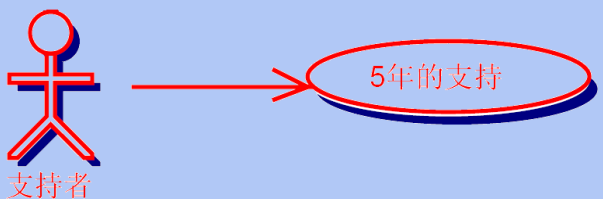
第 43 期（2004 年 11 月）

James Rumbaugh

X-Programmer 非程序员

软 件 以 用 为 本



您的默默支持，使UMLChina走过了第一个五年

下一个五年，UMLChina需要您的继续支持

联系：think@umlchina.com

http://www.umlchina.com/

聆听 James Rumbaugh 的演讲

[2004/11/11]

今天参加了IBM2004 开发者大会，上午聆听了James Rumbaugh的演讲“模型驱动架构”。演讲现场笔者自己录制的一些录音下载（现场效果很好，录音质量一般是设备关系）。

（本图摘自网易）

下午，协助《程序员》杂志的记者 ljw，myan，一起采访了 James Rumbaugh，访谈实录将刊登在 2004/12 期《程序员》杂志。

（think 文）

Borland 的 ALM 产品 2004 年赞誉如潮

[2004/11/11]

软件开发优化（Software Delivery Optimization）方面平台无关解决方案的领袖公司 Borland 软件公司 (Nasdaq:BORL) ，宣布 2004 年是其产品标志性的一年，这一年中 Borland 的解决方案获得了来自全球的潮水般的赞誉和各种奖项。锐意革新的评价将激励 Borland 在 2005 年以更大的动力前进，继续保持技术领先、关注顾客价值。

同样引人注目的是 Borland 的软件开发优化策略，它使得软件开发从一个不可预测的状态变为一个更加可管理的可重复的业务过程。该策略首先公布于 2004 年的 Borland 的大会，建立于应用生命周期（Application Lifecycle Management，ALM）领域，协同了公司在业务和 IT 之间的业务过程和管理能力。第一个发布的版本代号为“Themis”（西弥斯，希腊神话中司法律与正义的女神），它将整合 Borland 的各种 ALM 解决方案。

Borland 软件产品部副总裁 Boz Elloy 指出，“我们给自己树立了高标准的要求，为我们的顾客提供高质量的软件解决方案和服务，尽可能地做到创造性和高效的生产率。这些奖项显示了我们在满足或者超越这些标准上的努力，但它们更多地是送给我们的顾客的礼物。荣耀属于他们，正是他们支持和反馈使得我们年复一年地创新前进，赢得荣誉。”

2004 年，Borland 被众多刊物和杂志评价为技术革新、经验丰富、产品策略得当的公司。Borland 的产品覆盖了应用生命周期的各个种类，从个人到团队开发、从建模到测试。得到的荣誉也跨越了多种平台和语言。这一切背后，正是 Borland 对顾客选择的承诺。

Java 奖项

——10 月 JavaPro 现场大会上 Java Pro 杂志授予 Borland Optimizeit(TM)套件读者选择奖“Best Java Optimization/Profiling Tool”。Borland 产品也进入了"Best IDE" 、"Best Java Development Suite" 、"Best Java Modeling Tool"、"Best Java Testing Tool"、"Best Team Development Tool"、"Best Mobile Development Tool"、"Best Web Services Development Toolkit"和"Best Java Data Access Tool"等奖项的决赛阶段。

——SD 杂志授予 Borland JBuilder(R)10 月份读者选择奖"Most Robust Tool"第一名。

——Java Developer's 杂志授予 Borland JBuilder 读者选择奖"Best Java IDE Environment"，授予 Together(R) ControlCenter(TM) 读者选择奖"Best Java Application"。

——PC Magazine 最近授予 JBuilder 2005 开发环境方面编辑评选奖，并评为 4.5 星（满分为 5 星）。今年初，JBuilder 赢得该杂志的"Best of Development Tools"类 2003 年度头名，并指出 JBuilder 世界级的 EJB 开发能力以及对开放标准的支持。

——Programmer's Paradise 评选 Borland JBuilder Enterprise 为销售最佳的单一语言 IDE。

——中国信息产业部、中国软件协会和软件世界杂志的评选中，Borland Enterprise 以其对 J2 EE 平台、CORBA 以及 Web Services 标准的支持赢得中间件类别的软件金奖。

——VSJ (UK)年度读者评选中，读者投票评选 JBuilder 为"Best Java Tool or Environment 2004"。

——JBuilder, Together 和 Optimizeit 最近赢得 Java Magazine 读者评选的多个奖项。JBuilder 赢得"Best IDE"、Together 赢得"Best UML-Modeling Tool"、Optimizeit 赢得"Best Java Profiling Tool"。

Microsoft .NET 奖项

——Together Edition for Microsoft Visual Studio .NET 被 MSD2D.com 的第三届评选中被用户评选为最佳"Modeling Tool"。

——Borland Delphi(R)同样赢得了 MSD2D 的"Best IDE/Editor"奖项。

——本年度 TechEd 欧洲会议上，Delphi 赢得开发工具类"Best of Show"奖项。

——Delphi 也被 Web Services Journal 的"Best GUI for a Web Services Product"读者评选中进入决赛阶段。Delphi 以其符合用户直觉的接口和行业标准 Web services 的快速生成闻名。

——最后，今年初 Visual Studio Magazine 授予 Borland C#Builder（现在被纳入 Borland Delphi 2005 中了）"Best Development Tool"奖项。

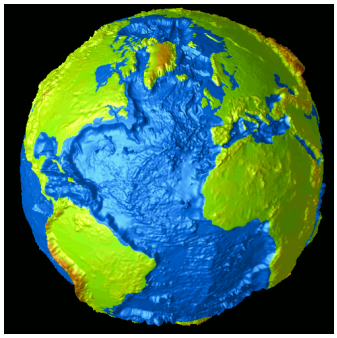
（UMLChina 袁峰 摘译，不得转载用于商业用途）

IBM 掀起 Atlantic 大潮

[2004/11/1]

IBM 公司在上个月揭开了其开发平台下一版本的面纱，增强了 Rational 及其它工具的相互协同以及在 Eclipse 3.0 框架下运行的方便性。

IBM 承诺今年年底发布代号为 Atlantic 的开发平台。该开发平台基于 Eclipse3.0，并包含了为项目管理层、架构师、开发人员、测试人员以及其它管理者等各种角色量身定做的工具集，这里面也包括了所有希望了解业务关键（business-critical）的开发项目的进行情况的各种人员。



IBM 的软件开发平台和微软公司 5 月份宣布的 VSTS(Visual Studio Team System)类似，类似的计划还包括 Borland 软件公司 9 月宣布的 Themis。IBM 希望年底可以发布，而 Borland 和微软计划都是在 2005 年发布。“这将是我们在四五年里最重要的一次发布，”IBM Rational 的主管 Mike Devlin 说，“我们正在转换并简化开发，我们的工作是前无古人的”。

该平台将支持“业务驱动的开发”，Devlin 说，“我们说到应用生命周期管理时，都是开始于业务，然后直到部署和操作”，Devlin 提到，Atlantic 将完成 IBM 自 2002 年收购 Rational 依赖的规划。

IBM 软件开发平台包括：Portfolio Manager，项目管理者和业务生产线执行人员执行任务并监控项目开发中方方面面的工具；Software Architect，基于 UML2.0 的高端建模以及工具集，还包括应用开发、Web 开发、软件配置管理等工具；Software Modeler，提供基本的 UML2.0 建模能力；Manual Tester，QA 专业人员和其它人员用来在开发生命周期的各阶段进行测试的工具。另外还包括：IBM Rational Application Developer for WebSphere（过去的 WebSphere Studio Application Developer）以及 IBM Rational Web Developer for WebSphere Software（WebSphere Studio Site Developer）。除了新的名字之外，这些工具中都增加了对 Eclipse3.0 的支持，以及减少人工编码的各种新功能。

IBM 号称这四个工具都是新的，但 Devlin 承认 Software Architect、Software Modeler 和 Manual Tester 都是基于已有的各种建模、应用开发和软件配置管理工具。

Architect 标价 US$5,500/用户，它面向那些希望得到 UML2.0 支持的用户，其中还包括了 Application Developer、Web Developer、Software Modeler 以及 IBM 的配置管理工具 Rational ClearQuest 和 ClearCase。

Modeler 标价 US$1,795/用户，适合于那些建模的新手，其中包括 UML2.0 的基本的一些图如类图和顺序图的支持。“太多的技术会把这些用户淹没的”，Devlin 提到，一些开发团队并不需要所有的建模能力。

Manual Tester 标价 US$1,500/用户，从 Rational 的测试工具发展而来，包括 Functional Tester，也包括在 Eclipse 测试项目中开发的 Hyades 中的一些功能。Portfolio Manager 基于 IBM 计划从 Systemcorp ALG 中收购的技术，这家公司是位于加拿大蒙特利尔的一家私营公司，该产品的标价现在还没有给出。

IBM 还宣称将继续支持并加强其现有的建模产品 Rational Rose 和 Rational XDE。

（自 sdtimes，UMLChina 袁峰 摘译，不得转载用于商业用途）

征稿

http://www.umlchina.com/xprogrammer/xprogrammer.htm

Accelerated 为使用 UML 的嵌入式开发人员提供应用开发平台

[2004/10/26]

由 IBM 率先提出的 Eclipse 开放源码工具又有新的进展，与此同时，OMG（Object Management Group）的 UML（Unified Modeling Language）2.0 也有新的突破。

Mentor Graphic (Nasdaq:MENT) 的分公司 Accelerated 科技(R)宣布，他们已经将广泛应用的可执行、可转换的 UML（xtUML）和他们的扩展的原型开发产品集成起来，为嵌入式软件开发人员提供生产力上飞跃式的进步。

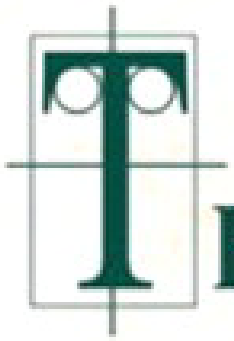
UML 建模套件 The Nucleus(R) BridgePoint(R)和 the Nucleus SIMdx 原型环境一起为嵌入式应用的开发提供硬件无关的完整平台。现在，开发人员可以在开发机器上开发完整的应用并图形化地检验了，开发平台的实用性得到了魔术般的提高。

更高抽象级别的建模在嵌入式世界的应用使得开发人员可以更多地关注于应用，而不是其所部署的硬件。XtUML 语言使得系统模型可以在设计时进行执行和检验，然后被翻译为 C 或者 C++的代码，直接编译到某个嵌入式系统上。通过在 the Nucleus SIMdx 原型环境中引入翻译后的代码，我们可以得到新层次的系统原型。

the Nucleus SIMdx 原型环境中支持嵌入式开发人员搭建完全的硬件平台在主机上（host-based）的模拟版本，包括外设、用户接口、软件协议以及 IO。虚拟的硬件平台可以在模拟硬件上运行由模型得到的代码（译者注：文中都称为是 modeled code），并可以进行完全的系统或者制订模块/构件的测试，而无需硬件的支持。一旦模型和原型平台搭建完毕，他们就可以部署为一个可重用的应用开发平台，供所有的软件开发人员使用。这降低了对硬件原型的依赖，同时帮助全球的软件开发团队可以在同样的平台上编写软件。

Mentor Graphics 的嵌入式系统分部市场负责人 Robert Day 指出，“Accelerated 科技在嵌入式市场是唯一提供真正的建模解决方案的，另外，还提供 the Nucleus SIMdx 原型产品。通过整合这些技术，嵌入式开发人员现在可以无需访问硬件，而可以检验他们的嵌入式系统，缩短开发周期、更块地向市场发布产品。”

（自 rednova，UMLChina 袁峰 摘译，不得转载用于商业用途）

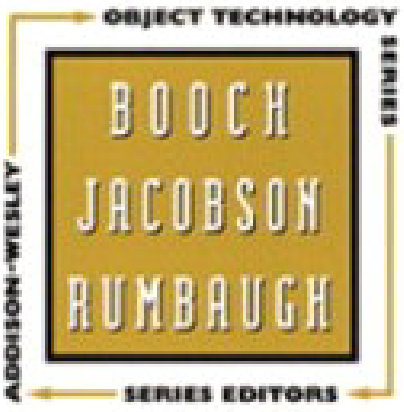


THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

Covers UML 2.0

UMLChina 译
（王海鹏、李嘉兴）



CD-ROM included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社出版

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于
领域模型没有捕获到
业务的深层内涵

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

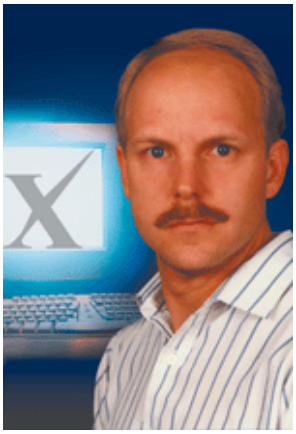
机械工业出版社即将出版

Scott Ambler

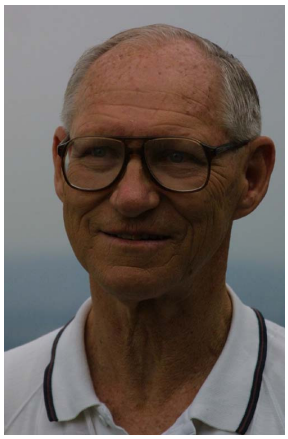
通过纪律达到敏捷——一场辩论

Kent Beck、Barry Boehm 著，米全喜 译

讨论



VS



Kent

有些人把软件开发中的敏捷看作是一种权衡，一种类似对飞机翼梁的切削。放弃了安全、严格或责任，我们可以进行得更快。然而，从极限编程的本质来看，“敏捷或安全”或者是“敏捷或纪律”并不是对立的。从极限编程的本质来看，获得我们所寻求的结果的唯一途径是将世界看作是“两者同时”而不是“非此即彼”。为什么要将它们分开呢？

我的字典（www.m-w.com）中是这样定义纪律的：3. 一个研究领域，4. 改正、影响或提高思想能力或道德品质的训练，5. (a)通过强制或命令获得的控制，(b)安排或规定好的行为或行为方式，（c）自我控制；6. 控制行为或活动的规则或系统的规则。

让我们在原型极限编程小组环境中看依次看一下这些定义。

*一个研究领域。*极限编程描述了在成为软件开发团队中高效成员的道路上，系统开发人员或客户应该掌握的技巧。

> **对于每个人来说，只有通过更好的纪律（Discipline）才能达到敏捷（Agility）。**
>
> **—Kent Beck**

*改正、影响或提高思想能力或道德品质的训练。*我不会声称道德品质，但是测试驱动的开发、重构、结对编程、不断地集成和每周的计划（以及对计划的估计与跟踪）都构成了“完美”的脑力活动。

*通过强制或命令获得的控制。*这个描述不适合极限编程。小组对它自己的规则负责。事实上，XP 的第一条规则就是“它们只是规则而已。”这个规则不是任意行为的借口，而是提醒小组要保持它自己的规则，并根据积累的经验不断地优化它们。

*安排或规定好的行为或行为方式。*小组成员知道相互间期待什么，所以小组的行为是有秩序的。然而，制订规则的人同时也是执行规则的人。

*控制行为或活动的规则或系统的规则。*极限编程当然是这样。XP 小组知道他们计划、开发、集成和部署的规则。尽可能多地将规则嵌入到工具中。例如，有一条规则是说：如果没有通过所有测试，就不能发布代码。这条规则可以嵌入到脚本中，自动运行测试，仅当所有测试都通过时才发布变更。

如果把极限编程说成是没有纪律，它仅仅在“通过强制或命令获得的控制”这方面来说是对的。如果使用了纪律的这个狭窄定义，我要对没有纪律的进程感到高兴。强迫开发人员和客户按照对结果不负直接责任的人制订的过程来工作，无一例外都失败了。然而，如果给出一个更广义的规则，极限编程比大多数的过程都更讲纪律，提供了一个更清晰的总画面，告诉人们期望什么活动，以及更多的机会学习。确实，如果不遵循“纪律”的正面想法，极限编程的社会契约将会迅速解体。

—Kent Beck

Barry

从前，在一个叫“隐喻”的岛上，住着一只猴子和一只大象。他们都住在一条宽阔、水流湍急的河水的一边。河水两岸长满了果树。猴子很敏捷。他能够爬到果树的顶部，尽情地吃水果。大象很高。他能够用躯干够着果实并尽情享受。



但是果树越长越高。很快大象就够不着果实，也就吃不饱了。但是他很强壮，能够自给自足。他发现当他饥饿的时候，他可以把树扳倒，吃果实。

猴子看到大象扳倒了大部分果树。他很不高兴。他对大象说：“不要那样做！我可以爬上树帮你取得果实。”

大象说：“我饿了。我很强壮。我自己能做到。”

猴子说：“你这愚蠢的大象。很快就会没有树也没有果实的。”

大象说：我一次只解决一个问题。事情可能变化。也许会有更多的果实出现。或者果树可能变矮。如果不是这样的话，我可以再想出一个解决方法来。”

猴子只能同意他。他也是一次只解决一个问题。

很快地，在他们居住的河岸就没有果树了。大象说：“我有办法。我将过到河对岸，把那边的树放倒。”

猴子说：“你这愚蠢的大象。那样做在河这边行不通，在河那边也行不通。你会让我们俩都挨饿的。让我们决出个胜负。我敏捷，我比你快得多。

大象说：“没问题。我魁梧、强壮，我会压扁你。”

> **你不能通过舍弃“纪律”的部分定义来扩展它的定义。**
>
> —Barry Boehm

于是他们开始战斗。猴子比大象快得多。但他无法阻止大象对他的攻击。

大象试图用他强壮的躯干和四肢攻击猴子，但是猴子太敏捷了，大象每次都落空。

天气很热。猴子和大象很快就累了。他们坐下来，决定下一步做些什么。

猴子说：“我敏捷。我将跃过河水带回一些果实。”

他跑向河水，快速移动，但是河水开始将它冲走。

“那个愚蠢的猴子！”大象说。他走入河水中，捞起了猴子，把它放到背上，并回到岸上。在猴子晾干身体的时候，他们坐下来想了很长时间。

最后猴子说：“可能这是一个办法。你可以把我背到河对岸。等过了河，我可以爬到树上，摘下足够咱们俩吃的食物。

大象想了一下，回答说：“听起来这个方法值得一试。”于是他们进行了尝试，效果不错。

此后他们生活得非常快乐，直到终老。

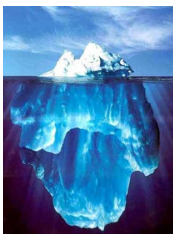
—Barry Boehm

这个故事的一些寓意：

- 猴子能够完成大象无法完成的事情。
- 大象能够完成猴子无法完成的事情。
- 一次解决一个问题可能不是一个好主意。
- 决出胜负可能不是一个好主意。
- 寻求一个合作的双赢方案可能是一个好主意。

Kent Beck 对 Barry Boehm 的回应

我从 Barry Boehm 的故事中获得的最主要的寓意是：他将他称为“面向计划的”方法与我称为“面向现实的”方法看作是竞争关系。我恐怕不能那样看待问题。既然他引入了故事的形式，我也以类似的方式回应，虽然我不敢奢望与两个聪明的生物学习如何相处的故事想媲美。

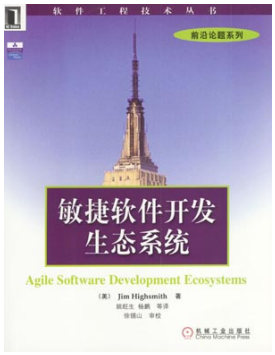

从前有一只恐龙和一个小的、有点类似于老鼠的哺乳动物。冰川时代来临了。恐龙不能调节它自己的温度，死掉了。而老鼠有更适应环境的新陈代谢，生存了下来。

恐龙和老鼠不是竞争的。他们是两个不同的生物，吃不同的食物，生活在不同的地方。当然，恐龙死去会给许多小生物带来更多的生存机会，但是与老鼠没有什么关系。如果不是老鼠，也可能是一只蟑螂。

敏捷主义者如我们看到的那样对商业、技术和社会环境做出反应。如果您处的气候与 Barry Boehm 做出建议的气候一致，没有问题，你可以遵循它。但如果变得冷起来，就要想一种其他可能更有效的方法了。

Barry Boehm 对 Kent Beck 的回应

Kent Beck 认为敏捷与纪律不是对立的，这是对的。他还说XP需要大量纪律，这也是对的。他认识到大项目可能需要与XP中不同的纪律，这同样正确。在《解析极限编程》一书中（Addison-Wesley, 2000, 第157页），Kent说：“很显然，项目的大小很重要。你可能无法让上百个程序员运行XP项目。也不能让五十个，或是二十个。十个就够多了。”



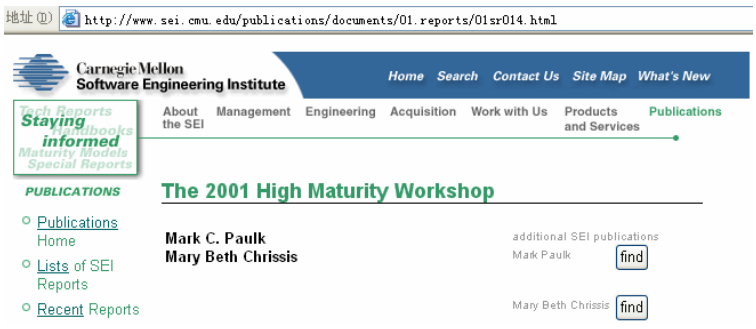
正如 Jim Highsmith 在《敏捷软件开发生态系统》一书（Addison-Wesley, 2002, 第358页）中指出的那样，世界上大约60%的软件项目是不超过10个人。但是用项目百分比乘以项目大小来确定完成的相对工作量，这些小项目只占了全世界软件工作的大约17%。

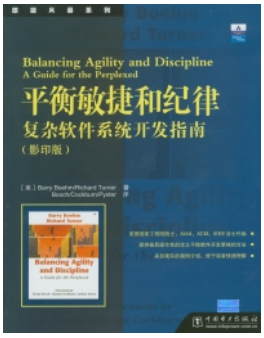
其他83%的工作是如何使用纪律的？一个好的例子是Amr Elssamadisy和Gregory Schalliol在《极端编程中“坏味道”的发现与响应》（Proc. ICSE 2002, pp. 617-622）一文中描述的ThoughtWorks租赁管理案例研究。为了使XP在一个50人的项目中成功地工作，他们发现这样做是非常重要的：“……坚持在开发与数据更新中包括更传统的图示与图表来显示整个应用程序……”、“创建一个简明的必须完成的任务列表……，并严格、诚实地监督它，”使用一个工厂模式而不是对可预见的新功能的简单设计。

这听起来象是“通过强制或命令获得的控制”吗？当然。

我不知道 Kent 从哪里获得数据来支持他的“强迫开发人员和客户按照对结果不负直接责任的人制订的过程来工作，无一例外都失败了。”的观点。美国国防部要求合同商达到CMM3才能完成合同就是一样例子，而它产生不少成功的故事。关于数据，可以参见 SEI Report CMU/SEI-2001-SR-014, The 2001 High Maturity Workshop” (www.sei.cmu.edu/publica-tions)，和我将要和Richard Turner出版的新书：平衡敏捷与纪律：疑问指南(Addison-Wesley, 2003)（译者注：**Balancing Agility and Discipline: A Guide for the Perplexed**，电力出版社影印出版了此书，书名为《平衡敏捷和纪律：复杂软件系统开发方法指南》，在 www.china-pub.com上看了一下书评，有一位读者认

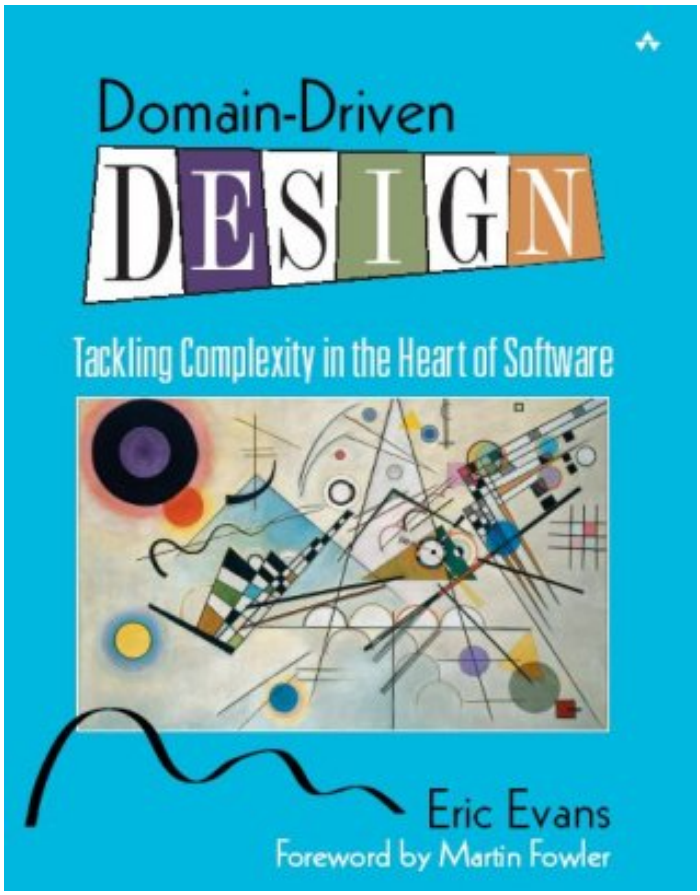
为此书名的翻译不准确，我在书店里翻了一下这本书，也觉得翻译欠妥。在文章中暂译为《平衡敏捷与纪律：疑问指南》)。





