第 23 期（2003 年 3 月）

【新闻】

1 JBuilder套件用Together Modeler取代Rose/RUP…

【方法】

4 UML序列图开发

21 过程模式的UML描述

37 小项目的RUP应用

52 把业务对象连接到关系数据库 （上）

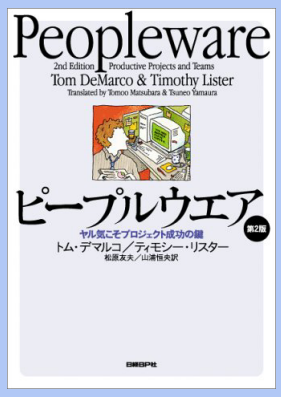
79 现实世界电子商务系统的数据库设计

【人件】

86 人的问题：关于《人件》

92 开发人员是人吗？

95 办公室空间，下一场革命



X-Programmer 非程序员

软 件 以 用 为 本

投稿：editor@umlchina.com
反馈：think@umlchina.com

http://www.umlchina.com/

本杂志免费下载，仅供学习和交流之用
转载需注明出处，不得用于商业用途

《非程序员》1-22 期目录分类索引

IT 之源 张恂 整理

文章	作者、出处	译者	期号
UML 概述			
傻姑之路：通往职业初段—前言	think	-	21
家用报警器的 UML 设计	M.W.Richardson	liuweiw	20
驳 UML 三大“硬伤”论	张恂、think	-	18
电梯系统的 UML 文档	Lu Luo	王君	18
用户界面的 UML 建模	Paulo Pinheiro da silva, Norman W. Paton	李巍	17
使用 UML 和 Rhapsody 开发导航控制系统	G. R. de Boer	ottafei	16
如何用状态图进行设计	Dr. Doron Drusinsky	Jill	15
用 UML 描述工作流管理系统规约	Pavel Hruby	wind.deng	10
使用模式集成 UML 视图	Alexander Egyed	davidqql	7
使用 UML 设计数据库应用	Michael Blaha	张启鹏	5
数字录音机设计方案	Ivan Porres Paltor，Johan Lilius	乐林峰	5
用 UML 设计 Java 应用程序	Hans-Erik Erikkson, Magnus Penker	刘忠	2
OO 方法			
Web 应用模型中的抽象和复用机制	Gustavo Rossi	ydr	21
致面向对象技术初学者的一封公开信	Alistair Cockburn, 1996/02	袁峰	19
轰然巨响	Martin Fowler	Jady	18
CRC 建模方法－－跨跃开发者与用户之间的交流障碍	Scott Ambler	huang_shen	18
重构过程中的行为保持	William Opdyke	透明	17
根据合同进行分析--录像店案例研究	Richard Mitchell	zhen_lei	16
处理对象的特性	Martin Fowler	Xu Zhiling	16
对象-关系数据库之间的映射	Scott Ambler	Adams Wang	15
利用重构改进面向对象设计	Lance Tokuda, Don Batory	koun	14
建模鸡汤	Scott Ambler	乐林峰	6
面向对象分析和设计技术	Martin Fowler	easehawking	5
需求工程			
用例：十年风雨	Alistair Cockburn	think	22
通往职业初段—使用用例组织需求(上)	think	-	22

为什么用例如此难用?	Gary K. Evans	zhen_lei	13
面向对象开发中的本质用例及其职责	Robert Biddle	zhangxxin	11
利用角色扮演和用例卡片进行需求复审	Robert Biddle	caorui	11
怎样避免用例陷阱	Susan Lilly	qian_x_j	5
勇于直面需求变更	Windy. J	-	4
用例的使用误区:管理需求之一	Dr. Timothy Korson	James Zhong	4
构造用例过程:管理需求之二	Dr. Timothy Korson	June Fourteenth	4
创建有用的用例	Dr. Timothy Korson	June Fourteenth	4
用 Use Cases 捕获需求	Pete McBreen	苏康胜	1
模式			
分析模式应用——帐务模式	Windy.J	-	22
分析模式学习笔记:LOG—日志记录模式	Windy.J	-	17
分析模式学习笔记之四（下）	Martin Fowler	Windy J	11
模式的讨论 FAQ	Doug Lea	Qian_x_j	3
《分析模式：可重用对象模型》前言和介绍	Martin Fowler	Windy J	
《分析模式：可重用对象模型》学习笔记之二：责任模式	Martin Fowler	Windy J	2
《分析模式：可重用对象模型》学习笔记之三：观察与测量	Martin Fowler	Windy J	2
分析模式学习笔记之四	Martin Fowler	Windy J	4
预订和使用可重用实体的分析模式	Eduardo B. Fernandez, Xiaohong Yuan	Shane	7
业务资源管理模式语言	Rosana T. Vaccare Braga	zhen_lei	12
对于模式的"十大误解"	John Vlissides	透明	10
用创建方法封装子类	John Vlissides	透明	12
提取创建类	Joshua Kerievsky	透明	14
用创建方法取代多个构造子	Joshua Kerievsky	透明	9
停止过分设计！	Joshua Kerievsky	透明	13
设计模式之路(一)	Jun Qian	-	14
软件设计模式的非软件例子	Michael Duell	wuccc	12
设计模式的理解	Adams Wang	-	8
Abstract Class 模式	Bobby Woolf	透明	3
Alternator 模式	John Liebenau	透明	3
Authenticator 模式	F. Lee Brown, Jr.	透明	3
Chain Constructors	Joshua Kerievsky	透明	8
Document-View-Presentation 模式	Ku-Yaw Chang	透明	3
Role Object 模式	Dirk Bäumer	透明	3
Matcher-Handler 模式	Frank Metayer	透明	3
Reactor 模式——同步事件复用和处理调度的对象行为模式	Douglas Schmidt	Tonny Tam	16

