

第1章 建模和UML

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

您在阅读《软件方法》时如果发现错误，欢迎通过下面提供的邮箱、微信或QQ告知。如果作者认为有道理，决定在下一次发布时根据您的意见修改，将付给您5.12元报酬，并在书中说明您的贡献。报酬通过支付宝或微信支付。（1）任何您认为的错误都可以，包括错别字。（2）同一错误仅支付最先指正者报酬。（3）请根据最新版本作指正。QQ和QQ邮箱：1493943028@qq.com，微信：umlchinapan

1.1 粗放经营的时代已经远去

中国刚改革开放时，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？我们每想到一个“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：

扫码或访问http://www.umlchina.com/book/quiz1_1.htm完成在线测试，做到全对以获得答案。



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. 软件开发做设计工作的目的是

- 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，这就是创新的动力之源。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“增加销售”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。

★高焕堂在[高 2008]中讲到：用例是收益面，对象是成本面。本书发展了他的思想。

如果需求和设计不分，利润就会缩水。从需求直接映射设计，会导致功能分解得到重复代码。如果从设计出发来定义需求，会得到一大堆假的“需求”。

拿自古以来就有的一个系统“人体”来举例。人体能提供的功能是会走路，会跑步，会跳跃，会举重，会投掷，会游泳……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”、“血液循环子系统”、“神经子系统”、“内分泌子系统”……。很多人体的“子系统”是不能从需求直接映射出来的，需要设计人员的想象力。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”。

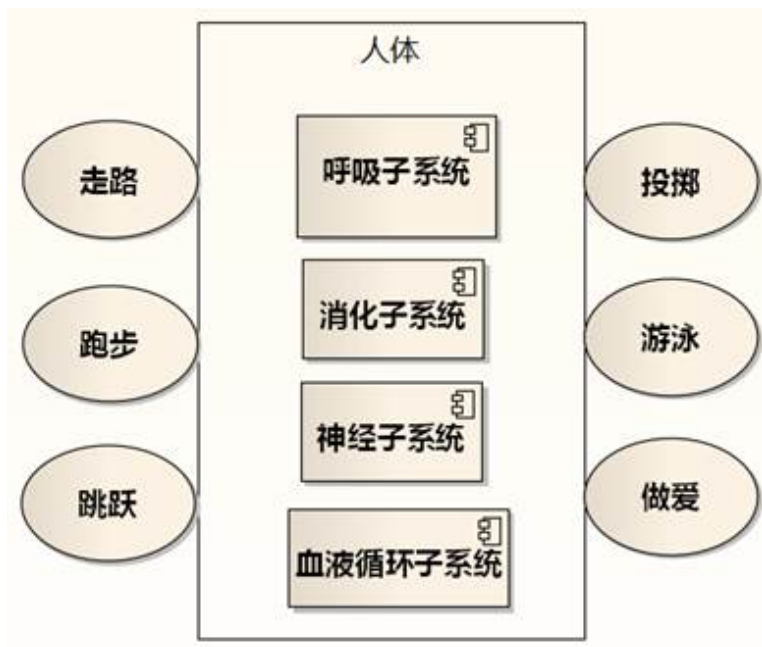


图1-1 人体的需求和设计

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要求来找，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.....”需求包是从外部对系统功能所做的简单分包，子系统应根据部件的耦合和内聚切割得到。

我把需求和设计的区别简要列举如图1-2，后面的章节我再慢慢阐述这些观点。

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

1.3 建模 workflow

要迈向“低成本制造好卖的系统”的境界，并非喊喊口号就能达到，需要静下心来，学习和实践以下各个建模 workflow 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。如果能学会通过业务建模去推导新系统的需求，而不是拍脑袋得出需求，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领域机制。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，可以帮助我们获得基于核心域的复用。

4. **设计**——为了满足质量需求和设计约束，核心领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，就会把这些工作通通称为“设计”或者“文档”。例如问开发人员在做什麼，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，不了解他在业务建模和需求方面有多少研究。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，见图1-3。如果清楚了解这一点，即使用C#也照样可以表达需求，用Word也可以“编码”。

