软件方法（草稿）

发布日期：2011.4.27

请用 9.0 版以上 Arcobat Reader 阅读，否则可能无法使用书中的测试题。

UMLChina 腾讯微博

http://t.qq.com/UMLChinaPan

UMLChina 新浪微博

http://t.sina.com.cn/2117056262

潘加宇

自序

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；1986

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和形式都算是糟透了，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。

目前（2011 年 4 月）为止，我已经上门为已经超过 140 家的软件组织提供需求和设计技能的训练和咨询服务。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来和我讨论。”虽然一再这样强调了，有的学员还是经常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啊。

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，目前已经有三十多本软件工程书籍上有 UMLChina 的标记了。不过我一直没有自己写一本书，主要原因还是觉得自己的积累不够，思考的深度也不够，对软件开发的认识还在不断变化。如果没有自己成型的东西，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 后来开始采取了“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将近十年，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，开发团队还是高估了自己的“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。感谢曾经选择过我的服务的伙伴们。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。

目前还没有和任何出版社商议出纸书事宜。本书先以电子版方式公布，不定期更新版本，您可以到

http://www.umlchina.com/book/panjiayu.htm 查看新的版本。因为我经常为《程序员》杂志写文章，所以本书中的一些文字您可能在《程序员》上看到过。

每一章的后面，我会提供一些针对该章内容的自测题，读者感兴趣可以测试一下。这些测试题以嵌入 Flash 的方式提供，请留心一下您正在使用的 PDF 阅读器是否支持 Flash。

一些作者喜欢在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，所以我也效仿一下，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；如果是截取真实模型的图片，可能会涉及到各种工具。我不像 Robert C. Martin 那样，女儿已经长大到可以帮画插图，所以非 UML 模型的插图，我都自己用 Wacom 笔来画，可能丑了一些，请见谅。

关于本书的任何反馈，请发邮件到 umlchina@gmail.com。

第 1 章 建模和 UML

脚下没有路，我们走出来，狂风暴雨过天边

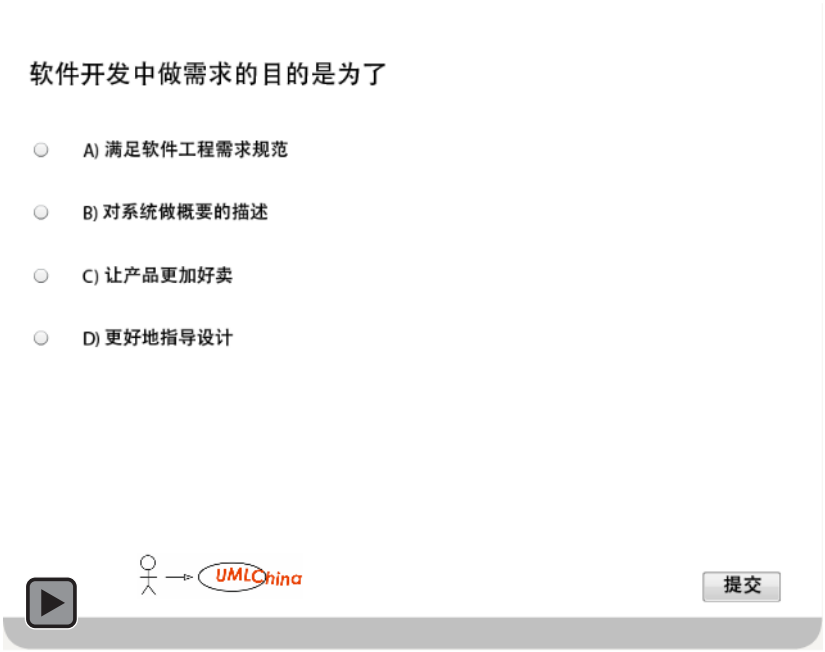
《新空气的声音》；词曲：张全复、毕晓世、解承强，唱：新空气；1988

粗放经营的时代已经远去

中国刚迈入改革开放时，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。为什么？当时的市场几乎空白，竞争非常少。农民企业家思路很简单：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）。软件开发行业也是一样，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，所以开发财务软件就能赚钱。现在呢？我们每想到一个“点子”，可能有上千人同时在这样想；我们要做一个东西，可能发现市场上已经有许多类似的产品，你卖高价，他就卖低价，你卖低价，他干脆就开源。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。

