、许鑫、厉景宇、沈志坚、龙志超、邹盛荣。

1999—2000年间，最早为UMLChina做贡献的（按时间先后）： 欧阳巨星、Adams Wang、陈英、kflee、bug、mirnshi、Windy J、Frank Gu、张恂、wenjl、杨健、宋怡、苏康胜、wsb、lhf、wbj、马成长、鸿雁南飞、张利锋、sealw、davidqql。

2000—2004年间，在UMLChina讨论组担任过组长的（按时间先后）： abug、mouri、vcc_cn、sealw。

2001—2005年间，为《非程序员》提供稿件最多的（按稿件数量多少）： 透明、杨德仁、刘庆、甄镭、Huang Yin、金哲凡、李巍、Windy J、刘巍、michael、王念滨。

1999—2004年间，带给我许多启发的新浪IT业界论坛坛友（排名不分先后）： 书生意气、it99、狂马、青梅、ie98、徐远明、徐远光、徐运光、胡不悦、悠悠散步、网络机器人、Microhelper、纸马、notthreenotfour、宽带应用专家、胡不乐。

合作过的出版社同仁（按姓氏）： 陈冀康、符隆美、傅志红、胡顺增、江立、李万红、李艳波、李阳、刘金喜、刘立卿、刘映欣、麻众志、隋曦、汤斌浩、王定、温莉芳、谢晓芳、熊妍妍、姚蕾、尤晓东、张春雨。

UMLChina翻译书籍译者（按合作时间先后）： 汪颖、方春旭、叶向群、熊节、李巍、李嘉兴、章柏幸、王海鹏。

合作过的CSDN和《程序员》杂志同仁（按姓氏）： 才子英、常政、丁莹、董世晓、杜倩、付江、高松、霍泰稳、贾菡、刘洪洁、刘龙静、孟岩、孟迎霞、欧阳璟、吴志民、熊节、闫辉、郑柯、邹震。

不拘一格把我带进中科院研究生院兼职教了很多年课的：潘辛苹博士。

我的太太Sicilia。我们好不容易才在一起，没想到婚后却经常吵架，幸好，近年少多了。

潘加宇

2017年10月

每当变幻时，便知时光去。

《每当变幻时》；词：卢国沾，曲：古贺政男，唱：薰妮；1985

前　言

2013年写的第一版前言，现在看来依然可以用，所以除了修改一些随年份变化的数字之外，我把第一版前言附在后面，本次版本的前言就尽量写得简短一些。

在主要思想不变的前提下，我结合最近几年的进展，几乎把整本书重新写了一遍，从文字到图形基本上都换了。每一章的内容更细致，道理讲得更严谨，例子和练习也更丰富。总之，希望能给读者带来一本更有用的书。本书出版之后，将继续投入未写完的《软件方法（下）：分析和设计》。

18年过去，弹指一挥间。我已经在这一个狭窄的领域泡了十八年了，也许累计的时间已经超过了一些前辈。希望还能再研究十八年，和大家分享更多有价值的东西。

潘加宇

2017年10月

光阴匆匆似流水，它一去不再回。

《浪子归》；词：黄小茂，曲：崔健，唱：崔健；1985

前言（2013版）

1999年还是一名程序员时，我创建了UMLChina，从那时开始关注软件工程各方面的进展。2001年12月，阿里巴巴的吴泳铭来email询问是否有UML方面的训练，我开始准备训练材料。2002年3月，我去杭州给阿里巴巴做了这个训练。虽然与后来我给阿里集团各公司做的许多次训练相比，这第一次讲课从内容到形式都算是糟透了，但是我现在还记得当时的心情——迈出自己事业第一步的心情。

截至2013年7月，我已经上门为超过190家软件组织提供需求和设计技能的训练和咨询服务（2017年注：2017年10月的数字为超过260家）。训练结束后，学员们常会问：“潘老师，上完课后我们应该看什么书？”我总是回答：“先不用看杂七杂八的书，还是要复习我们留下的资料，那些幻灯片、练习题、模型就已经是最好的书了，按照改进指南先用一点点在具体项目上，带着出现的具体困惑和我讨论。”虽然一再这样强调，有的学员还是情不自禁地拿着一本《***UML***》之类的书来问我问题，不管书上说得对不对。看来写在正式出版物上的效果就是不一样。

其实现在出书也不难，UMLChina一直在和出版社合作推介国外优秀

的软件工程书籍，目前UMLChina的标记已经出现在三十多本软件工程书籍上。不过我一直没有自己写一本书，主要原因还是觉得积累不够，思考的深度也不够，对软件开发的认识还在不断变化。如果没有自己成形的东西，不能站在别人的肩膀上看得更远，只是摘抄别人的观点，这样的书有什么意义呢？

另外一个原因是，UMLChina后来采取了“隐形、关门”的策略，秉持“内外有别”的原则。我关闭了已经有4万多人的Smiling电子小组（也是为了降低某些风险），网站不再有公开的社区，在网站上也找不到“客户名单”，所有更细致的服务以非公开的方式对会员提供。在这种情况下，出一本书也不是那么迫切。

现在距离第一次提供服务已经超过10年（2017年注：已经超过15年了），也有了一些积累，所以硬着头皮也要开始写书了。在这些年的服务过程中，和开发团队谈到改进时，我发现一个有趣的现象：很多开发团队（不是每个团队）或多或少都会有人（不是每个人）或明或暗地表达出这样的观点——自己团队的难处与众不同，奇特的困难降临在他们身上，偏偏别人得以幸免。

尽管UMLChina一直强调自己的服务是“聚焦最后一公里”，坚信每一个开发团队都会在细节上和其他团队有所不同，而且也应该有所不同，但很多时候，我还是感觉到，开发团队高估了自己的“个性”，低估了“共性”。本书就是归纳这样一些“共性”，作为我的一家之言，供大家参考。感谢曾经选择我的服务的伙伴们。他们一次次地给我机会来实践、发展和锤炼技艺，才有了这本书。

本书中所讲述的技能集合也是我本人身体力行的。例如，您可能已经注意到，为本书写推荐序的正是本书的“老大”，他不是什么大师专家名人，而是一名经历了入职、升职和创业，不断成长的软件开发人员。

一些书籍作者喜欢在书中每一章的开头放上和该章内容相关的一幅画或一句名人名言，我也效仿一下，不过没那么“高雅”——每章的开头放

上和该章内容相关的一句歌词。

书中的模型图，如果是我为了讲解知识而画的，用的建模工具是Enterprise Architect 9（2017年注：改为Enterprise Architect 13）；如果是截取真实模型的图片，可能会涉及各种工具。我不像Robert C. Martin那样，女儿已经长大到可以帮画插图，所以书中的手绘插图，我都自己用Wacom笔来画，可能丑了一些，请见谅。

潘加宇

2013年10月

联系方式	号码	二维码
QQ号和QQ邮箱	1493943028 1493943028@qq.com	
微信	umlchinapan	
QQ群	647242431	
公众号	UMLChina	
微博	UMLChina潘加宇	

欢迎您对本书的任何反馈，联系方法如上。

张生带上仆人阿梁，挑着圣贤书两大箱。

《张生记》；词：高晓松，曲：高晓松，唱：曹颖；2006

推荐阅读

在为软件组织提供服务时，我一直采取拿来主义的做法，不拘泥于流派或风格，着力于细节和应用。如果硬要说出本书的几个主要思想来源，我认为应该是Ivar Jacobson、Alistair Cockburn、Peter Coad和高焕堂。

下面是我推荐大家阅读的需求和设计书籍和资料。这些书籍和资料我都读过，否则就没有资格在此处推荐了。您可能会发现，一些有名的著作如Brooks的*The Mythical Man-Month*、GoF的*Design Patterns*等不在其中，不是因为我没有读过——事实上，需求和设计书籍只要有中文译本或者能有渠道找到英文电子版的，绝大多数我都阅读过。只是我认为，对于需求和设计技能的提升，阅读以下推荐的资料帮助更大。

另外要说的是，要用发展的眼光看问题，不能搞“原教旨主义”。某种思想或方法起源于某人，不意味着某人最初对该思想或方法的认识永远是最正确的，也不意味着某人在以后的岁月中针对该思想或方法发表的各种观点都是正确的。Ivar Jacobson的*Object-Oriented Software Engineering*出版于1992年，Peter Coad的*Java Modeling In Color With UML*出版于1999年，Alistair Cockburn的*Writing Effective Use Cases*出版于2001年。不否认这些书

中思想的光芒，但毕竟世界在进步，在实践的大浪淘沙之下，有些细节值得商议。小教派式的“教主崇拜”，由一些编辑捧出来的圈子文化以及廉价“大牛”“大仙”“大神”式的称呼，不值得提倡。鉴于此，本书不会称呼先行者们为“大师”“大牛”“大仙”“大神”，我想他们的贡献不会因此埋没。