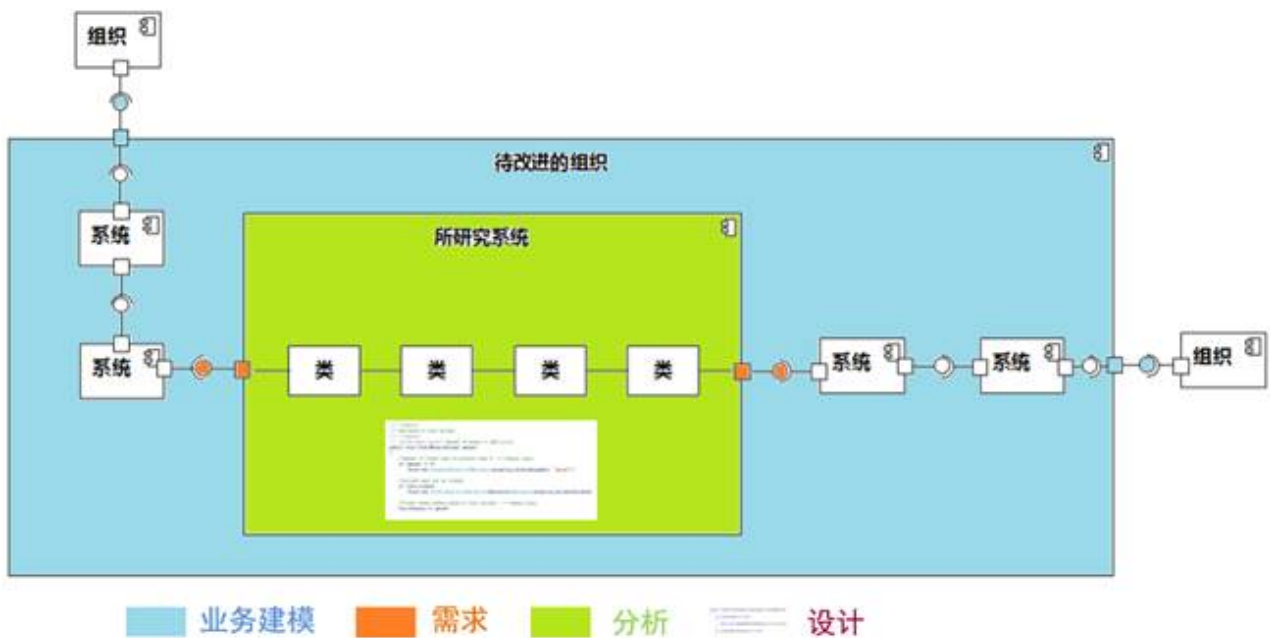


图1-3 建模 workflow 思考边界

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，找出最分值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。这个学渣就是中国足球。

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，参加更残酷的竞争，就体会到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。一种是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。

扫码或访问http://www.umlchina.com/book/quiz1_2.htm完成在线测试，做到全对以获得答案。



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 workflow

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 workflow

```

/// <summary>
/// Add money to this account
/// </summary>
/// <param name="amount">Amount of money to add</param>
public void CreditMoney(decimal amount)
{
    //Amount to Credit must be greater than 0. --> Domain logic.
    if (amount <= 0)
        throw new ArgumentException(Messages.exception_InvalidArgument, "amount");

    //Account must not be locked.
    if (this.Locked)
        throw new InvalidOperationException(Resources.Messages.exception_AccountIsLocked);

    //Credit means adding money to this account. --> Domain Logic
    this.Balance += amount;
}

```

- 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Email、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，软件开发的方法学一直在进化。从最开始没有方法，到简单的功能分解法，再到数据流/实体关系法。进入1990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，用不同方法学符号各自表达如图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，一同参与这项工作。他们三个人被称为“三友”（three amigo）。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。