但最需要的不是在言辞上“分出高下，”而是将敏捷与计划驱动方法中最好的部分综合起来以应对未来的挑战，以同时达到软件的高度可依赖性、敏捷性和可调节性。

这又把我们带回到了猴子和大象的故事中。



众多问题根源在于：领域模型没有捕获深层业务内涵！

《领域驱动设计》中译本即将由清华大学出版社出版，UMLChina 辅助教材

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

探索需求

设计前的质量

Donald C. Gause
Gerald M. Weinberg / 著
章柏幸 王媛媛 谢攀 / 译

Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

清华大学出版社

业务建模 vs 系统建模

think 著

业务建模可以看作软件开发之前的第零步工作流程。随着统一过程被越来越多的团队所采纳，基于用例和对象技术的业务建模成为团队所关注的重要技术。以下是笔者被问到过的一些有关业务建模的问题。把它们归纳出来，加上我自己的理解，希望对大家能有帮助。

业务 vs 系统

业务：指某个组织或者组织单元。

系统：软件系统。

严格来说，**【业务】**也可以看作一种包含了人、机器、资源的**【系统】**，所以也可以使用一些软件思想（用例思想、对象思想）来描述它，这就是业务建模。

业务建模并不意味着必须有软件开发项目跟随。业务建模是要想办法描绘出当前的现实业务。描绘现实业务的目的通常是为了改进业务，使之更具有竞争力，达到这种改进的可选解决方案包括重构业务流程，包括引进软件系统——我们软件开发人员所指的【系统】。

提问：我要做业务建模，却不知道应该如何考虑建模的“业务”范围。是对整个公司建模呢，还是对里面的一个部门？

回答：

如何选定业务建模时**业务**的范围，就要看改进组织的任务情况。如果任务涉及改进整个组织，那么把“业务”定为整个组织，把业务边界定在该组织的周围是合适的。如果任务只涉及改进某个部门，那么把“业务”定为该部门，把业务边界定在该部门的周围是合适的，这个时候，同一公司的其他部门就成为业务执行者。如果这种改进任务是一个软件项目，简单地想想软件项目的名称——“××企业信息管理系统”“××人力资源管理系统”——也基本可以判断出来它们涉及的业务范围。

这里要注意的，上面提到的业务边界指的是责任的边界，不是物理的边界。同一办公地点的人也许分属不同业务，同一业务的人可能分散在不同地方。这个边界的概念同样适用于系统边界。

提问：如果我要开发一个mp3播放器软件，需要业务建模吗？怎么业务建模？

回答：

1.业务建模并非软件项目的必要环节。对软件开发来说，业务建模作为一个辅助环节，能起到帮助理清软件需求的作用，例如，当你要为一个大的组织引进软件系统来改进业务，你一开始对业务流程如何，其中哪些环节需要优化、能够而且值得引进软件系统来优化，并不是很清楚。这个时候，业务建模就有必要了。业务流程环节越多，涉及的人或组织越多，业务建模就越有必要。如果涉及的人少，业务流程环节也少，就可以不需要业务建模。例如，mp3播放器这样的软件。

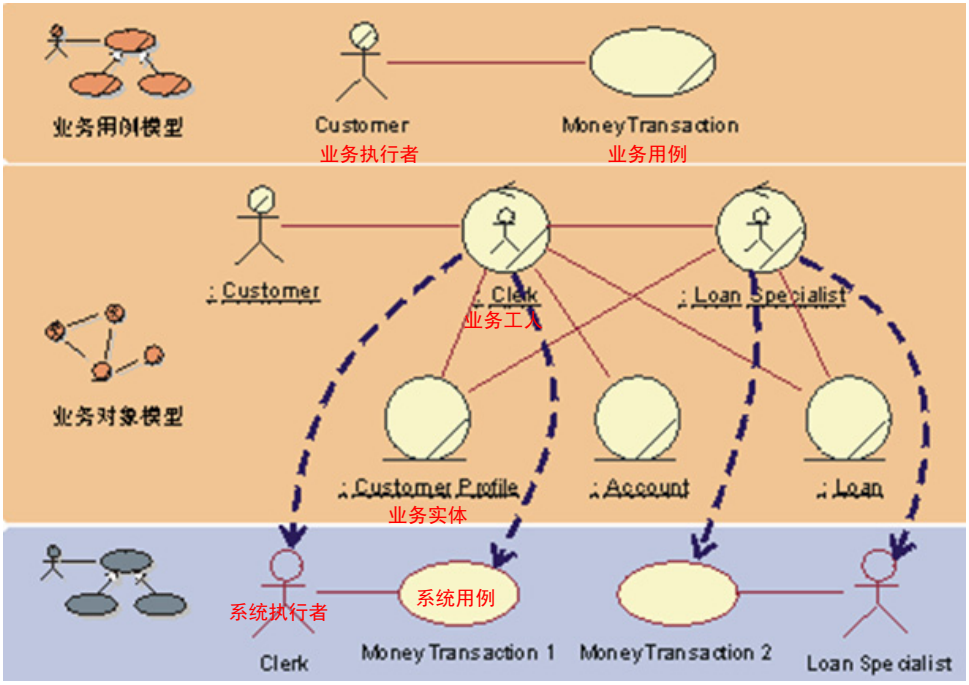


图1 业务建模帮助理清软件需求

2.其实对于这种非特定客户的产品研发，业务建模也是很重要的。在市场经济下，如何定位自己的产品十分重要。业务建模有助于描述出“战场”的真实态势，帮助你从中寻找软件的切入点。以“mp3播放器”为例，如果把考虑问题的边界放宽一点，来个业务建模，也许就会发现，业务在mp3音乐到达听者的耳朵之前就已经开始了。

mp3是通过什么渠道到达播放器的？——切入点：在播放一首mp3的时候，能了解这首歌的来处（盗版大本营、P2P大本营...）吗？如果歌曲从专辑中摘出来，专辑里还有哪些曲目呢？

用户是怎样用现有的播放器听mp3的？——切入点：mp3播放器能在播放操作上、收听效果上有什么改进？只能听mp3一种格式吗？

听腻了以后会怎样？——切入点：哪首歌在听众的系统里停留的最长——亦即他们曾经最喜爱的歌是哪些？

业务执行者 vs 系统执行者

业务执行者：在业务外和业务交互的人或组织。

系统执行者：在系统外和系统交互的人或事物。

问题：医院里的医生是业务执行者吗？

回答：有关业务执行者的识别，最常见的错误认为业务执行者就是在组织里干业务活的人，这样就会误把业务工人（Business Worker）当作业务执行者（Business Actor），例如以医院为研究对象，可能会得出下图的业务执行者：

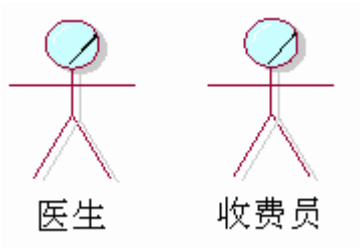


图2 错误的业务执行者

这里有“不甘心”的因素在里面，医生这么重要，不给一个业务执行者怎么说得过去？破除这种思想，就是要认清业务工人的本质，业务工人就是“打工仔”，医院付给他们工资，他们干活，而干活的目的是为了向外面的业务执行者（如病人）提供价值（治好病）。换句话说，如果有一所超级医院，不用医生，病人只需从前门进去后门出来，所有病痛一扫而光，医院老板还会需要招收那么多医生吗？

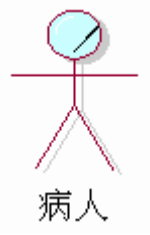


图3 正确的业务执行者

医生、挂号员、收费员、药房是业务工人，它们会在详述业务用例时出现。

提问：业务执行者一定是系统执行者吗？

回答：

如果业务执行者需要直接与要开发的软件系统交互，它就会成为系统执行者。但通常更多的系统执行者来自业务工人，系统把业务工人的工作计算机化。从这里，我们也可以进一步破除“执行者迷信”：他这么重要，怎么不是执行者？省省吧，系统执行者就是“打工仔”变来的，就是整天摸键盘弄鼠标的人，你觉得重要人物会那么喜欢摸弄键盘鼠标吗？恐怕更喜欢摸高尔夫球杆吧？

提问：系统执行者只包括和系统直接交互的人，那些不直接交互的重要人物怎么办？不会被漏掉吗？象下面的例子：

旅行者来到订票点，告诉订票点工作人员买什么航班，工作人员使用机票预订系统出票。如果用例建模成：

图4

这样会不会忽略了旅行者的利益，难道工作人员不是在执行旅行者的指令吗？

回答：

恰恰相反，这个用例图恰好揭示出了这个简单的道理，在这个事件中，系统的直接目的是为工作人员提供价值，改善他帮助旅行者订票的工作。我们做一个假想，如果该订票点计算机系统崩溃，不能出票。在市场经济下，会发生什么情况呢？一种情况是工作人员手工出票、或者打电话让人送票、或者自己跑去可以出票地方的拿票。

另一种情况是，旅行者用脚投票，到另外的点去买，从此不再光顾此点。

旅行者呢？他不重要吗？当然，十分重要，他是一类涉众。工作人员在帮助旅行者订票的时候，好多双眼睛盯着看呢。

旅行者——担心出错票，更担心拿到了错票自己还不察觉；担心票是打出来了，信息却没有传到后头；担心没那个航班的票…

订票点老板——担心工作人员玩忽职守，出错影响声誉；担心工作人员匿客户的钱，出假票，然后拿钱开溜…

工作人员——担心操作太复杂，一天下来手指麻木；担心不好用，出错票被老板扣工资…

机场——担心出错造成纠纷…

航空公司——担心……..

用例必须在它的路径、步骤、数据需求、业务规则、非功能需求、设计约束……中考虑体现这些涉众的利益。

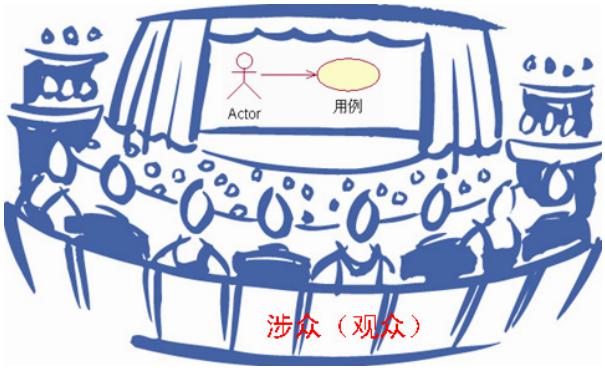


图5

关于用例，Ivar Jacobson的定义是：用例实例是在系统中执行的一系列动作，这些动作生成执行者可见的价值结果。一个用例定义一组用例实例。Alistair Cockburn进一步发展了用例思想，正如《编写有效用例》开篇所言，用例是什么？用例是涉众之间就系统行为达成的契约。笔者尝试把它补充一下，作为用例的进阶定义：**用例是涉众之间就系统行为达成的契约，以执行者为达成特定目标和系统交互的方式演绎。**笔者认为：只有在涉众契约这个层次上理解用例，才能发挥出用例的最大价值。有关用例的历史，不妨看看《程序员》2004年5月的《用例：十年风雨》。

随web等技术的出现，软件系统为业务外的人提供了更多的接口，这个时候业务执行者也会演变成系统执行者。整个业务流程也改变了。如果系统提供网上订票的价值，旅行者就是系统执行者。用例图变为：

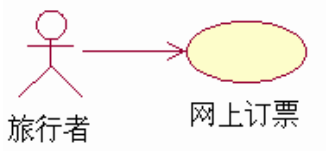


图6

注意：这个时候执行者改变了，用例的步骤也变了，订票点老板、工作人员不再成为涉众，但旅行者、机场、航空公司等涉众的利益却没有变。

业务用例 vs 系统用例

业务用例——在业务中执行的一系列动作，这些动作为业务执行者产生可见的价值结果。通俗一点，就是业务执行者和业务打交道是为了达到什么目的？

系统用例——在系统中执行的一系列动作，这些动作为系统执行者产生可见的价值结果。通俗一点，就是系统执行者和系统打交道是为了达到什么目的？

提问：

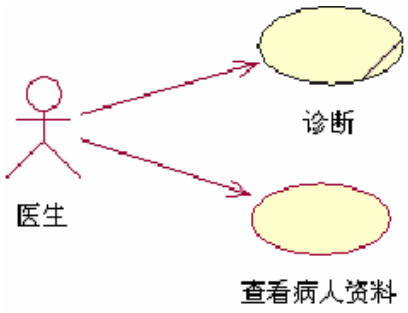


图7 “诊断”变成了业务用例

一个医院管理系统。医生通过系统来查看病人资料，所以“查看病人资料”是系统用例。医生诊断不是通过系统来完成，所以“诊断”是业务用例。是吗？

回答：

这是对业务用例的最常见误解。认为现实中要做但没有计算机化的事情就是业务用例。

业务用例就是使用用例的观点来研究业务，把业务内部的各种流程看成是为业务执行者提供价值的方式。这种思路对改进业务流程有非常大的帮助：先归纳出业务对外提供什么价值，再思考如何更好地优化内部流程来实现这些价值。Ivar Jacobson在这方面做出了主要的贡献，可以参看他的著作《软件复用》

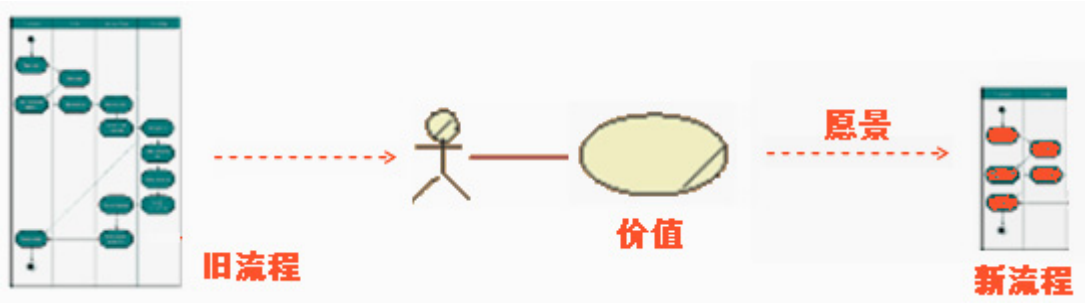


图8 重新构造业务流程

“医生→诊断”只是业务用例“病人→求诊”里的一个步骤，不是业务用例。

提问：公交公司运营调度工作中的一项重要工作是制订运营计划，由调度员负责，计划哪条线路配备多少台车，配备多少司机，每天安排多少个运行班次，制订每班车的发车时间和发车间隔。从上面的说法看调度员是业务工人不是业务执行者，业务用例是业务对外提供价值。到底“调度员→制订运营计划”这个工作属于哪个业务用例呢？“乘客→坐车”？不象。

回答：

思路是：思考调度员为什么要“制订运营计划”，如果他领着薪水却偷懒不“制订运营计划”，他的直接上司可能会炒他；那么他的上司为什么要炒他呢？因为他需要这个，没有这个计划他就会作不了一些事情，那么他为什么需要调度员给他这个计划呢，是谁让他做什么事，需要这个计划的帮助呢－－－一直问到业务之外为止－－很可能是“公司拥有者：提高公交线路效率”。

提问：从业务用例能确定系统用例吗？

回答：

这个问题实际上是问：业务建模的结果能自动映射到软件需求吗？答案是不能。业务模型和系统模型之间确实可能存在以下的角色变换：

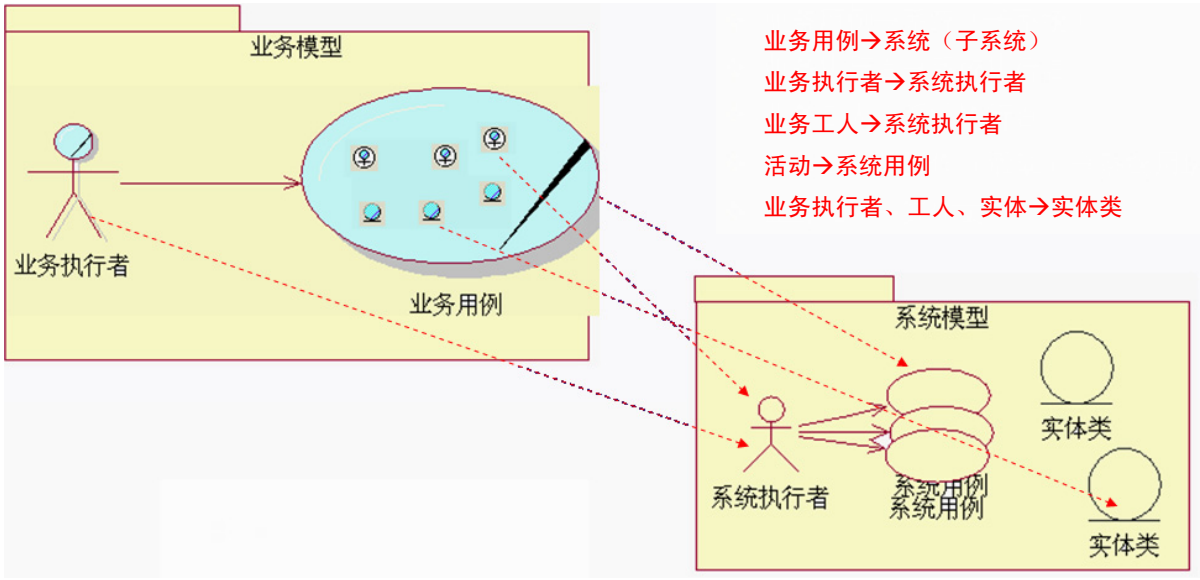


图9 可能的角色映射（注意图中，一个业务用例映射了一个系统或子系统，认识到这一点，也可以进一步帮助我们理解业务用例的概念。）

业务建模是为了描述出业务现实（恰恰这一点常被视而不见，开发人员很容易在详述业务用例时有意无意地臆想系统已开发出来之后的场面，得到一个“假”的业务流程。这样的模型已经失去了它的价值。只有描绘出真实的现实，加上真实的愿景，才有可能得到真实的软件需求）。就像侦探柯南所说的，“真相只有一个”。现实摆在那里，但为改善现实而开发的软件却可以千差万别，其中的差别就在于系统的愿景（Vision）。了解了“现实是什么”（业务模型），还需要了解“我希望引进系统来帮我达到什么，我的钱包有多鼓…”（愿景），才能导出需求。所以，光从业务模型不能确定系统需求。

业务建模是为了描述出业务现实（恰恰这一点常被视而不见，开发人员很容易在详述业务用例时有意无意地臆想系统已开发出来之后的场面，得到一个“假”的业务流程。这样的模型已经失去了它的价值。只有描绘出真实的现实，加上真实的愿景，才有可能得到真实的软件需求）。就像侦探柯南所说的，“真相只有一个”。现实摆在那里，但为改善现实而开发的软件却可以千差万别，其中的差别就在于系统的愿景（Vision）。了解了“现实是什么”（业务模型），还需要了解“我希望引进系统来帮我达到什么，我的钱包有多鼓…”（愿景），才能导出需求。所以，光从业务模型不能确定系统需求。

一个业务用例可能会导出多个系统用例：“病人→求诊”的业务用例中的各个步骤如果需要、值得而且有钱来把它们计算机化，就可能有：

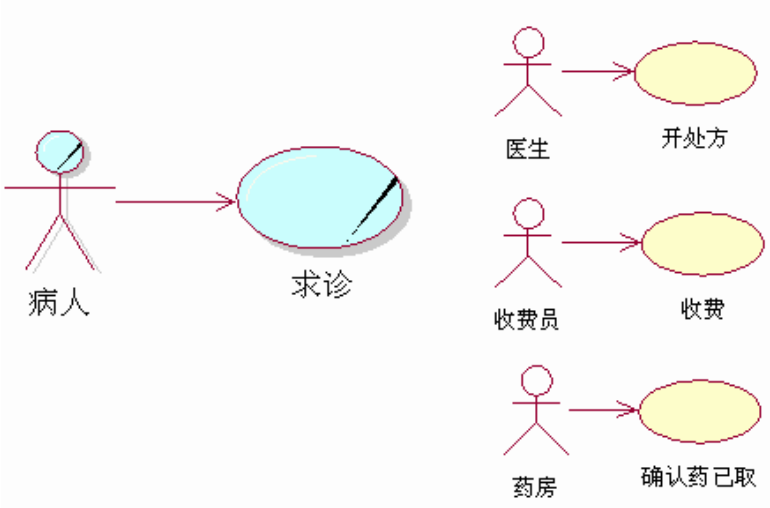


图10 一个业务用例，多个系统用例

很多个业务用例可能导出一个系统用例：

装电话和装ADSL对用户来说是两件差别很大的事，但对营业员来说，通过“营业受理系统”受理什么业务，并没有价值上的区别，都是让他肩膀累手酸，同时带给他薪水的工作。

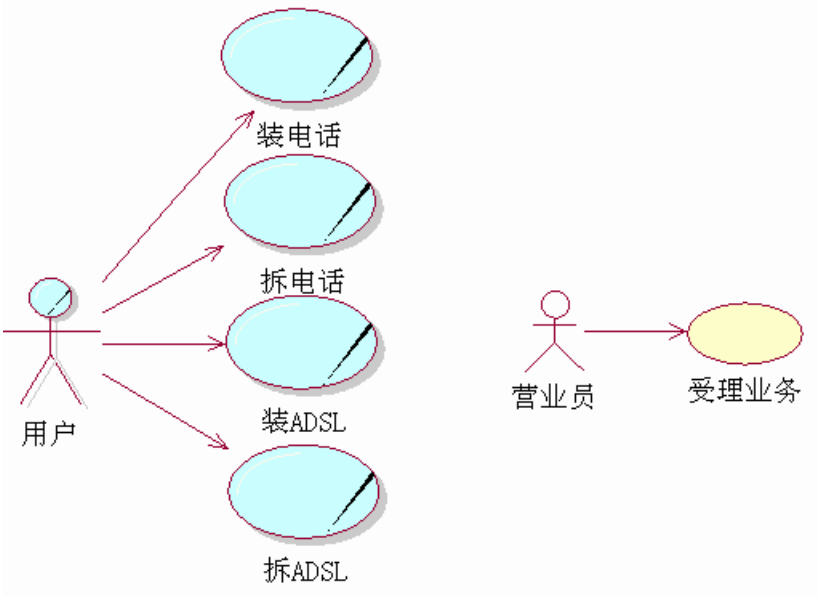


图11 多个业务用例，一个系统用例

业务对象模型 vs 系统对象模型

业务对象模型：用面向对象思想描述现实中各种业务执行者、业务工人、业务实体之间的关系

系统对象模型：用面向对象思想描述系统中各种对象之间的关系

提问：我是要拿类图去和客户交流的，那么怎样和客户交流设计呢？

回答：

不需要和客户交流设计，和客户交流的只会是需求。类图并不意味着设计。

用面向对象的思想去剖析现实业务，可以得到业务建模的另一个工件：业务对象模型。拿去和客户交流的类图扮演的就是业务对象模型的角色。许多开发人员需要和客户“交流设计”，原因只是前面的业务建模和需求没有做好，漏了许多数据需求和业务规则，不得不在设计阶段拿着设计类图去和客户交流。