书名	ISBN	出版年	作者	中译本
Software Reuse: Architecture, Process and Organization for Business Success	978-0201924763	1997	Ivar Jacobson M. Griss P. Jonsson	软件复用：结构、过程和组织
Use Cases: Requirements in Context: 2nd Edition	978-0321154989	2003	Daryl Kulak	用例：通过背景环境获取需求
Writing Effective Use Cases	978-0201702255	2000	Alistair Cockburn	编写有效用例
Exploring Requirements: Quality Before Design	978-0932633132	1989	Donald C. Gause Gerald M. Weinberg	探索需求——设计前的质量
Mastering the Requirements Process: Getting Requirements Right (3rd Edition)	978-0321815743	2012	Suzanne Robertson James Robertson	掌握需求过程(第3版)
Positioning: The Battle for Your Mind	978-0071373586	2000	Al Ries Jack Trout	定位
Serious Creativity: Using the Power of Lateral Thinking to Create New Ideas	978-0887306358	1993	Edward De Bono	严肃的创造力
How to Win Friends and Influence People	978-0671027032	1936	Dale Carnegie	人性的弱点
历史深处的忧虑	978-7108010186	1997	林达	/
为什么是市场	978-7508601045	2004	秋风	/
Case Studies in Object-Oriented Analysis and Design	978-0133051377	1996	Edward Yourdon Carl A. Argila	实用面向对象软件工程教程

（续表）

书名	ISBN	出版年	作者	中译本
Object Models: Strategies, Patterns, and Applications (2nd Edition)	978-0138401177	1996	Peter Coad David North Mark Mayfield	对象模型：策略、模式与应用（第2版）
Java Modeling In Color With UML: Enterprise Components and Process	978-0130115102	1999	Peter Coad Jeff de Luca Eric Lefebvre	彩色UML建模
Analysis Patterns: Reusable Object Models	978-0201895421	1997	Martin Fowler	分析模式：可复用的对象模型
Object-Oriented Software Construction (2nd Edition)	978-0136291558	1997	Bertrand Meyer	/
The Data Model Resource Book, Vol. 1: A Library of Universal Data Models for All Enterprises	978-0471380238	2001	Len Silverston	数据模型资源手册（卷1）
The Data Model Resource Book, Vol. 2: A Library of Data Models for Specific Industries	978-0471353485	2001	Len Silverston	数据模型资源手册（卷2）
The Data Model Resource Book, Vol. 3: Universal Patterns for Data Modeling (Volume 3)	978-0470178454	2008	Len Silverston Paul Agnew	数据模型资源手册（卷3）——数据模型通用模式
Model Driven Architecture with Executable UML	978-0521537711	2004	Chris Raistrick Paul Francis John Wright Colin Carter Ian Wilkie	MDA与可执行UML
Holub on Patterns: Learning Design Patterns by Looking at Code	978-1850158479	2004	Allen Holub	设计模式初学者指南
Data Model Patterns	978-0932633743	2011	David C. Hay	/
Domain-Driven Design: Tackling Complexity in the Heart of Software	978-0321125217	2003	Eric Evans	领域驱动设计

（续表）

书名	ISBN	出版年	作者	中译本
Pattern-Oriented Software Architecture Volume 1: A System of Patterns	978-0471958697	1996	Frank Buschmann Regine Meunier	面向模式的软件架构，卷1：模式系统
Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects	978-0471606956	2000	Douglas Schmidt Michael Stal	面向模式的软件架构，卷2：并发和联网对象模式
Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management	978-0470845257	2004	Michael Kircher Prashant Jain	面向模式的软件架构，卷3：资源管理模式
Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing	978-0470059029	2007	Frank Buschmann Kevin Henney	面向模式的软件架构，卷4：分布式计算的模式语言
Pattern Oriented Software Architecture Volume 5: On Patterns and Pattern Languages	978-0471486480	2007	Frank Buschmann Kevin Henney	面向模式的软件架构，卷5：模式与模式语言
Pattern Languages of Program Design	978-0201607345	1995	James O. Coplien Douglas Schmidt	程序设计的模式语言，卷1
Pattern Languages of Program Design 2	978-0201895278	1996	John Vlissides James O. Coplien	程序设计的模式语言，卷2
Pattern Languages of Program Design 3	978-0201310115	1997	Robert C. Martin Dirk Riehle	程序设计的模式语言，卷3
Pattern Languages of Program Design 4	978-0201433043	1999	Brian Foote Neil Harrison	程序设计的模式语言，卷4
Pattern Languages of Program Design 5	978-0321321947	2006	Dragos Manolescu Markus Voelter	程序设计模式语言，卷5
OMG Unified Modeling Language Version 2.5		2015	OMG	/

（续表）

书名	ISBN	出版年	作者	中译本
UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)	978-0321193681	2003	Martin Fowler	UML精粹（第3版）
Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems	978-0750687065	2008	Miro Samek	/
Objects, Components, and Frameworks with UML: The Catalysis	978-0201310122	1998	Desmond Francis D'Souza Alan Cameron Wills	UML对象、组件和框架——Catalysis方法
Working With Objects:The Ooram Software Engineering Method	978-0134529301	1998	Wold Reenskaug Trygve Reenskaug O. A. Lehne	/

目　录

牵着你走进傍晚的风里，看见万家灯火下面平凡的秘密。

《情歌唱晚》；词：黄群，曲：黄群，唱：曹崴；1994

第1章　建模和UML

1.1 粗放经营的时代已经远去

改革开放初期，中国出现了许多农民企业家，他们不用讲管理，也不用讲方法，只要胆子大一点，就能获得成功，因为当时的市场几乎空白，竞争非常少。农民企业家的思路很简单：人人都要吃饭，所以开饭馆能够赚钱。现在这样的思路已经行不通了。市场竞争已经足够激烈，十家新开张的饭馆恐怕只有一家能撑下来，所以农民企业家已经很少见（连农民都越来越少了）。软件业也一样，最开始的时候，会编程就了不得，思路也很简单：每个公司都要做财务，所以开发财务软件能赚钱。现在呢？我们想到一个“点子”，可能有上千人同时想到了；我们要做一个系统，可能发现市场上已经有许多类似的系统。你卖高价，他就卖低价，你卖低价，他就干脆免费。机会驱动、粗放经营的时代已经远去，为了在激烈的竞争中获得优势，软件开发组织需要从细节上提升技能。

本书聚焦于两方面的技能：需求和设计。关于需求和设计，开发人员可能每天都在做，但是否理解背后的道理呢？我们来做一些题目：

本书不提供练习题答案，请扫码或访问http://www.umlchina.com/book/quiz1_1.htm完成在线测试，做到全对，自然就知道答案了。

1. 软件开发中需求工作的目的是_____。

A）让系统更加好卖　　B）更好地指导设计

C）对系统做概要的描述　　D）满足软件工程需求规范

2. 软件开发中设计工作的目的是_____。

A）对系统做详细的描述　　B）更好地指导编码

C）降低开发维护成本　　D）满足软件工程设计规范

1.2 利润＝需求－设计

利润＝收入－成本。不管出售什么，要获得利润，需要两个条件：

（1）要卖出好价钱；

（2）成本要低。

妙就妙在，价格和成本之间没有固定的计算公式，这正是创新的动力之源。放到软件业上，我也炮制了一个公式：

利润＝需求－设计

在软件开发中，需求工作致力于解决“提升销售”的问题，设计工作致力于解决“降低成本”的问题，二者不能相互取代。能低成本生产某个系统，不一定能保证它好卖。系统好卖，如果生产成本太高，最终还是赚不了多少钱。

如果需求和设计不分，利润就会缩水。从需求直接映射设计，会得到大量重复代码；如果从设计出发来定义需求，会得到一堆假的“需求”。

拿自古以来就有的一个系统“人体”来举例。人体的功能（能做什么）是走路、跑步、跳跃、举重、投掷、游泳……但是设计人体的结构时，不能从需求直接映射到设计，得到“走路子系统”“跑步子系统”“跳跃子系统”……人体的“子系统”是“呼吸子系统”“消化子系统”“循环子系统”“神经子系统”……“子系统”不是从需求直接映射出来的，需要设计人员的想象力——本例子的设计人员就是造物主了。同样，也不能从设计推导出需求：因为人有心肝脾肺肾，所以人的用例是“心管理”“肝管理”（见图1-1）。

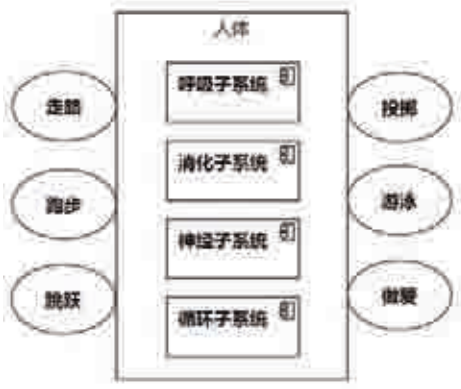


图1-1　人体的需求和设计

水店老板要雇一个民工送水（即租用一个人脑系统），他只要求这个民工能跑能扛就行，管他体内构造是心肝脾肺肾还是一块电路板；民工找工作也要从市场的需要来找，而不是从自己的内部器官出发来找——“老板，我有心脏管理功能，你请我吧！”