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	

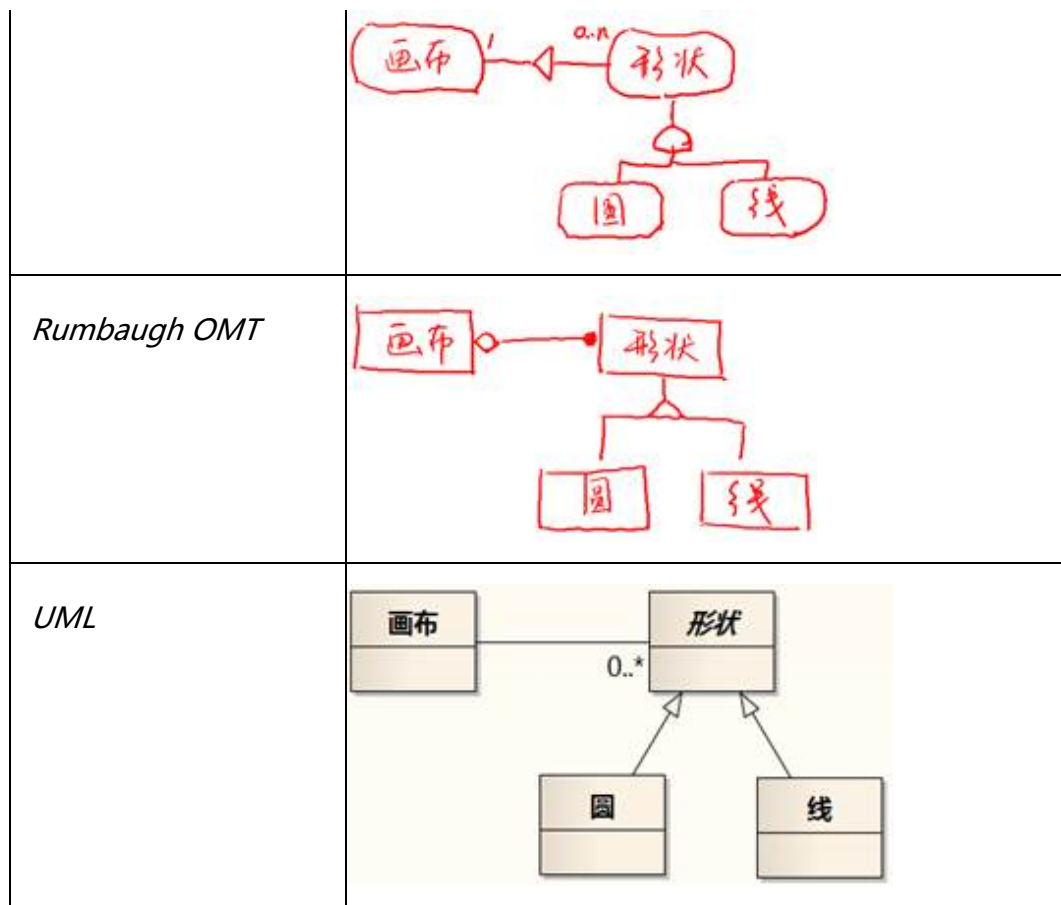


图1-4不同方法学图形比较（同样一个三角形符号，在Coad/Yourdon方法学中用于表示关联，而在OMT方法学中用于表示泛化。）

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/CMML、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，基本上是方法学家所写。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、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 workflow

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）。

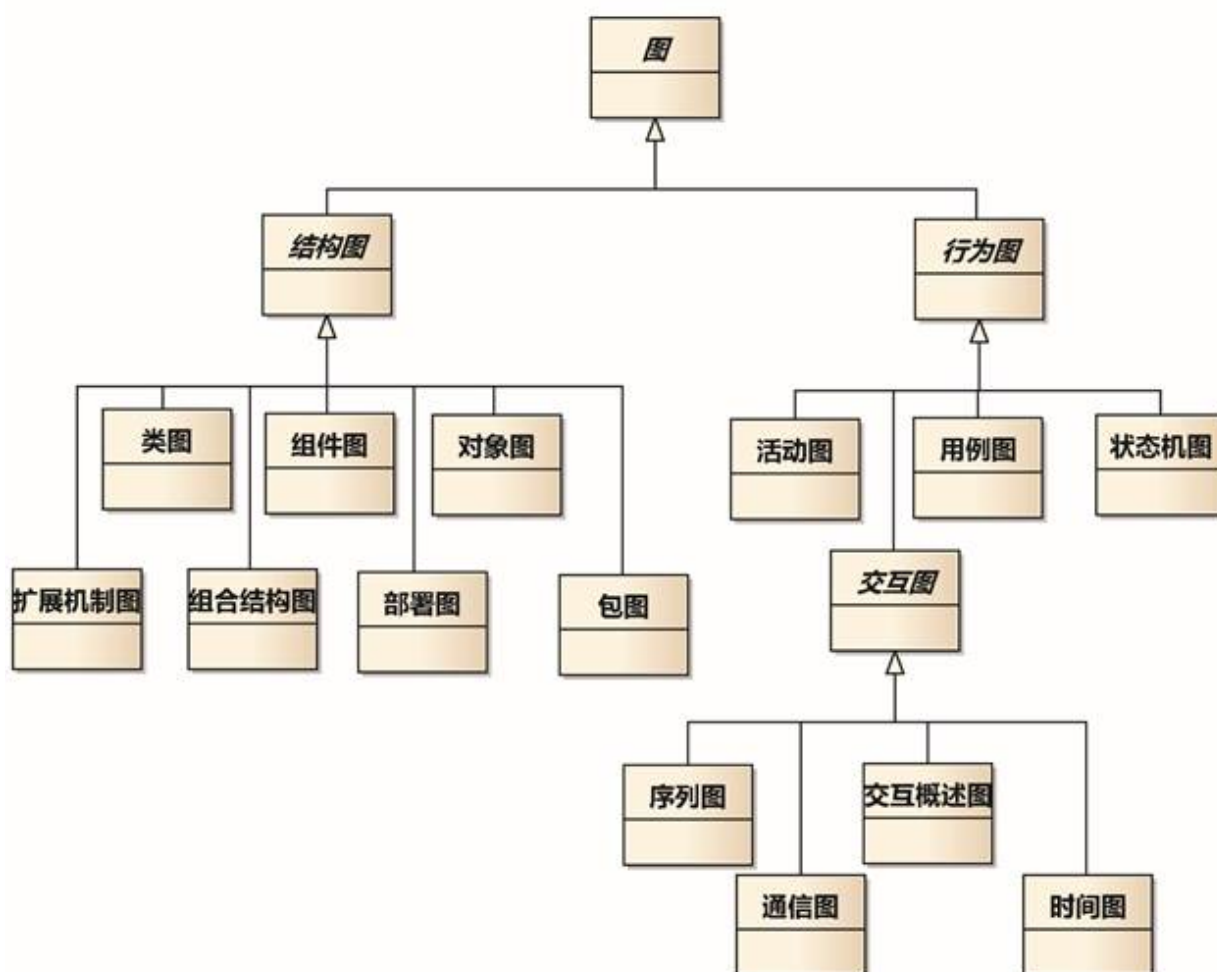


图1-5 UML图形（根据UML2.5规范绘制）

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，里面各种工具都有。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，完完整整地实施了整套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了？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 workflows 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	包	扩展	序列	通信	状态	活动	时间	交互	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（●：优先使用，√：可以使用）