Java 中的 Singleton（上）	石一楹	-	6
Temporal（时效） 模式	Andy Carlson, Sharon Estepp, Martin Fowler	透明	6
构建 EJB 应用—模式集合（上）	Eberhard Wolff	杨德仁	18
构建 EJB 应用—模式集合（下）	Eberhard Wolff	杨德仁	19
电子商务应用系统的几种模式	Dragos A. Manolescu, Adrian E. Kunzle	杨德仁	17
网上商店的模式	Eduardo B. Fernandez	Huang Yin	11
一种在线拍卖管理的模式语言（上）	Paulo C. Masiero	vicky wei	18
一种在线拍卖管理的模式语言（下）	Paulo C. Masiero	vicky wei	19
仓库管理器：一个库存的分析模式	Eduardo B. Fernandez	邓克	7
一种关于物品修理的分析模式	Eduardo B. Fernandez, Xiaohong Yuan	刘巍	19
一个产品的订货和配送（系统）的分析模式	Eduardo B. Fernandez, Xiaohong Yuan, Sandra Brey	li.sining	14
保险系统的部分模式	Wolfgang Keller	liwenhua	15
GUI 设计			
GUI 设计精髓：交互	Weinschenk	Ma Yaohua	21
界面耻辱纪念堂－术语	iarchitect	金哲凡	17
界面耻辱纪念堂--颜色的使用	iarchitect	freeyourmind251	15
界面耻辱纪念堂--隐喻使用不当	iarchitect	金哲凡	14
界面耻辱纪念堂--可视元素	iarchitect	wanghongbao	14
糟糕界面集锦－控件篇	iarchitect	bucher	10
用户界面软件	Jens Coldewey	Tom.X	17
隐喻的使用及误用	Larry L. Constantine	whoisred	13
发明软件	Larry L. Constantine	zhen_lei	12
GOF 模式用于 GUI 设计	James Noble	cntang	12
用户界面设计从抽象到实现	L. Constantine, Helmut Windl, James Noble, Lucy Lockwood	Lin George	12
针对用户界面设计的用例结构和式样	L.Constantine,Lucy A.D.Lockwood	zhen_lei	11
返璞归真：通过简化用例来简化用户界面	L.Constantine,Lucy A.D.Lockwood	zhen_lei	10
用户界面的交互模式	Martijn van Welie, Hallvard Tretteberg	James Zhong	10
掌握可用性规则	Lucy Lockwood	Nothing	4
用户需要什么-软件的工程可用性	Larry L. Constantine	Huang Yin	2

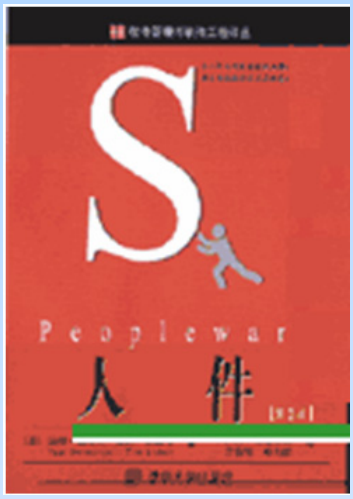
用户需要什么-软件的工程可用性（二）	Larry L. Constantine	Huang Yin	3
敏捷过程			
过程敏捷性和软件可用性：以使用为中心的轻量级设计	Larry L. Constantine	huang_shen	21
使用模式和 XP 构造复杂的面向对象系统	Eduardo B. Fernandez	曲俊生	20
XP 的价值和局限	张恂	-	15
大型项目中的 XP－开发者角度	Amr Elssamadisy	simaetin	13
极端编程中"坏气味"的发现与响应	Amr Elssamadisy，Dr. Gregory Schalliol	herman	6
UML 与 XP	Alan Cameron Wills	刘巍	22
模式与 XP	Joshua Kerievsky	Gigix	7
过程改进			
通过 CMM 4 级和 5 级的组织	SEI, 2002/05	shiyy	13
通过 CMM 评估的战略	Victor Stachura	zhoufang	9
在小型软件开发组织中使用 CMM	Mark C. Paulk	张俊	4
深刻理解 CMM－成功的关键	蓝尔公司	-	2
参与变革	Lisa J. Roberts,SDMagzine,2000/11	mirnshi	1
项目管理			
功能点过程	Adams Wang	-	6
项目经理面试指南（上）	Patricia L. Ferdinandi	zhoufang	7
项目经理面试指南（下）	Patricia L. Ferdinandi	zhoufang	8
项目管理规范-RUP 管理实施 (1)	李杰	-	2
项目管理规范-RUP 管理实施（二）	李杰	-	3
项目管理规范-RUP 管理实施（三）	李杰	-	4
项目管理规范-RUP 管理实施（四）	李杰	-	5
项目管理入门	Karl E. Wiegers	mirnshi	3
成功项目管理的秘密	Karl Wiegers, SDMagazine	Shids	2
更好地领导一个项目的诀窍	Warren Keuffel,SD Magazine,1999/09	think	1
创建成功的工程	Bruce Eckel	Hairui	1
回顾过去，展望未来	Karl Wiegers，Johanna Rothman	亚玲	6
统一变更管理的威力	Brian White	James Zhong	5
别把开发人员当成牲口	Mike Gunderloy	think	22
读书			
设计死亡了吗	Martin Fowler	?	15
总把新桃换旧符——写在《最后期限》	透明	-	21