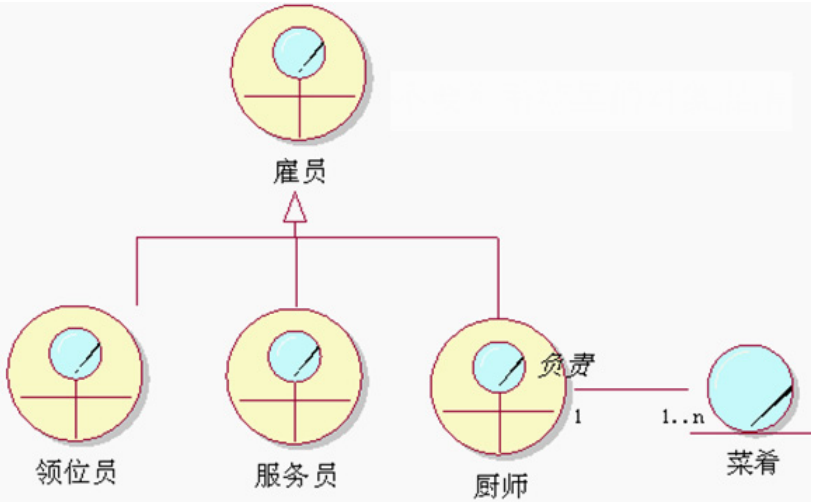


图12 业务对象模型只是勾勒出现实中的人、事物、关系，并不暗示系统中会有哪些类

业务对象模型本身并不能直接映射到系统对象模型（分析类模型、设计类模型）。业务对象模型里的类，不一定就在系统中存在；系统对象模型中的类，业务对象模型里不一定有。分析只能从需求得到，不能抛开需求来天马行空地分析，业务对象模型只是起到辅助理解的作用。甚至，面向对象的业务建模并不意味着面向对象的分析，面向对象的分析并不意味着面向对象的设计。

工件的形式和开发工作流之间并不是一一映射的：

类图可以用于业务建模（业务对象模型）、分析（分析类模型）、设计（设计类模型）

用例可以用于业务建模（业务用例）、需求（系统用例）、测试（测试用例）

顺序图可以用于业务建模（业务用例实现）、分析（分析顺序图）、设计（设计顺序图）

活动图可以用于业务建模（业务用例实现）、设计（描述算法）

本文首发于《程序员》杂志 2004 年第 7 期，经《程序员》允许转载。

征稿

http://www.umlchina.com/xprogrammer/xprogrammer.htm

软件工程技术丛书

设计系列

企业应用架构模式

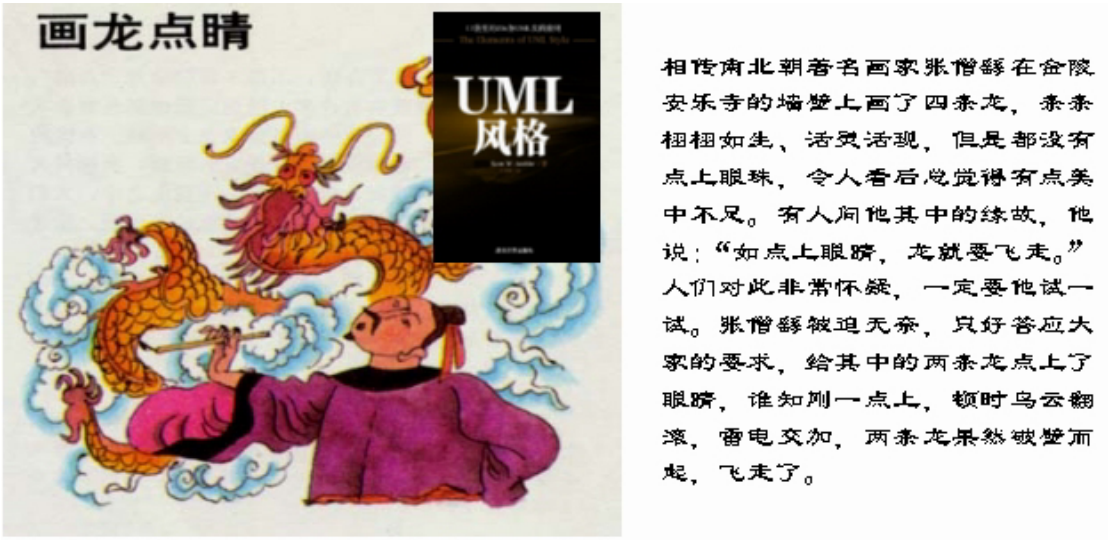
Patterns of Enterprise Application Architecture

(美) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校

UMLChina 训练辅助教材



它不讲概念，它假设读者已经懂了概念。

它不讲工具，它假设读者已经了解某种工具。

它不讲过程，它假设读者已经了解某种开发过程。

它只是在读者已经了解方法、过程和工具的基础上，提醒读者在绘制 UML 图时需要注意的一些细节。

在这本类似掌上宝小册子中，Ambler 提出了 200 多条准则，帮助读者在画龙的同时，点上龙的眼睛。

业务建模实践：使用 RUP、WBI Modeler 和 Rose/XDE

Maria Ericsson 著，李胜利 译

讨论

本文讨论了业务建模工作多方面的上下文，提供了对这些技术的全面评述。IBM 通过电子商务采用的三个阶段提供支持业务转换：功能，集成和按需计算。

当一个组织正转向变得更敏捷和按需服务时，采用工程方法连接业务与 IT 部门的需要正在增加。

如果你对理解为什么业务建模可能是有用的感兴趣，并且希望了解 IBM 的业务建模解决方案，请读下去。理想情况下，假设你已经对由 RUP 定义的业务建模的概念有基础的认识。几个关于业务建模概念的介绍性文章能在 Rational Edge 文档中找到。

本文的目的探讨业务建模工作的上下文，并且提供了对 IBM 提供的支持技术的概要说明。在进行过程中，我将谈到多种提供具有与这次讨论相关能力的 IBM 软件产品。

基本概念

包括业务工程，业务建模和业务流程管理。RUP 讨论使用如 Rose/XDE 的 UML 建模工具进行业务建模，并且讨论如何支持业务工程。WBI Modeler 与组成 IBM WebSphere Business Integration 工具集的其他工具相结合支持业务流程管理。我们依次回顾下面概念。

业务工程

业务工程是一个公司根据特定目标设计业务的技术集。业务工程可用于业务重组、业务改进和业务生成。

业务重组。这种形式的业务工程产生改变，这些改变基于你的现有业务的全面观点和它怎样及为何实现功能的彻底评估。你对所有现有的业务流程提出疑问，并且试图找到重构他们的新方法，以实现根本的改进。它的别名是业务流程重组（BPR）和过程创新。

业务改进。这种形式的业务工程产生跨越所有业务的局部的改变。它包括调整成本和交货时间，并集中监视服务和质量。

业务生成。当目标是生成一个新的业务流程、新的业务线或一个新的组织时，使用这种形式的业务工程。

业务建模

业务建模围绕你能用来可视化业务建模的所有建模技术。这些是一个你能用来执行业务工程的技术的子集。

业务流程管理(BPM)

业务流程管理是一个概念，它持续定义、分析和改进业务流程。

业务建模工作的目标

在一个竞争日益激烈的业务社会中，很多公司经常发现他们的组织架构和流程不允许他们适应足够快的变化。这些公司的许多已经决定采用支持他们的现代业务实践和 IT 技术，一种影响组织的许多方面的改变。下面的挑战是共有的：

- 业务流程对在组织内工作的人是不可见的。如果没有共同的起始基线，很难讨论改变什么。
- 一个组织内的不同部门使用相同流程的不同变体。在实践上和技术上的基准的缺乏引起混乱，并且使得评估应该改变什么变得困难。
- 技术已变成业务怎样运行的基本要素。无论是业务管理还是 IT 观礼团对都看到了比传统要更加协作的需要。不习惯认为他们的产品和提供的服务是软件的公司现在发现软件是产品如何表现的。例如，在过去的五到十年，通过提供在线基本服务，使很多金融和银行产业已经发生改变。

业务建模，指的是在组织内关于流程和结构的可视化和合理化技术，因而变得更加重要。目前，没有明显的业务建模符号标准；然而，正努力统一几个通用实践。标准化工作由BPMI（业务流程管理计划，www.bpmi.org）牵头与OMG（对象管理组织www.omg.org）合作进行。更多的标准背景，参见附录C。

所以，让我们想象在最简单的场景中，你正在进行业务建模，这里你只想理解或统一业务流程。正如图 1 所描述的那样，你应该通过可视化流程来做，然后通过执行过程及与业务需求的理解做改进。你将看到所有依赖于观众和工作集中的建模工作形式的不同级别。有些从业者只使用白板和 PowerPoint，然而其他人可能使用如 Rose/XDE 的可视化建模工具制作更正规的 UML 模型。

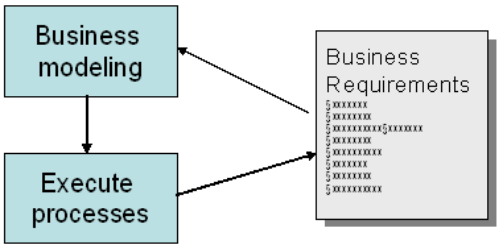


图 1 业务建模的简单场景

更复杂的场景包含与构建过程监控能力相结合的业务建模。如图 2 所示，随着过程的执行，收集数据和分析性能以便生成过程改变的业务需求。这是 IBM WebSphere Business Integration 工具集的强大之处，并且它提供的这种能力称为业务流程管理（如上面的定义）。目前用来业务建模和过程监控的符号是几年前 IBM 从 Holosfx 产品继承得到的。WBI Modeler 也使记录需要做工作流的模拟的任务的成本和时间变得容易，。

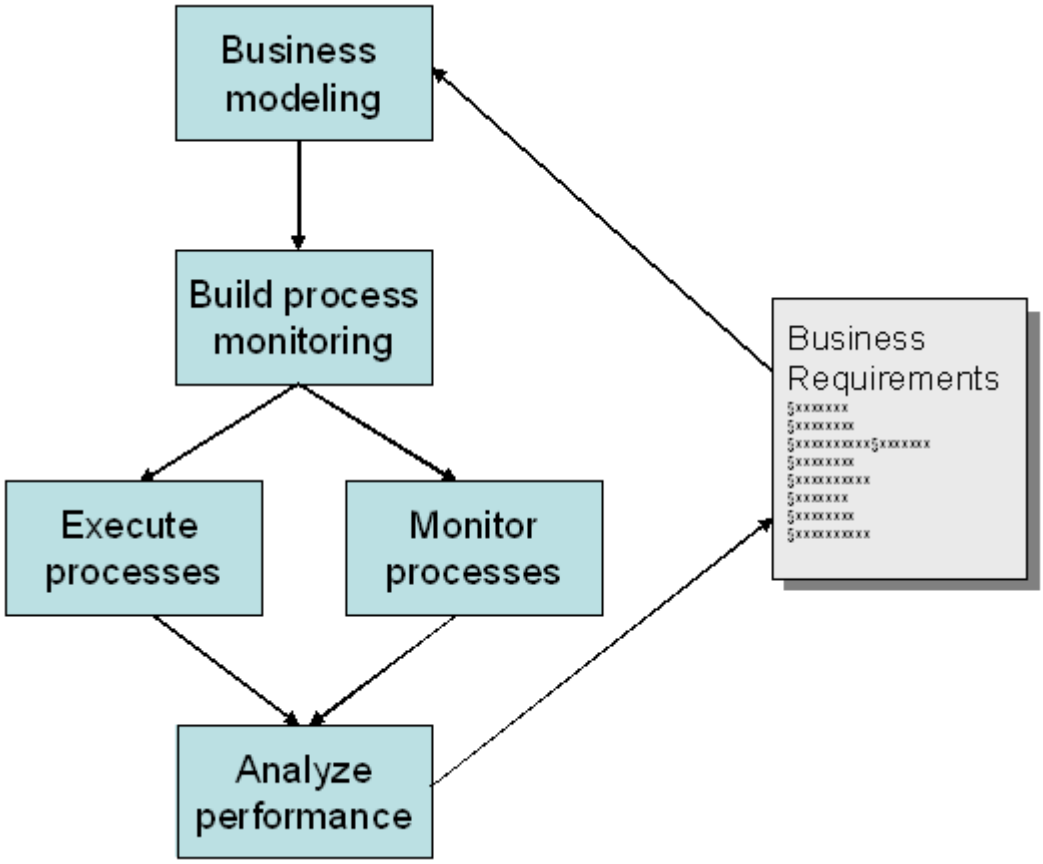


图 2 使用性能分析为业务建模工作细化业务需求

一个更加诡辩的变化联合了业务建模和过程监控，具有应用再配置能力（在业务模型中基于变化的应用的自动再配置）。变更的业务需求可能需要现有软件应用程序的功能上的改变（IS 解决方案）。正如这样，如图 3 所示，软件需求将导致应用程序运行参数的变化。再次说明，这种类型的再配置由 WebSphere Business Integration 工具集支持。

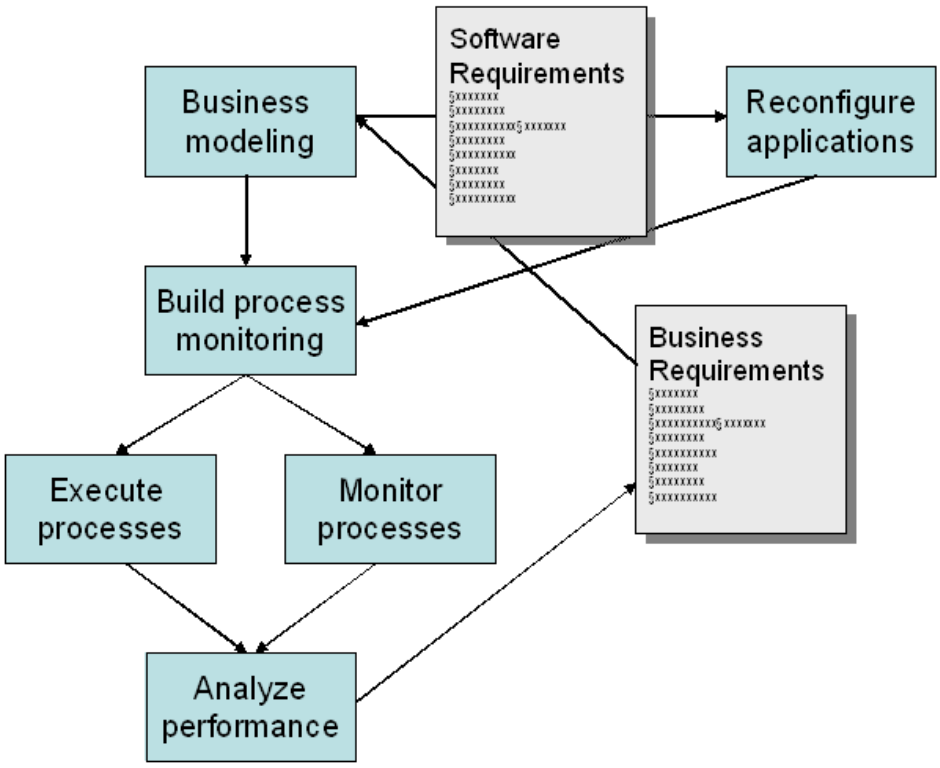


图 3 通过引进软件需求到分析流，作为业务需求匹配到实际性能的结果，应用程序能够自动再配置

最后，你也许要与应用程序开发结合作业务建模。如果你的应用中的业务需求需要新功能，或者如果业务贯穿一个自动工作，为了生成软件需求，业务建模应当输入到应用开发工作中。这就是 IBM Rational 软件开发平台的强大之处。用来表达全面流程的业务建模符号和工具集基于标准的 UML。

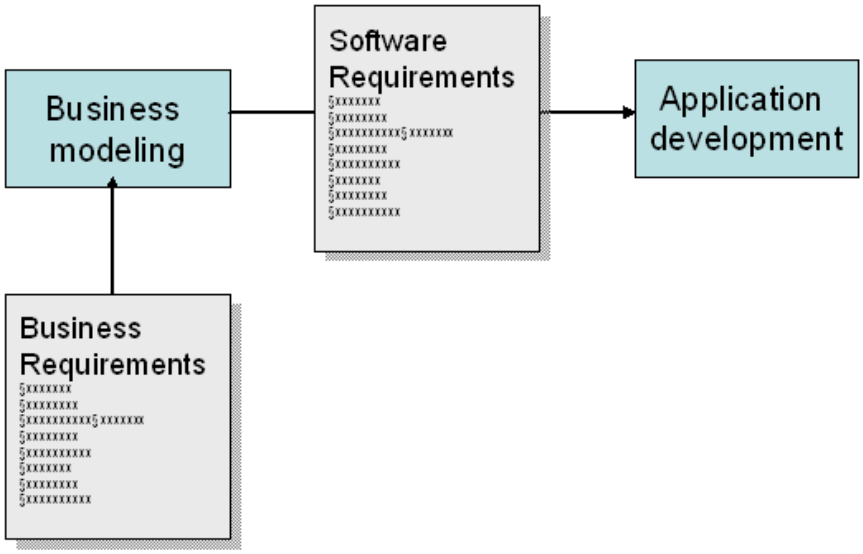


图 4 当应用开发增加到你的基本业务流程中时，业务能够剪裁 IT 系统以直接关注已知的业务需求

现在，让我们看一下如图 5 所示的大意图。IBM Software Development Platform 结合业务建模和其他能力，对业务流程监控、再配置和构建应用提供支持。

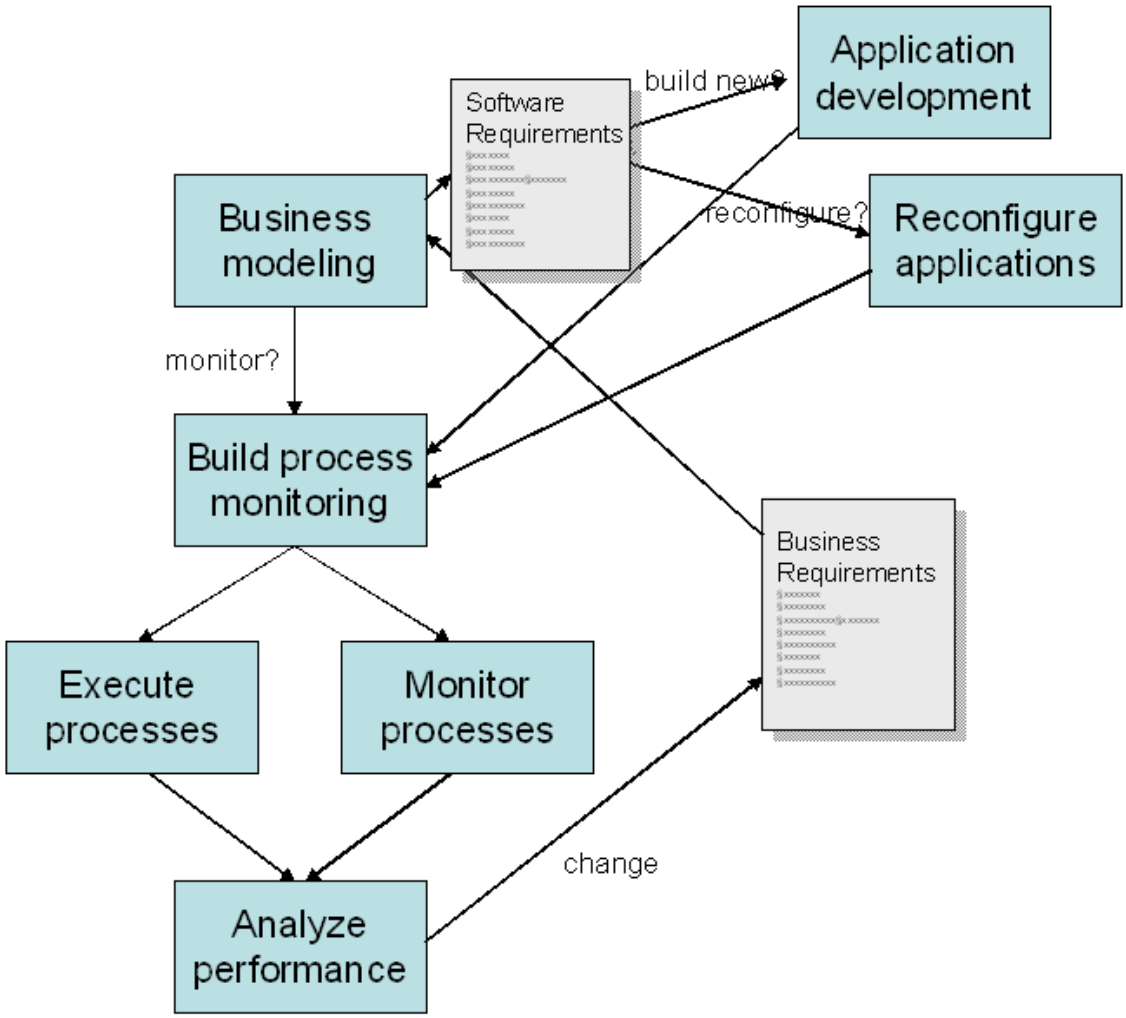


图 5 IBM Software Development Platform 的基本能力，引导业务 IT 系统改进。

你问为什么想这样做？

为了保持竞争优势，与我们合作的客户正走过图 6 中描述的电子商务采用的三个基础阶段：功能、继承、按需服务。

功能。典型的第一阶段，用来为业务中的领域和功能创建过程和 IT 支持。这是今天大多数组织所处的阶段。对业务流程和支持应用的理解趋向于特定的功能和领域。例如，国际性公司经常依赖于基于国家的开支报告和支持应用。这样做的缺点就是在特征上有一些重叠，并且在国家边界间的通信信息上可能有些困难。

集成。第二个阶段是集成特定领域的 IT 系统来消除重叠，并且使过程和组织间的信息结构一致。这应当导致维护更少的应用系统和不同边界或传统业务边界间易于通讯。然而，随着这些工作的实现，在适应功能和领域需要上，业务趋向发现更高一级的问题，并且机构感到更加官僚。例如，由于它现在影响整个组织中使用的应用，增加或修改开支报告系统以符合当地法律的变化和加班工作的制度可能突然变得相当复杂和昂贵。

按需服务。第三个阶段是以在市场上适当定位为焦点，生成连接两个世界最好方案，不带有在企业间为了反应的效率和速度进行的业务主导的流程和 IT 性能结合的能力。已经做到这点的公司非常少。如果不是为了保持竞争力他们不得不做，甚至没有人愿意去尝试它。但是越来越多的公司为了生存正努力达到这个阶段。这就是 IBM 的按需愿景的全部内容。

在上面描述的三个阶段的每一个阶段里，业务建模的要求是不同的；随着你走过这些阶段，业务工程变得日益重要。为了发布合适的技术，理解业务路程和业务架构是基本的；因而，业务建模对 IBM 的按需策略是至关重要的。

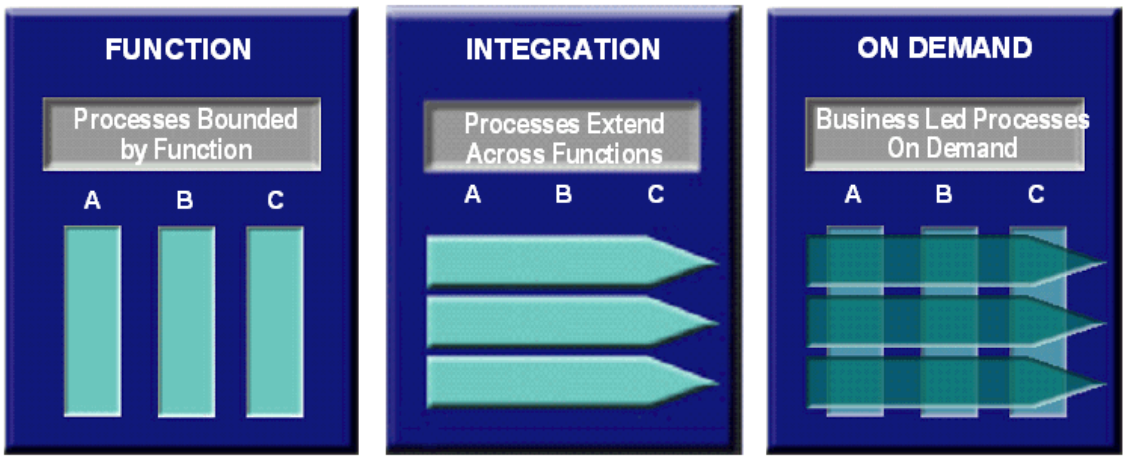


图 6 电子商务采用的三个阶段

通过电子商务采用的移动工具

目前，存在两个 IBM 产品集支持业务建模。这些产品集是相关联的，尽管他们拥有不同的目的：

- IBM Business Performance Management Framework，包括使用基于工作流专用符号的用于业务建模的 WBI Modeler。WBI Modeler 能够导入和导出 UML 模型。它也能够为 WMQ Workflow 工具导出模型信息。

- IBM Software Development Platform。包括使用 UML 和 Rational 可视化建模工具集（Rose/XDE 和 RUP）进行业务建模的方法。由于名称预示，这个工具集以软件开发者为目标用户；尽管解决方案的业务建模部分是以业务开发人员和系统分析和架构师为目标的。

如果你看到了一个完整的帮助客户在电子商务的各阶段迁移的画面，你将看到三个主要组成部分：

- 一个组成所有应用程序基础的中间件平台。在这个平台中，你可以看到像 WebSphere 应用服务器那样的企业应用服务器，也可能看到像 WebSphere 基础服务器那样用来在业务流程执行监控的工作流引擎。