从“推荐UML元素”一栏中可以看到，常用的UML图形用例图、类图、序列图三种就够了。针对特定类型的项目，可以按需要添加图形。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 workflow 目前推荐的做法是不需要画UML图，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意装13“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。

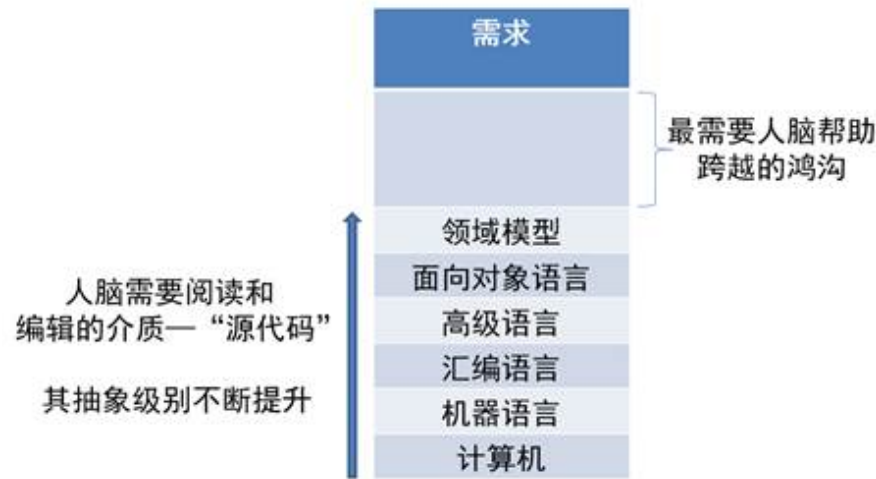


图1-7 “源代码”的发展历程

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，但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。

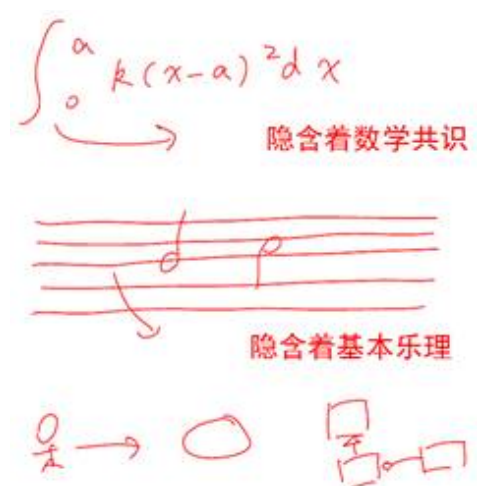


图1-8 符号背后隐含基本共识

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形成了共识，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，非被打得落花流水不可。

★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在第7章详细叙述。

1.7 建模和敏捷 (Agile)

敏捷运动在1990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。我有专文《敏捷传播的最佳实践》[潘 2012]比较敏捷和社会主义在理论背景以及成功经验方面的共通之处。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开发人员。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有大楼的结构、所用的材料、所在位置的地质环境等，但这些都涉及到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、工人是否结对洗澡，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，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”，稍为深入了解，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”、“敏捷”成为偷懒的庇护所。

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。

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，天下的武功又不仅仅只有少林这一派”，以为这样一说，自己就可以摇身一变成成为武当派高手了。其实，少林派武功学精了，如果对武当派武艺感兴趣，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。

民科苦苦思索得到的宝贝发现，其实很多年前已经写在教科书上了。

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“自己搭个狗窝不需要建模，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，搭狗窝和盖摩天大楼的复杂度是一样的。狗窝有狗窝的品牌，摩天大楼有摩天大楼的品牌，飞机有飞机的品牌，汽车有汽车的品牌，滑板车有滑板车的品牌——世上无易事，市场没有小系统。

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。也就是说，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。