很多时候我们说“本系统分为八大子系统……”，其实说的是“本系统的功能需求分为八大需求包……”需求包是基于涉众视角对系统功能分包而得到的，子系统是基于内部视角根据系统部件的耦合和内聚情况切割而得到的。

需求和设计的区别简要列举如图1-2所示，在后面的章节中再慢慢阐述这些区别。

需求	设计
卖的视角	做的视角
具体	抽象
产品当项目做	项目当产品做
设计源于需求，高于需求	

图1-2　需求和设计的区别

高焕堂在他的书《Use Case入门与实例》中说过：用例是收益面，对象是成本面。本书基于他的思想做了扩展。

1.3 建模工作流

要达到“低成本制造好卖的系统”的目标，并非喊喊口号就可以，需要静下心来学习和实践以下各个建模工作流中的技能。

1. **业务建模**——描述组织内部各系统（人脑系统、电脑系统……）如何协作，使得组织可以为其他组织提供有价值的服务。新系统只不过是组织为了对外提供更好的服务，对自己的内部重新设计而购买的一个零件。组织引进一个软件系统，和招聘一名新员工没有本质区别。如果通过业务建模推导新系统的需求，而不是拍脑袋得出，假的“需求变更”会大大减少。

2. **需求**——描述为了解决组织的问题，系统必须具有的表现——功能和性能。这项技能的意义在于强迫我们从“卖”的角度思考哪些是涉众（Stakeholder）在意的、不能改变的契约，哪些不是，严防“做”污染“卖”。需求工作流的结果——需求规约是“卖”和“做”的衔接点。

3. **分析**——提炼为了满足功能需求，系统需要封装的核心域机制。可

运行的系统需要封装各个领域的知识，其中只有一个领域（核心域）的知识是系统能在市场上生存的理由。对核心域作研究，可以帮助我们达到基于核心域的复用。

4. 设计——为了满足质量需求和设计约束，核心域机制如何映射到选定平台上实现。

软件开发人员如果缺乏软件工程方面的训练，对以上工作流没有概念，就会把这些工作产生的工件通通称为“设计”或者“文档”。例如问开发人员在做什么，回答“我在做设计”“我在写文档”，其实他的大脑可能正在思考组织的流程（业务建模），或者在思考系统有什么功能性能（需求），或者在思考系统要包含的领域概念之间的关系（分析），但他通通回答成“在做设计”“在写文档”。后来又有牛人说了：代码就是设计。本来“设计”在他脑子里就是“代码以外的东西”，这么一推导，不就变成了：代码就是一切？

很多大谈“编码的艺术”的书籍和文章，其实探讨的根本不是编码的技能，而是分析技能甚至是业务建模技能，只是作者的大脑里没有建立起这些概念而已。编码确实有编码的技能，就像医院里护士给患者输液也需要经过训练，但如果患者输液后死亡，更应该反思的是护士的输液手法不过关，还是医生的检查诊断技能不过关？

把工件简单分割为代码和文档（或设计），背后还隐含着这样的误解：认为模型（文档）只不过是源代码的另一种比较概要或比较形象的表现形式。这种误解不只“普通”的开发人员会有，一些著名的UML书籍作者也有。Martin Fowler[①]所著的UML畅销书《UML精粹》，认为UML有三种用法：草稿、蓝图和编程语言，也是仅从编码的角度来说的。从Fowler写作的其他书籍《重构》《企业应用架构模式》《分析模式》等可以知道，他的研究工作集中在分析设计工作流，特别是设计工作流，在业务建模和需求方面研究不多。

① 鉴于Fowler在某些社群的心目中如大神一般存在，此处专门提到了他。

不同工作流产出的工件之间的区别不在于形式，而在于内容，也就是思考的边界，如图1-3所示。如果清楚了解这一点，即使用C#，照样可以表达需求，用Word也可以“编码”。

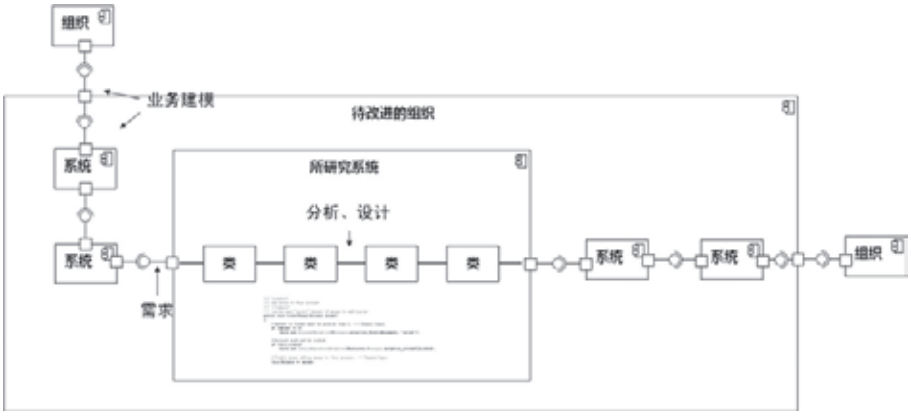


图1-3　建模工作流思考边界

有的开发人员思维刚好是颠倒的，先拍脑袋实现，然后再从实现反推前面的内容，例如下面的对话。

顾问：这个不应该是系统的用例。

开发人员：是的！我都写好了，运行一下给你看，这个系统确实提供了这个用例。

顾问：这两个类关系不应该是泛化，而是关联。

开发人员：是泛化，不信我打开代码给你看，或者逆向工程转出类图给你看。

是否系统用例应该以“好卖”来判断，是否泛化关系应该以“符合领域内涵”来判断，而不是先写好代码，再用代码来证明。

一些不了解以上概念的软件开发人员干脆以“敏捷”“迭代”为名，放弃了这些技能的修炼。就像一名从护士成长起来的医生，只掌握了打针的技能，却缺少检查、诊断、拟治疗方案等技能，索性说：“唉，反正再高明的大夫，也不能一个疗程把患者治好，干脆我也别花那么多心思了，

先随便给患者打一针看看吧，不好再来！”“迭代”只是一个底线，确实，再高明的大夫也没有把握一个疗程就治好患者，所以要按疗程试试看，但是在每一个疗程中，依然要尽力检查、诊断、拟治疗方案。检查、诊断等技能越精湛，所需要的疗程就越少。

唱曲的名家，唱到极快之处，吐字依然干净利落；快节奏的现代足球，职业球员的一招一式依然清清楚楚；即时战略游戏高手要在极短时间内完成多次操作，动作依然井然有序。

有的开发团队用“项目时间太紧”作为放弃建模的理由，这个理由是不成立的。以高考做类比，如果一年之后要高考，学霸会认真做一年的计划，再做一周的计划，再做每天的计划，找出分值最大而自己又最薄弱的地方先复习，并且随进度调整计划，不断提高，最终考了700分。学渣则浑浑噩噩，零敲碎打，最终考400分。假设教育部门突然下令，一周之后就要高考！学渣可能心里暗喜，以为自己翻身的机会来了。学霸依然会认真做一周的计划，再做每天的计划，找出分值最大而自己又最薄弱的地方先复习，并且随进度调整计划，不断提高，最终考得550分。学渣虚度了一周时间，考了350分。有一个著名的学渣，每隔四年失败一次。每次失败后总结教训“备战时间仓促”，其实再给学渣四年，它也是混四年，最终还是那个样子——你猜这个学渣是谁？

在激烈竞争的年代需要快速应变，掌握技能才能真敏捷。世上无易事，偷懒要不得。不能把“敏捷”“迭代”作为偷懒的庇护所。

有些互联网开发人员鼓吹“试错大法”，主张拍脑袋拿出去让市场试错，这同样是很荒谬的。客户确实不管你是怎么开发的，他只需要挑好用的就行，但对于开发系统的组织来说，有没有被挑中则是致命的。就像奴隶主看角斗士厮杀，奴隶主无所谓谁胜谁负，反正过程精彩，最终嘉奖冠军就行，而每一个角斗士却要如履薄冰，想方设法让自己坚持到最后。可悲的是，互联网公司是角斗士，却偏偏有“我是奴隶主”的错觉。

刚入行的开发人员体会不到建模的重要性，是正常的。就像下象棋，

初学者面对单车对马双士、单马对单士等已经有共识的局面还需要思考良久，最终还拿不下来，甚至输掉。这时中局和布局的书在他看来多半是枯燥无味的，还不如把一本实用残局汇编看熟了，学到一些雕虫小技，也能在菜市场赢几盘棋。不过，要迈向职业棋手的境界，参与更残酷的竞争，就体现出中局和布局的重要了。

从我的观察所得，以上四项技能，大多数软件组织做得较好的是设计（也就是实现），前面三项都相当差，特别是业务建模和分析，没有得到足够的重视。很多软件组织拍脑袋编造需求，然后直扑代码，却不知“功夫在诗外”。

本书中的“需求”和“设计”两个术语有两种用途。一种用于表达建模得到的结果，例如“需求和设计不是一一对应的”；另一种用于表达建模的工作流，即需求工作流和设计工作流，例如“我正在做需求”。希望下面的话能帮助理解：为了得到需求，需要做的建模工作流有业务建模和需求，为了得到设计，需要做的建模工作流有分析和设计。

到目前为止，我没有谈到UML。只要您思考过上面四个工作流的问题，就是在建模。可以用口头表达，也可以用文本、UML、其他表示法或自造符号来表达。在每一个项目中，开发团队肯定会思考和表达上面这些问题，只不过可能是无意识地、不严肃地做。现在，我们要学习有意识地做，而且把它做出利润来。使用UML来思考和表达是目前一个不坏的选择。

本书不提供练习题答案，请扫码或访问http://www.umlchina.com/book/quiz1_2.htm完成在线测试，做到全对，自然就知道答案了。

1. 开发人员说“根据客户的需求，我们的系统分为销售子系统、库存子系统、财务子系统……”，这句话反映了开发人员可能有什么样的认识错误?