出版之前			
《最后期限》：一本软件开发小说	?	Huang Yin	21
《最后期限》各章精选	Tom Demarco	透明	21
软件项目管理小说《最后期限》（草稿）节选	Tom Demarco	透明	9
Frederick Brooks 传	?	?	20
《人月神话》软玉生香	刘天北	-	20
《人月神话》与实践	Adams Wang	-	20
《人月神话》20 周年纪念版评论集	……	陈懋成	20
《人月神话》20 周年纪念版评论集	……	陈懋成	16
《人月神话》争论：软体尚方宝剑何在	欧阳进	-	20
《人月神话》出版后的网友评论	ChinaPub, cnforyou	-	20
《人月神话》节选	Frederick Brooks	Adams Wang	19
《人月神话》节选	Frederick Brooks	Adams Wang	7
《人月神话》各章精选	Frederick Brooks	汪颖	20
烧毁这本书，别让员工看到—《人件》评论集	Tom Demarco, Tim Lister	柳林	18
《人件》节选	Tom Demarco, Tim Lister	方春旭、叶向群	11
《敏捷软件开发》翻译草稿样章	Alistair Cockburn	Jill	16
《面向对象项目成功之道》（草稿）节选	Alistair Cockburn	乐林峰	8
国内出版的软件工程书籍一览	UMLChina	-	1
I-LOGIX 公司 BRUCE POWEL DOUGLASS 在实时设计模式领域的新书	I-Logix	?	20
访谈			
Jutta Eckstein：没有客户就没有项目	UMLChina, 2002/08	曲俊生	21
钱五哥答疑录（一）	UMLChina, 2002/11	-	20
高焕堂：何谓世界软件之神	UMLChina, 2002/10	-	19
高焕堂答疑录（一）	UMLChina, 2002/10	-	19
高焕堂：恢复中华民族设计自己产品的信心	UMLChina		8
尤克滨：简单正是用例的价值	UMLChina, 2002/10	-	19
Marko Boger： XP、UML 和 Poseidon	UMLChina, 2002/08	李巍	17
Peter Merel：妻子告诉我该睡觉了	UMLChina, 2002/08	extreme	17
David Van Camp：模式、构架和 XP	UMLChina, 2002/03	李文华	16
Bruce Powel Douglass：实时系统和 UML	UMLChina, 2002/06	Windy.J	15
Alistair Cockburn：选择你所需要的	UMLChina, 2001/09	谭红霞	13
James J. Odell：UML 的将来	UMLChina, 2002/04	think	13
Alan Shalloway：面向对象设计的新视角	UMLChina, 2002/04	gigix	13
Scott W.Ambler：空手道和太极拳	UMLChina	think, gigix	12

Roger S. Pressman：CMM 不是宗教	UMLChina	钱五哥	10
Martin Fowler 访谈	UMLChina	gigix	10
Kent Beck：只是一种正确做事的方法	UMLChina	Areca Chen,think	9
Alan Cooper：垃圾，都是垃圾！	UMLChina	dewen	9
Ivar Jacobson 访谈	Adriano Comai	Tang Xiaoming	8
交互设计之父 Alan Cooper	uidesign.net	亚玲	7
《设计模式》作者 John Vlissides	UMLChina	透明，think，Qingrun	6
Dassault Aviation 为业务的成功实施面向对象技术和 UML	Nasser Kettani, Rose Architect,1999/06	think	1
新闻			
年度回顾：开发工具的 UML/MDA 趋势	vnunet	风自由	21
Visual Studio .Net 2003——“Whitehorse”蓄势待发	CRN	风自由	21
《人月神话》与软件开发基本问题研讨会在京举行	……	-	21
IBM 收购 Rational	Rational	think	20
建模动力:UML2.0 使模型驱动的开发更加容易	computerworld	袁峰	
Borland 以 1.85 亿美元收购 UML 工具厂商	adtmag	think	19
Chick-fil-A 统一使用 Together ControlCenter 进行 Java 开发	Togethersoft	风自由	22
Sun 整合第三方 UML 建模工具	infoworld	think	22
SteelTrace 公司集成 Catalyze 套件与 Rational Rose	tmcnet	卓锐	22
CASE 工具			
使用 Rational Rose 全景追踪	Richard Felsinger	盖九宇	13
Rational ClearCase LT 使用指南	Jim	-	13
CASE 工具赛马	Gary K. Evans	张启鹏	6
UML 相关产品价格	ObjectsByDesign	Cliff	2
选择一种 UML 建模工具	ObjectsByDesign	think	1
其他			
程序调试的智力游戏	David Burns	史彦军	8
为什么还不编码？	?	张俊	8
威赛儿商务通系统开发员手册	苏康胜	-	4
千年决心	Scott Ambler	abug	1

《非程序员》文章仅作为学习参考之用，不得用于商业用途

《人件》：老板，别把开发人员当成牲口!!!



Tom Demarco，Tim Lister

翻译：UMLChina 方春旭 叶向群

http://www.peoplewarecn.com

人件中文版网站

Frederick P. Brooks, Jr--《人月神话》第 19 章

近年来，软件工程领域的一个重大贡献是 DeMarco 和 Lister 在 1987 年出版的《人件》，我衷心地向我的读者推荐这本书。

Edward Yourdon

《人件》在 1987 年出版后，立即成为最畅销的作品。当人件第一版出版时，我写了一份评论，“我强烈推荐你买一份人件给你或你的老板，如果你是一个老板，那么为你部门的每一个人买一份，并给自己买一份”。这建议在 12 年后的第二版依然有效，并且更加热烈。

Mark A. Herschberg

我只学了办公环境的章节，就辞掉了原来的工作！

Joel Spolsky

我想，微软成功的原因之一就是公司里的所有经理都读过《人件》

Steve McConnell

由于本书第 1 版的赫赫声名，新版的《人件》是我不用看就会决定购买的少数几本书之一。

中文译本即将发行！

业务流程领域的 Popkin-Intalio 联队

[2003/2/11]

周二，企业建模开发商 Popkin 软件公司和 Intalio 公司宣布了双方的合作：将 Popkin 的建模工具和 Intalio 的业务流程执行引擎整合在一起。

该方案将把 Intalio 的 n/3 业务流程管理系统[BPMS]和 Popkin 的 System Architect 工具集整合在一起。

通过此项合作，已经采用了 Popkin 建模软件的公司实际上将具备执行模型和转换模型的能力，模型的执行通过图形化映射到后端的 IT 系统来实现。

据官方消息，采用新的解决方案，Popkin 的 System Architect 可以由流程模型产生输出的 BPML,而不论该模型是采用 IDEF（一种结构化过程建模和分析技术）这样的传统技术还是新定义的 BPMN 创建的，然后，BPML 可以在 Intalio n/3 Designer 上执行或者部署到 Intalio n/3 Server 上去。