- 一个应用开发工具集，像 Rational RequisitePro 那样的需求管理工具，像 Rational Rose/XDE 那样的可视化建模工具，像 WSAD 那样的构造工具。

- 一个像 WBI Modeler 或 Rose/XDE 那样的业务建模工具集，它可能结合像 WebSphere MQ Workflow 那样的工作流编排工具。

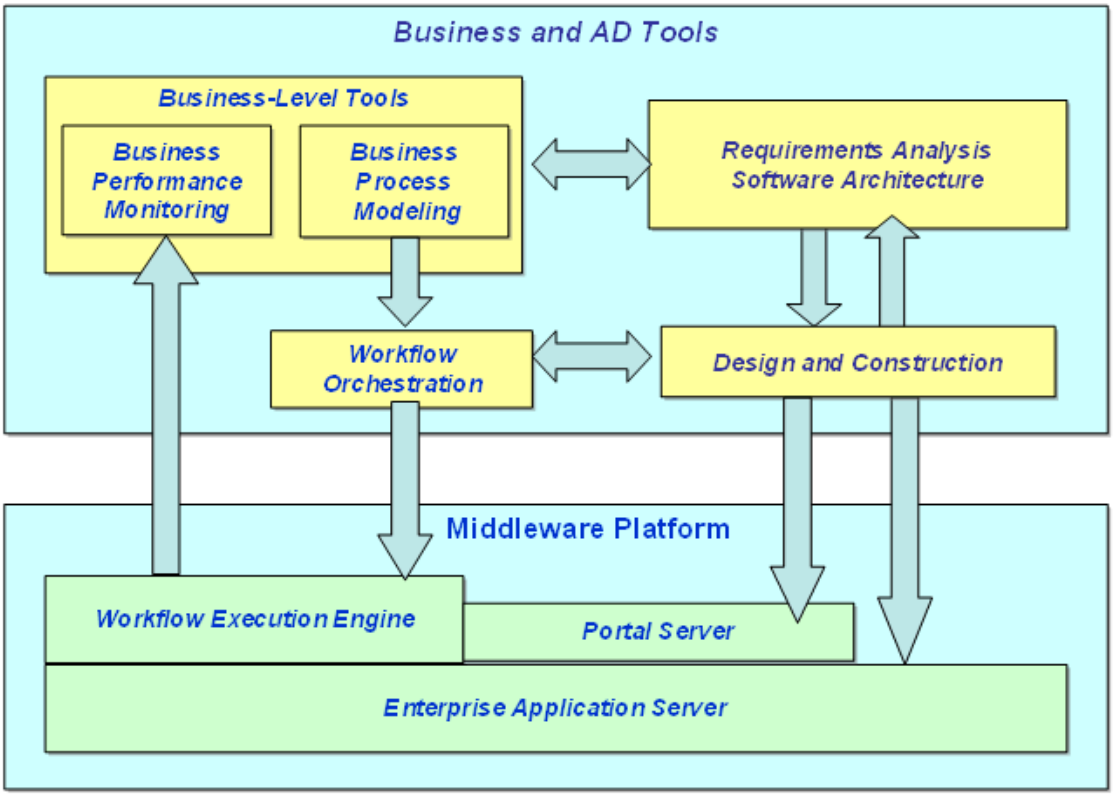


图 7 电子商务采用的工具。位于图左侧的 IBM Business Performance Management Framework 的强大之处，虽然 IBM Software Development Platform 聚焦在右侧。

流程特征的影响

要考虑的其他因素是被描述业务流程的特征。IBM Business Performance Management Framework 产品本质上借用了自身来建模和管理可重复的流程。在这些类型的流程中，如熟知的交易流程，带有较小变化的相同顺序的步骤在每次流程执行时重复。例如，这是典型的生产流程和订单处理流程。IBM Business Performance Management Framework 产品借助自身来建模和管理现实中可重复的流程。

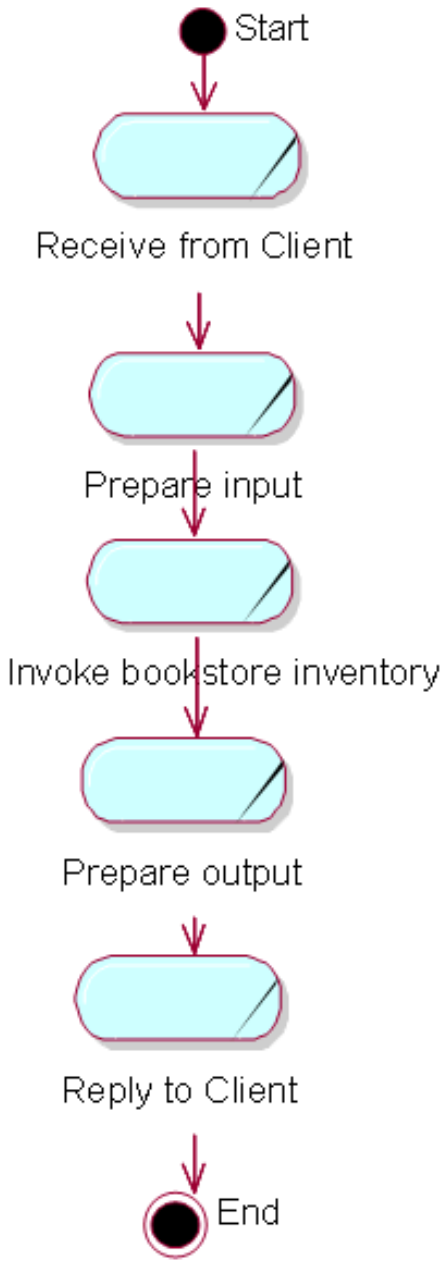


图 8 一个事务流程例子，每次流程执行得到重复的相同的步骤顺序

但是实际上也有很多现实中很难预测的流程。例如，在一个软件开发流程中，很难定义活动的可预测顺序。 在基于 UML 建模中实用的活动图和工作流图从来不是规范的，只是说明性的，在 RUP 中的工作流图也是同样的情况。对这些流程建模需要关注这种方式实现的结果，需要实现的职责，并且不是一个活动的规范的顺序。我们称这些流程为非事务性；UML（及其他事务）的优点是它支持类图和交互图以及活动/工作流图。

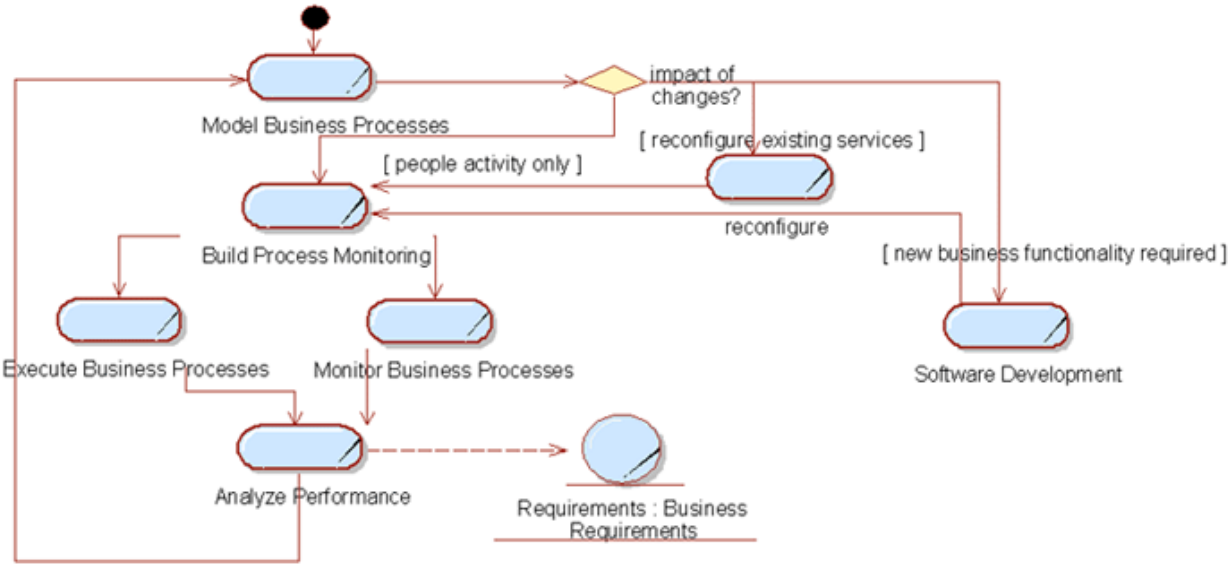


图 9 UML 活动图实例

IBM 的业务流程管理技术（IBM Business Performance Management Framework 软件）正如前面所提到的，对事务性的业务流程是最有效的。较难预测的流程的监控和管理需要业务符号的扩展集和像 IBM Rational Rose/XDE/UML 方式作为一部分提供的监控技术。IBM Rational 监控技术(IBM Rational Project Console) 集中了关注流程发布状态的项目管理和里程碑的实现，而不是已经进行的流程步骤。

对客户工作在 IBM Business Performance Management Framework 提供的和 IBM Rational RUP/Rose/XDE 提供所需的场景来说，IBM 提供在 WBI Modeler 和 IBM Rational RUP/Rose/XDE 的集成。注意这两种方式在他们的符号形式上有差异；附录 A 分别提供一个在 WBI Modeler 和 IBM Rational RUP/Rose/XDE 中转换的关键概念。这些符号有重叠，但提供如上指出的不同长处。

业务建模过程

RUP 中的业务建模训练

早期在“业务建模工作的目标”部分勾画的业务建模场景中，任何组织可能发现自身正进行多个场景。一种“普通的工作方式”渐入真实生活意味着一种如何根据使用场景的变化进行业务建模的公共框架。为了能够收获和重用经验，使用标准的术语、方法和符号是非常重要的。随着你从业务建模工作中积攒经验，你可以收集一个典型使用场景的改编框架集。这些改编应当基于你组织内的经验。例如，RUP 中的业务建模训练描述的工作流如图 10 所示。

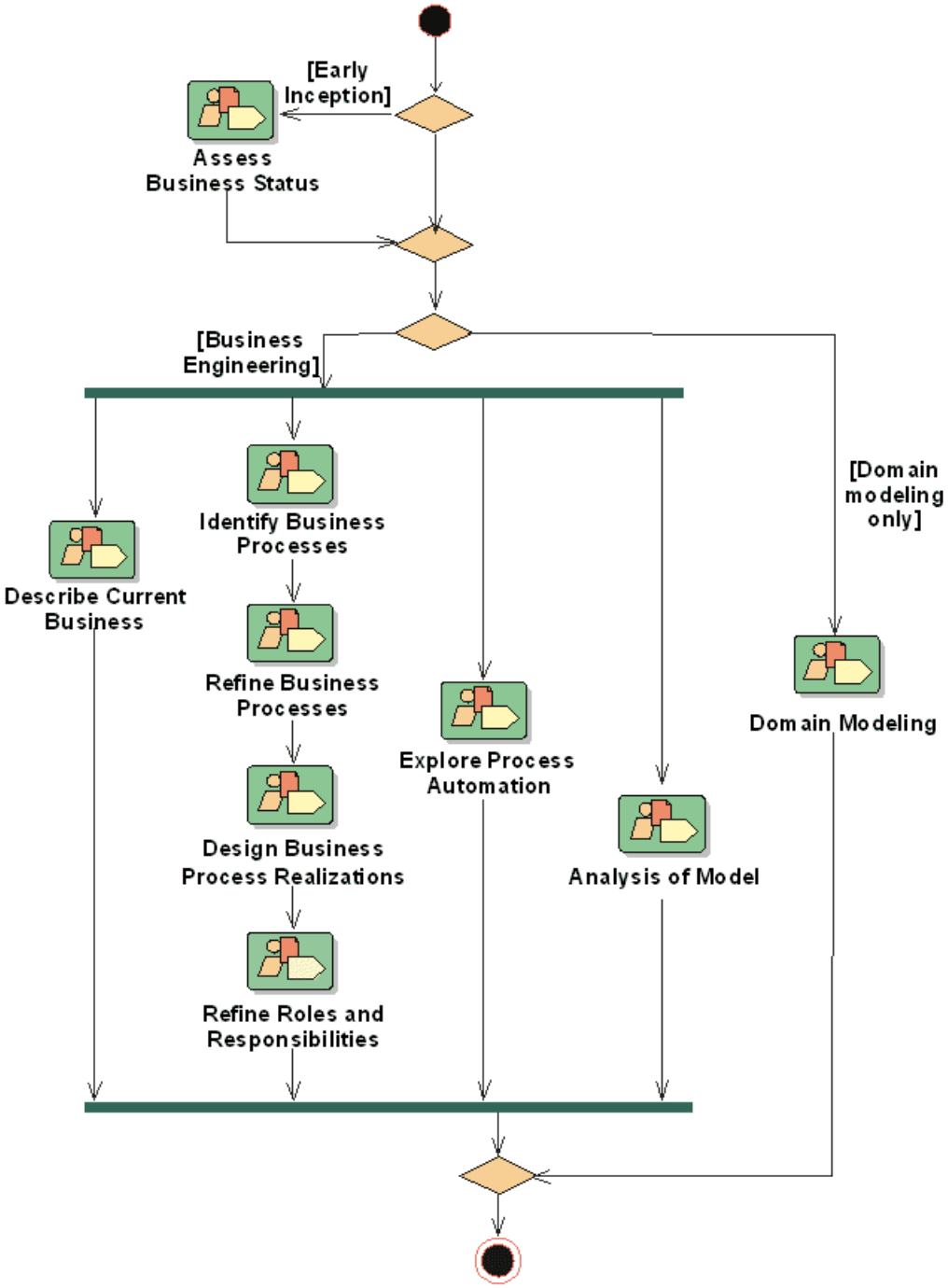


图 10 RUP 工作流图

尽管这些步骤的重点在每个场景中是不同的，但这张图能够为所有业务建模的使用场景作为一个框架服务。来自 RUP 中的原始工作流已经被“模型分析”的工作流细节扩展，来进一步强调模型如何被研究用来分析期望的行为。RUP 在如何做上提供了一些指导，然而 WBI 的附加物和它的指导拓宽了这种指导。

让我们考虑在图 10 中显示的与建模相关的工作流细节的子集（对业务建模工作流的全部细节，你应当参考 RUP 本身）。

描述当前业务。这一步的主要益处是对当前业务上下文管理的理解，业务需要中的首要洞察力，和建模工作的范围的定义。能够通过 Rose/XDE 生成的图形的例子是如图 11 所示的上下文关系图（类图的一种变化）。

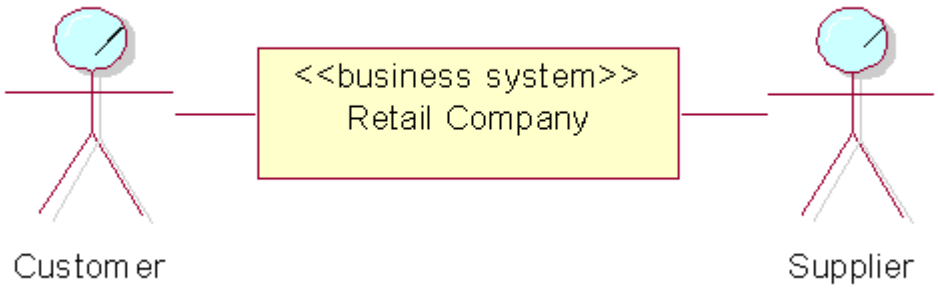
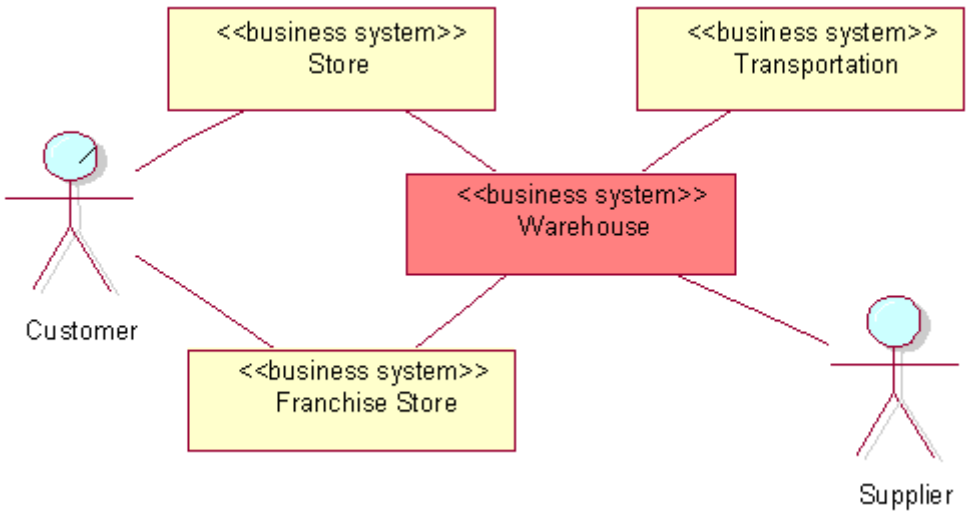


图 11 一个简单的上下文图实例

在一些情况下，你也许发现业务建模工作只与部分业务相关，在这些情况下，一个更详细的关系图可能生成



如图 12 所示。

图 12 更复杂的上下文图实例

识别业务流程. 这一步的主要益处是与当前流程相比，流程区域需要变化的理解。这通过生成显示高层流程的总图来实现，如业务用例和业务角色。这里你构建的（在“细化业务流程定义”工作流中你也许要继续进行的）是 RUP 提供的，作为业务用例模型。使用 Rose/XDE 生成的这种模型的例子如图 13 所示。

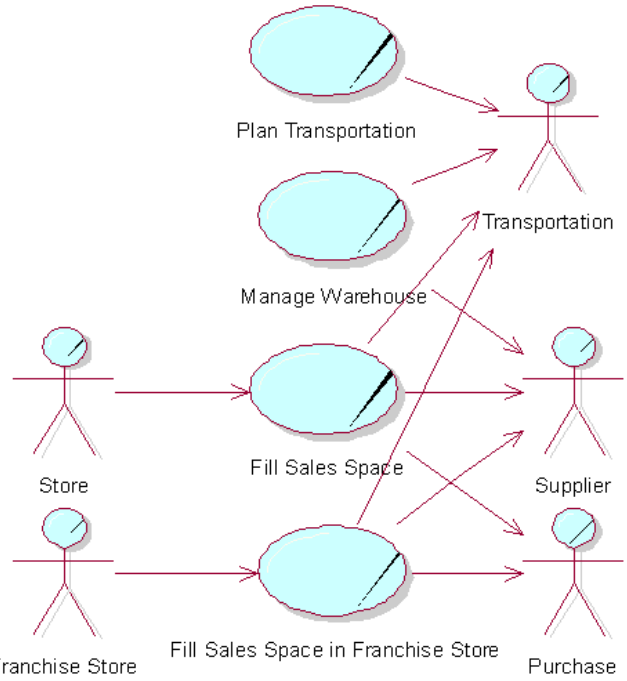


图 13 业务用例模型实例

细化业务流程定义。这一步的益处是一个更详细的业务流程定义，但是对一些组织外的人员它仍然是可理解的。这是从一个外部观点描述的流程是什么，不包含例如信息结构或涉及角色的内部细节。典型地，一般通过文本文档来完成，尽管一些也使用简单的活动图描述业务流程流。

设计业务流程实现。这一步的益处是角色如何协同工作来完成流程的描述，和用到、管理和生成哪些信息对象。这就是从哪里进入流程细节和开始创建 RUP 所指的业务分析模型。这一步的通常输出是一个或多个根据你的上下文关系，使用 WBI Modeler 或 Rose/XDE 生成的活动/工作流图。例如，如果流程是高度事务性的，并且可用工作流引擎实现，你最好选择 WBI Modeler。这个结果的例子如图 14 所示。