A）开发人员没有认识到面向对象设计的重要性

B）开发人员直接从设计映射需求

C）开发人员直接从需求映射设计

D）开发人员没有用UML模型来描述子系统

2. 打开开发人员写的需求规约，发现用例的名字都是“学生管理”“题库管理”“课程管理”……，这背后可能隐藏的最大问题是什么?

A）用例的名字不是动宾结构，应改为“管理学生”……

B）用例粒度太粗，每一个应该拆解成四个用例，“新增学生”“修改学生”……

C）开发人员直接从需求映射设计

D）开发人员直接从设计映射需求

3. 以下这些经常在开发团队里使用的词汇，都是不严谨的。其中_____混淆了需求和设计的区别。

A）功能模块

B）详细设计

C）用户需求

D）业务架构

4. 以下描述最可能对应于软件开发中的哪个工作流?

每个项目由若干活动组成，每项活动又由许多任务组成。一项任务消耗若干资源，并产生若干工件。工件有代码、模型、文档等。

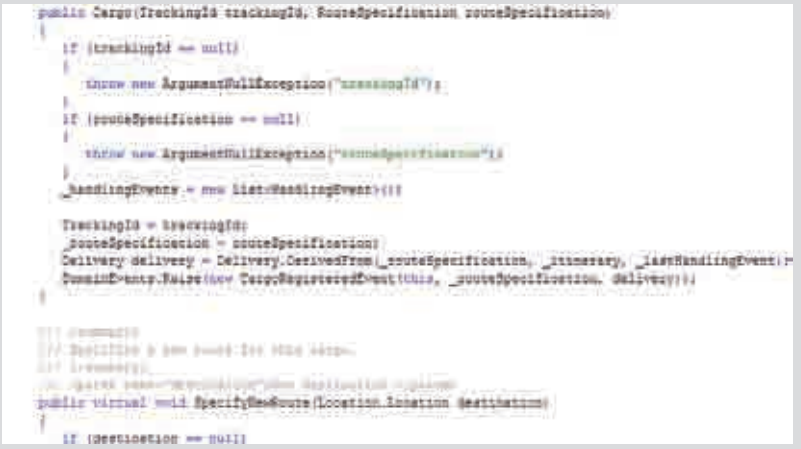
A）业务建模

B）需求

C）分析

D）设计

5. 以下描述最可能对应于软件开发中的哪个工作流？



A）业务建模　　B）分析　　C）需求　　D）设计

6. 以下描述最可能对应于软件开发中的哪个工作流？

系统向会员反馈已购买商品的信息。

A）业务建模　　B）分析　　C）需求　　D）设计

7. 以下描述最可能对应于软件开发中的哪个工作流？

某集团向优马神州经理提出举办讲座的请求后，经理根据请求决定请哪一位专家，并拟定讲座计划，交给组织工作人员执行。组织工作人员根据经理提供的专家资料通过E-mail、电话等各种方式联系专家，和专家商议讲座的时间和主题。

A）业务建模　　B）分析　　C）需求　　D）设计

8. 如果问开发人员“你在做什么”，他说“我在写文档”，那么他有可能（本题可多选）_____。

A）不了解软件开发各工作流的区别

B）把自己的工作简单分为“代码”和“文档”

C）认为文档就是代码的叙述性文件

D）知道“文档”和“代码”的真正区别是什么

9. 以下说法和其他三个最不类似的是_____。

A）如果允许一次走两步，新手也能击败象棋大师

B）百米短跑比赛才10秒钟，不可能为每一秒做周密计划，凭感觉跑就是

C）即使是最好的足球队，也不能保证每次进攻都能进球，所以练习传球配合是没用的，不如直接大脚开到对方门前

D）虽然大家都考不及格，但考58分和考42分是不一样的

1.4 UML简史

随着市场所要求软件的复杂度不断增大，软件开发的方法学也在不断进化。从没有方法到简单的功能分解法，再到数据流/实体关系法。进入20世纪90年代，面向对象分析设计（OOAD）方法学开始受到青睐，许多方法学家纷纷提出了自己的OOAD方法学。流行度比较高的方法学主要有Booch、Shlaer/Mellor、Wirfs-Brock责任驱动设计、Coad/Yourdon、Rumbaugh OMT和Jacobson OOSE。

这种百花齐放的局面带来了一个问题：各方法学有自己的一套概念、定义和标记符号。例如现在UML中的操作（Operation），在不同方法学中的叫法有责任（Responsibility）、服务（Service）、方法（Method）、成员函数（Member Function）……同一个类图，不同方法学也有各自的符号表达，如图1-4所示。这些细微的差异造成了混乱，使开发人员无从选择，也妨碍了面向对象分析设计方法学的推广。

1994年，Rational公司的James Rumbaugh和Grady Booch开始合并OMT和Booch方法。随后，Ivar Jacobson带着他的OOSE方法学加入了Rational公司，一同参与合并工作。这项工作造成了很大的冲击，因为在此之前，各种方法学的拥护者觉得没有必要放弃自己已经采用的表示法来接受统一的表示法。

Rational公司的这三位方法学家被大家称为“三友”（three amigo）。

1996年，三友开始与James Odell、Peter Coad、David Harel等来自其他公司的方法学家合作，吸纳他们的成果精华。1997年9月，所有建议被合并成一套建议书提交给OMG。1997年11月，OMG全体成员一致通过UML，并接纳为标准。

从2005年起，UML被ISO接纳为标准。相当于UML 1.4.2的ISO标准是ISO/IEC 19501，相当于UML 2.1.2的ISO标准是ISO/IEC 19505。2012年，ISO继续接纳UML 2.4.1为ISO/IEC 19505-1:2012和ISO/IEC 19505-2:2012，接纳OCL 2.3.1为ISO/IEC 19507:2012。

2011年，中华人民共和国也发布了统一建模语言国家标准GB/T28174。

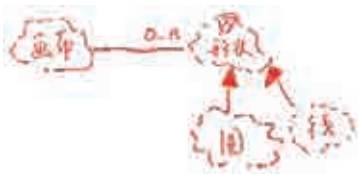
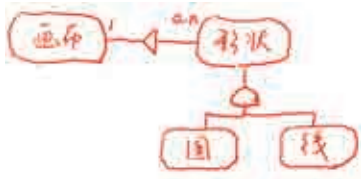
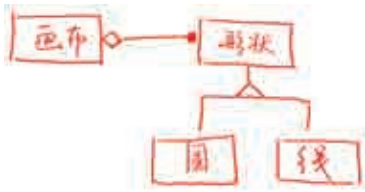
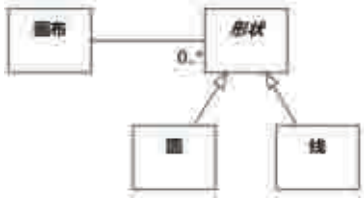
Booch	
Coad/Yourdon	
Rumbaugh OMT	
UML	

图1-4　不同方法学图形比较（同样一个三角形符号，在Coad/Yourdon方法学中用于表示关联，而在OMT方法学中用于表示泛化）

UML的最新版本是OMG于2015年6月通过的UML 2.5，相关网址如下。

OMG UML 2.5规范：http://www.omg.org/spec/UML/2.5/PDF

ISO UML规范：

http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=52854

中华人民共和国国家标准：http://www.chinagb.org/ChineseStandardShow-197675.html

时间过去二十年了，UML不断发展，在表示法上已经获得了胜利。随便打开一本现在出版的软件开发书，里面如果提到建模，使用的符号基本都是UML，即便在纸上随便画个草图，样子也是UML的样子。各种主流的开发平台也相继添加了UML建模的功能。OMG还和各种行业标准组织如DMTF、HL7等结盟，用UML表达行业标准。