Intalio 和 Popkin 的合作意味着业务流程管理迈上了一个新的台阶，Intalio 的官方宣称，通过使用这些集成的产品，用户可以应用 System Architect 来为他们的流程建模，然后利用 Intalio 的技术来得到流程部署的方法。

位于纽约市的 Popkin 提供业务流程建模软件已有十数年之久。按照其 CEO Jan Popkin 的说法，该公司同时也是 BPMN 标准的开创者。

而 Intalio 则是 1999 年才创建的相对要新一些的公司。一月，该公司宣布了其 n/3 BPMS 第二个周期，同时也宣布了和 Popkin 的合作。

Intalio 同样和 Intalio 的竞争对手 IDS Scheer AG 有着紧密的联系。

Popkin 和 Intalio 将在下个季度正式发布其联合解决方案。

（自 eweek，风自由 摘译，不得转载用于商业用途）

JBuilder 套件用 Together Modeler 取代 Rose/RUP

[2003/3/1]继以 1.85 亿美元购买 TogetherSoft 之后的一个月，Borland 开始集成这两家公司的产品线。第一个成果：为 Borland’s JBuilder IDE 设计的 Together 建模工具最新版本，一个升级的 Enterprise Studio for Java suite，其中，用 Together modeler.取代了 Rational Rose 和 Rational Unified Process。

新的建模工具叫 Together Edition for JBuilder,，价格为$3,999-per-seat，作为 JBuilder 的插件。TogetherSoft 已经为 IBM 的 WebSphere Studio 提供了类似的产品。包括带有同时双向工程的 UML 建模，支持设计模式和审核、度量。

Borland’s Together 业务部门的首席科学家，前 TogetherSoft,副总裁 Todd Olson 认为，开发人员仍然继续使用 JBuilder IDE 和 Together 建模界面，二者的界面之间应该无缝连接。“你有两个应用在运行，你使用 JBuilder 写代码，使用 Together 画图，奇妙之处是这两者之间是同步的”。

Olson 把 Together Edition for JBuilder 和 Together Control Center（TCC）作了对比。“TCC 是一个为软件架构师设计的单独的应用，它能够支持多种语言，让架构师能更关注高层设计工作。Together 则集中于需要购买 JBuilder 的 Java 开发者，TCC 和代码环境的集成度没有 Together Edition for JBuilder 那么高。”

新版本的 Together 是 Enterprise Studio 5 for Java 的焦点。发布于 2002 年的 Studio 4，包含 JBuilder 7，Optimizeit 4.2 和 Borland’s Enterprise Server J2EE 应用服务器的开发版本，还有 Rational Rose 的 Java 专用版本和 RUP。它同时也包含 Borland 的 JdataStore 数据库和 Macromedia Dreamweaver MX Web 开发工具，可以用来写 Java，ASP 和 JSP 页面。

现在已经上市的 Enterprise Studio 包括 JBuilder 8，Optimizeit Suite version 5.0。另外，Borland 抽走了 Rational 的建模工具，代之于 Borland 自己的 Together Edition for JBuilder，并拿走了 Dreamweaver。

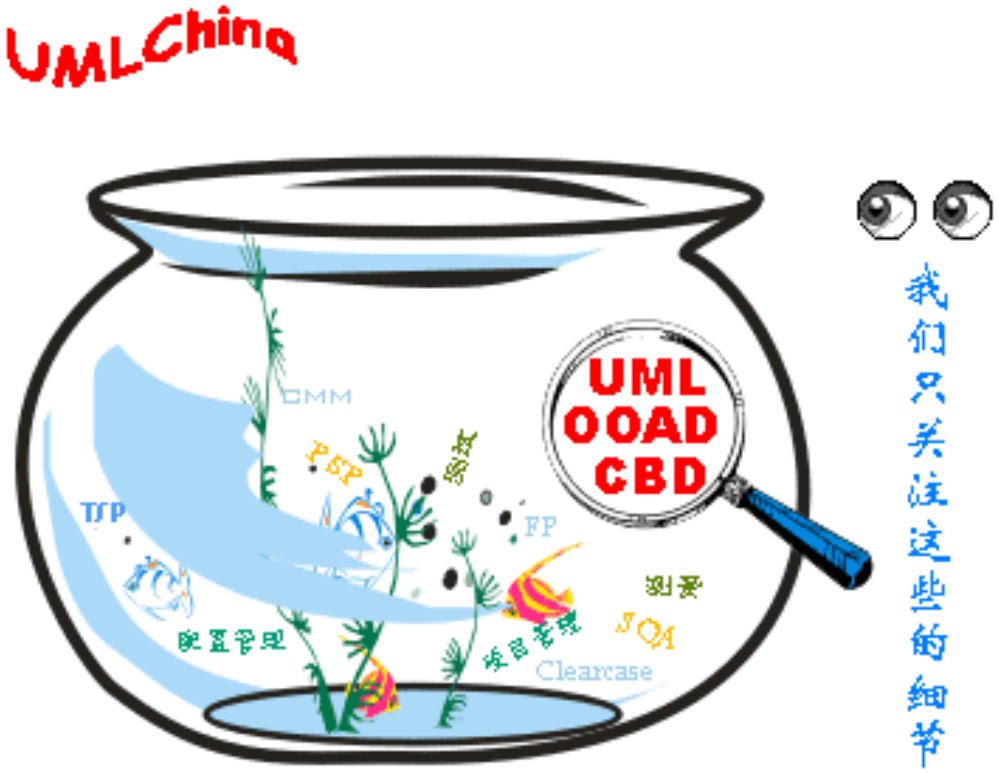
挪走 Rational 产品情有可原，因为这是 TogetherSoft 的直接对手，但为什么拿走 Dreamweaver？Java 业务部的产品经理 Bill Pataky 说，“我们的研究发现 Borland Enterprise Studio 的 Java 用户通常已经有了这样的工具，可能是 Dreamweaver，也可能是 Dreamweaver 的竞争产品”。