PET PARADISE猫狗窝	小伙伴猫狗窝	中恒猫狗窝	利其尔猫狗窝
乐佳猫狗窝	小不点猫狗窝	舒乐猫狗窝	豪迪猫狗窝
托比猫狗窝	咪它猫狗窝	顽皮猫狗窝	耐威克猫狗窝
海洋之星猫狗窝	哈格猫狗窝	它它猫狗窝	贝享猫狗窝
天宠猫狗窝	Ason猫狗窝	丽丝猫狗窝	宠博士猫狗窝
哈根猫狗窝	卡地全猫狗窝	魔全猫狗窝	宠爱猫狗窝
KOJIMA猫狗窝	艾比猫狗窝	皇家猫狗窝	天元猫狗窝
草柳人家猫狗窝	凯瑞猫狗窝	瑞西屋猫狗窝	雅皮狗猫狗窝
创逸猫狗窝	爱斯克猫狗窝	爱宝嘉猫狗窝	中彩猫狗窝
上上猫狗窝	卡地全猫狗窝	魔全猫狗窝	宠爱猫狗窝

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。

1.9 案例介绍

本书的案例讲述UMLChina自己的故事。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。例如，2002-2004年间我们和出版社合作翻译了《人月神话》、《人件》等软件工程书籍，后来这方面的书籍就不再做了，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在商业方面的事宜由公司负责运作，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 workflow 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 workflow 来组织。

本书推荐的模型组织方式是按 workflow 组织，如图1-10。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，不过使用其他EA版本打开编辑应该没有问题。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。以前Rational Rose缺省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，无意识地改变研究范围，思考的边界一会在系统之间，一会在待开发系统的边界，一会跳到系统里面类的关系，一会又细到某个操作内部的代码。所以，不推荐按视图组织模型。