许多开发团队里面往往会有一些高手，他们是项目的顶梁柱。这些“高手”在职业道路的初期做项目也是失败的，但经过在失败中不断积累经验，慢慢开始能够成功完成项目。不过，“高手”靠的是头脑里面的隐式知识，这些知识没有经过整理，也不一定都正确，而且“高手”**潜意识**里出于利益的考虑，并不愿意积极和大家分享，本书希望能够讲述一些能够被整个团队共享的显式知识，使团队有可能在不同的项目中复制成功。

本书聚焦于两方面的技能：需求和设计。关于需求和设计，开发人员可能每一天都在做，但是否理解背后的道理呢？我们来做一些测试：



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）制造的成本要低。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“产品好卖”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。您能低成本生产某种软件产品，但不一定能保证它好卖。您的某种产品好卖，但如果生产成本太高，或者在市场需要新型号时，无法复用之前的组件，又要投入大量人力物力去重新制造，最终还是赚不了多少钱。

需求设计不分，利润缩水。例如从需求直接映射设计，会导致功能分解得到重复代码。如果从设计直接找需求，会导致得到一大堆假的“需求”。

拿自古以来就有的一个系统“人体”来举例。人体对外的功能是会走路，会跑步，会跳跃，会举重，会投掷，会游泳…。但是设计人体的内部结构时，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”…..。这些“子系统”中很多是不能从需求直接找出来的，需要设计人员的想象力。水店老板要雇一个送水工（即租用一个人肉系统），他只要求这个工人能跑能扛就行，管他体内构造如何。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”。送水工能这样找工作吗：老板，我有心脏管理功能，你请我吧！

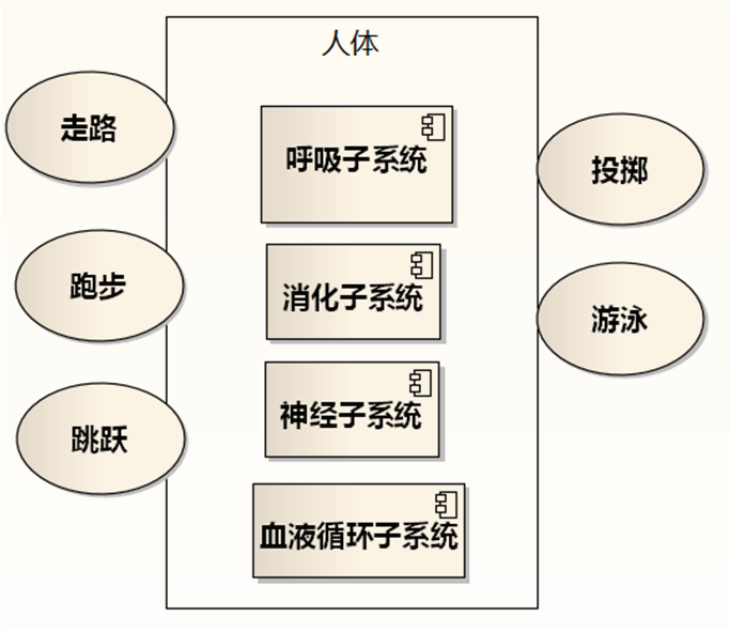


图 1-1 人体的需求和设计

需求要具体，设计要抽象。或者说，需求，要把产品当项目做；设计，要把项目当产品做。后面的章节我再慢慢

阐述这些观点。

核心工作流

要迈向“低成本制造好卖的各款产品”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核心工作流中的技能：

1. **业务建模——**描述组织内部各系统（人肉系统、机械系统、电脑系统...）如何协作来为组织的“客户”提供服务。新系统只不过是组织为更好地满足客户，对自己的内部重新设计而购买的一个零件（和招聘一个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——**聚焦于待开发系统的边界，详细描述系统要卖得出去必须具有的外部表现——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——需求规格说明书是“卖”和“做”的衔接点。