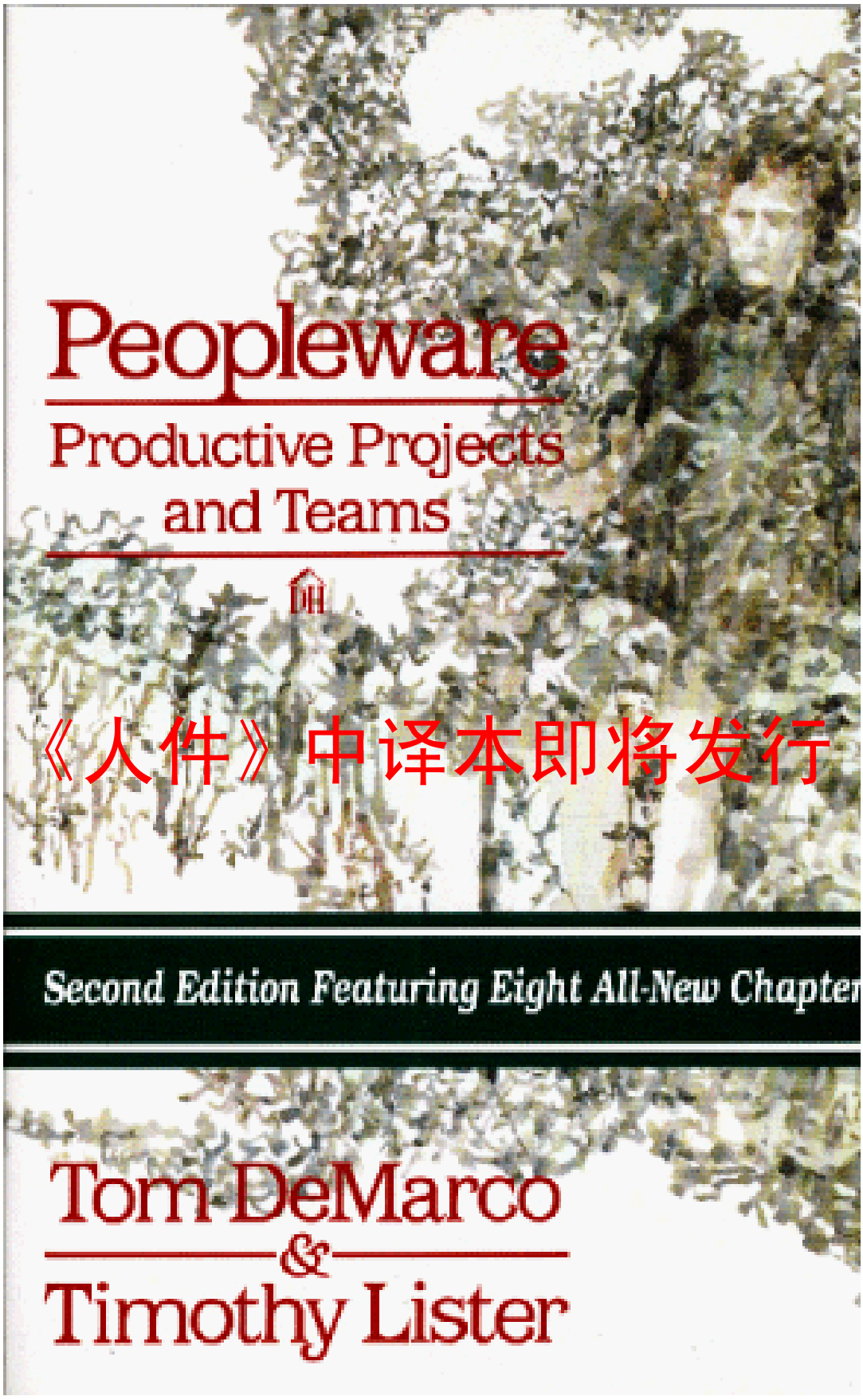
“比起 Rose，Together 结合得更紧密”，Borland 主管 Java 业务的副总裁 George Paolini 说。这个套件也和 CaliberRM 集成。CaliberRM 是 Borland 在 2002 下半年从 Starbase 购买的需求管理工具。这样就允许你从 CaliberRM 到 Together 再到 JBuilder 跟踪需求。

随着这次升级，Enterprise Suite 的价格也从$5,999 升到了$6,999 per seat。

别了，VISUALCAFÉ

Borland 已经停止了对 VisualCafé的支持。VisualCafé是 TogetherSoft 在 2002 年从 WebGain 购买的。Pataky 说，Borland 希望 VisualCafé能够迁移到 JBuilder。可以从 http://info.borland.com/new/jb7/vcafe.html 获得免费的转换工具。同样，他说，“Borland Enterprise Studio 5 for Java 是 WebGain Studio 的替代品”。




Peopleware
Productive Projects
and Teams
《人件》中译本即将发行
Second Edition Featuring Eight All-New Chapter
Tom DeMarco
&
Timothy Lister

UML 序列图开发

Il-Yeol Song 著，李巍 译

查看评论

摘要

UML（统一建模语言）已经被广泛接受为面向对象分析和设计的一种标准语言。在 UML 图中，开发难度最大以及耗时最长的图便是对象交互图（object interaction diagram, OID），其表现为一张序列图或协作图。我们的经验表明，开发者在 OIDs 的理解和开发上会有很大的困难。在本文中，我们提出了一个开发序列图的有效的十步启发法，并结合一个案例学习来演示本方法。通过本方法，我们展示了一种在开发序列图时正确使用控制对象和边界对象的方式。在我们的启发法中，在一个单独用例中多个序列图之间的关系，可使用控制对象来优雅地表示。我们发现开发者使用本启发法可以高效地进行序列图的开发。

介绍

UML（统一建模语言）所宣称的是一种语言，而非一种方法。UML 提供了开发面向对象软件或系统所必需的一组标记和概念。UML 包括九种相互关联的图，它们用来对所建模系统的不同方面进行建模。这些图之间的关系可如图 1 所示。在 UML 图中，开发难度最大以及耗时最长的一个图便是对象交互图（object interaction diagram, OID），其表现为一张序列图或协作图。通过展示系统组件如何进行交互以完成在用例设计[2]中所定义的核心任务，OIDs 对动态的行为进行建模。然而许多设计初学者注重于静态模型，他们常常不注重对动态模型的使用，动态模型对于对象间正确的职责分配[8]是非常重要的。交互图的意图如下[6,2;14;5,8;19,3,10,15]：