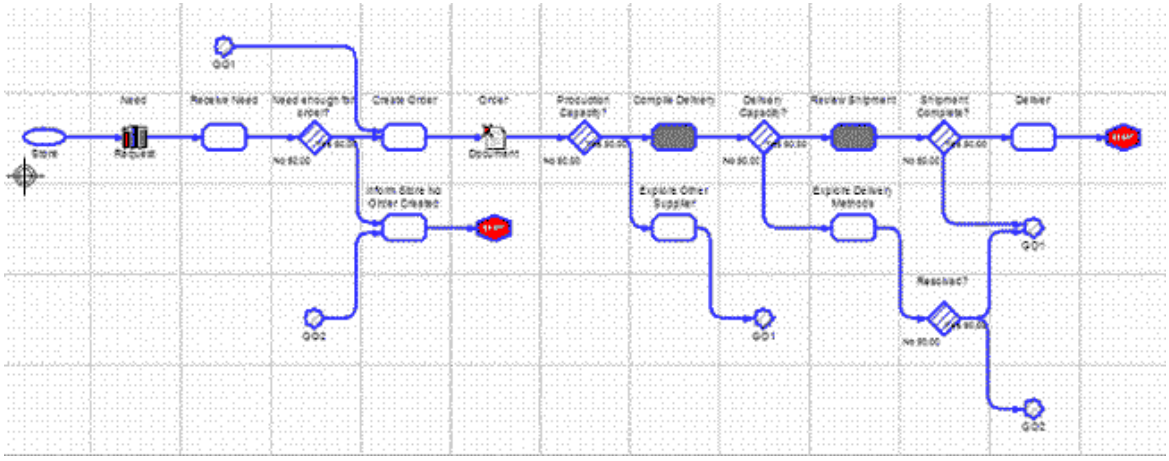


图 14 使用 WBI Modeler 的业务流程实现建模实例

在 Rose/XDE 中使用 UML 的相同的活动/工作流图如图 15 所示。

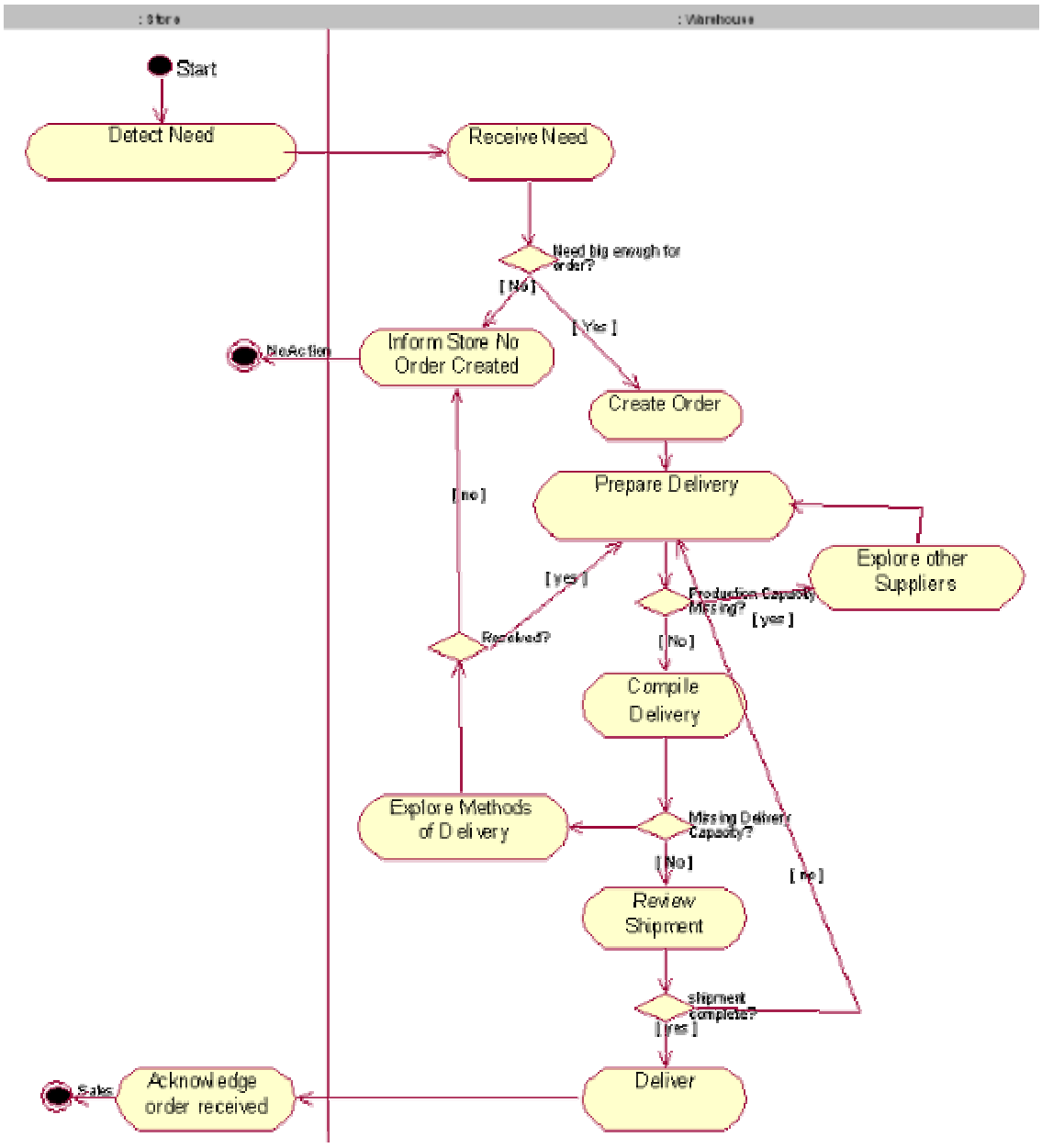


图 15 用 Rose/XDE 和 UML 画的，与图 14 中描述相同的活动/工作流。

对技术读者来说，使用得更多的其他图类型是顺序图，能够通过如图 16 所示的 Rose/XDE 生成。

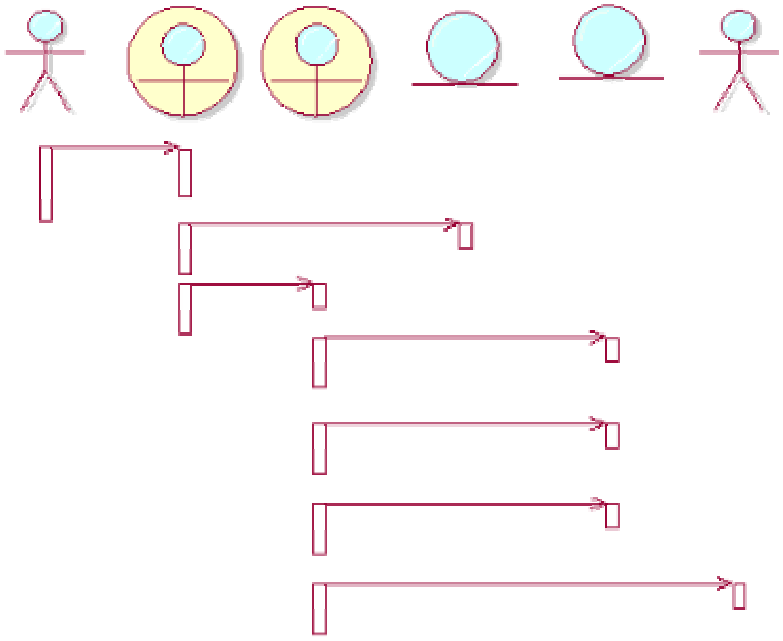


图 16 为技术读者设计的顺序图提供被建模业务流程的逐步实现视图。

到目前为止，我们只关注流程的动态性。弄清使用类图的类中的静态结构也是有用的。例如，在业务用例或流程中的参与对象视图如图 17 所示。

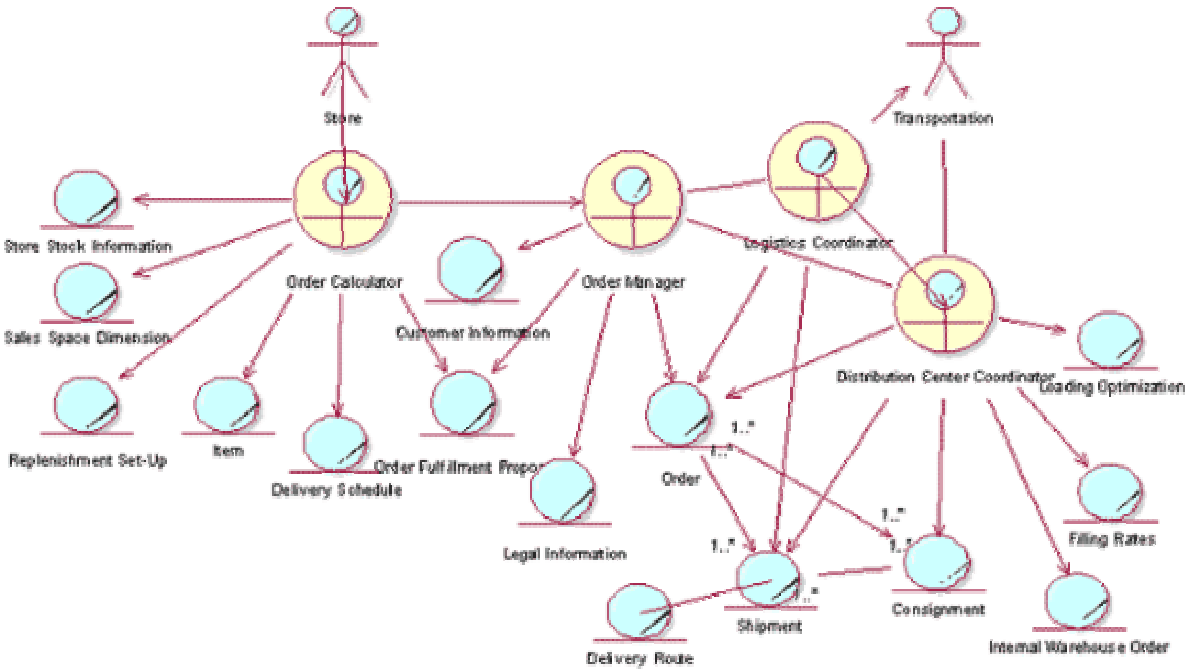


图 17 类图的例子。显示业务用例或流程中的参与对象的视图

你也可能研究信息中的结构，在 Rose/XDE 中表达的业务实体如图 18 所示。

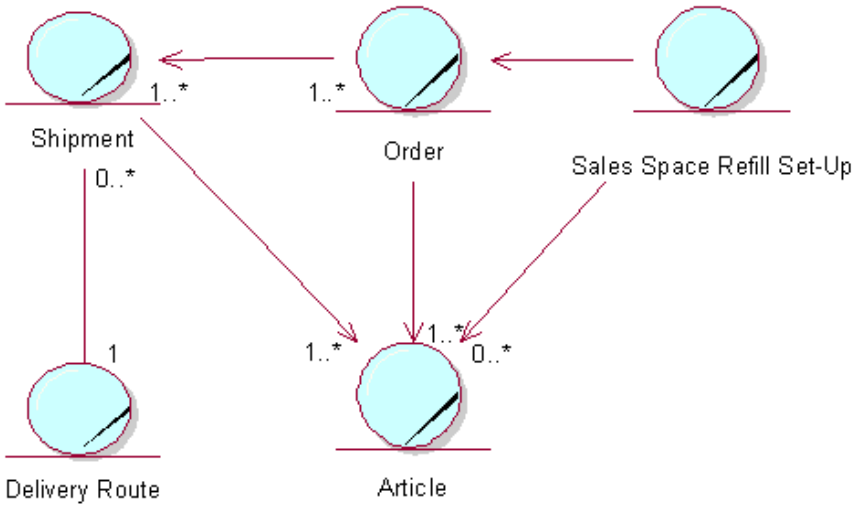


图 18 Rose/XDE 允许建模信息结构表达为业务实体

提炼职责和角色。如果你不需要仿真你的工作流，典型情况下，你不在业务分析模型中的类的详细描述花费很多时间。相反，你应该调查状态变化来研究如图 19 所示的状态图的关键业务实体（Rose/XDE）的生命周期。

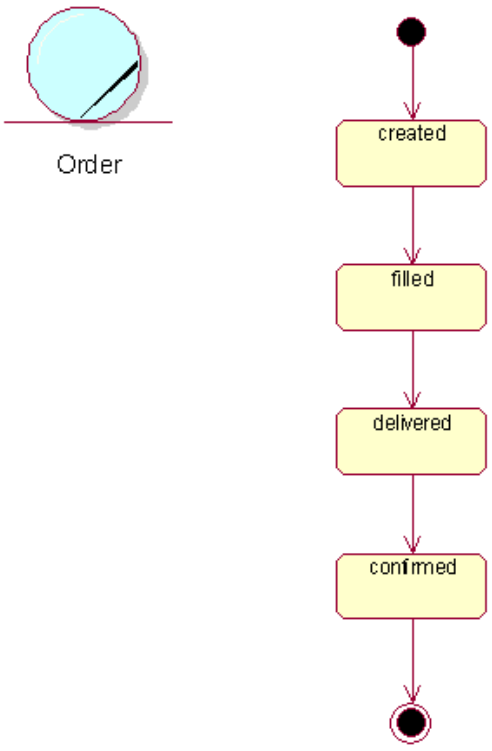


图 19 在 Rose/XDE 中画的状态图（状态转换）例图

如果你正在 WBI Modeler 中构建模型，并且计划做仿真，你将需要抓住围绕资源的成本和日程的细节。图 20 显示一个业务工人/角色在 WBI Modeler 中如何定义的例子。

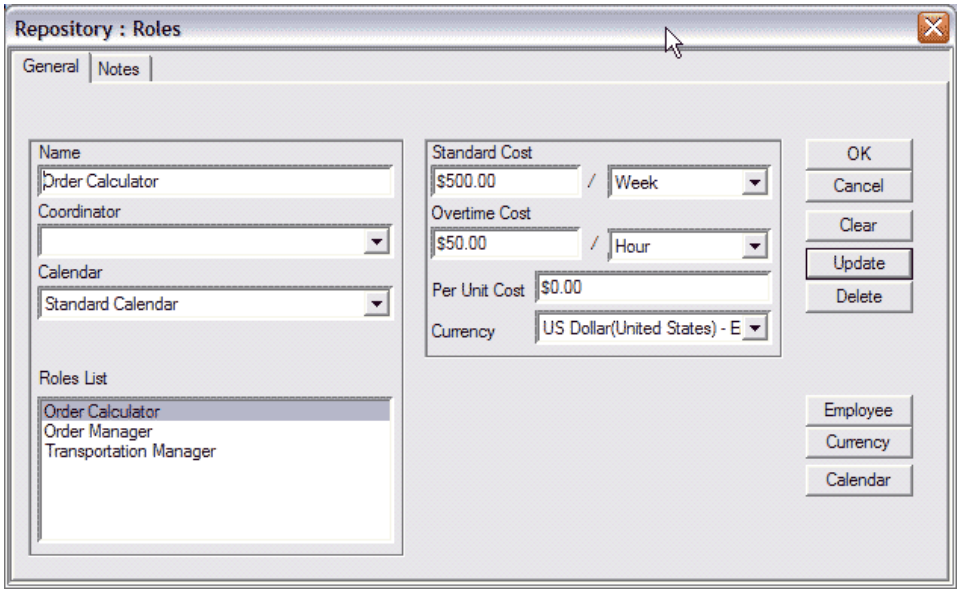


图 20 WBI Modeler 帮你捕捉围绕资源的花费和日程的细节。这部分的用户界面显示业务工人/角色是如何定义的

模型分析。RUP 目前不提供分析中的许多重点或模型的仿真。有一些描述基于活动的 WBI Modeler 工具支持的一系列成本的一些内容。假如你已经抓住如前面步骤描述的资源成本和活动持续时间的细节，WBI Modeler 允许你生成分析工作流场景执行的多种图表。图 21 显示一个工作流（场景）实例中每活动的成本例图。

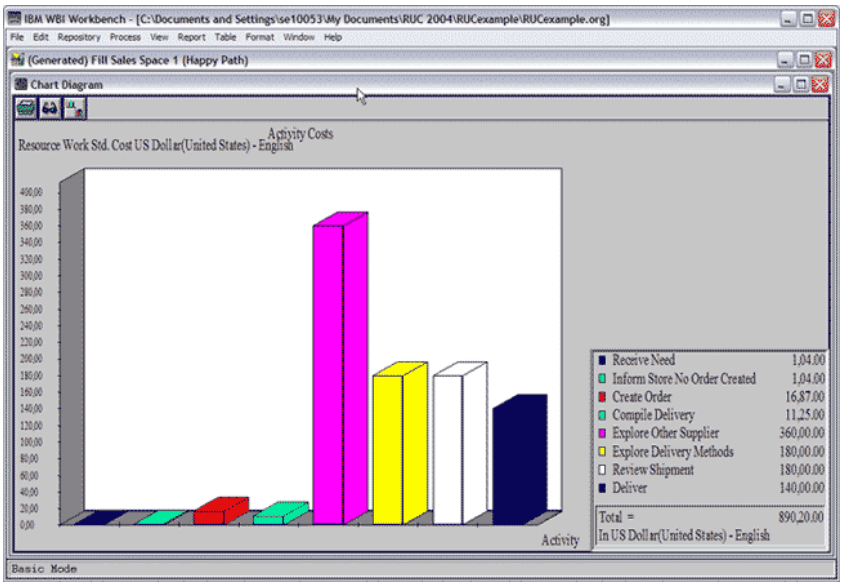


图 21 WBI Modeler 允许生成多种图表来分析工作流场景的执行。这是一个工作流（场景）实例的每活动花费的例子图表

分析的其他变化是模仿过程的进展。图 22 显示一个生成用来在 WBI Modeler 工具内部的可视化参考的过程“快照”。在活动/任务后面的小红盒子的较大数值表明在快照生成的时间点上每一任务的较高的工作量。

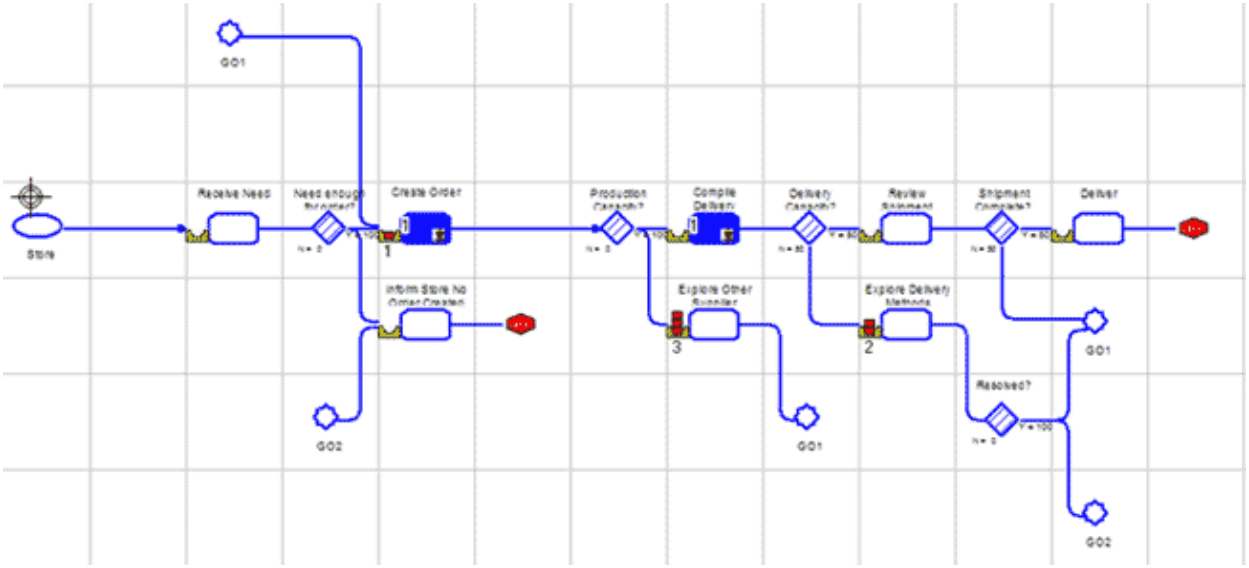


图 22 WBI Modeler 允许模拟流程进展

可重用的模型框架

IBM 的业务咨询服务组目前正通过开发一个我们称为组件业务模型（CBM）的方法来关注客户的业务工程需要。今天相互联系的公司面对一个多层次挑战的业务环境；组织机构和战略联盟持续变化以响应急速的市场变化。这个设想是通过为敏捷的或按需业务提供框架来加速转换。目前，CBM 首先可供金融业的用户使用。但从普遍意义上说，像业务系统的定义那样的一些 CBM 原则和技术在今天的 RUP 中是可用的。

集成项目 vs. 开发项目

在业务建模中要考虑的其他因素的集合不得不利用项目的总体目的；那就是，无论你是否准备集成过程和应用，或开发或自动化过程。应考虑以下事情：

- 在传统的应用开发中，工作需要包括架构规格（界面，机制，组件）以及像类和方法那样的底层构建。UML 提供对这些活动的全部支持。

- 集成方案的组成有些不同：适配器，业务目标，过程的单个步骤地代表框图等都是关键的构建。在集成项目上的工作经常围绕与现有人员，过程和应用（或许与一些附属开发一起支持终点集成）联系在一起。

因此，如果你打算在开发工作中以详细的业务建模工作为结果的话，实现技术的选择在生成的建模工作的格式（以及潜在的全部内容）上有很大的影响。

在一个高层次上，可以认为 IBM Business Performance Management Framework（使用 WBI Modeler）方便实现对项目集成的交易过程建模，IBM Software Development Platform（使用 Rose/XDE 和 UML）方便实现新系统功能和架构的过程建模。这两个工具集能够一起使用（例如，Rose/XDE 用来建模业务需求和业务分析模型，WBI 用来实现交易业务流程）。

结束语

业务建模是一个转向按需企业的关键。IBM 集中业务建模来推动：

- 流程监控
- 流程集成
- 流程创建
- 流程自动化

目前，IBM 软件组提供两个用来指导业务建模的补充产品组：

- IBM WBI Modeler，它与 IBM 的流程配合与监控的产品捆绑。WBI Modeler 支持在 RUP 业务建模规则中定义的建模活动的一个子集，主要是业务用例的可视化实现。WBI Modeler 也提供了允许分析模型的模拟能力。
- IBM Rational Rose/XDE，支持全面的 UML 业务建模，并且捆绑在 IBM 的软件开发平台和由 RUP 业务建模规则定义的所有建模活动中。

根据你的业务建模的上下文，你可能选择这些建模方法的各种结合。

参考文献

- BPMI (Business Process Management Initiative) - www.bpmi.org
- OMG (Object Management Group) - www.omg.org
- Continuous Business Process Management with HOLOSOFX BPM Suite and IBM MQSeries Workflow—an IBM Redbook, www.Redbooks.com
- CBM (Component Business Modeling)—IBM Executive Brief
- The Rational Unified Process 2003

致谢

我想感谢 Jasmine Basrai, Sr.，WBI Modeler 团队中的产品规划者，2004 年在 Dallas 与他合作为 Rational 软件开发用户大会作了一个报告，本文基于该报告。

感谢 IBM Rational 的 Simon Johnston，他帮我挑选出需要的标准。

感谢 IBM Rational 和 IBM 的杰出工程师 Alan W. Brown，他教给我们很多工作并一直帮助我们。

感谢瑞典 AstraZeneca 公司的 Christina Gerde 和 Anna-Lena Henriksson，他们对我们理解客户在业务工程、业务建模和业务流程管理上有很大的贡献。

关于作者

Maria Ericsson 是 IBM Rational 战略服务部（SSO）的首席咨询师。 她于 1990 年在 Objectory AB 公司开始在软件工程和对象技术领域的工作，并与 Ivar Jacobson 合著《*The Object Advantage: Business Process Re-engineering with Object Technology*》一书。自从 1995 年加入 Rational 以来，她一直作为在过程、业务建模和需求管理方面的顾问和培训讲师，并且花费了三年的时间作为 Rational 统一进程或 RUP 的成员。作为 SSO 的成员，她目前关注方案部署策略和 IBM Rational 领域培训团队的服务。作为在瑞典的常驻代表，她工作在位于 Kista 的办公室。

本文最初发表于 IBM developerWorks & RationalEdge，经 IBM developerWorks 允许刊登。

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

机械工业出版社即将出版

Scott Ambler

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于
领域模型没有捕获到
业务的深层内涵

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

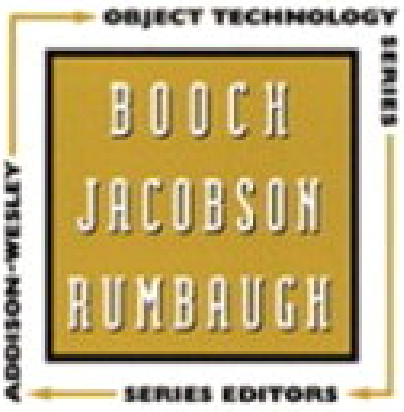
即将由清华大学出版社出版，UMLChina 审稿

THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

Covers UML 2.0

UMLChina 译
（王海鹏、李嘉兴）



CD-ROM Included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社出版

需求问题的根源－需求启发

think 著

讨论

用例是表达需求的一种形式，例如：

表达了这样一个要开发的系统的需求：“值班人员”需要使用系统来达到“派单”的目的，或者说，“派单”是系统应该为值班人员提供的一种价值。但问题来了：**我们怎么知道“值班人员”使用系统来“派单”是一个合适的需求？**答案只能来自涉众，和“派单”这个事情利益相关的人们。

需求只能从涉众中来，涉众包括最终用户、客户、政府、法律、文化、开发人员、管理人员、竞争对手…但是，需求不能直接从涉众中来。需求工程师往往不能通过询问涉众“你的需求是什么”来得到需求。经常有人抱怨“客户什么都不懂”，或者纳闷“客户需要懂得 UML 或者用例吗？”。其实需求工程师应该把启发的责任揽过来，而不应该过分强求客户。想想有这么一类无知的涉众：婴儿。婴儿只会哭和笑，不会直接表达需求，但出色的公司仍然可以从婴儿身上探索出“天线宝宝”这样的需求。通过各种手段探索到客户“想要的是什么”，是需求工程师最重要的能力。

启发障碍

涉众往往存在以下启发障碍：

1. 涉众无法陈述自己的需要。即使说出来了，也可能是解决方案不是需求。涉众说“我要基于 web 的三层架构系统”（从报纸上看的或从别的地方听来的），其实他真正想要的很可能是“想让分布在各个地方的员工能一起工作”。

2. 对于某些新产品的研发，开发者往往是第一个想到的人，涉众不会直接构想出新的工作方法。这种情况，更不可能从涉众那里得到直接的需求。

3. 涉众的利益矛盾。开发一个系统，往往会涉及很多人的利益。有的人会获得权力，有的人会丧失权力。不同立场的涉众之间必然会存在矛盾，如果启发工作安排不当，这种矛盾会成为一种启发障碍。更不用说有些系统的开发可能会使某一类涉众下岗或者使得头脑里面的经验不再重要，导致这类涉众抵制变更。

所以，要根据不同的情况，采用不同的启发技术来探索涉众的真正需求。

常用启发技术

文档研究

文档研究往往是需求调研的第一步，主要是为了获取业务领域的初步知识，为下一步的启发工作作准备。研究的文档资料可以是工作中的表格、文件、便函、工作报告、作业日志…等

文档研究的时候要注意：文档资料往往会比较多，有价值内容的相对比例较少，如果碰到有价值的信息，随时做笔记，或者把该页复印、扫描甚至撕下来。

问卷调查

当需要调查的人群分布较广时，随意挑选几个人来访谈或观察是不够的。例如，要去调研一个即时消息软件的需求，如 QQ，如果在自己公司随便找几个人问，事实上这几个人很可能都属于一类涉众“软件开发人员”，他们的利益诉求不能代表即时消息软件其他类型的涉众，例如“中学生”、“新闻工作者”，“网络管理部门”。如果涉众的分类尚不明确时，就需要做一些问卷调查（电子的或者纸张的），根据问卷调查的结果来给涉众分出类别，然后再从各类中选取样本，进行下一步的启发工作。

访谈

访谈是最重要也最常用的需求启发技术。需求工程师和涉众直接交流以收集信息。我们可以分几部分来说这种技术。

1. 需求工程师

需求工程师的态度要让涉众觉得自己被尊重。首先是言语上的礼貌：例如，不能表露出“你不懂软件！我才是专家” 的意思，“懂得软件”不是涉众必须的素质，涉众只需要清楚自己的利益和关注点。另外，访谈的涉众往往处在一个从业人员平均学历和能力都比软件业低得多的行业，涉众可能在接受访谈时潜意识中有一种自卑感，如果需求工程师的态度不礼貌，更容易引起抵触心理。另外，不可忽略这个事实：系统的出现可能对受访者不利。要么是头脑里的经验不再那么重要，甚至职位还可能会取消。所以在言语中暗示“我们来这里是为了让你下岗”的意思是不适当的。即使是表示“我们来这里是为了帮助您把工作做得更好”也不合适。他的工作做得很好！并不需要我们帮助。而应该把自己摆在一个低姿态的位置上 “我们来这里是为了帮助您更方便地完成工作”－－方便，想方便的时候就方便一下。

其次在行动上，访谈的时候身体应前倾，不时点头，发出声音“嗯－嗯”，手上做一些笔记（表明重视），适当的时候作两句总结。这样，涉众的心里多半会大为受用，向你倾出心中所想。

访谈的时候光听肯定是不行的，要想办法记录。可选的方案有：

自己做笔记－－手上肯定会有做笔记的动作，但记录的速度是比不上说话的，而且还会因为问答，节奏经常被打断，不能因为低头专注记笔记而影响交流。所以，边问做笔记只能记住一些关键点和做出尊重涉众的姿态。