另外，以UML为契机，掀起了一股普及软件工程的热潮，在UML出现后的几年，不但有关建模的新书数量暴增，包括CMM/CMMI、敏捷过程等软件过程改进书籍数量也出现了大幅度增长。制定UML标准的角色（OMG）、根据标准制作建模工具的角色（UML工具厂商）、使用UML工具开发软件的角色（开发人员）这三种角色的剥离，也导致建模工具的数量和种类出现了爆炸性的增长。而之前的数据流等方法从来没有像面向对象分析设计方法一样，出现UML这样的统一表示法，从而带动大量书籍和工具的产生。

最开始一批UML书籍，基本上由方法学家所写的。最近几年，以“UML”为题的新书大多为高校教材或普及性教材。这并不是说UML已经不重要，而是没有必要再去强调，焦点不再是“要不要UML”，而是要不要建模、如何建模。

根据UMLChina的统计，UML相关工具最多时达168种，经过市场的洗礼，现在还在更新的还有近百种。有钱买贵的，没钱就买便宜的或者用免费或开源的，可参见UMLChina整理的UML工具大全：http://www.

umlchina.com/Tools/Newindex1.htm。

1.5 UML应用于建模工作流

UML 2.5包含的图形如图1-5所示，一共14种（泛化树上的叶结点）。

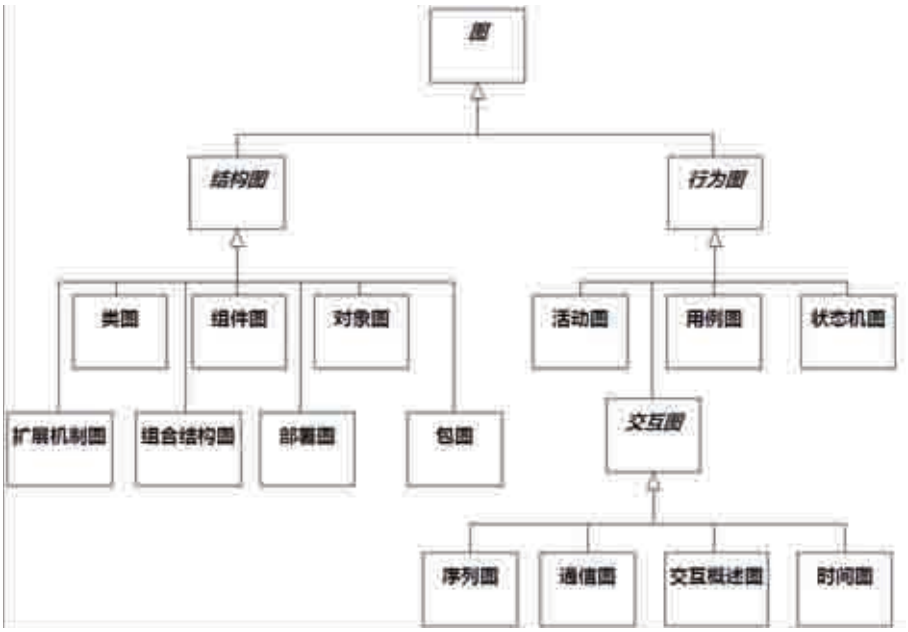


图1-5　UML图形（根据UML2.5规范绘制）

可能您看了会说，哇，这么多图，学起来用起来多复杂啊。其实，UML像一个工具箱，里面有各种工具。建模人员只需要根据当前所开发系统的特点，从这个工具箱中选择合适的工具就可以，并不需要“完整地”使用UML。这和编程语言类似。很多人说“我用Java”，其实只是用Java的一小部分，而且很长时间内也只会用这一小部分。

经常有学员问：潘老师，能不能给一个案例，完完整整地实施整套

UML？这是一种误解，这样的案例不应该有。这相当于问：有没有一本经典的小说，把字典里所有的单词都用上？有一些建模工具自带的案例模型会造成误解，一个模型里把所有的UML图都给用上了，但这是工具厂商出于展示其工具建模能力的目的而提供的，不可当真。

各建模工作流可以选用的建模图形以及推荐用法，如图1-6所示。

工作流	思考焦点	用例	类	组件	对象	部署	组合结构	包	扩展机制	序列	通信	状态机	活动	时间	交互概述	文本
业务建模	组织内系统之间	●	●							●			√		√	
需求	系统边界	●								√		√	√			●
分析	系统内核心域		●		√		√			●	√	●	√	√		
设计	系统内各域之间		√	√	√	√	√			√	√	√	√	√		●

图1-6　可选和推荐的建模元素用法（●表示优先使用，√表示可以使用）

观察图1-6中标有●的地方可以知道，掌握用例图、类图、序列图这三种图基本上够用了，可以先重点学习这三种图，其他图形暂时不管。针对特定类型的项目，如果有必要，可以按需添加图形。例如，开发复杂组织的运营系统，如果在业务建模时不喜欢用序列图，可以用活动图取代；对于系统中的核心类，可以着重画出状态机图来建模类内部的逻辑；针对质量要求很高的系统，每一个类可能都需要画状态机图，甚至还要画时间图。

另外，设计工作流目前推荐的做法是不需要画UML图，而是用文本来表达实现模型，即所谓“代码就是设计”——编码就是一种建模工作。计算机运行的是二进制指令，源代码实际上也是“模型”。之所以被称为“源代码”，是因为它是人脑需要编辑的最低形式模型。这个最低形式模型随着时代的发展不断变化。

如图1-7所示，最初的源代码是机器语言。程序员在纸带或卡片上打孔来表达0和1。后来发现这样太累了，于是发明了一些助记符，这就是汇编语言。今天会有开发人员故作谦虚，“这些我不太懂唉，我是做底层的，用C编码”，可是C语言却被归类为“高级语言”，因为类似C这样的语言

出现的时候，大多数程序员编辑的是汇编语言，C相对于汇编来说，当然很高级。今天的一名企业应用程序员，需要编辑的可能有Java代码、配置脚本、SQL语句等，这些就是现在的“源代码”的形式。

图1-7 “源代码”的发展历程

如果人脑只需要编辑UML模型就可以实现系统，那么“模型就是源代码”。例如用带有设计级调试和强大代码生成能力的工具IBM Rational Rhapsody开发实时嵌入系统，人脑只需要编辑和调试UML模型（类图和状态机图）。

1.6 基本共识上的沟通

不少开发人员并不喜欢用UML，更喜欢在白板上画个自造的草图，似流程图非流程图，似类图非类图，然后说“来，我给大家讲讲！”这样的做法有巨大的“优点”：怎么画都是对的，关于这个草图的解释权归“我”所有。同事不好批评“我”，项目要依赖于“我”头脑中的隐式知识——要是“我”不“给大家讲讲”，大家就玩不转了。这样，“我”在团队里的地位就提高了。上面这种现象，在有一定资历、但又不对项目的

成败承担首要责任的“高手”身上表现得更明显。


开发人员让我看他的模型时，如果开口说“我先来给你讲讲”，我都会拦住，“如果还需要你先讲讲，说明你所想的没有体现在模型中”。

这种做法的本质是想通过形式上的丑陋来遮掩内容上的丑陋。动乱年代，数学家在牛棚中用马粪纸做数学推导，不代表就可以因为演算工具简陋而允许自己胡乱使用符号和概念；过去的作家没有电脑，不意味着可以随意写错别字和犯语法错误。开发人员故意选择简陋的形式为简陋的内容开脱，就如同作家故意选择不好的纸来掩盖自己文字功力不足的事实，并不是好现象。UML没有强调一定要用多么昂贵的工具来建模，即使开发人员在海边用手指在沙滩上建模，模型所体现的概念依然要清晰。

如图1-8所示，数学里的积分符号、五线谱的小豆芽，幼儿园小朋友也会画，但背后的道理需要经过艰苦的训练才能理解。就像数学符号背后隐含着数学的基本共识，五线谱背后隐含着基本乐理一样，UML背后隐含着软件建模的一些基本共识，这些共识需要一定的训练才能掌握。

掌握统一的建模语言之后，开发团队在基本共识上沟通，会大大提高沟通的效率和深度，有意无意遮掩的脓包也会强制露出。开发人员如果习惯于画“草图”，用“模块”“特性”等词汇含糊不清地表达思想，在严谨建模思维的追问之下，往往会千疮百孔，暴露许多之前没有想到的问题。这是一些“高手”潜意识里不愿意直面UML的深层原因——如果有“高手”不同意，欢迎把所画的草图发过来，我告诉您背后隐藏的“脓包”。

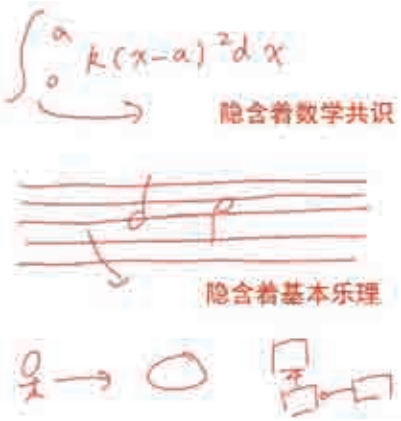


图1-8　符号背后隐含基本共识

面对一个棋局，下一步怎么走？在业余棋手看来到处都是正确答案，在职业棋手眼里，值得讨论的选项只有两三种，因为职业棋手针对一些基本的技能达成了共识，大大减少了思考中的浪费。