3. **分析——**提炼系统内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。对核心领域作研究，可以帮助我们获得基于核心域的复用。

4. **设计——**将核心域知识和非核心域知识结合，最终实现系统。说“代码就是设计”指的就是这狭义的“设计”。代码确实是设计，但代码不是分析，不是需求，不是业务建模。很多时候开发人员乱用“设计”这个词，把“编码以外的所有工作”统统称为“设计”。后来又有牛人说了：代码就是设计。这么一推导，不就变成了：代码就是一切？

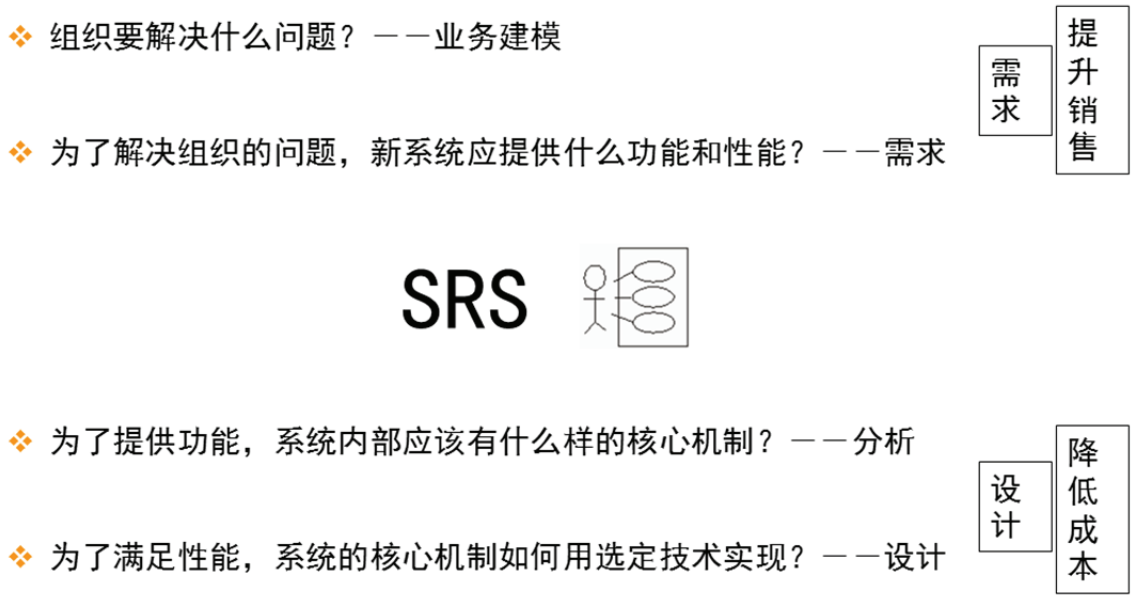


图 1-2 核心工作流

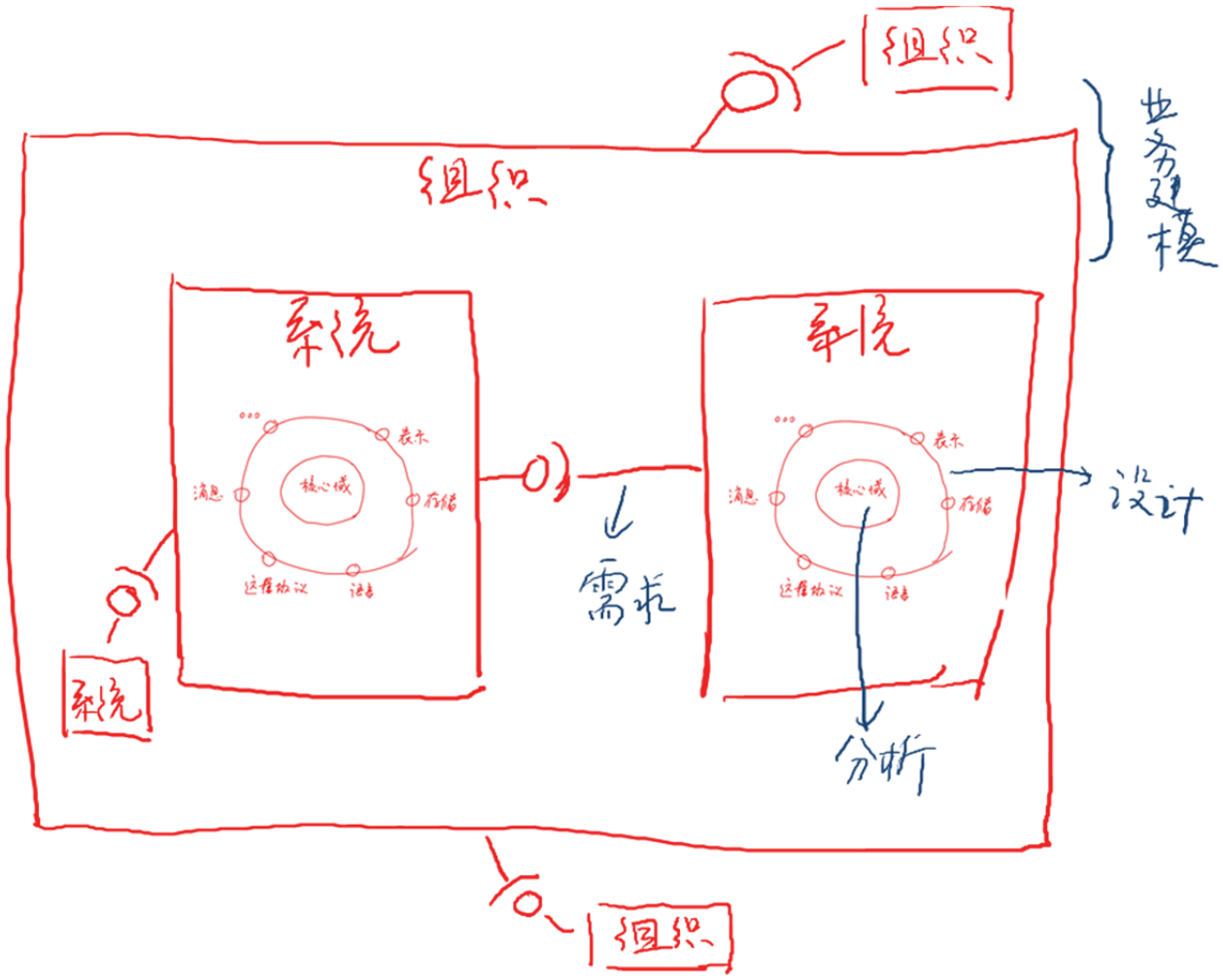
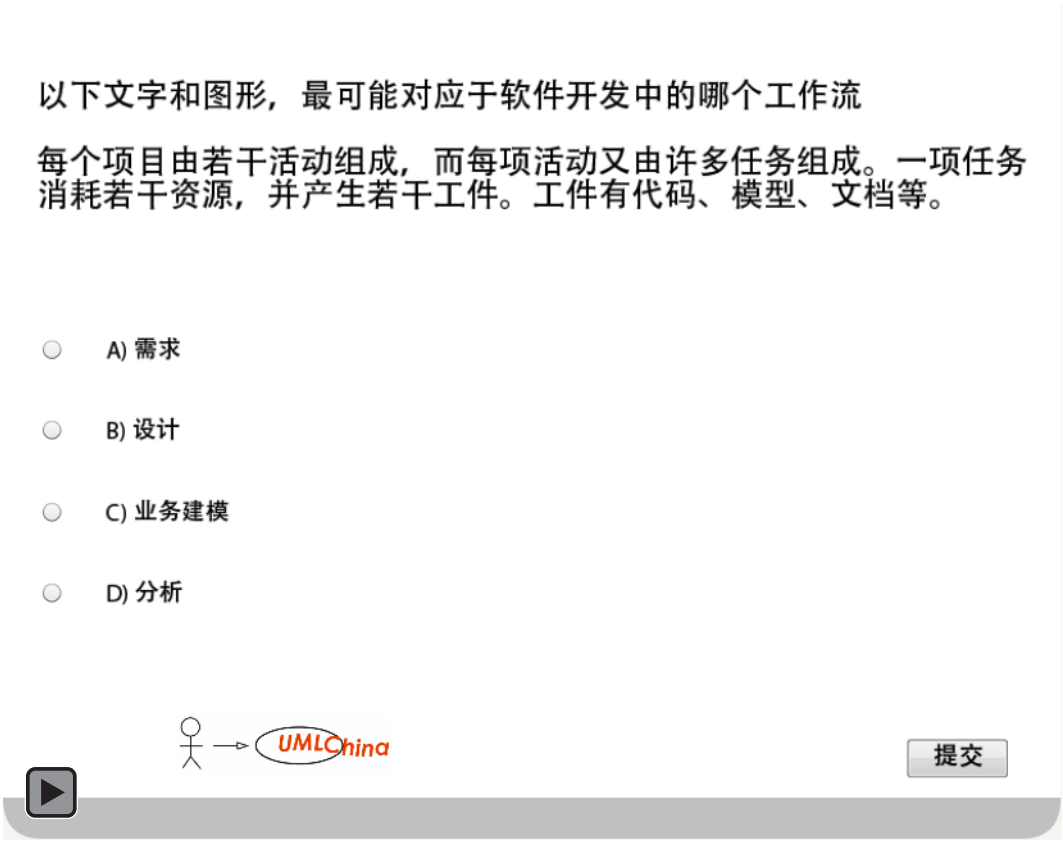