同事做笔记－－如果有两个人去访谈，可以一人主问，一人主记，然后再掉换角色，最后将笔记合并。

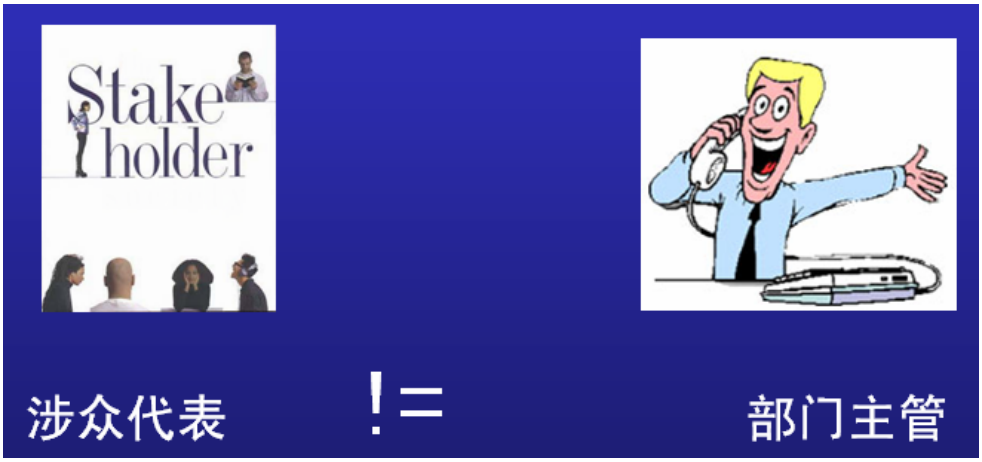
录音－－现在录音设备越来越小了，放在桌面上基本不会对访谈造成影响，是一种非常好的记录方法，如果说有什么不足之处，就是只记录了声音，回去后无法研究涉众的表情来确定涉众的真实意思。

录像－－可以看到肢体语言，但在有一个镜头对准的情况下，受访者往往受到影响。

有可能的话，记录手段应该采取双备份。录音笔会出故障的，因为去一趟不容易，保险一点为好。但要把录音录像当作好像不存在，该倾听倾听，该记笔记记笔记。

2. 涉众

涉众代表必须名副其实。“涉众代表”不等于“部门主管”。要访谈车间的操作工，就不能用车间主任来做代表。操作工岗位的酸甜苦辣，只有操作工自己才知道。应该把“车间主任”看作是另外一类涉众。实际工作中常见的是把甲方的“信息中心主任”头衔的涉众作为主要的需求来源，认为他们既懂电脑，又懂业务，其实大谬。



不同类型的涉众，应该尽量单独访谈，如果图省事把有利益冲突的两类涉众集中到一起访谈，受访者言辞之间可能就会有顾忌。

3. 访谈问题

内容上，访谈就是要收集用例的素材，所以用例思维在这个地方就可以起到指导作用。例如，问“为了完成这项工作，您需要哪些材料？”，可能就对应到 “用例的前置条件”，问“这项任务要是做得不好，会影响到谁？”可能就对应到“涉众利益”，问“如果您不做这项工作，公司会受到什么损失？” 可能就能追索到“业务用例”。

形式上，和新闻记者提问一样的：5W＋1H。谁（Who）、什么（What）、什么时候（When）、什么地点（Where）、为什么（Why）、怎么进行（How）。提问的时候尽量采用业务词汇，不要采用技术词汇。

4. 环境

访谈的环境。要在涉众的工作环境里进行。涉众在自己的工作环境中会想起许多工作中的喜怒哀乐，如果把他搬到软件公司或者宾馆的会议室，环境发生变化，一些本来深有感触的东西，因为环境的因素，一时之间会想不起来。

（访谈要在涉众的地盘进行）

观察

观察就是需求工程师亲身去体验涉众的工作。这是最直接的技术，也是最费时间的技术。

要挑选经验丰富的涉众来观察。因为系统的开发就是为了分担他的经验。经验丰富的业务工人在长期工作中归纳出一套行之有效的方法，这套做法也许就是系统应该借鉴的。

观察时，重点关心：完成一项工作所需时间、操作次数、出现的的错误和混乱。所需时间越多、操作次数越多、出现的错误越多，系统如果能在这项工作有所帮助，必将会给涉众带来强烈的改进感觉。

观察在今天的产品研发越来越重要。在市场竞争激烈的体验经济时代，客户的口味提高了，光是有个东西给他用是不行的，市场上有十几二十种同类的产品。对精益求精的产品开发者来说，观察是不得不花代价去做的启发技术。

开会

开会就是把不同类型的涉众集中到一起，往往用于验证需求和解决冲突和妥协。不同涉众有不同的利益诉求，归纳成需求文档后，可能就会发现利益之间的冲突导致需求无法确定。这时，把大家召集在一起，验证写好的需求文档和集中商讨其中的冲突。

开会时值得注意的是小心少数人主宰讨论。如果高级别的与会者不断插话，身为下属的与会者可能就不敢发表意见。所以，事先可以就此提醒高级别的与会者。

研究竞争对手

这是做产品型软件的厂商最重要的考虑。如果说以前产品的需求是开发人员导向的，现在软件公司也逐渐意识到需要以客户为导向，所谓“客户是上帝”。问题在于，这只是必要条件，而不是充分条件。所有公司都知道要“面向客户”，这个时候客户如何才能从这么多家中选择你？难道是你有我也有？你加一个功能，我也加一个功能；你反应时间 0.5 秒，我反应时间 0.4 秒？最终只能在无奈的军备竞赛中打转。

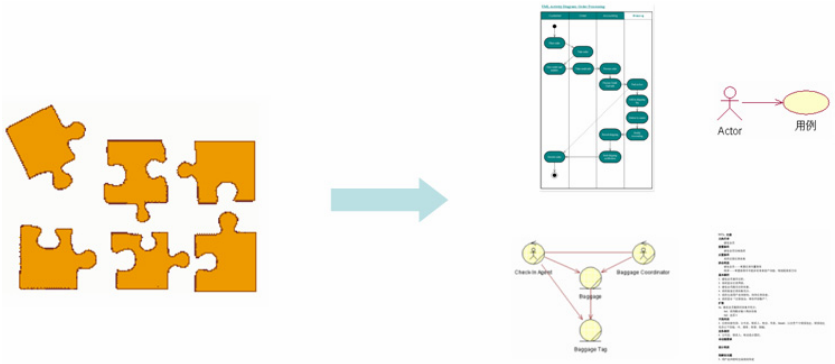
（冷战时的“星球大战”计划）

研发就是战争，战场是客户的头脑。必须研究竞争对手，做出对应的策略，而这些策略直接影响了需求。市场经济里，一个领域可能都会有以下战局。市场领先者负责防御，扎紧自己的篱笆，不断更新自己。几家跟随者，它们负责进攻领头羊。其他的众多公司想尽办法占据一小块细分市场。只有研究清楚对手和战局，才能了解自己的位置和应该采取的战略。

可以借用齐白石的话，“学我者生，似我者死”。不是“更好”，而是“不同”。研究竞争对手时，思路就是：研究对手的优点，但不是模仿，而是在关键的地方反其道而行之。例如在中国，MSN 的“不和陌生人说话”进攻 QQ 的“爱陌生人”。

从素材到需求

得到素材之后，需求工程师需要结合软件的愿景，把各种素材碎片归纳成需求。并不是所有涉众的需要都会变成软件需求。象不符合愿景的要求。例如：目标是改进进货流程的效率。访谈到供应商时，提出了自己库存管理上的一些要求，这和愿景就不相关了，所以这样的要求不能变成软件的需求。还有不是计算机所擅长的要求。计算机最擅长做什么？一是信息的自动流转，如：患者做检查后检查图像自动流到医生桌面；二是演绎复杂业务逻辑，如：比较价格寻找最优价；三是访问和操作业务实体，如：把病历电子化以方便管理和检索……对一些需要人去判断的地方，计算机实现是很困难的－－对这一点，涉众有时并不了解，经常提出过分的要求。

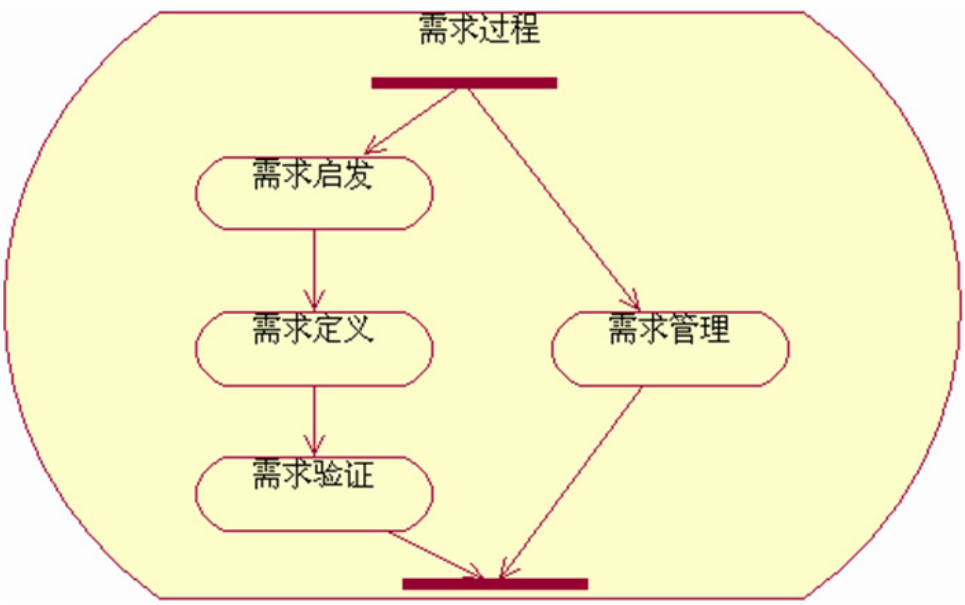


（把素材归纳成需求）

素材本身是跳跃多个边界的。要在不同边界来回思考，需要需求工程师有强的归纳合成能力，并且掌握一些思考的工具。温伯格的《探索需求》介绍了一些思考的工具，此书是目前少数几本专注于“启发需求”方面的书。

结语

软件开发中的需求过程可以通过活动图简要表示如下：



需求启发：从涉众那里得到需求的素材。

需求定义：把有价值的素材组织成需求文档（用例文档）。

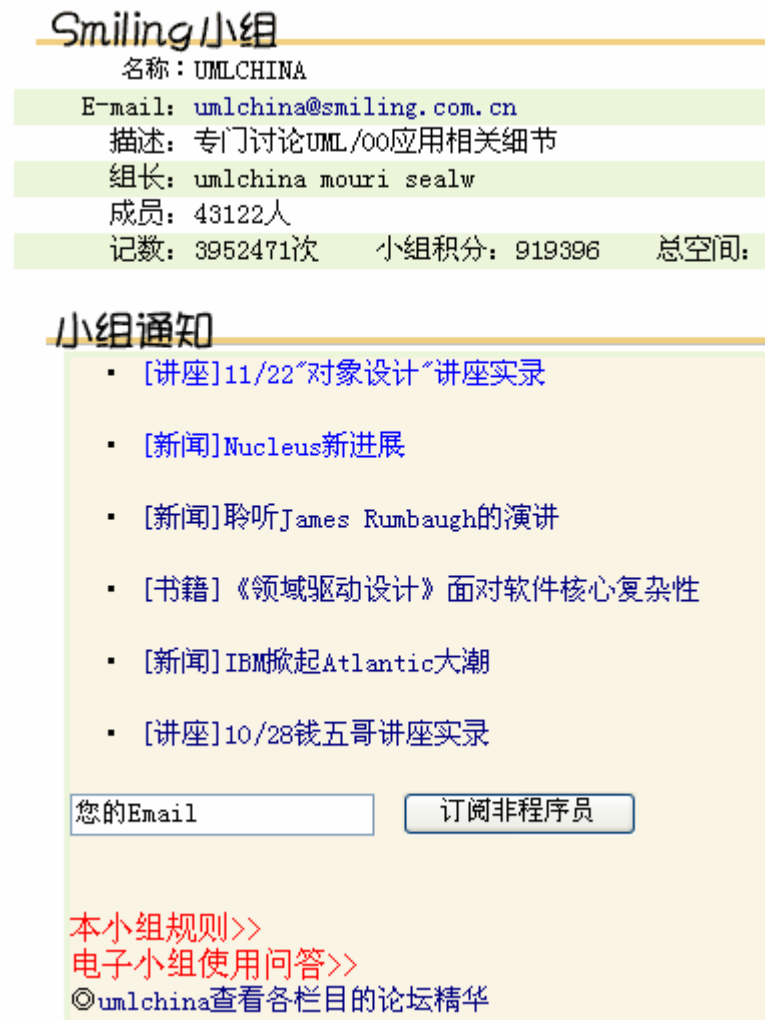
需求验证：让涉众验证需求文档。

需求管理：跟踪需求，管理变更…

这四个环节里面，除了需求管理环节目前已经有一些工具如 Borland 的 CaliberRM，Rational 的 RequisitePro…来辅助之外，其他环节上，工具很难起到大的帮助。需求启发和需求验证的绝大部分工作是在人和人（需求工程师和涉众）之间进行，强调的是人际交流的能力，需求定义需要的则是思考的能力。很难想像有这样的工具：客户只需呆呆坐着，把它套在客户头上 5 分钟，就可以知道客户“想要的是什么”；也很难想像这样的工具：它能够从一大堆乱七八糟的笔记和录音里，自动整理出一份整齐的需求文档。

当我们关起门来“编写有效用例”的时候，当我们激烈争辩一个用例的“粒度”的时候，一定要记住，答案根本不在我们这里，在涉众那里。到第一线去，用我们上面提到的各种技术，探索涉众心中所想，才是需求的正道。

本文首发于《程序员》杂志 2004 年第 11 期，经《程序员》允许转载。



Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

清华大学出版社

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

探索需求

设计前的质量

Donald C. Gause
Gerald M. Weinberg / 著
章柏幸 王媛媛 谢攀 / 译

Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社

从基本用例到对象

Robert Biddle、James Noble、Ewan Tempero 著，李胜利 译

讨论

导言

用例的识别和使用现在已经成为软件开发中的常见惯例，并且用例是一个在建模语言和开发过程中被认可的概念。基本用例作为一般概念的精练而被介绍，最初的目的是为满足在用户界面设计中表达独立于技术的需要。我们感到基本用例值得作为一个更广泛的角色，并且在一般的面向对象软件开发中都已在探索基本用例的应用。

为一般系统开发考虑基本用例的动机起始于基本用例一些简单的优点，特别是其抽象使得过早考虑界面细节能被避免，并且使之更为简洁。当我们在开发各类系统应用基本用例时，我们逐渐认识到许多深层的益处。

我们开始使用基本用例作为我们首选的需求收集工具，并且已经在最近三年多的时间里使用这种方法完成了许多系统开发工程。我们认为基本用例普遍适用于面向对象软件开发，并且极大地优于常规用例。

在本文中，我们简要回顾基本用例的本性，然后讨论两个我们正在探讨的新技术。第一个技术涉及需求分析，并且涵盖基本用例自然支持的用于用例评审的角色作用的使用；第二个技术涉及设计，并且涵盖由基本用例中辨别出的系统职责分布，生成一个组成可跟踪需求的面向对象设计所涉及的对象和他们责任的集合。

背景：基本用例

用例的一般想法是描绘系统（即使尚未实现）和系统外部世界间可能的交互的序列。用例在RUP中的定义是“对特定的角色来说，系统执行产生一个可观察的结果值的动作包括变量的集合和序列描述”。用例的定义对整个系统开发过程都是有益处的，从早期的分析阶段开始持续到测试和维护阶段。

基本用例是以用法为中心设计方法的一部分，是由Larry Constantine和Lucy Lockwood开发的，Constantine和Lockwood支持用例，并且同意关于用例优点的许多主张。他们同时也看到了用例的局限性：“在特别情况下，对于仍在设计中的用户界面表单，一般用例典型地包含太多的内在假设，它们经常是隐藏的或含蓄的。”无论是因此而过早强迫作出设计决策，还是因此而嵌入到这些需求中的决策，这对于用户界面设计来说都是有问题的，使得到其后难于修改或适应。

基本用例被设计用来克服这些问题。这个术语“*基本*”涉及到基本模式“是通过技术选择自由，理想化和抽象的描述来捕捉问题的本质”。Constantine和Lockwood把基本用例定义如下：

> *基本用例是一个结构性的叙述，使用应用域和用户语言来表达的，组成一个任务或交互的简单的，一般化的，抽象的，非技术的和与实现无关的描述，从用户的观点来看，在某个角色或与系统相关的一些角色中，这个任务或交互是完整的，有意义的和明确定义的，并且使构成交互作用的基础的目标和意图具体化。*

基本用例以代表用户和系统间对话框的格式被书写。它类似于Wirfs-Brock1993年所使用的两列格式。在这种格式里，列表指的是动作和响应。尽管Wirfs-Brock讨论过用例中抽象层次框架的问题，然而她提出的这种两列用例包含了用户和系统交互的具体步骤。

相反，基本用例格式标出了用户意图与系统职责列。这些新标签显示基本用例如何通过无需描述用户接口细节而允许交互文档化来支持*抽象*。注意这种抽象并不真正与用例作为一个整体相关，而是更多与用例步骤相关。使用这种方法，基本用例真正指明交互序列，而不是抽象步骤的序列。

Constantine和Lockwood给出了图1和图2中显示的例子。图1中的对话框是为一个使用动作和响应术语描述的对话用例设计的。图2的对话框是为一个使用意图和职责术语描述的基本用例设计的。基本用例的步骤是比较抽象的，并且允许具体实现方法的多样性。理解对话仍是容易的，然而基本用例是比较简略的。

取现金

用户动作	系统响应
插卡	
	读取磁条，申请PIN
输入PIN	
	验证PIN， 显示事务选项菜单
按键	
	显示账户菜单
按键	
	提示数量
输入数量	
	显示数量
按键	
	退卡
取卡	
	分发现金
取现金	

图1 从自动柜员机取钱的用例对话

用户意图	系统职责
自身标识	
	验证标识
	提供选择
选择	
	分发现金
取现金	

图2 从自动柜员机取钱的基本用例

基本用例和需求

我们在使用基本用例上的第一个探索是在系统需求的工作中。我们想使这项活动对于新手和非技术人员易于使用，并且想使这项活动在一种多人作为团队一起工作的方式下使用。

在目前很多时候，我们已经在面向对象开发后来的阶段使用和赞美CRC（类职责协作者）卡技术了。为了使用例更易于使用，我们决定采用简单的方法。我们的基本设想是对用例使用索引卡，并且一定程度上也涉及到角色扮演。本文在这里提出一个大概的想法。

用例卡和角色扮演

我们决定每个用例卡应当代表一个单独的用例，应当显示用例名称和对话脚本的步骤。我们跟踪一个基本用例的规范表达，并且区分两个角色，我们从中间分割卡片，在左边写上用户列，在右边写上系统列。最初我们在卡片画一条垂直线，但是不久发现这样难于区分用例卡和CRC卡。经过一些试验后，我们决定在中间画一条暗示交互作用的锯齿线，这样易于区分用例卡和CRC卡（参见图3）。

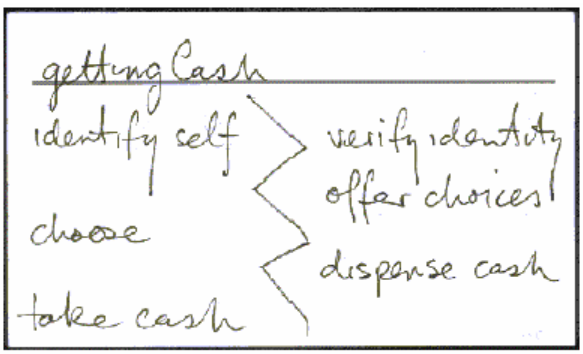


图3 一张基本用例卡

基本用例的形式具有很多优点。首先，因为使用这种类似于简单脚本的简单对话形式，这种两列的格式简化了角色扮演。脚本中有两种角色，用户和系统，所以用例角色扮演能够简单涉及两个人，每个拥有脚本的一部分。另外一个重要的格式上的优势涉及到基本用例的提取。这种抽取保持了基本用例简洁，并能适合索引卡。这种抽取也帮助避免对不相关实现细节不必要的争论，所以允许更快的进展。这种意图和职责的焦点包含有我们以后将要讨论的重要结论。

用例卡和角色扮演从根本上帮助细化用例“主体”和决定交互的步骤。在这之前有必要标识用例。这最初涉及为用户可能与系统交互的不同方式提出建议的背景分析。在大型的系统开发中，尽管用例标识能够组成一个大尺度和范围的活动，我们使用分析技术识别大量的候选基本用例，然后把它们按优先次序区分，选择多个“焦点”基本用例，他们代表早期原型中必要的关键交互。

然后我们的角色扮演技术开始了。我们通过使用卡片和角色扮演在这些焦点用例上进行细化步骤来着手。我们通常建议由一个3到6人的小团队做这些工作，对CRC卡片的工作也类似。我们通过在卡片上写上焦点用例的简要名称开始工作。例如，一个银行系统可能有一个称为存入现金的焦点用例，它用来检查账目结余和抽出现金。

然后我们选择一个卡片并设计出对话。我们选择用户和系统角色的人员，并且使用一种探测性的角色扮演演练。团队一起工作，决定对话中的步骤。当设想看起来合理时，角色扮演者通过脚本进行表演。团队的其他成员审查，然后讨论对话和做些必要的改进。文档投影机的有效性增强意味着卡片能够容易地投射到大屏幕上，供较大的团队跟进（见图4）。

图4 通过投影用例卡来执行用例角色扮演

这个过程重复使用直到团队对对话代表的用户和系统交互的方式满意为止。有时它可帮助一个用例与其它用例一起工作，随后返回以改进早期的用例。

角色扮演的一个用途是用来做探索性的和迭代的用例开发。我们也使用角色扮演来评审。这样的评审可能由一个项目中不同方面对等的开发者或者涉众或者系统领域的专家引导进行。

在索引卡片和角色扮演已经帮助确定用例主体后，全部的细节可能被记录在需求文档或案例工具数据库中。用例能够用来驱动系统设计、评审、以后的系统测试和试运行。这些卡片和角色扮演可能仍然证明在以后的一些活动中有用，他们即时的和轻量级的天性支持快速评审和可选择的交互开发。

体验和讨论

我们对基本用例的体验显示许多简单的优点已经实现，并且也还有更深远的效果。正如我们所希望的，抽象真正保持用例对索引卡足够简洁，并且帮助我们避免关于实现的不成熟的争论。

我们已经明确地为团队成员寻找主动的和直接的约定。就CRC卡来说，这种索引卡的具体单元连同其角色扮演的行为单元一并导致了注意力的集中和指令的积极参与。这种独一无二的特性是有价值的，因为它保证了实际整个团队在决定需求上的集中。

索引卡也以另外一种方式帮助我们，就CRC卡来说，相对于它们作为卡的简单性质来说，他们是离散且具体的，并只是一个轻量级的投入。卡片能够在桌面上被重组或按次序排列，需要的卡片被选中而其他的放在一边，并且在角色扮演期间卡片可保存；一个卡片可被改变或不需要时很容易销毁，并且这种改变可立即进行。

角色扮演的优势可以更加深入。甚至非开发者的人员也能擅长于下面的对话。而且他们熟悉角色扮演的生动的设备并且乐于中止对戏剧的怀疑。参演人员通常能够观看角色扮演，并且对最终系统的最终实现设想交互。这是一个快速原型形式，它允许快速评审和允许迭代改进的反馈。

在较深关卡的层次上，我们看到在铸造用户角色上作为表达意图和系统角色作为表达职责的启发式的优点，合起来说，这些导出了更好地反映用户动机和捕捉系统需求的用例对话，同时避免过早的设计决策。

在我们的经验中，决定需求中一个特别早期的关键的问题仅仅决定了系统的边界，这就把系统该做什么与不该做什么区分开。涉及到所波及的人们—分析师和涉众时，达成必要的一致来作出区分是一件困难的事情。

用例卡和角色扮演已经证明在决定系统边界上非常有用。我们的方法是在设计上应用一个熟悉的方法：我们将系统看作一个“黑盒”。系统内部工作无须指定，但系统应用方式需由用例指出。每个用例的这种双栏的对话表单清楚地从系统角色中区别显示了用户角色，并且这些角色的分割线就是系统边界。

在用例角色扮演帮助决定系统边界上有多种方式。在早期的探索角色扮演中，如果团队成员在理解系统需求上有差异，它能够快速清晰。阅读同样的分析文档，人们能产生不同的理解，并且对于在技术设计者和业务分析员或者领域专家间出现的差异特别常见。尽早发现和解决这样的差异是重要的，并且为角色扮演决定实际的交互顺序来暴露这些差异。

探索角色扮演也可暴露关于对用户和系统角色的不合理的假设。对用户角色，它能显示出在用例中指定的动作与一个实际用户可能去做的不一致。对系统角色，因为关键信息不可用，它能显示出所需动作的不可能。实际上，如果用例没有被细心阅读的话，角色的表演动作似乎作出非常明显的异常。