有些“敏捷”软件组织，抛弃了所有“文档”（前文已说过，如果使用“文档”这个词，说明概念不清楚），更喜欢口头交流，动不动就开会，你一嘴我一嘴，场面看起来热热闹闹，其实沟通的效果不好，更谈不上思考的深度和知识的沉淀。对比一下街坊老大爷下象棋的热闹和职业棋手比赛时的沉静就知道了。“敏捷”的理由也不成立，街坊老大爷判断局势的速度快还是职业棋手判断局势的速度快？野路子棋手不看棋书不打谱，摆个路边摊也许够用，如果参加职业比赛，会被打得落花流水。

有的开发人员的“十年工作经验”实际上是“一年工作经验用了十年”，一直在热热闹闹的民工层次徘徊，没有积累和成长。

不过要注意一点：使用UML沟通仅限于软件组织内部，UML模型不是用来和涉众沟通的！这个道理以及和涉众沟通的技能将在第7章详细叙述。

1.7 建模和敏捷（Agile）

敏捷运动在20世纪90年代中期兴起，当时敏捷过程被称为轻量（lightweight）过程。2001年，Kent Beck、Martin Fowler等人聚集在犹他州的Snowbird，决定把“敏捷”作为新的过程家族的名称，并提出以下宣言：

个体和交互　　胜过　过程和工具
可以工作的软件　胜过　面面俱到的文档
客户合作　　胜过　合同谈判
响应变化　　胜过　遵循计划

敏捷运动可以看作是“政治正确”的左翼思想在软件开发领域的一次演练。这次演练和在人类社会其他领域已发生的左翼演练一样，迅速取得了成功，带来了一批“有信仰”的软件从业者。此处不再细谈，只谈谈口号和方法的区别。

有口号有方法，有口号无方法，无口号无方法，这三种情况哪一种最坏？可能有的人认为无口号无方法最坏，其实不然，无口号无方法地呆在原地，可能会慢慢衰落，但不是最坏的。历史上各种最坏的大悲剧往往和“有口号无方法”有关。

最坏的事是“有口号无方法”的“好人”做的。偷盗抢劫的坏人知道自己做的是坏事，会暗自收敛，事后会内疚，甚至做一些善事来弥补以求心安；而“好人”认为自己是做好事，所以会做得很极端，如果有口号无方法，大悲剧就发生了。例如，有人谈论社会上存在的（□□此处作者删去三十二字□□）问题，列举的大都是事实，结果给出的解决方案却是（□□此处作者删去二十八字□□）。有一句名言“通往地狱的道路通常是由善意铺就的”，就是这个意思。

软件开发领域也有不少有口号无方法的场景。

【口号】我们只做最重要的需求，尽快把系统推向市场。

【问题来了】怎么知道哪个需求最重要？拍脑袋？

【口号】设计要分离变和不变，这样可以减少变更的成本。

【问题来了】怎么知道哪些变哪些不变？抓阄？

建模，为口号提供了方法；愿景、业务建模方法，帮助迅速定位最重要的需求；领域分析方法，帮助厘清各种概念的变和不变。

开发团队要警惕有口号无方法的成员。这些害群之马擅长喊口号打鸡血，上班时间端着茶杯大谈老子、庄子、孙子、禅、道……吐槽软件项目中的各种痛苦。这些痛苦确实是事实，结果给出的解决方案却是荒谬的。

有两幢大楼，地震中一幢倒塌了，另一幢没倒塌。倒塌的直接因素可能是大楼的结构、所用的材料、所在位置的地质环境等，但这些都涉及比较艰深的工程力学、材料学和地质学知识。有人懒得思考，干脆就直接嚷“有人吃回扣了”，就算经过调查没人吃回扣，他也会从工作服的颜色，工人是否结队洗澡，施工队开会时是否站立等方面来找原因，因为这相对容易。

本书内容针对的是影响软件质量的直接原因——缺少各种建模技能。许多团队实施过程改进容易流于形式，根源往往就在于建模技能的不足。如果把改进的焦点先放在建模技能上，开发人员技能提升了，适用什么样的过程自然就浮出水面，没有必要去生搬硬套某过程。或者说，技能增强了，更能适应不同的过程。UML建模没有绑定到特定过程。当前主流的软件过程都是强调增量和迭代开发，应该把前面所讲的业务建模、需求、分析、设计看作是一个迭代周期里的工作流，做一点业务建模，做一点需求，做一点分析，做一点设计……不可误解成“做完了所有的业务建模才能做需求”……

很多时候方法和过程经常被混淆，有人会把“敏捷”说成“敏捷方法”，其实“敏捷”是一个过程家族。之所以造成这个误解，也许和Martin Fowler把他介绍敏捷过程家族的文章起名为“新方法学（*The New Methodology*）”有关。另一个常见的误解来自Robert C. Martin的书《敏捷软件开发——原则、方法与实践》，书中主要讲的是面向对象设计的一些

方法（原理、原则和模式），这些方法并非Robert C. Martin首先提出的，而且和敏捷过程没有必然关系，但是，经常会有开发人员误解面向对象设计的这些思想是敏捷人士提出来的。更有一些敏捷人士在没有学习和掌握软件工程知识的情况下，先高喊“砸烂一切”，吸引热血青年，然后发现砸烂一切是不行的，又偷偷拾起过去被自己砸烂的东西，加上自己的“敏捷”包装，有意无意地给不了解历史的新入行开发人员造成“这是敏捷发明的，那是敏捷发明的，所以把敏捷挂在嘴边的人最懂”的印象。

我刚开始为软件组织提供服务时，有一次和一个软件组织的经理交流，经理说“我们用的是面向过程方法”。我一开始信以为真，认为如果能做到用面向过程方法，从组织级、系统级到模块级层层分解也不错的。后来发现，经理所说的“面向过程方法”其实是随意的功能分解，也就是没有方法。

类似的场景还有：软件组织负责人说“我们现在采用的是敏捷过程”，稍微深入了解，多半会发现其实他所说的“敏捷过程”就是没有过程。不要让“面向过程”和“敏捷”成为偷懒的庇护所。

1.8 什么样的系统不需要建模

这是经常被问到的一个问题。前文已经说过，编码就是建模，所以什么样的系统都需要建模。这个问题真正要问的是“什么样的系统可以不用业务建模、需求、分析，而直接设计（编码）？”

1.8.1 市场没有小系统

过去的软件工程书籍中，谈到建模的重要性时，常会说：“自己搭个狗屋不需要建模，盖摩天大楼需要建模，因为后者更复杂。”这样的说法

并不正确。在市场经济的环境下，如果要挣到钱，搭狗屋和盖摩天大楼的复杂度是一样的。狗屋有狗屋的品牌，摩天大楼有摩天大楼的品牌，盖摩天大楼的公司要抢搭狗屋公司的市场可没那么容易——世上无易事，市场没有小系统（见图1-9）。

要是我问您，跑百米容易还是跑马拉松容易？这还用问！当然是跑百米容易了，是吧？其实我想问的是：亚洲运动员要拿奥运冠军，是跑百米容易还是跑马拉松容易？答案似乎就颠倒过来了。近邻韩国和日本都已经出过奥运马拉松冠军，比起拿百米冠军，概率要大多了。

不同形态的系统各自有各自的复杂性，看起来做一个电厂管理信息系统好像很牛，但做一块电表的学问也不小。到现在为止，我服务的组织覆盖了国内各个领域的领袖企业，包括通信、企业管理、电子商务、房地产、网络游戏、地理信息、物流、数码设备、医疗设备、工业控制等领域。业务建模、需求、分析等建模技能都适用于这些企业的项目。无论是上千万人同时使用的社交系统，还是行政人员使用的内部办公系统，还是埋藏在人体内的小设备，建模是否值得，和系统的运行形态无关，而是看软件组织有没有一颗冠军的心。