图 1-3 核心工作流思考边界

从我的观察所得，以上四项技能，开发团队做得较好的是设计（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。很多开发团队拍脑袋编造需求，然后直扑代码，却不知“功夫在诗外”。更糟糕的是，一些开发团队以“敏捷”为名，干脆就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，也不能一个疗程把病人治好，干脆我也别花那么多心思了，先随便给病人打一针看看吧，不好再来！

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；星际争霸高手要在极短时间内完成多次操作，动作依然井然有序。在激烈竞争的年代需要快速应变，掌握技能才能真敏捷。

上面的文字我没有提到 UML。也就是说，只要您思考过、表达过上面这些问题，就是在建模，用文本，用自造的符号来表达都可以。而且我相信，每一个项目，我们都会思考和表达上面这些问题，只不过可能是无意识地、不严肃地在做，现在我们要学习有意识地做，把它做出利润来。当然，使用 UML 是目前一个不坏的选择。



UML 简史

随着市场所要求的软件规模不断增大，软件的分析设计方法一直在进化。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”，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）... 这些细微的差异通常会造成混乱，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。

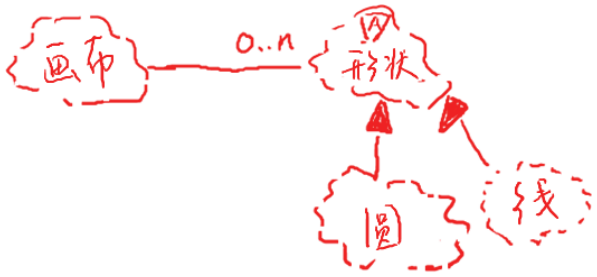
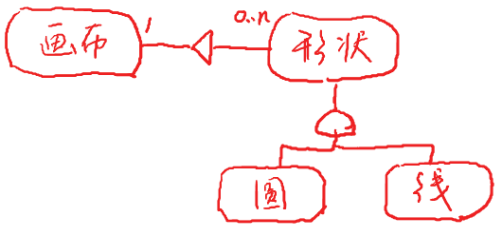
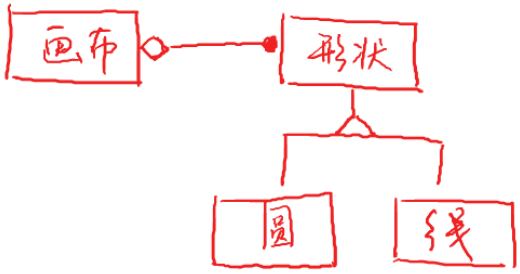
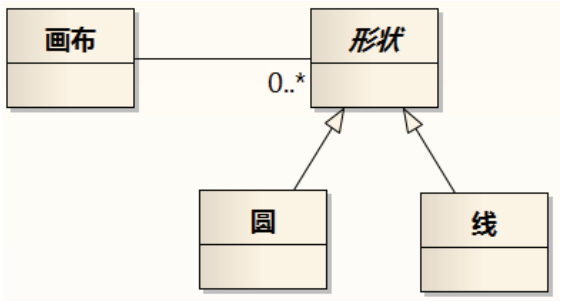
Booch	
Coad/Yourdon	
Rumbaugh OMT	
UML	