为解决这样的异常，用例步骤可能不得不修改，或者用例可能需要以另外的方式组织。这样的改变可能需要对其他用例的修改，并且甚至可能需要生成新的用例。这个重要的结果就是问题能在这么早的时间上标识和确定，而不是在开发后期更昂贵的时间点上。

角色扮演也可使用例交互对开发团队外的人员能够理解。这对于提交用例给繁忙的涉众特别有用。角色扮演快速、投入并且非常容易接受。不需要大量的投入，它使讨论问题、比较方案和可能缺陷检测成为可能。

我们已经发现某些考虑需要与基本用例一起工作。一个是用例角色扮演将是自身抽象，并且如果涉及到明确的界面细节，它又并非是现实的。在我们的经验中这已经不存在疑问了，尽管团队发现它有时有助于照顾用例的具体设定来检查他们的理解。基本用例的起源是用户界面设计。在典型的实践中，系统职责受限于直接相关的交互职责。系统行为的深层职责语义学没有研究过。然而，我们发现在必要之处包含这些更深层职责是可能的，例如，早期图形中的用例能很好地包含以维持账目平衡的完善，审计一连串的账目的职责。

我们一直在开发带有基本用例的支持需求分析的工具。特别是，我们已经开发出一个基于Web的工具，它使得记录基本用例变得容易，并且也能帮助使用基本用例开始设计流程。

基本用例与设计

基本用例一个明显的特征是在对话过程的提取，这里用户与意图相连，并且系统与职责相连。在面向对象的设计中，术语*职责（Responsibility）*已经是一个特殊的角色。特别地，在CRC卡和职责驱动的设计中，职责是一个关键的概念。因此，基本用例中的职责标识提供了一个连接用例和面向对象设计的新方式。

在CRC卡技术中，类职责帮助决定系统清晰度，并且现在看来，这个原始的教学工具在所有的系统开发项目中是有用的。在职责驱动的设计中，职责的思想被使用得更加彻底，并且被大规模应用。职责与对象相关，并且象征对象所包含的知识，或对象能执行的动作。当职责对可能的实施架构没有记载时就强调了抽象的动作。

CRC和职责驱动设计都涉及职责概念去指导制定涉及决策。特别是，这些技术建议在一个面向对象的系统内，每个对象或类应当有一个一致的和易于理解的职责集。在这种方式中，职责被用作启发式的工具来分割系统为带有全部合理分布的职责的协作对象。

面向对象设计的一个基本法则是对象包含的动作和信息一起工作。因为职责通常建议包含做某事的职责和去做的资源，所以职责是一个好的决定对象启发式的工具。职责也允许委托，允许大量的职责通过委托少量的职责来管理。职责既包含抽象也包含封装：

> *职责驱动的方法强调对象结构和动作的封装。通过聚焦于类的契约职责上，设计者能够延期实现时考虑的问题到实现阶段。*

在基本用例中，职责的想法是用来标识系统必须做什么以支持用例，不需制订约定作为系统必须怎样做。这个类似的对象封装，当对象内部不能直接从外部访问时，它具有同样的益处。这个用例中的职责角色与设计中的职责角色是完全一致的。两者都不需要描述实现来描述动作。这个公共点提出连接我们决定需求和决定设计时的工作方式的有价值的机会。

系统职责的分布

这样看来，职责是一个强有力的概念，并且已经在面向对象设计中作为一个启发式的工具为人们所熟悉。基本用例也与职责相关作为一个相似的启发式的工具。对控制连接来说，这似乎是一个有价值的机会。我们可以把基本用例当作一个设计的起始状态。从整个观点来看，系统像一个简单对象。这对完成用例是一个起促进作用的有益观点。因为其打消了在早期阶段对内部系统设计的关注。注意这是同我们在需求分析中使用的相同的法则，

当决定系统边界时，我们将系统当作一个“黑盒”。

将系统作为一个简单对象的想法能够被用来作为一个开始然后开发设计的方式。我们将系统职责认为基本用例中的文档，并且巩固这些内容作为一个连贯的整体。然后我们将整个系统当作一个拥有这个职责集的对象，同样在CRC设计中，每个对象都有一个职责集。则设计流程是在对象集中的单个对象内分布职责。我们称这种方式为系统职责分布（DSR）。

一个典型的CRC程序应当从一个候选类或对象集开始，每个候选类或对象都具有初始职责建议。然后跟踪用例来研究分布，迭代修正对象和职责。可选的DSR程序应当从简单系统对象和它的职责开始，然后跟踪CRC。就典型的CRC来说，我们也应该找出一个候选的对象和它们的职责集，并且我们也应该研究作为用例驱动的设计。

然而，我们开始包含简单对象的设计工作。然后我们通过共享候选对象或者代理职责与候选对象间的职责来研究。就更多典型的CRC来说，我们反复研究和修正对象和职责。这种方法似乎是每种方式CRC的真正灵魂，并且仅仅提供一个更确定的起始状态。

DSR开始时的优势在于我们从用例的职责着手，所以我们初始的CRC对象角色扮演忙于从基本用例来的系统交互。我们进行过程中，特别是接近完成设计时，它的优势在于对象的职责集必须一起仍然符合基本用例中标识的职责。

通常，DSR的优点是可跟踪性：我们应当能连接设计的职责到基本用例中标识的职责。这也是相关的第二个优点。当可跟踪性允许我们对设计检查需求时，DSR也提供设计时的操作指导。因而在迭代设计过程中的任一点上，一种可能的方法就是进一步分布基本用例职责，或者以不同的方式重新分布。

当然，我们承认严格的分布系统职责不是一个完整的设计技术。特别是在从已知对象到未知对象工作是不具有CRC卡的所有优点，也不具有在标识设计抽象高层次中的职责驱动设计的优点。然而，我们相信DSR真正在可跟踪和操作指南上提供有用的优点，它能与CRC卡或者职责驱动设计一起工作。

例子：Sokoban（仓库番）

在这一部分，我们给出一个小型的详细例子，它的工作通过系统职责的分布将一个基本用例集转化为一个可工作的面向对象设计。这个例子是一个“Sokoban”游戏程序。本文我们仅给出此例以支持上面的讨论。

Sokoban是一种涉及仓库中“箱子”、“架子”和“墙”的难题。当唯一可用操作是“推箱子”时，目标是将所有的箱子放到架子上。这里有一些变化。其中一个如下：

Sokoban游戏包含一个关卡集。每个关卡描述一个仓库，由一个墙、架子、箱子和工人的起始位置的集合组成。倘若箱子对面是空的或空架子，工人可能在任何空位置移动，或者移动到被箱子占有的位置。也就是说，工人只能“推”箱子。当所有的箱子在架子上时，这一关卡完成，下一关卡开始。一旦所有关卡完成，游戏结束。如果游戏结束时游戏者既没有重新开始一个关卡也没有跳过任何关卡，然后他们输入游戏排行榜中他们的名字。

这就是可能给第一年编程课程的学生的描述种类，并且作为一个推理，它或许比通常案例措辞更严谨。图5显示一种可能的方案。

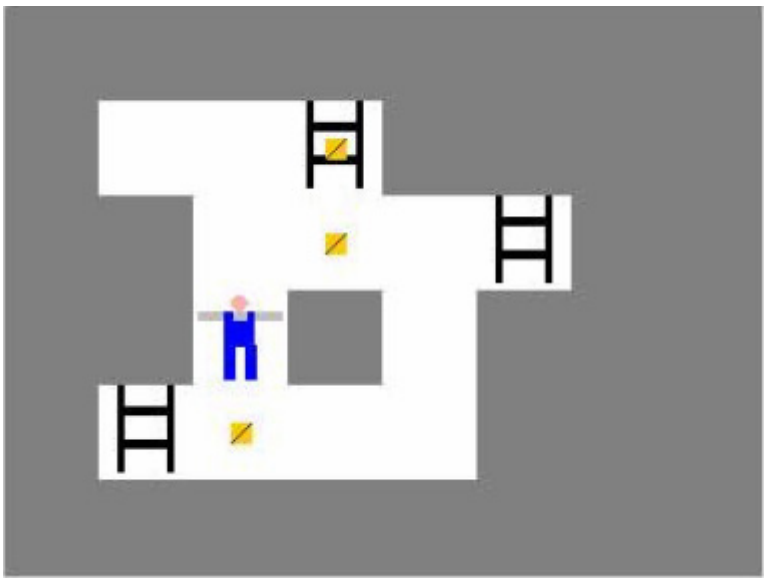


图5 三个箱子的Sokoban关卡

这一部分描述Sokoban系统需求为一个基本用例（EUC）集。EUC的约定在用例主体的描述上应尽可能简洁。因而，由于被EUC名暗指，他们典型地不从一个类似于“选择操作”的动作开始。它也经常是与参与者相关的用例，但在这个用例中只有参与者（游戏者）。Sokoban的基本用例如下：

开始游戏	
用户意图	系统职责
	需要游戏者名字
提供名字	
	记录游戏者名字 设置当前关卡为第一级 设置当前关卡为初始配置 显示当前关卡

当前关卡	
预先状态：关卡规格必须合法	
用户意图	系统职责
指定关卡	
	读取指定关卡描述 设置当前关卡为指定关卡 设置当前关卡为其初始配置 显示当前关卡 如果当前关卡是任一关卡而不是第一关卡，记录游戏者不能进入排行榜
重新开始	
用户意图	系统职责
	■ 设置当前关卡为其初始配置 ■ 显示当前关卡 ■ 如果当前关卡是任一关卡而不是第一关卡，记录游戏者不能进入排行榜
移动工人	
预设状态：指定方向必须合法	
用户意图	系统职责
指定方向	
	■ 如果在指定的方向上与工人相连的位置是空的，或包含一个箱子并且在指定方向上与它相连的位置是空的或包含一个空架子，如果在指定方向上移动工人，如果有一个箱子，箱子也移动 ■ 显示当前关卡 ■ 如果关卡中所有箱子在架子上，装载下一关卡 ■ 如果这一关卡是最后一个，并且游戏者在排行榜中合法，在排行榜中记录游戏者

显示排行榜	
用户意图	系统职责
	列出排行榜中的所有用户

我们现在给出Sokoban系统地面向对象设计的一系列迭代。这些设计作为CRC卡集给出。为了需求和设计间的可跟踪性能被跟踪，每个类的职责被加标签。这能进一步开发，但是目前我们仅关注无论何时职责被改变或被委托，我们使用这些标签标出它们与其他职责的关系。

在感兴趣的地方，我们这里使用完全代表其他类的职责在委托类中不显示的约定。然而只部分委托的职责仍将被列出（可能在某些方面改变）。

迭代 1：系统对象

设计的第一个迭代是系统对象，他们的职责作为系统职责在EUC中列出的单一类，这意味着这个设计用来保证提供需求中指定的动作。这里因为明显没有该类的协作者,我们仅列出职责（按字母顺序）。

系统对象	
SO1	如果所有箱子在架子上，则进入下一关卡
SO2	如果在指定的方向上与工人邻近的位置是空的，或者包含一个箱子并且在指定的方向上与其邻近的位置是空的或者包含一个空架子，则移动工人，如果有一个箱子，箱子也移动
SO3	如果当前关卡是最后关卡，并且游戏者挤进了排行榜，则在排行中记录的名字
SO4	如果当前关卡是除第一关卡以外的其他关卡，记录游戏者不能进入排行榜
SO5	列出排行榜中所有游戏者
SO6	读指定关卡的描述
SO7	记录游戏者的名字
SO8	请求游戏者名字
SO9	设置当前关卡到第一关卡
SO10	设置当前关卡到任一关卡
SO11	设置当前关卡到初始配置
SO12	显示当前关卡

迭代 2

检查早期部分中的系统对象，我们立即注意到“Level(关卡)”的频繁使用。这暗示着提供一个Level类可能减少系统对象承担的工作量。一旦我们完成这些，我们决定如何分派系统职责给这个新类。

在做的过程中，我们将改变系统对象职责。我们改变结果类的名字以更好地反映其在设计中的角色，即应用，也就是Sokoban的主界面。因为这个类给出了与系统对象相同的系统视图，所以它应当拥有相同的职责集，并且正好代表其他类中的一些职责。

Sokoban	
SO1	如果所有箱子都在架子上，则进入下一关卡（部分取代LEV1）
SO2.1	给定一个方向，在哪个方向上移动工人（代替SO2，部分取代LEV4）
SO3	如果当前关卡是最后关卡，并且游戏者挤进了排行榜，则在排行中记录的名字
SO4	如果当前关卡是除第一关卡以外的其他关卡，记录游戏者不能进入排行榜
SO5	列出排行榜中所有游戏者
SO6	读指定关卡的描述
SO7	记录游戏者的名字
SO8	请求游戏者名字
SO9	设置当前关卡到第一关卡
SO10	设置当前关卡到任一关卡
SO11	设置当前关卡到初始配置
SO12	显示当前关卡
合作者：关卡	

值得讨论的决定是SO2怎样分配到关卡上。这种分配可能有很多方式应当被完成。因为这个案例研究不是尽量产生“最好”的设计，但应是一个“好”的设计，这里的焦点是决定是否合理，而不是最好。

Level(关卡)	
LEV1	报告关卡中的所有箱子是否在架子上
LEV2	设置到初始配置
LEV3	显示关卡的所有细节
LEV4	给定一个方向，如果在指定的方向上与工人邻近的位置是空的，或者包含一个箱子并且在指定的方向上与其邻近的位置是空的或者包含一个空架子，则移动工人，如果有一个箱子，箱子也移动
LEV5	读描述并且生成

在这个案例中，我们打算利用设计中的Level类去处理特定关卡中所有已经发生和能够发生的各方面问题。这包括跟踪工人在哪里，以及相关的各方面位置信息。这就是用例，它似乎合理地代表了SO2的大部到这个Level类。

迭代 3

这次迭代集中在Level类。类中术语Location的重复使用暗示者Location类应当有用。这有了在设计上的下面影响（Sokoban没有改变）。

Location(位置)	
LOC0	生成给定位置和内容
LOC1	在位置上显示细节（根据内容着色）
LOC2	指出内容（空，箱子，空架子，满架子，工人，墙）
LOC3	移动工人进出（根据指定）
LOC4	移动箱子进出（根据指定），如果内容是空的或满架子，则作适当改变

这个设计决定是一个Level将包含一个Location集。每个Location由一个位置（将被着色）和内容。保持对相关Location如何不同的跟踪一直到Level类。

Level(关卡)	
LEV1	报告关卡中的所有箱子是否在架子上(部分取代LOC2)
LEV2.1	读当前关卡的描述，根据此生成位置并指定位置和内容（代替LEV2，部分取代LOC0）
LEV3.1	显示每个位置的细节（代替LEV3，部分取代LOC1）
LEV4	给定一个方向，如果在指定的方向上与工人邻近的位置是空的，或者包含一个箱子并且在指定的方向上与其邻近的位置是空的或者包含一个空架子，则移动工人，如果有一个箱子，箱子也移动（部分取代LOC2,LOC3,LOC4）
LEV5.1	（代替LEV5，部分取代LOC0）
协作者：Location	

迭代 4

回到Sokoban类，我们注意到维护游戏者细节的需要，建议一个Player类。

Player(游戏者)	
P1	请求和记录名字，符合排行榜要求
P2	使符合排行榜要求
P3	报告名字
P4	报告是否符合排行榜要求

Sokoban	
SO1	如果所有箱子都在架子上，则进入下一关卡（部分取代LEV1）
SO2.1	给定一个方向，在那个方向上移动工人（部分取代LEV4）
SO3	如果当前关卡是最后关卡，并且游戏者挤进了排行榜，则在排行中记录的名字（部分取代P3和P4）
SO4	如果当前关卡是除第一关卡以外的其他关卡，记录游戏者不能进入排行榜（部分取代P2）
SO5	列出排行榜中所有游戏者
SO9	设置当前关卡到第一关卡
SO10	设置当前关卡到任一关卡
SO11	设置当前关卡到初始配置（部分取代LEV2）
SO12	显示当前关卡（部分取代LEV3）
协作者：Level，Player	

迭代 5

最后，我们生成一个HallOfFame类，来接管更多的Sokoban类的职责。这是我们这次案例研究中的最终设计。虽然可能还有其他的迭代，我们所做的完全是一个良好的开端；这是一个我们确信能满足需求和职责的设计，它似乎具有相当好的分布性。我们应当使用它作为一个开发其他技术应用的起点，比如职责驱动设计。

HallOfFame(排行榜)	
HOF1	读取游戏者名字（从SO3，SO5）
HOF2	增加游戏者名字（从SO3）
HOF3	写游戏者名字（从SO3）
HOF4	显示游戏者名字（从SO5）

Sokoban	
SO1	如果所有箱子都在架子上，则进入下一关卡（部分取代LEV1）
SO2.1	给定一个方向，在那个方向上移动工人（代替SO2,部分取代LEV4）
SO3	如果当前关卡是最后关卡，并且游戏者挤进了排行榜，则在排行中记录的名字（部分取代P3，P4，HOF1，HOF2和HOF3）
SO4	如果当前关卡是除第一关卡以外的其他关卡，记录游戏者不能进入排行榜（部分取代P2）
SO5	列出排行榜中所有游戏者（部分取代HOF1和HOF4）
SO9	设置当前关卡到第一关卡
SO10	设置当前关卡到任一关卡
SO11	设置当前关卡到初始配置（部分取代LEV2）
SO12	显示当前关卡（部分取代LEV3）
协作者：Level，Player，HallOfFame	

结束语

用例在系统开发的许多方面都是有益的，并且基本用例的细化最初用于定位用户界面设计中的需要。我们一直在探索基本用例在一般系统设计中的应用。

我们最初的动机直接与基本用例的方方面面相关。因为对话的使用，所以基本用例很好地支持角色扮演。因为避免了关于实现细节的不必要争论，所以它们是简洁的、能快速开发的。我们也发现了与系统开发的方方面面相关的深层优点。最重要的是，基本用例的抽象来源于标识用户意图和系统责任，并且这些都是对系统开发有启发式的帮助作用。

一些来源于基本用例的优点来自用户界面设计的传统。所以在需求分析中，他们强调交互对话，强调以用户或演员为中心以激发合理的交互。在这种方式下，基本用例可以提供与用户界面原型的早期开发等效的事物作为系统设计的输入。在本文中，我们略述了用卡片和角色扮演做需求分析和评审的使用基本用例的方法。

其他优点涉及到基于用例的系统设计。系统责任也可扩展到包含超过用户交互的责任，并且能证明重要的前后功能关系。通过系统责任的分布，决定了的责任也可用来作为设计的起点。在本文中，我们略述了使用责任分布直接从基本用例来开发面向对象设计的方法，并给出了一个详细的例子。该方法提供了可跟踪的优点和一些设计上的操作指南。

软件工程技术丛书

设计系列

企业应用架构模式

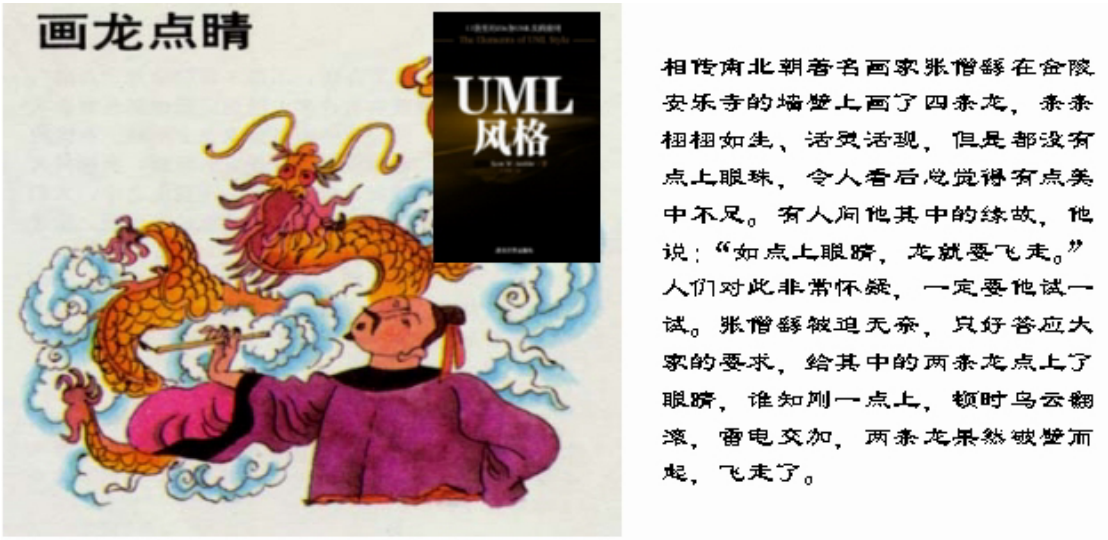
Patterns of Enterprise Application Architecture

(美) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校

UMLChina 训练辅助教材



它不讲概念，它假设读者已经懂了概念。

它不讲工具，它假设读者已经了解某种工具。

它不讲过程，它假设读者已经了解某种开发过程。

它只是在读者已经了解方法、过程和工具的基础上，提醒读者在绘制 UML 图时需要注意的一些细节。

在这本类似掌上宝小册子中，Ambler 提出了 200 多条准则，帮助读者在画龙的同时，点上龙的眼睛。

UML 重构浏览器

Marko Boger、Thorsten Sturm、Per Fragemann 著，elanzhou 译

讨论

摘要

像其他的敏捷过程一样，重构是极限编程的基石。自动支持的工具也已开始出现，一般是指重构浏览器。它们大部分是作为编辑器和集成开发环境的扩展，可以直接在代码层次上进行操作。本文讨论了重构的思想如何能被扩展到 UML 模型，并提出了一个集成在 UML 建模工具中的重构浏览器。也提出了一些对静态体系结构和动态行为模型的重构。

关键字：重构，重构浏览器，开发工具，UML

介绍

重构引起了广泛的注意，特别是在极限编程(XP)和敏捷过程社群。这个思想首先在 Opdyke 和 Brant 的工作中形成，由 Beck 推广，并由 Fowler 进行更深入的阐述。重构是在不改变软件的外部行为的情况下改善内部结构的技术或策略。XP 和其他敏捷过程建议采取两个迭代的步骤来开发软件。首先，应该实现要实现的行为，第二，在不影响其他行为的情况下优化代码的结构。这种方法，因为代码结构更简单，所以以后的变更也更容易，因此开发也就更敏捷。

重构描述了什么能被改动，不改动语义时如何改动，和改动时要注意哪些问题。重构浏览器有助于使刚才描述的步骤自动化并对可能的冲突发出警告。

到目前，重构经常被放在程序代码的上下文中讨论。所有的重构浏览器（据我所知）也是作用在代码上。尽管如此，重构自身经常由 UML 符号来解释。对我们来说，这引发了一个问题，重构是否能直接在模型层次上而不是代码层次上定义，重构浏览器能否在 UML CASE 工具的环境中而不是在集成开发环境下实现。本文阐释了我们的研究结果并讨论了我们的一些发现。

在第二节中本文讨论了哪些重构在模型层次上有意义，哪些重构在代码层次上没有意义但是在模型层次上有帮助。第三节描述了这些如何在工具中实现。第四节给出重构和重构浏览器是如何应用的例子。第五节以一个结论完成全文。

适用于 UML 的重构浏览器

双向工程已经发展到了可以把 UML 模型和程序代码看作是一样东西（后面简称为软件）的不同表达的程度。这样，重构的概念能被具体为改进软件结构的质量，而不仅仅是它的代码表示。

对一些重构来说，很自然的就能把它们应用在代码表示的层次。例如 *extract method*，将一个长的函数(method)分解，提炼出一些小函数，自然地应用在代码上面。其他的，像 *rename class* 或者 *move method up*，不管是应用在程序代码上还是在 UML 模型上，有着同样的效果。其他像 *replace inheritance by delegation* 或者 *replace type key with state or strategy*，看来似乎更适合于 UML 表示。

对后者定义各种类型的模型上的重构，并且为它们的 UML 工具提供重构浏览器，对面向图形界面的开发者更有益处。此外，因为重构现在是从代码中提炼出来的，如果应用到模型，在工具支持方面也许会有新的重构方法和利益。本文关注于这些应用于软件的结构信息的重构，而不是那些显然是从代码中来的重构。

冲突检测（Conflict Detection）

对可能的冲突进行检测是在重构过程中至关重要的一部分。Martin Fowler 倡导使用单元测试来防止不希望出现的副作用慢慢渗透到工作码（working code）中去。但是他留给用户去寻找在特定的情形下会出现什么副作用。

我们的工作关注冲突的自动检测。如果应用不得当，任何重构都会将很多错误带入代码中。因此下面讨论的每一个重构都被严密检查过，以便确定因为它的潜在使用而可能带来的冲突。