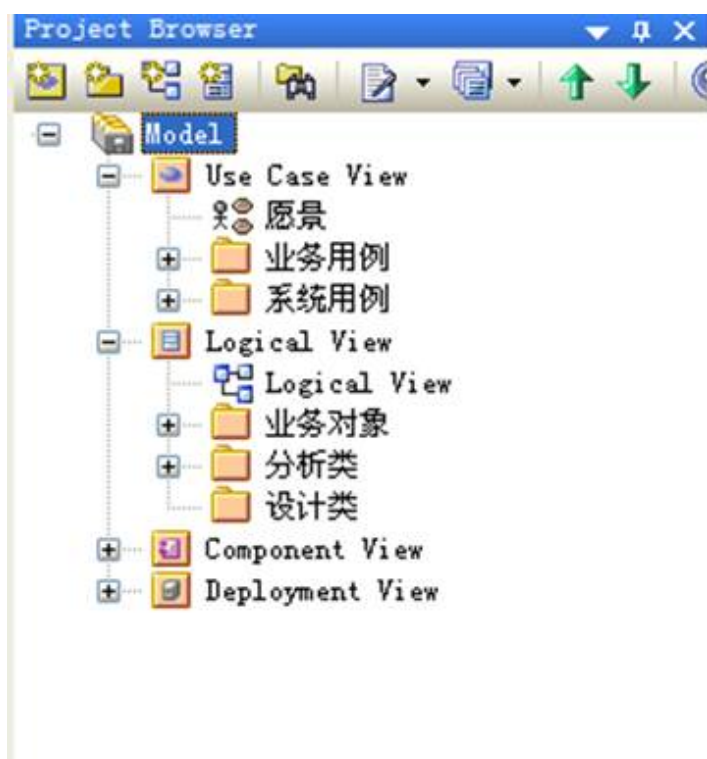


图1-11按视图组织模型（不推荐）

访问http://www.umlchina.com/book/quiz1_3.htm完成在线测试，做到全对以获得答案。

1. UML三友是？

- 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（本题可多选）

- 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。

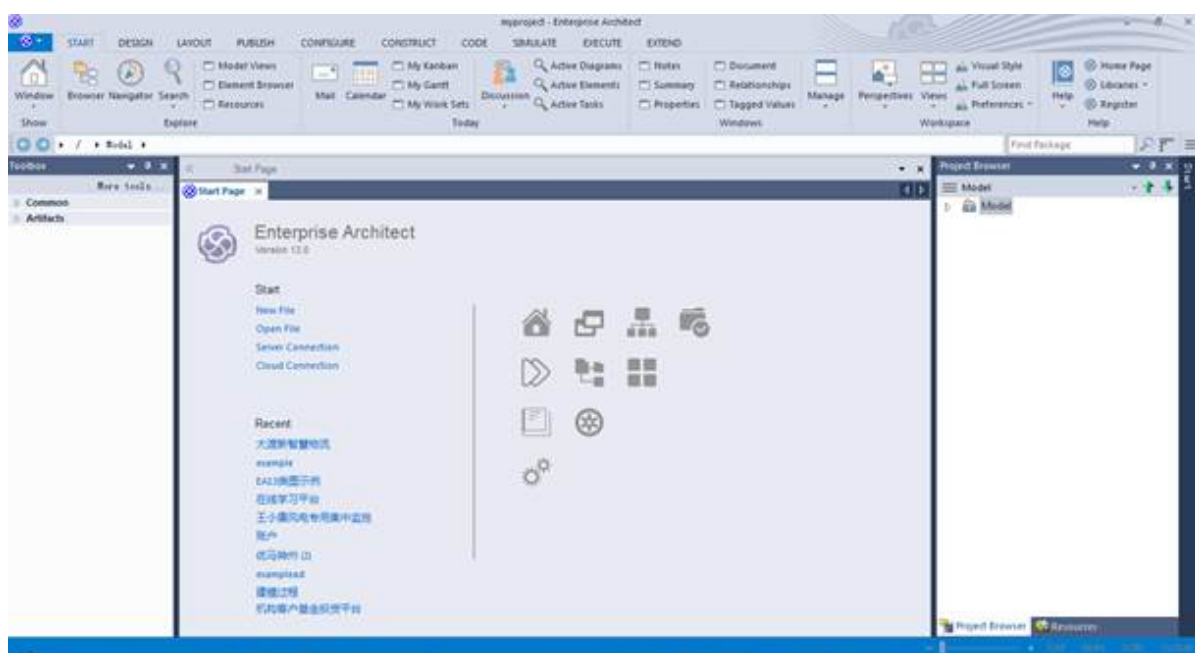


图1-12 启动EA

【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。与在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。

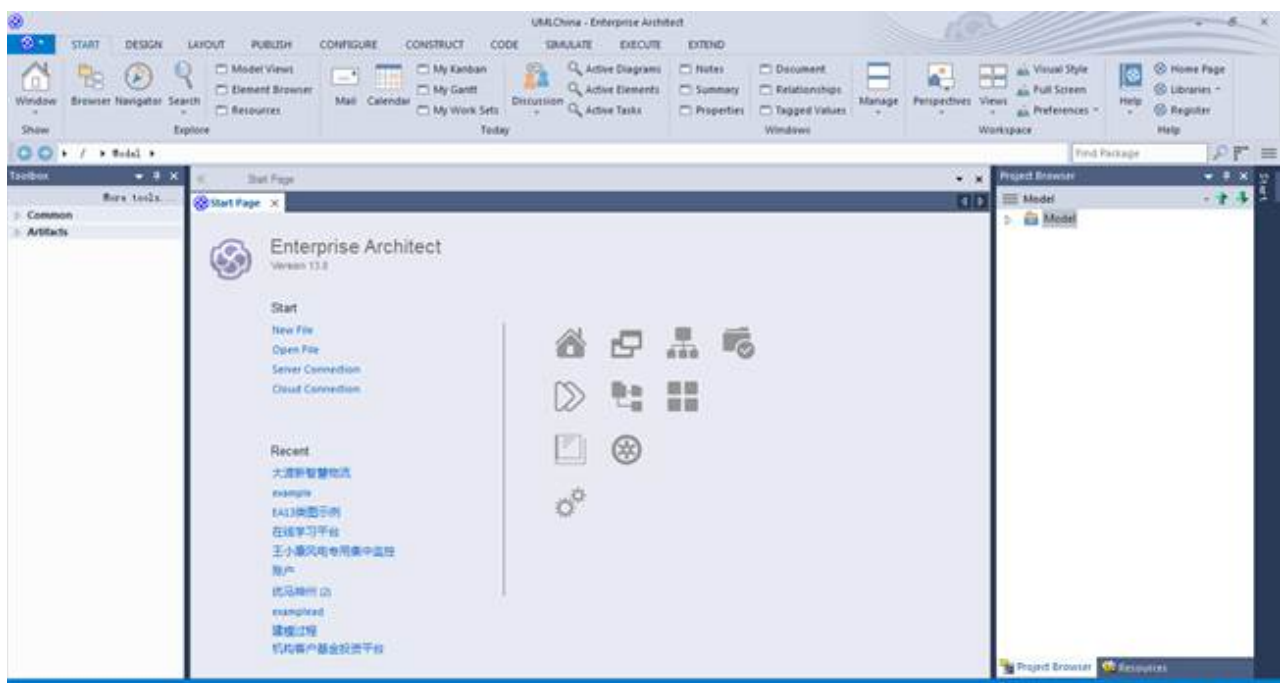
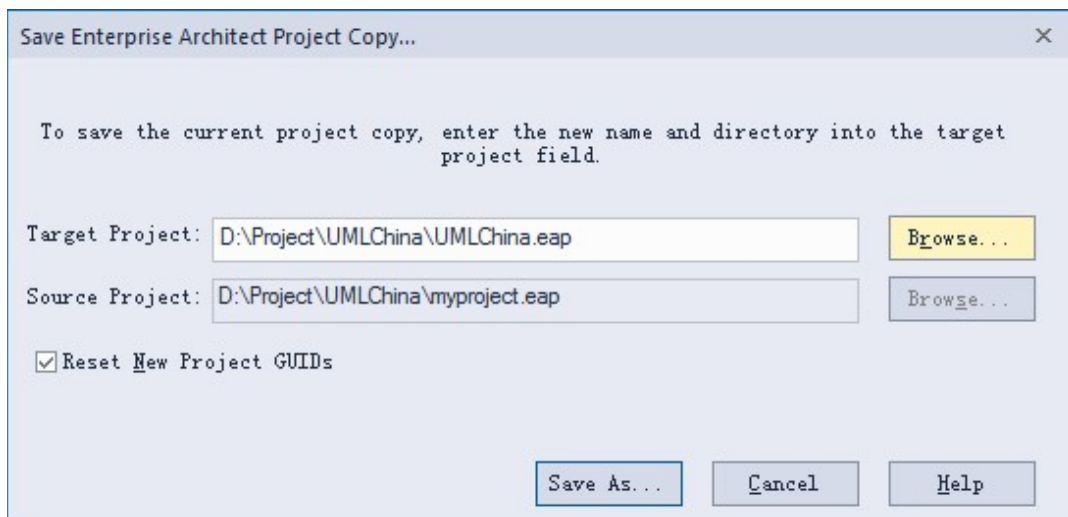
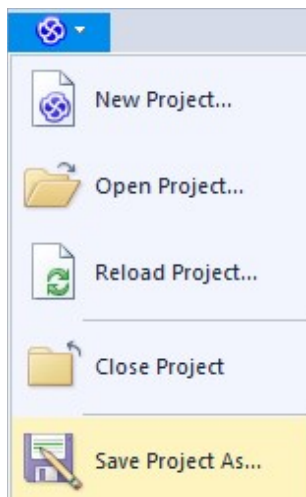