- 建模对象之间的交互，
- 辅助理解一个系统（一个用例）实际是如何工作的，
- 验证一个用例描述能够被现有的对象类所支持，
- 识别职责/操作并将它们分配给类，
- 建模在实时系统中所传递的同步/异步消息

然而从表面上看来，构建一个 OID 的方法没有在文献中更多地加以描述。我们的经验表明，初学者在理解和开发 OIDs 上会有很大的困难。大多数 UML 书只是简单地阐述了 OIDs 的标记和语义，以及介绍事先已构建好的序列图。

有些作者提供了序列图开发的简单指导原则。我们发现这些简单指导原则对于许多初学者而言是不够的。基于作者多年从事面向对象分析和设计的教学经验，我们提出了一个有效的开发序列图的十步启发法，并结合一个案例学习来演示本方法。

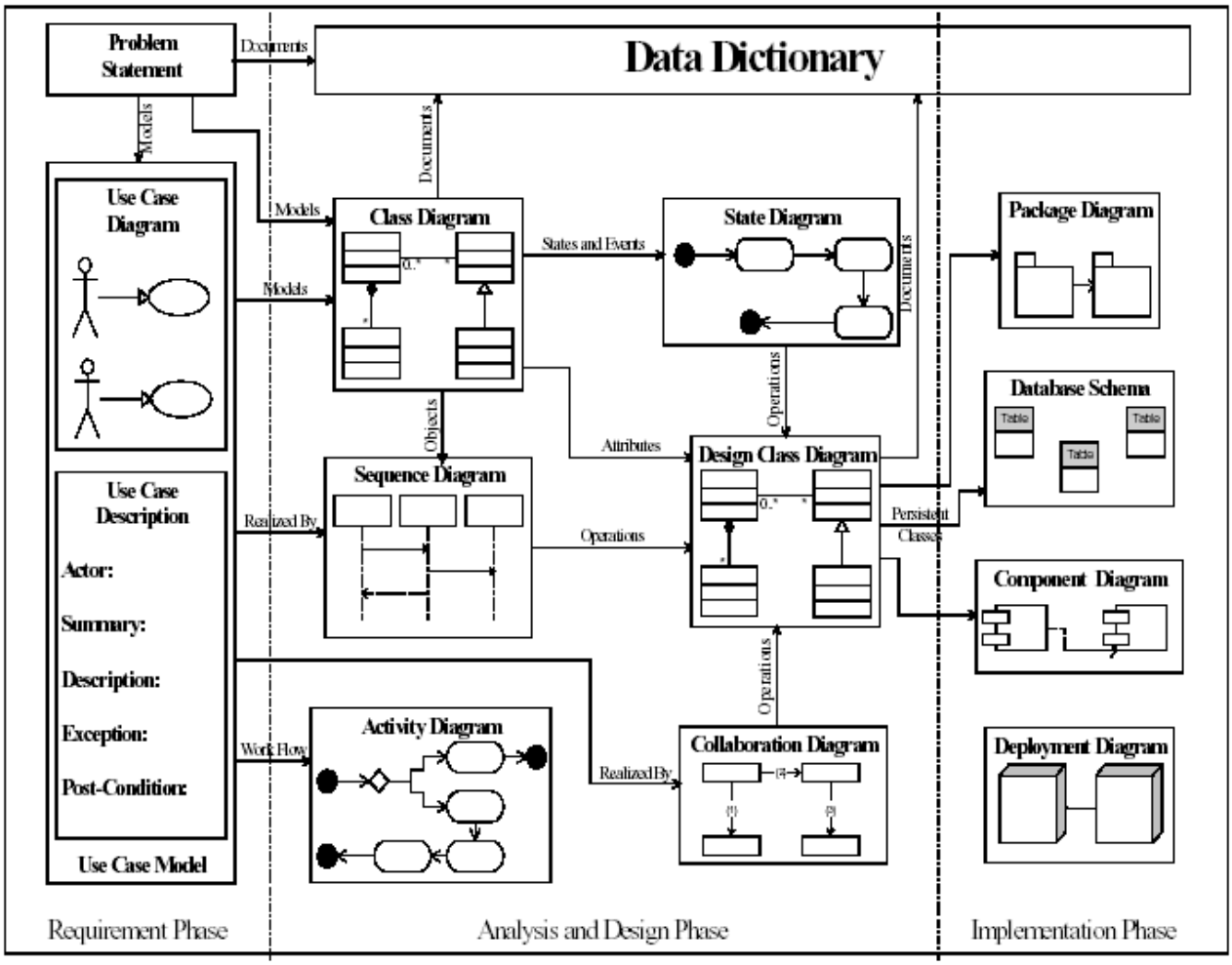


图 1：UML 图之间的关系

理想情况下，我们的启发法假定开发者已经开发出用例图，用例描述和类图。我们基于每个基本用例（primary use case）来开发交互图。我们的启发法将会利用构造型对象（stereotype object）—实体（entity），边界（boundary）和控制（control）对象—的特性。我们假定 actor 仅能与边界对象直接进行通信，不能与控制对象或实体对象直接进行通信。我们为每个基本，包含（included）或扩展（extended）用例都创建一个控制对象。该约定使得我们能够优雅地表示在一个单独用例中多个序列图之间的关系。

由于序列图能够很容易地转化成协作图，我们的启发法同样适用于协作图的开发。在本文的其余部分中，我们将交替使用术语*交互图*和*序列图*。

本文的其余部分将按如下组织：第 2 节概要描述序列图的标记和概念。第 3 节回顾序列图领域的研究活动，以及各个作者所建议的多种开发方法。第 4 节描述构造型对象及其关系的使用。第 5 节提出启发法，第 6 节则结合一个录像租借系统作为案例学习来演示本启发法。