图 1-4 不同方法学图形比较

1994 年，在 Rational 工作的 James Rumbaugh 和 Grady Booch 开始合并 OMT 和 Booch 方法。随后，Ivar Jacobson 带着他的 OOSE 方法学也加入了 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，吸纳了他们的成果精华。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，并采纳为标准。从 2005 年起，UML 被 ISO 吸纳为标准，UML1.4.2 即 ISO/IEC 19501，UML2.1.2 即 ISO/IEC 19505。

UML 诞生时，Martin Fowler 就作了如下预测一：

你应该使用 UML 吗？一个字：是！旧的面向对象符号正在快速地消逝。它们还会残留在 UML 稳固前出版的书上面，但新的书、文章等等将会全部以 UML 作为符号。如果你正在使用旧的符号，你就应该在 1998 年间转换到 UML。如果你正要开始使用建模符号，你就该直接学习 UML。

——Martin Fowler 著，easehawking 译，面向对象分析和设计技术，《非程序员》第 5 期，2001。英文原文在网络上已搜索不到。

时间过去十多年了，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象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 多种，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。

参考链接

UML 新闻：http://www.umlchina.com/News/News.htm

UML 工具大全：http://www.umlchina.com/Tools/Newindex1.htm

各工作流中的 UML

UML 现在的版本是 2.4，包含的图形如图 1-6 所示，一共 14 种（泛化树上的叶结点）。

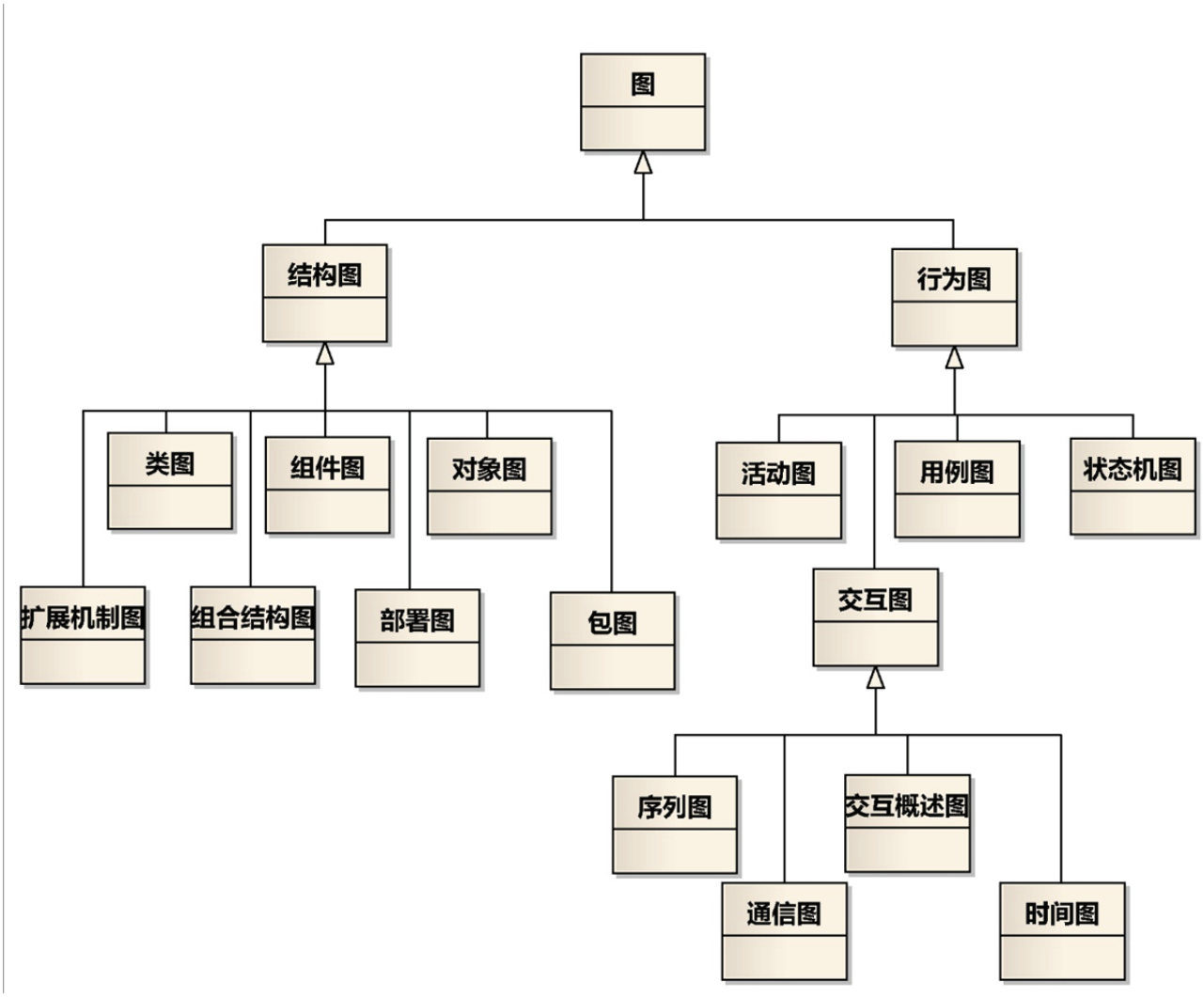


图 1-5 UML2.4 图形，根据 UML2.4 规范重新绘制

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。您只需要从这个工具箱中选择对您的项目类型合适的工具用上就可以，并不需要“完整地”使用 UML。不只是 UML，对编程语言也一样的。很多人说“我用 Java”，其实只是用 Java 的一小部分，而且很长时间内只用这一小部分就够了。