冲突分为警告和错误。警告表示模型在适当（well-formed）的状态下，重构可能会带来副作用。例如，对方法改名以便重载父类的函数，在某些情况下是保留了原来的行为的，但是对其他（类）来说带来的主要是不希望出现的设计变更。两种情况下都是可编译的。另一方面，错误表示操作将对代码或模型带来损害。

一些 UML 的重构很可能带来警告冲突，即使重构不会改变模型的行为。其他的主要报告主要的模型变更。我们已经得出结论：当所有的冲突都呈现给用户，用户必须可以忽略重构浏览器的警告冲突。这样他就能在单个重构不能而全部（重构）就可以保留模型的行为的情况下，一次执行好几个重构，或者通过其他方式来解决（警告冲突）。

静态结构的重构

UML 类图用来设计和可视化软件系统的静态体系结构。很明显一些面向代码的重构可以直接应用到类图。但是代码是一种线性的表达方式、不能很好的显示出依赖和结构，同时 UML 是二维和图形的表示方式。在类图中比在代码编辑工具中能更简单的表达出来对体系结构的改进。并且，确定要执行哪些可能的重构后，能在 UML 中更好的概览应用的结果。

因为面向对象软件结构的错综复杂的依赖关系，许多重构像重命名，删除，函数、类和属性的移动比仅仅是局部修改更有影响。通常继承和多态导致了这些问题。例如函数重命名，就有继承体系中的父类方法被（新名字）重载了或者不再被（旧名字）重载了的影响。

改变结构的重构，例如用委托关系来取代继承关系或者从一系列类中抽取出共同的接口，在模型层次上更明显。

但是很重要的是记住 UML 不仅仅是由类图组成的。类图或许是 UML 中最重要也的确是最常用到的部分，但是 UML 中不仅仅只有它。大部分重构应用在静态结构，并且它们的效果只在类图中很明显。然而，类图既不能表达动态的行为也不能表达业务过程或者业务需求。代码也不是表达软件的这些方面的最佳选择。

在几种不同类型的图的帮助下，UML 提供了表达这些的机制。软件不仅仅由代码组成，也包括软件的这些方面的观点正在传播开来。讨论完类图后，本文只限于讨论活动图和状态图。

状态机的重构

状态图通常用来表达‘类的操作集应该如何被使用’的协议。就是说，它定义了操作应该以什么顺序被调用，哪些操作的顺序是不允许的。对这样的协议进行重构意味着在不改变协议本身的同时改变协议的使用方式。

以下是一些状态图方面的重构的选集，我们按实现来划分：

状态合并（Merge States）用来把一系列状态组成一个。去掉了状态之间的内部转移，把外部转移重定向到新的状态，把内部转移和活动移到新状态中。如果多个所选状态和组外有转移，或者发现唯一的进入动作（entry action）没有在所选状态组的虚初始状态（virtual initial state）中，就得提出警告。

分解顺序复合状态（Decompose Sequential Composite State）把组合状态中包含的元素移出来然后将它们去掉。进入和退出动作被移动到合适的状态，如果需要的话，离开和到达组合状态的事务将被重定向或者被复制。如果在进入和退出动作被检测到或者活动（do-activity）将被删除时，就得提出警告。这种重构主要和 Form Composite State 一起使用。

组成复合状态（Form Composite State）创建一个新的复合状态并将选中的状态移进去。公有的转移被抽取到新状态的边界上，适当的缺省和完成状态被链接到初始状态和结束转移。如果没有缺省状态，或者有多个缺省状态或结束转移都适合时，就得提出警告。

序列化并发复合状态（Sequentialize Concurrent Composite State）为并发状态创建产品自动机并将并发状态包含的元素移出来。它主要和状态合并（merge states）一起使用，以便在保持行为的同时简化复杂的模型。这种重构只适用于良构的（well-formed）的并发状态，所以在不是由 fork 直接产生（并发）的地方，会做初始状态校验。

活动图的重构

活动图经常被用来描述业务过程。它们也能被用来以更图形化的方法来描述算法。但是在算法通常更适合于用代码来表达的同时，没有充分的方法来表达业务过程。对业务过程获得清楚的理解的价值是不用置疑的，活动图是对它们建模的一种很好的方式。对活动图的重构不会改变过程而仅仅是改变它在模型里的表现形式。这里有一些已实现了的重构的选集：

让活动并发（Make Actions Concurrent）创建一个 fork 和 join 的伪状态。并将数组顺序执行的活动放到两个状态之间，这样使它们能够并发执行。重构过程通过检查“组之间有活动转换”或者“组中有状态不能通过合适的转换路径到达组的结束状态”等错误，来检查更改是否最终导致良构的（well-formed）模型。如果一个组对一个只能被其他组访问的变量有写操作，就得提出警告。

Sequentialize Concurrent Actions（将并发活动顺序化）模式去掉 fork 和 join 伪状态对，并把里面各组活动状态一个个连起来。如果有发现公用变量，将会产生警告，（一个真正的并发算法并不能简单的顺序化），检查包还会检查非良构的转换。

用于 UML 的重构浏览器

上文描述的重构已经在 UML 重构浏览器中实现了。一些实现方面和开发接口的描述将在本节阐述。

UML 不仅仅是一套图形符号，UML 工具也（通常）不仅仅是专门的画图工具。代码编辑器通常仅仅是高级的文本编辑器，而 UML 编辑器已经带有丰富的语义内涵，是知识库，至少一些工具甚至是基于标准的 UML 元模型。能在元模型结构上开发和操作对 UML 模型的重构。这使得 UML 重构浏览器的实现比基于代码的更进了一步，因为基于代码的不存在预先定义的元结构。

本文描述的实现已经作为 Gentleware tool Poseidon for UML 的一部分完成了。这个工具是源于开源项目 ArgoUML，其知识库是从 UML1.3 元模型直接产生出来的。重构是作为在此模型上的控制者（controller）来实现的。

大部分的基于代码的重构浏览器的用户界面简单实现为上下文菜单（context menu）。我们的选择不同，能够在使用图表和浏览区的同时看到（重构）区。

第一部分，将会基于用户当前的选择来给予重构的提示。例如，如果选择了一个函数，将会提示函数重命名（rename method）。如果同时选择了父类，将会给予函数上移（move up method）的重构建议。第二，选择一种重构，就在显示参数输入框（例如方法的新名字或方法所在的应用程序）的同时，显示该重构的简短描述和重构的效果。第三，会展示执行重构将会带来的可能的冲突。它也有执行重构和撤消重构的按钮。

示例

对 UML 模型进行重构是怎么样帮助提高设计质量的呢？使用重构浏览器是如何能通过指出可能的冲突来帮助避免重构的缺陷的呢？下面举出一些小例子来说明。所有的场景都是从 Gentleware's Poseidon for UML 附带的例子中拿来的。一个假想的叫做 Softsale 的公司在互联网上销售数字产品。示例 UML 模型对订单和客户关系的处理进行了建模。在本文中，仅限于对新客户的验证过程建模举例。

UML 模型的一部分是对客户自己进行建模。有两个类，Client 与 DeliveryAddress 之间有两个关系，任务名（role name）分别为 deliverAddress 和 invoiceAddress。如图 1：

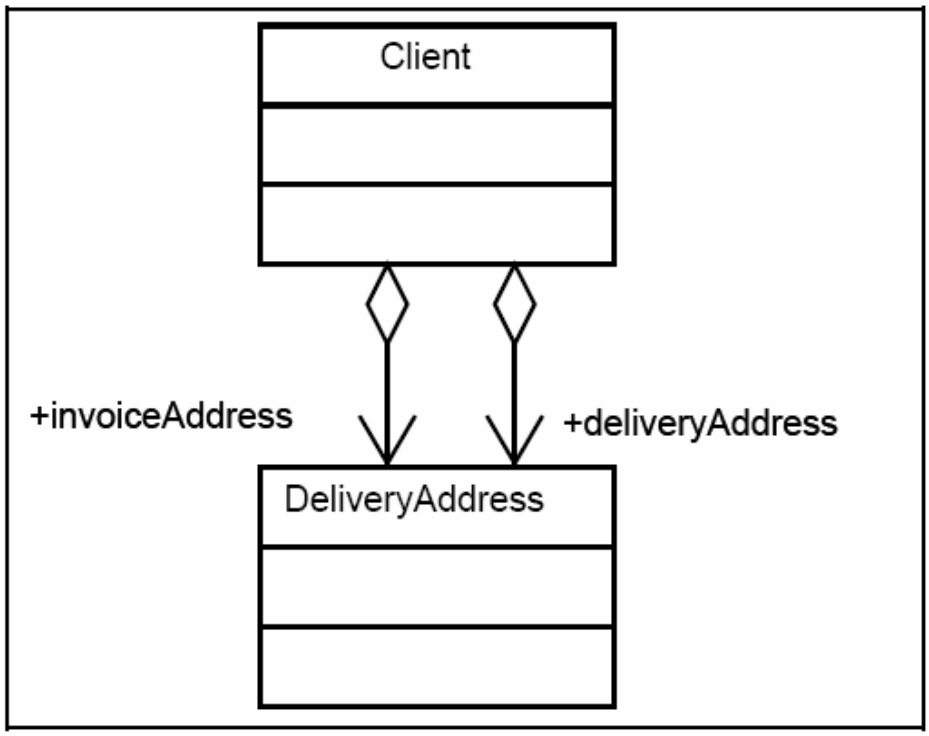


图 1 对类重命名前的类图

其中一个 role name 是和类名本身一样。这样看来 DeliveryAddress 这个类名不是很合适。使用 rename class 重构，把类名改成 Address，这样就更好的反映出它的用法。因为这是一个非常简单的重构，不会出现冲突。如图 2：

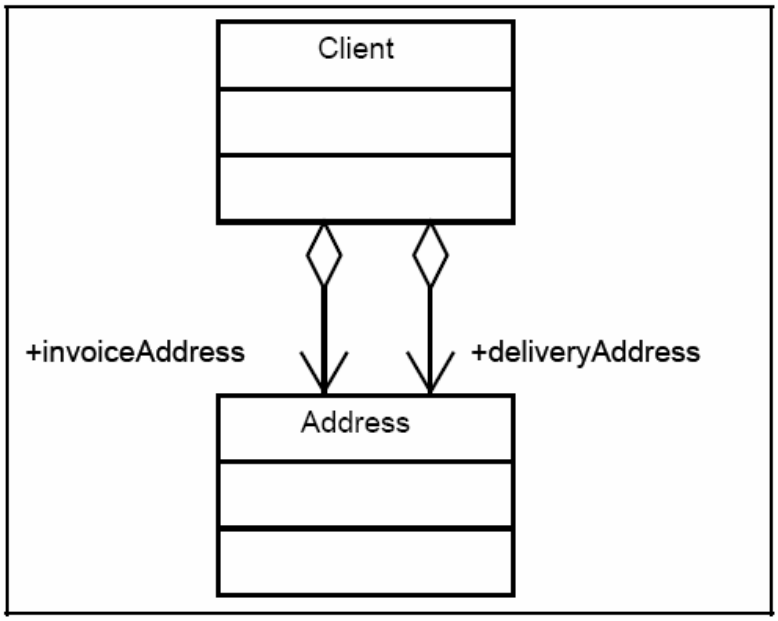


图 2 对类重命名后的类图

新用户的投递地址的验证过程是从请求适当的地址并对接收到的信息进行验证开始的。因此，地址从 *not verified* 状态开始，经过验证过程（*requested, retrieved*），并根据验证结果，返回到 *not verified* 或者移动到 *verified* 状态。如图 3：

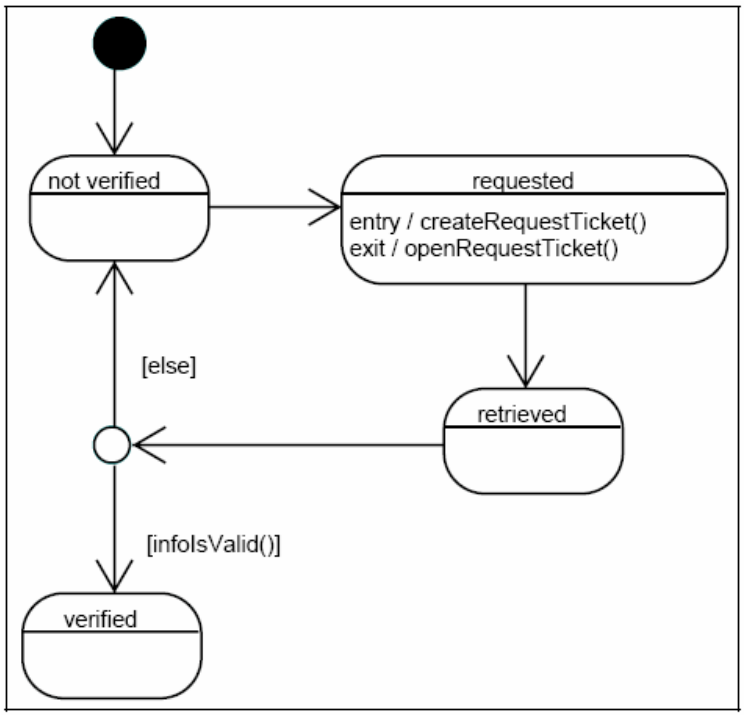


图 3 merge state 前的状态图

状态 *requested* 和 *retrieved* 是顺序执行的，合并它们不会改变处理逻辑。首先使用 merge state 的重构方法将它们合并到名为 *requested, retrieved* 的新状态。为了更好的反映它的目的，第二步把新状态的名字改成了 *verifying*，在第二步中没有使用重构浏览器（如图 4）。

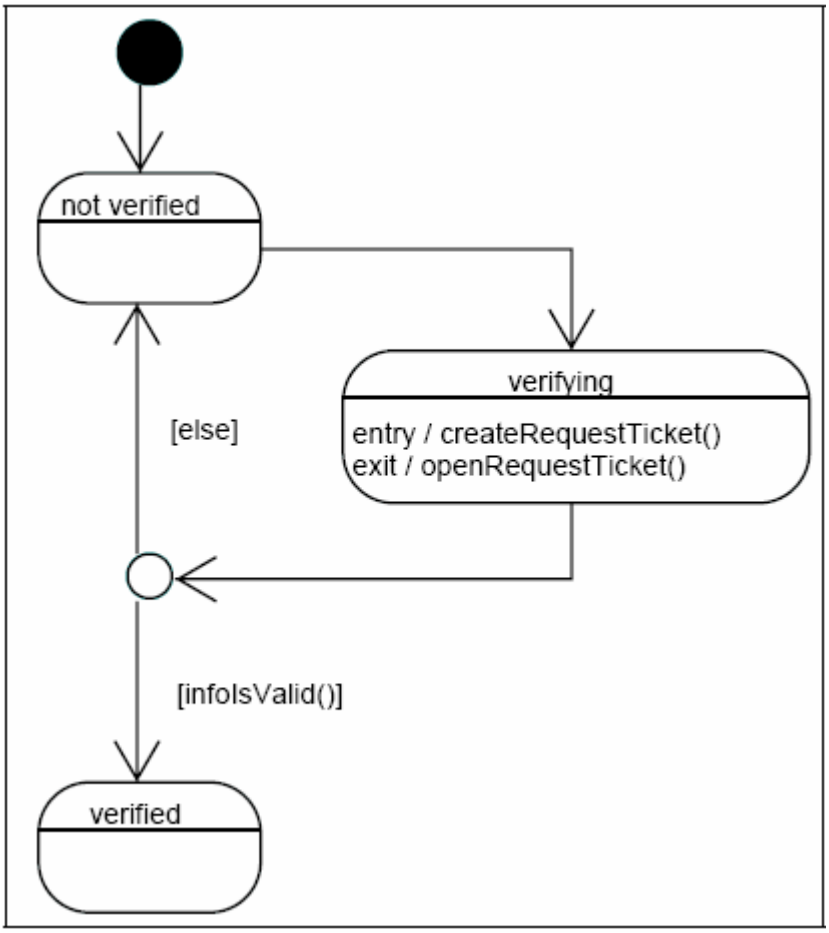


图 4 merge state 后的状态图

requested 状态有一个进入动作和一个退出动作，其中退出动作在重构过程中引起了冲突，这在重构浏览器中有显示，还有一段简短的解释，如图 5：

图 5 工具发出的警告

现在就需要用户来考虑是重新审视想要做什么或者等执行重构以后再解决这个冲突，因为退出动作也能放在合并后的状态。

新用户验证的整个过程在一个活动图中表达。验证从验证 email 地址开始，跟着是验证邮政地址。如图 6：

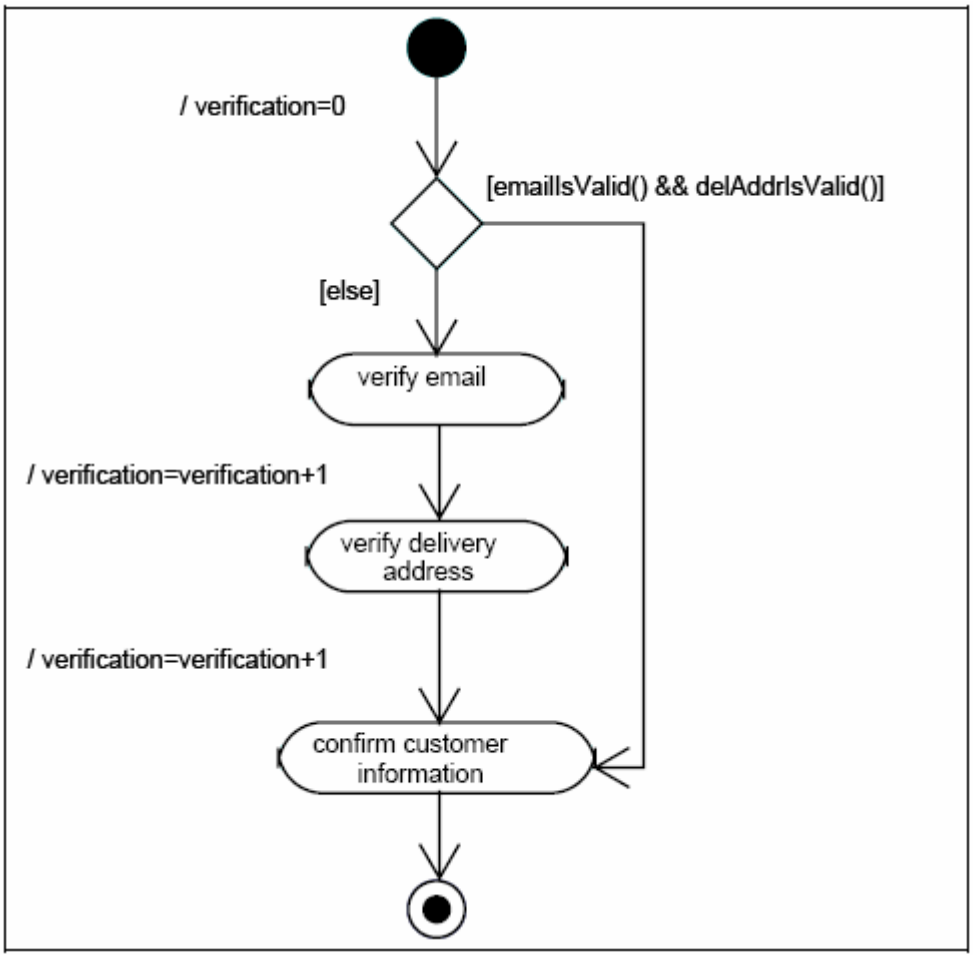


图 6 make actions concurrent 前的活动图

因为两个验证处理部分相互之间没有依赖，使活动并发重构能用来改善模型，如图 7：

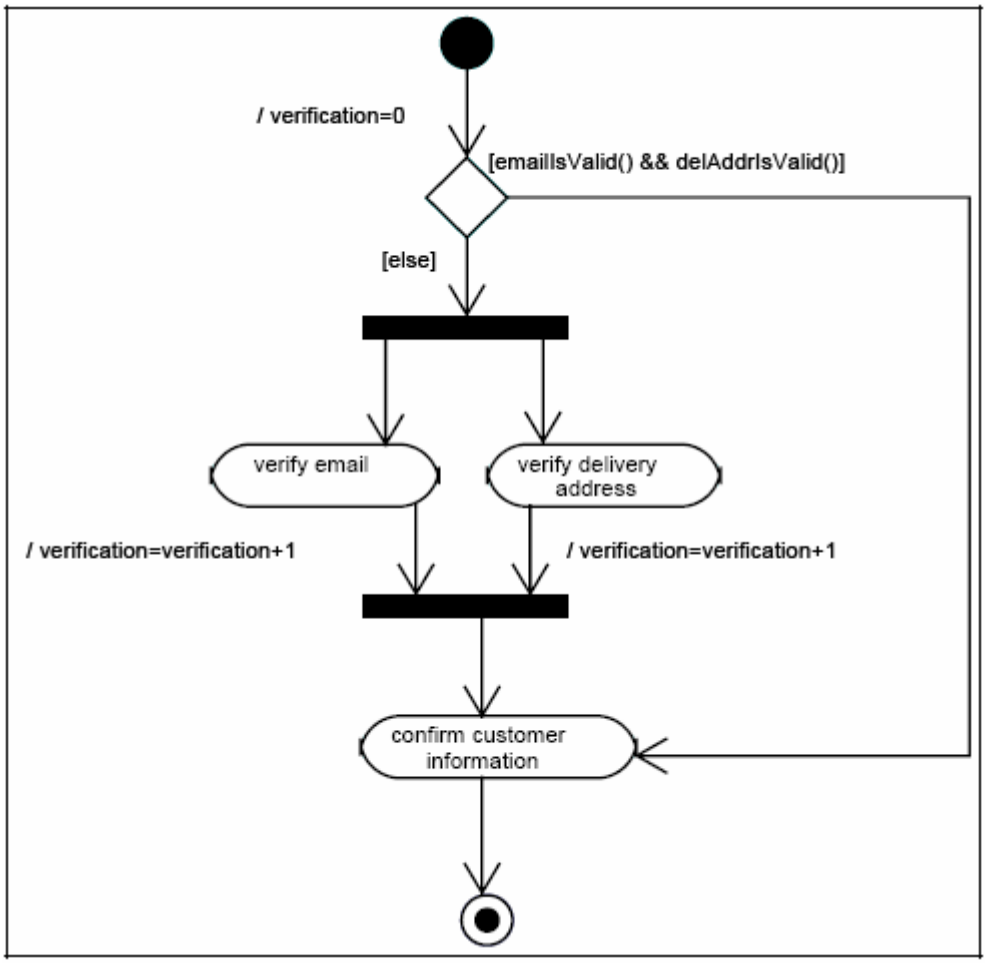


图 7 make actions concurrent 后的活动图

因为活动 *verify email* 和 *verify delivery address* 都使用了变量 verification，重构浏览器检查到了冲突。因为变量 verification 是递增的，我们能忽略冲突并继续重构。

图 8 工具提出的警告

上面的例子显示，不光是在模型静态结构上，在状态和活动图上的重构，都有改善（模型）的空间。

结论

重构作为在不改变代码行为条件下改善代码结构的有条理的过程，也能应用于 UML 模型。许多已知的重构能直接转移到 CASE 工具中实现。查找应用于静态结构的可能的重构，在二维的图形表现形式中会更简单些。另外也能找到一些新的活动图和状态机上的重构方法。

上文描述的工作已经在一个示范原型中得到了完全的实现。详见参考资料 8。它将开发成 Gentleware 公司发布的 UML CASE-tool Poseidon for UML 的插件。

参考资料

1. ArgoUML. ArgoUML, Object-oriented design tool with cognitive support, Open Source Project, at http://www.argouml.org

2. Beck, Kent. Extreme Programming explained: Embracing Change. Reading, Mass., Addison-Wesley, 1999

3. Boger, Marko et al. Extreme Modeling, in Succi, Gand Marchesi, M. Extreme Programming Examined, p.175, Addison-Wesley, 2000.

4. Boger, Marko and Sturm, Thorsten. Tools-support for Model-Driven Software Engineering, in Evans, A. etal. Proceedigs of Practical UML-Based Rigorous Development Methods. Workshop at <<UML>>2001 conference. Gesellschaft für Informatik, 2001.

5. Brant, John and Roberts, Don. Refactoring Browser, at http://stwww.cs.uiuc.edu/~brant/RefactoringBrowser.

6. Fowler, Martin. Refactoring, Improving The Design of Existing Code, 1999.

7. Fowler, Martin. Refactoring Home Page, at http://www.refactoring.org

8. Fragemann, Per. Refactoring von UML-Modellen. Diploma Thesis (to be published). University of Hamburg, Germany, 2002.

9. Gentleware AG. Poseidon for UML, a UML CASE tool based on ArgoUML, at http://www.gentleware.com.

10. Object Management Group. Unified Modeling Language, Version 1.3, at http://www.omg.org.

11. Opdyke, William F. Refactoring Object-Oriented Frameworks. Dissertation, University of Illinois at Urbana-Champaign, 1992.