序列图

本节中，我们定义了 UML 序列图中所用到的术语和标记。

序列图的流行（最初称为对象交互图），归功于 Jacobson 等人[6]。序列图专注于时间上的序列，或者消息的时间次序，或者发送消息的次序。这些图强调先发生什么，其次发生什么，等等。它们以图形方式来表示时间的经过。一个典型序列图的示意结构和标记可如图 2 所示。

这些图具有两个轴：水平轴显示对象，垂直轴显示时间。此外，序列图具有协作图所无法展现的两个特征：一个对象的生命线（lifeline）和控制的焦点（focus of control）[2]。在序列图中使用的对象生命线表示了在一个场景（scenario）中对象的生存期。大多数对象将会在整个场景中存在，但有时会在该场景期间创建或删除对象。例如，创建定单，和删除预约。

序列图的局限是它无法明确地展示对象间的关系或链接（link）。这些关系则是协作图所最需要关注的重点。UML 开发背后的一个目标就是，对所构建系统的特性谱进行建模的过程，要尽可能简化[2]。但是，这将会比以前所开发的面向对象方法更加复杂，因为其往往会更加广泛。因此，常常很难进行 UML 图的开发。在第一个序列图草案的开发过程中，需要确定许多重要的设计决定。难于对序列图进行开发还因为 UML 并没有提供一个可以遵循的过程或具体的步骤。这将由设计者自行选择或开发一种方法来辅助他/她创建一张高效的图。

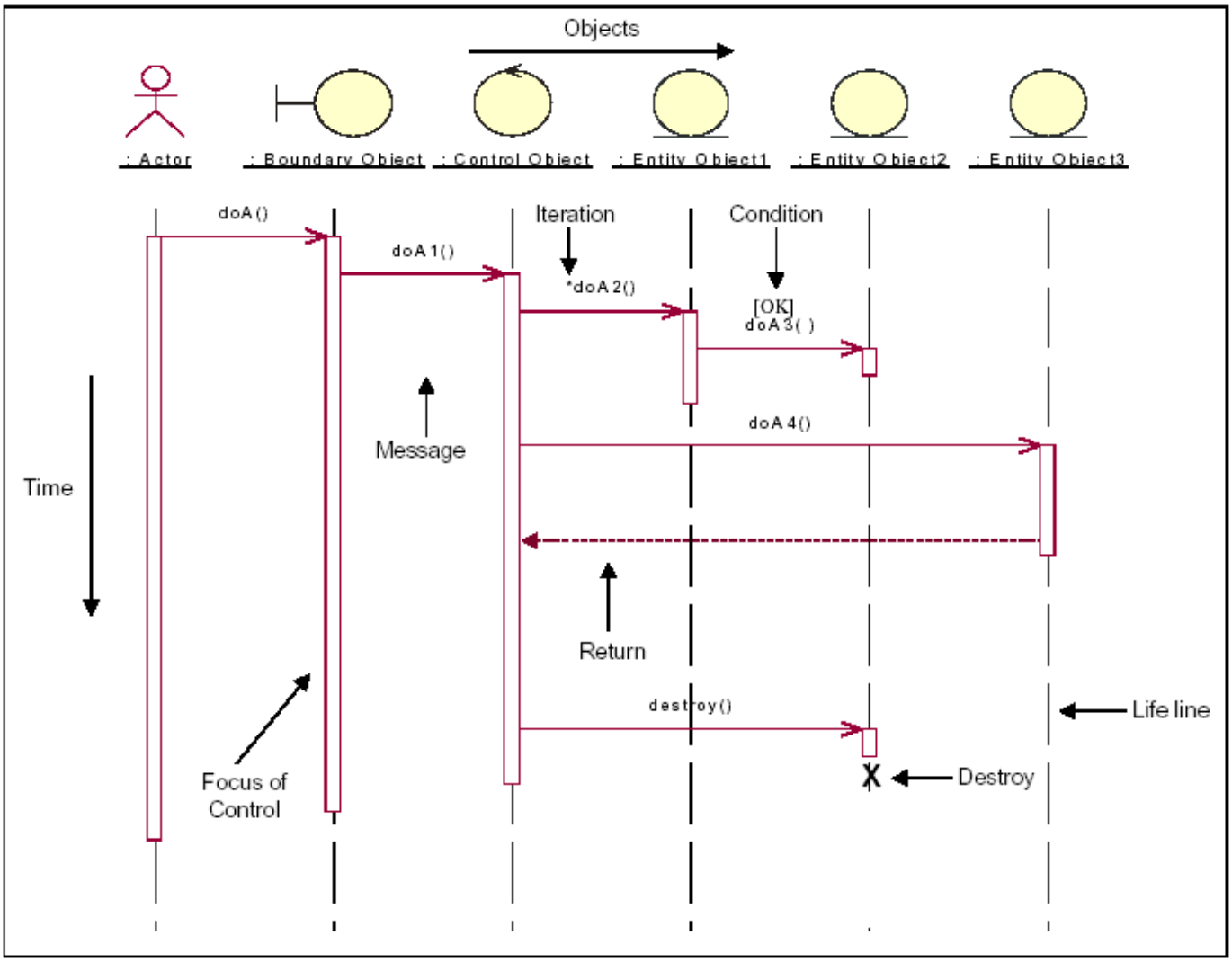


图 2：序列图的结构和标记

Booch 等人[2]指出，序列图是对系统某些动态行为的方面进行建模的一种手段，其可用的上下文包括整个系统，一个子系统或其所相关的一个用例。有人认为，应该至少为每个用例绘制一张序列图[6,19,5,18]。当为一个用例开发一张序列图时，至少用例描述可被用来开发一张序列图的最初草案。贯穿整个设计过程中，用例图可以基于序列图的结果来进行修正，反之亦然，直到这两种模型被调整得当为止[8,10]。我们可从用例描述中识别出操作，并将它们表达在序列图中（请参见图 1）。