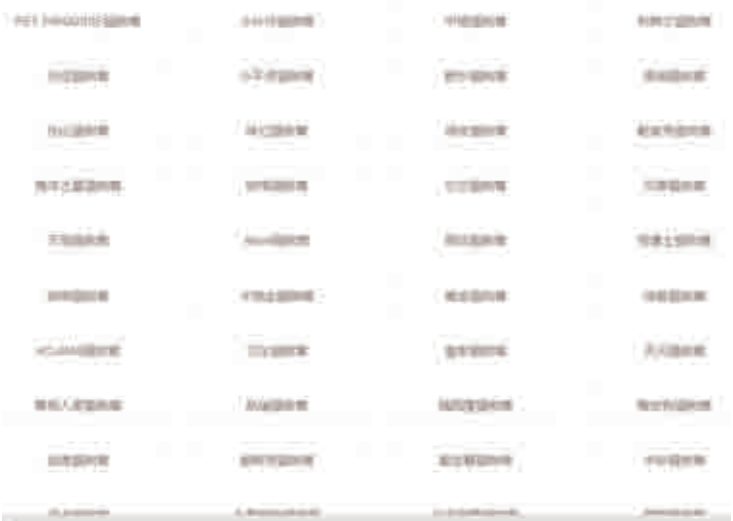


图1-9　商城中的猫狗窝品牌

1.8.2 你的系统不特别

还有一种以为不需要建模的情况是，开发团队经常认为自己做的系统“比较特别”，以此作为懒得深入思考的理由。如果学习了本书的建模技能，就会发现之前认为特别的项目其实没有什么特别，包括所谓的“遗留系统”、二次开发系统、内部系统、移动互联网系统、政绩工程……一些互联网开发人员动不动就鼓吹“互联网思维”，拼命强调自己所开发的系统有多特别，无非是偷懒和为失败寻找借口而已。


见识少的病人总以为自己得了怪病，其实到医院让医生一看，太普通了。

还有一个常听到的偷懒庇护所是“软件开发是艺术”。软件开发是不是艺术，我不知道，不过就算软件开发到了极高境界真的是艺术，恐怕也不是大多数人目前有资格谈的。下棋到很高境界，也有各种流派风格，但那是在通晓了基本棋理的基础上演变出来的，连基本棋理都没有掌握的初学者，把自己的胡思乱下也当成“流派”就不合适了。

我在指点建模人员改正他所画的模型的时候，偶尔会有建模人员不服气，“老师，难道一定要按照你这个规范吗？我自己有一套规范不行吗？”这不是规范的问题，是背后的基本道理。

师父纠正少林弟子武功招数的细节，弟子懒得去了解为什么按师父教的会好一点，反而说：“不要纠结于细节，天下的武功又不是只有少林这一派，”以为这样一说，自己就可以摇身一变成为武当派高手了。其实，少林派武功学精了，如果对武当派武功感兴趣，转起来容易很多。如果某人学少林派武功时面对细节总是以“不纠结”为由拒绝进一步思考，很难相信他学习武当派武功时会好到哪里去。

本书到现在为止，已经说了很多回“偷懒”，就是强调世上无易事，好的方法应该能强迫您思考，强迫您付出心血和汗水来获得竞争优势，反之就是忽悠，就像前些年一些甜得发腻的敏捷宣传。

1.9 案例介绍

本书的案例讲述UMLChina如何改进其内部业务系统的故事。我在序言说到，UMLChina秉持“内外有别”的原则。在外面看来，UMLChina的网站其貌不扬，十几年如一日，UMLChina组织的信息化其实藏在背后。

UMLChina一开始把领域放在软件工程，后来发现，这个领域已经太大了，没有能力在这么大的范围内做到最好，于是缩小范围，专注于和UML相关的建模技能。2002—2004年我们和出版社合作翻译了《人月神话》《人件》等软件工程书籍，这方面的书籍后来不再做了，只做建模相关的书籍。

我为UMLChina所做的定位是：**做世界上最小和最好的建模咨询公司**。咨询工作适合做精，不适合做大。UMLChina没有招揽或培养满屋子的“年轻资深咨询师”，通过扩大规模来提高收入，有的机构这样做了，也获得了更多的收益，但那不是我们的路。

一旦从深度上定位，就会发现要做的事情非常多，2005年，我决定着手开发UMLChina的业务系统。经过这些年持续改进和升级（也仍将继续改进和升级），虽然现在离我希望的“武装到牙齿”的境界还有很大的距离，但这个和UMLChina业务结合在一起的系统确实能够让UMLChina不必请更多的人就可以保持正常运作。因为很多业务逻辑已经封装在系统中，所以也比较容易分权给助理。

后面给出的素材绝大部分是真实的，如果有一些地方做了模糊处理，那是出于商业上的考虑。UMLChina在商业方面的事宜有公司负责运作，本书隐去公司真实名字，仍把这个公司叫做UMLChina。

1.10 模型的组织

从前面的图1-6可以知道，建模工作流和所用的UML元素不是一一对应的。模型可以按照UML元素的种类组织，也可以按照工作流来组织。

本书推荐的模型组织方式是按工作流组织，如图1-10所示。本书提供了一个初始Enterprise Architect（以下简称EA）模型，该模型已经按照图1-10的组织方式建好了包，并且添加了一些业务建模和分析的构造型（Stereotype）。我建议读者直接从本书提供的初始模型开始做，按部就班填空即可。初始模型的下载地址是http://www.umlchina.com/training/myproject.rar。初始模型使用EA13编辑保存，使用其他EA版本打开编辑也应该没有问题。您在熟练掌握本书的建模技能以后，如果体会出对您的项目更合理的组织方式，可以抛弃本书所推荐的方式。如果您平时使用的工具不是EA，而是RSA、StarUML、VP-UML等，也可以自行按照图1-10的方式组织模型。UML工具（包括EA）一般都会预置一些模板，建议先无视它们。

图1-10　按工作流组织模型（推荐做法）

另外一种常见的模型组织方式是按视图来组织，如图1-11所示。以前Rational Rose默认的组织方式就是这样，最开始我也是这么做的，但是后来发现开发人员容易出问题的地方不是用什么图，而是目前在做什么。开发人员的思维经常跳来跳去，无意识地改变思考的焦点——正在讨论系统之间的协作流程，突然就跳入某个系统的内部讨论类的关系，甚至类的某个操作内部的实现。所以，本书不推荐按视图组织模型。

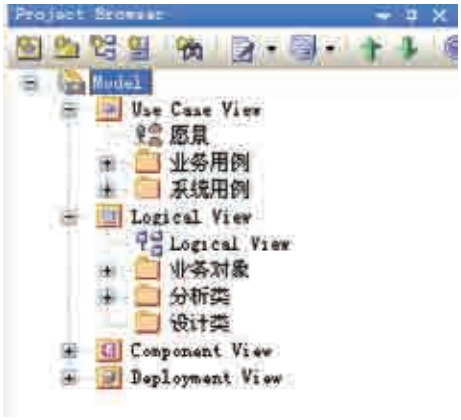


图1-11　按视图组织模型（不推荐）

本书不提供练习题答案，请扫码或访问http://www.umlchina.com/book/quiz1_3.htm完成在线测试，做到全对，自然就知道答案了。

1. UML三友是哪三位?

A）Messi、Neymar JR和Luis Suárez

B）Luciano Pavarotti、Placido Domingo和Jose Carreras

C）Martin Fowler、Kent Beck和Alistair Cockburn

D）James Rumbaugh、Grady Booch和Ivar Jacobson

2. 以下不属于OOAD方法学的是_____。

A）Booch方法　　B）Demarco方法

C）Rumbaugh OMT　　D）Coad/Yourdon方法

3. 以下不属于UML图形的是_____。

A）流程图　　B）状态机图

C）序列图　　D）通信图

4. 以下不属于本书推荐常用的UML元素的是_____。

A）用例图　　B）组件图

C）序列图　　D）类图

5. 以下不是UML工具的是_____。

A）Enterprise Architect　　B）DOORS

C）Astah　　D）MagicDraw

E）Plato　　F）Rhapsody

6. 一些开发人员更喜欢画“草图”，然后说“来！我给大家讲讲”，深层原因是_____。

A）这样更敏捷，现在流行“敏捷”

B）草图更自由，有发挥的空间

C）想通过形式的粗陋遮掩内容的粗陋

D）亲身讲解胜过模型文档交流

7. 经常被当作“偷懒庇护所”的说辞有（多选）_____。

A）软件开发是艺术，艺术是没有道理可讲的

B）我们敏捷了

C）建模带来竞争优势

D）不管用什么方法，把项目做成功就是好方法

8. 以下软件开发名人中，和前央视主持人小崔（崔永元）同龄的是_____。

A）Martin Fowler　　B）Kent Beck

C）Ivar Jacobson　　D）Peter Coad

E）James Rumbaugh　　F）Grady Booch

9. 以下说法正确的是_____。

A）在项目中可以只挑选一部分UML元素来使用

B）UML模型的最佳案例就是建模工具附带的例子

C）团队引进UML时，努力达到的最终目标应该是完整应用所有的UML元素

D）UML是软件开发人员和客户之间沟通的绝佳工具

10. 以下说法正确的是_____。

A）功能很少的系统不需要建模

B）类很少的系统不需要建模

C）市场上已经有很多现存产品的系统不需要建模

D）ABC都不正确

1.11 工具操作

在开始项目实作的讲解之前，我们先建立模型，并对Enterprise Architect做一些设置。

【步骤1】在Windows资源管理器双击模板文件myproject.eap，Enterprise Architect启动并打开myproject.eap（见图1-12）。