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。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的建模能力的目的而提供的，不可当真。

各工作流推荐的 UML 用法如图 1-7 所示。

工作流	思考焦点	推荐 UML 元素
业务建模	组织内系统之间	用例图、类图、序列图（或活动图）
需求	系统边界	用例图、文档
分析	系统内核心域	类图、序列图、状态图
设计	系统内各域之间	不画，代码即设计

图 1-6 推荐的 UML 用法

特定类型的项目，可以按需要添加图形，这一点后文再描述。

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！”。这样的做法有一个巨大的“优点”——因为怎么画都是对的，关于这个草图的解释权归“我”所有，同事也不好批评我，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。这一点，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。

但是，这样的做法更像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就可以允许自己胡乱使用符号和概念；过去的作家没有电脑，不意味着作家可以随意写错别字犯语法错误。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。

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 背后隐含的是对于软件建模的一些基本共识。这些符号幼儿园的小朋友都会画，但背后的共识需要一定的训练和学习才能掌握。在基本共识上沟通，效率会高得多，无效的低水平争论也会少得多，背后的脓包也会强制性露出来。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。

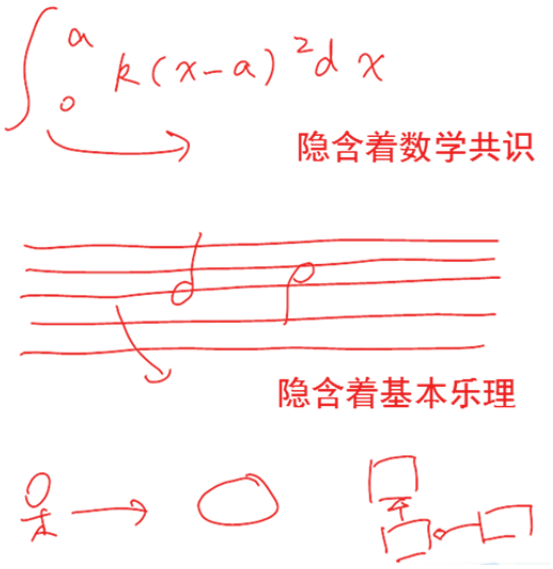


图 1-5 符号背后的基本共识

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，答案只有两三种，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。