图1-13 从模板项目创建新项目

【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**为合适的缺省字体。本书的选择是大小为14的微软雅黑。

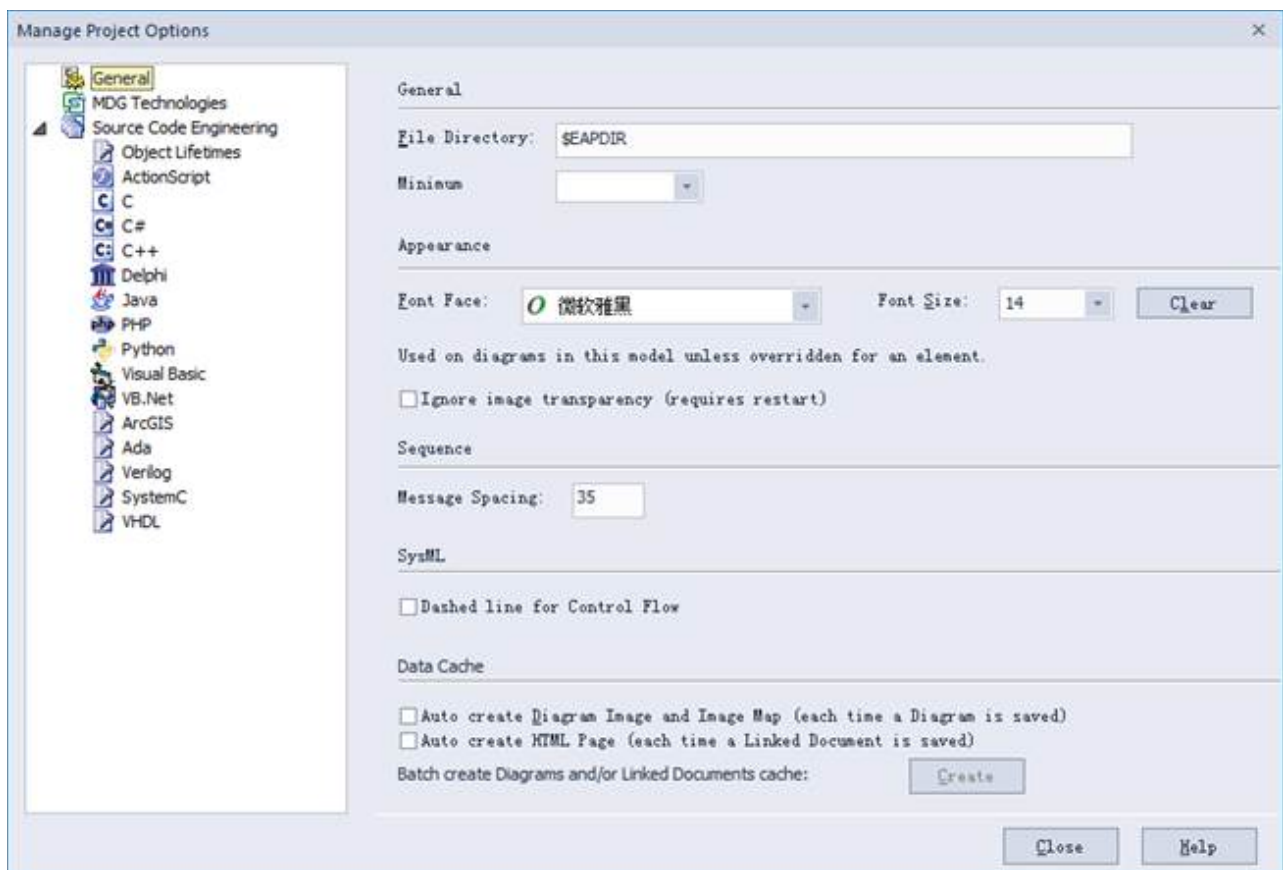
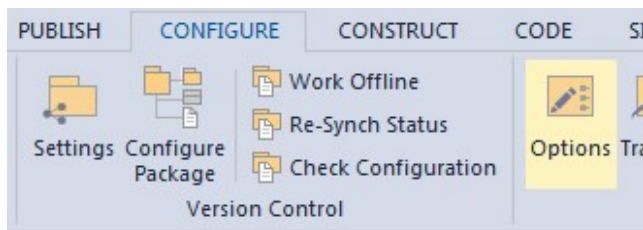
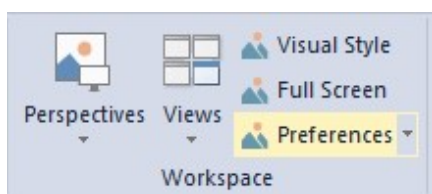