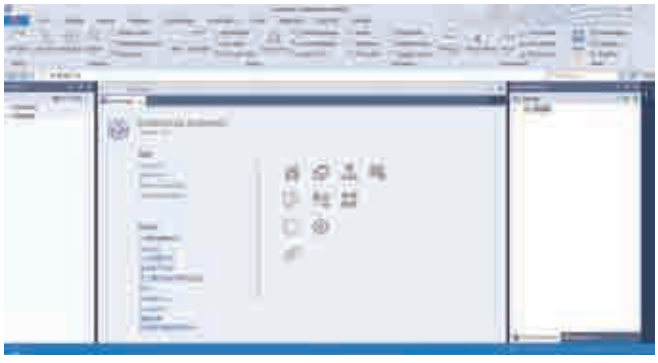


图1-12　启动EA

【步骤2】在左上角菜单选择Save Project As...，在对话框Target Project栏中设置新建模型文件的位置和名称，确认选中Reset New Project GUIDs复选框，单击Save As按钮。与资源管理器里复制模板文件相比，以上做法可以重置所有标识值，保证模型元素不冲突（见图1-13）。

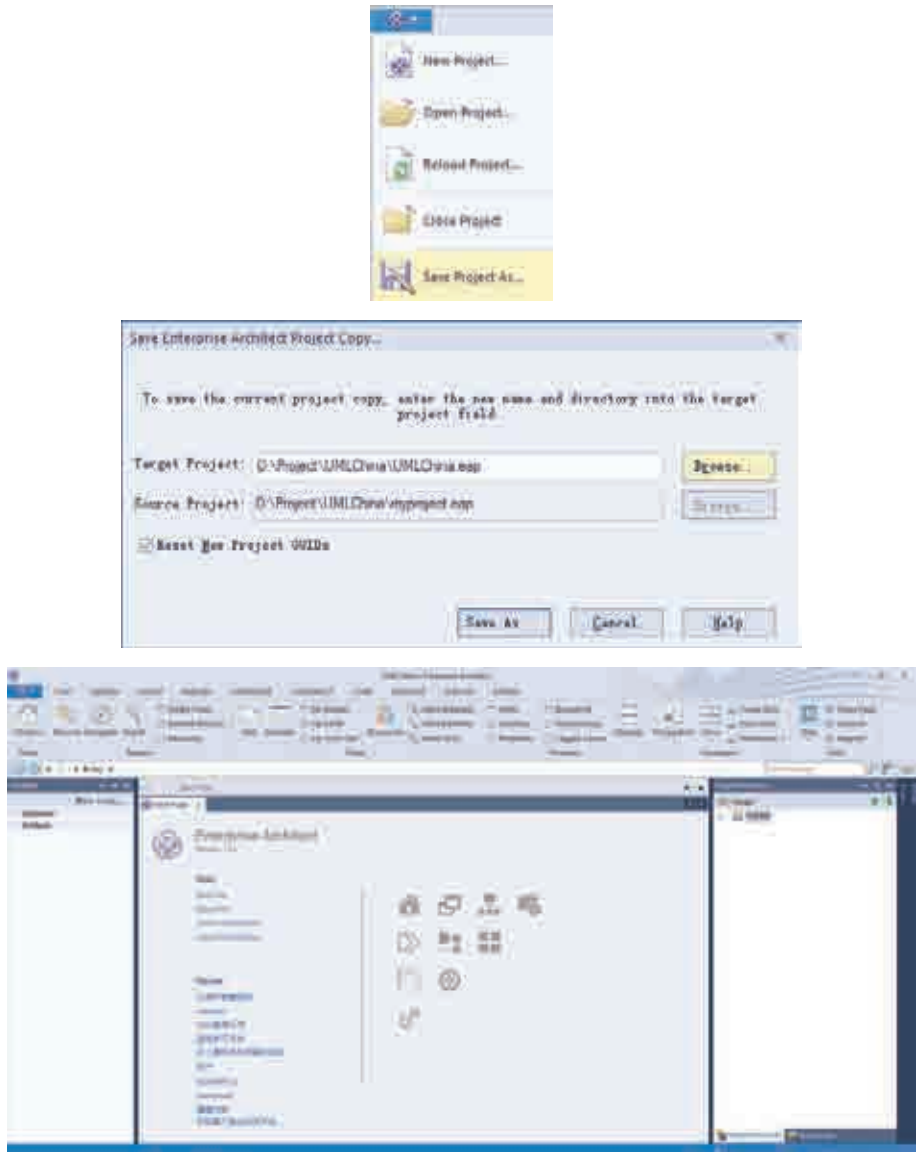


图1-13　从模板项目创建新项目

【步骤3】单击CONFIGURE菜单，选择Model组中的Options，在Manage Project Options对话框中选择General页签，设置Font Face和Font Size为合适的默认字体。本书的选择是大小为14的微软雅黑（见图1-14）。

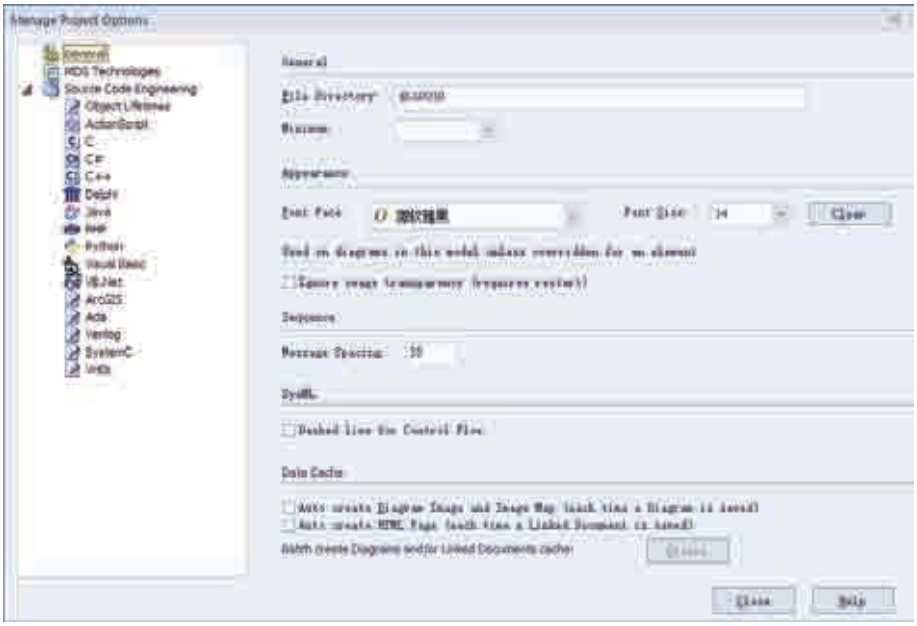
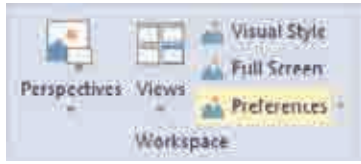


图1-14　设置默认字体

【步骤4】单击START菜单，选择Workspace组中的Preferences，在Diagram | Theme页签的Diagram Theme栏选择Monochrome for printing，单击Save。这一步把图形风格设成黑白色，如果不喜欢，可以跳过（见图1-15）。



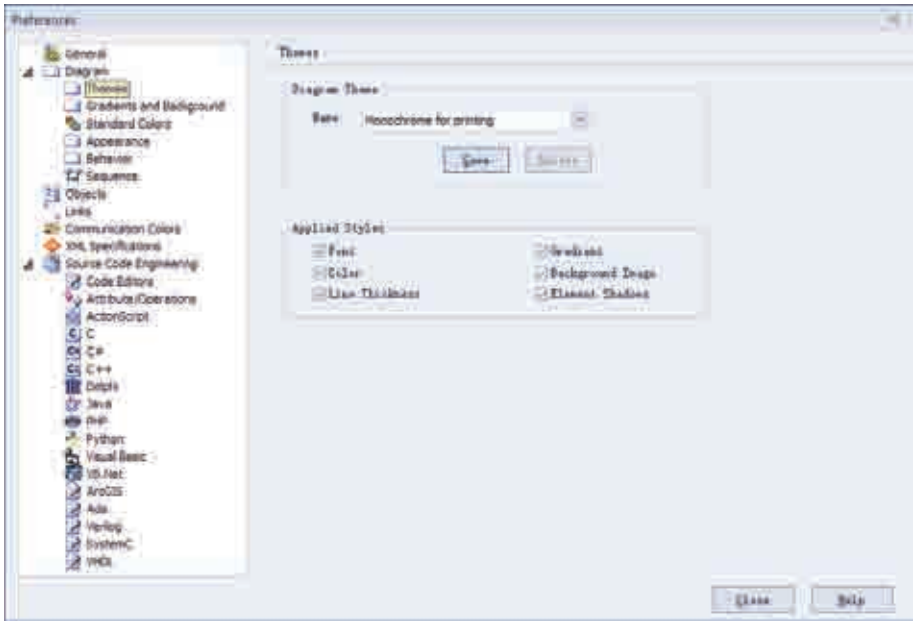


图1-15　设置图形颜色风格