方法和过程

本书讲的是方法（技能），不强调具体某一种过程。拿足球打比方，本书讨论的是射门、运球、传接、抢截、定位球、配合的技能，不讨论战术、更不讨论更衣室团结、俱乐部运营和俱乐部文化。

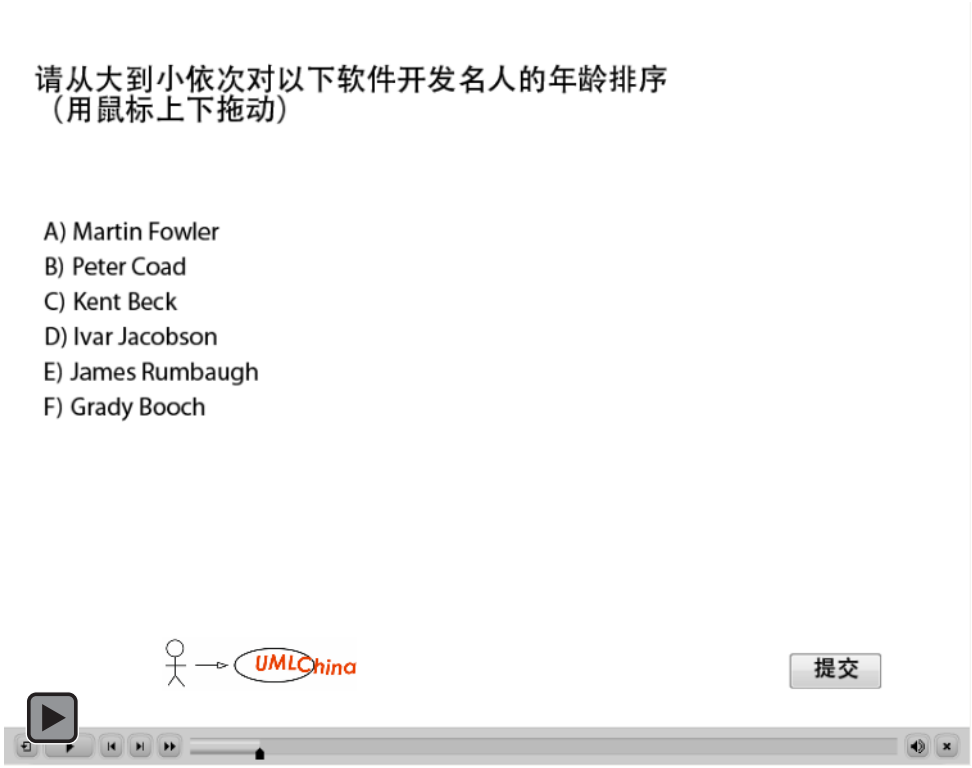
团队实施过程改进容易流于形式，根源也就在于技能的不足。如果把改进的焦点先放在技能上，开发人员技能提升了，适用什么样的过程自然就浮出水面，没有必要去生搬硬套某过程。或者说，技能增强了，更能适应不同的过程。

很多时候方法和过程经常被混淆，现在经常说“敏捷方法”，其实“敏捷”是过程（家族）。之所以造成这个误解，也许和 Martin Fowler 介绍敏捷过程家族的文章起名“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。

我刚开始为开发团队提供服务时，有一次和一个开发团队的经理交流，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。

类似的场景就是开发团队负责人说“我们现在采用的是敏捷过程”，稍为深入了解一下，其实经理所说的“敏捷过程”就是没有过程。

没有方法不等于面向过程方法，没有过程不等于敏捷过程。面向过程是成熟的方法学，真正的敏捷过程也是很严肃的过程。不要让“面向过程”、“敏捷”成为偷懒的庇护所。



----------您可以到 http://www.umlchina.com/book/panjiayu.htm 查看是否有新的版本----------