图1-14 设置缺省字体

【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。



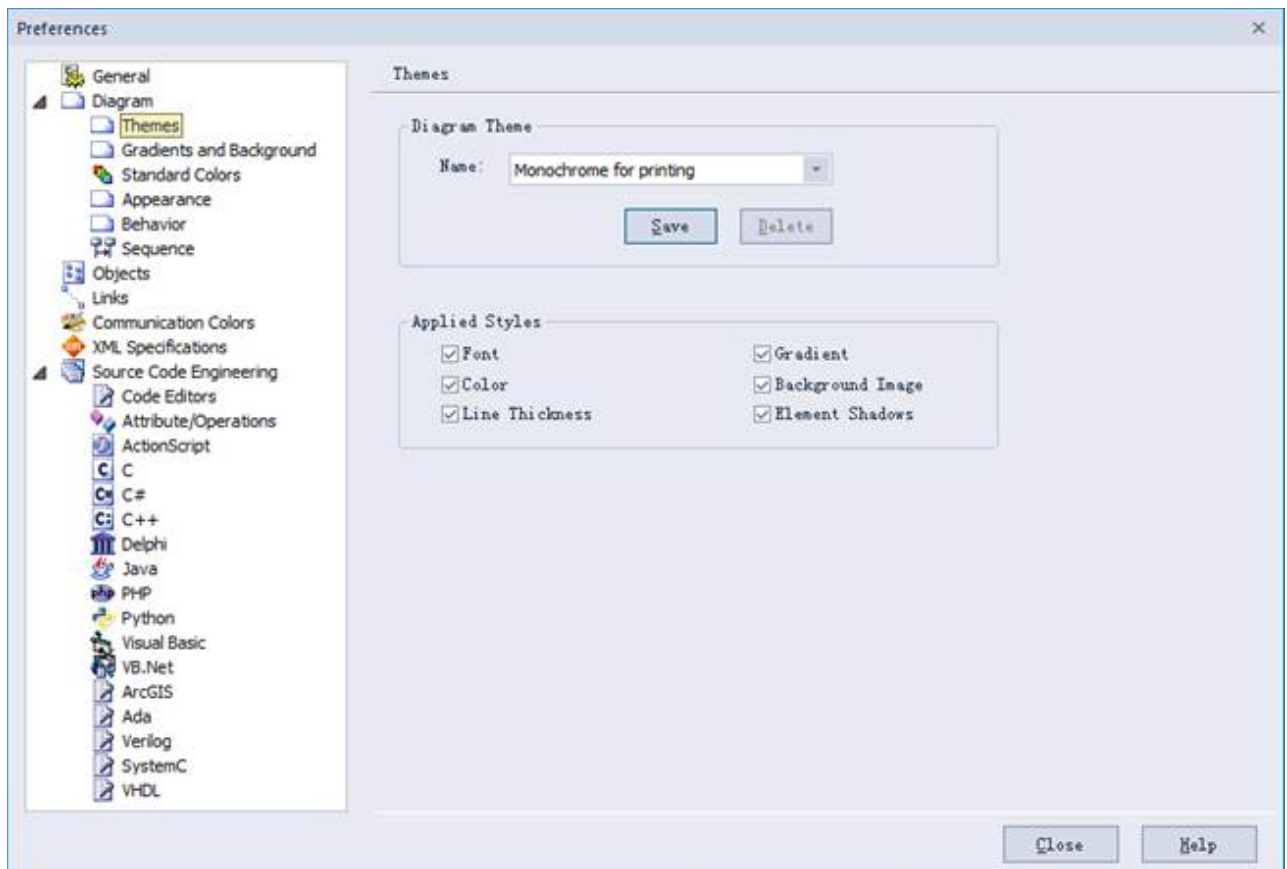


图1-15 设置图形颜色风格