由于可以为每个基本用例，*包含*用例和*扩展*用例都开发一张序列图，因此将会开发出许多序列图。但是，UML 并没有定义将这些图关联起来的标记。Song 等人则提议了这样一种标记的想法[17]。其建议是在序列图上插入一个连接器符号，既可“向下钻取（drill down）”至下一级序列图，又可“向上钻取（roll up）”至调用者。在本文中，我们提议使用控制对象来连接多个序列图。

序列图的概览及其开发

在本节中，我们概览了关于序列图的研究活动，以及多个作者所建议的开发序列图的指导原则。

我们发现大多数序列图的研究活动都集中在实时系统[16,9,13,4]，模拟[7]或行为驱动分析和设计[3]上。甚至很少有作者会提到用于开发高效序列图可能的方法，过程或步骤。

Amber[1]概述了其关于构建序列图的步骤的建议，总结可参见表 1。其步骤以一种非常简要的形式描述了在收集图的主体部分时需要做什么。但是这些步骤没有给出任何关于“如何去做”的细节指导。

1. 识别用例场景所起始的类。
2. 预演场景的过程逻辑，识别每个需要发送的消息及其要发往的对象。
3. 一旦识别出一个对象，则绘制其生命线。
4. 指明对象生命线上的活动框（activity box），其将发送一个消息或期望一个返回。

表 1：Amber 关于构建序列图的步骤[1,p.62]

Rosenberg[14]使用健壮性分析（robustness analysis）来论述序列图的开发，健壮性分析是由 Jacobson 等人[6]提出的。但是 Rosenberg 并未涵盖开发者所需遵循的逐步（step-by-step）指导原则。

Booch 等人[2]概要描述了一些单独的步骤来构造序列图（请参见表 2），和交互图。尽管有些步骤会相互重叠，但是有些不同的步骤则表现了这两种图之间的差异。这些步骤要比 Ambler 的更加详细，但其仍然是集中在需要做什么上，而没有涉及“如何”去做。

Booch 等人[2]也概要描述了一张构造良好的交互图的许多特征（表 3）和技巧，用来开发交互图（表 4）。尽管不是很具体，但却是非常有帮助的建议，尤其是对于初学者，应该引起他们的重视。

1. 建立交互图的上下文（系统，子系统，用例）。
2. 识别在交互中扮演角色的对象，并将那些最重要的放在左面。
3. 建立每个对象的生命线。
4. 从发起交互的消息开始，在生命线之间从上到下地对每个消息进行布局，展示每个消息的属性。
5. 修饰每条生命线上对象的“控制焦点”或活动期，如果需要其可视的话。
6. 修饰每条消息的时间和空间约束，如果需要的话。
7. 修饰每条消息的前置和后置条件，如果需要的话。

表 2: Booch 等人提出的构造序列图的步骤[2, P.251]

1. 是否专注于对系统的一个动态方面进行沟通。
2. 仅包含那些理解这个方面的最基本元素。
3. 提供了与其抽象级别相一致的细节，并且应该仅将那些在理解上是最基本的修饰成分暴露出来。
4. 其最低要求是会不会将一些重要的语义错误地传达给读者。

表 3: Booch 等人提出的一张构造良好的交互图的标准[2, P.256]

1. 给出一个名称，用以对其意图进行沟通。
2. 对其元素进行布局，使线之间交叉的数目最小。
3. 使用注释（note）和颜色作为可视化的提示，以将注意力集中在你图上的那些重要特征上。

表 4: Booch 等人提出的绘制交互图的技巧[2, P.256]

Jacobson 等人[6]根据结构（structure）对序列图进行了分类。*fork* 或集中式序列图包含一个基本的客户端操作（client operation），其控制了多个服务器操作（server operation）的信号流程。与此相反，*stair* 或分散式序列图则使用委托（delegation）作为基本的方式来构造多个对象之间的通信。更加明确地，Jacobson 等人将一个分散式 OID 描述为“每个对象仅仅知道少数的其它对象，以及知道哪些对象能够帮助其实现一个具体的行为。在这里我们没有‘中心’对象。”

Jacobson 推荐了封装性更强的 stair 配置，用来配置通信对象以某种等级的方式彼此相连的位置。另一方面，当操作能够改变一个操作序列，以及当在序列图中插入新操作时，则可推荐集中的 fork 结构。这两种结构可如[6]中的图 8.15 所示。

在开发序列图时，我们发现这些指导对于许多初学者来说是不够的。我们的经验表明，初学者在理解和开发序列图上有很大的困难。基于作者在教授面向对象分析和设计方面的经验，我们开发出了一个有效的十步启发法，用于序列图的开发。我们将会在第 5 节的一个案例分析中演示本方法。

边界对象和控制对象

由于我们的启发法使用了边界对象和控制对象的标记，我们将简要地描述它们的定义和特征。

实体，控制和边界对象是由 Jacobson 等人[6]第一次提出，用来开发一个健壮系统的构架。其思路是将一个用例描述中规定的行为分布成这三种对象类型。这三种对象类型的标记可参见图 2。我们采用了[6,19,14]对这些对象的定义和特征。

*实体*对象类标识现实生活领域的对象或概念，其对于系统而言是内部的。例如 Customer（客户），Rental（租借）和 Video（录像）类。外部的 actor 通常不会直接与实体对象发生联系。相反可通过边界对象来访问它们。*边界*对象类处理与外部 actor 之间的通信，并且它们封装了与环境相关的行为，从而保护了实体对象的完整性。例如用于输入输出的窗口，屏幕和菜单。*控制*对象类为事务处理类（transaction class），其负责捕获一个操作序列。这些对象捕获了业务规则和策略。Jacobson 等人[6]和 Larman[8]使用控制对象来处理一个用例。控制对象也经常被称作处理者（Handler）和控制者（Controller）[8,14]。

在[19,14]中，这三种对象之间的关系概括如图 3 所示。actor 仅能与边界对象进行通信。控制对象作为边界对象和实体对象之间的桥梁。不推荐边界对象和实体对象直接进行通信。不推荐实体对象直接发送消息至边界对象，因为这样会增加与一个系统的具体接口之间的耦合[8]。

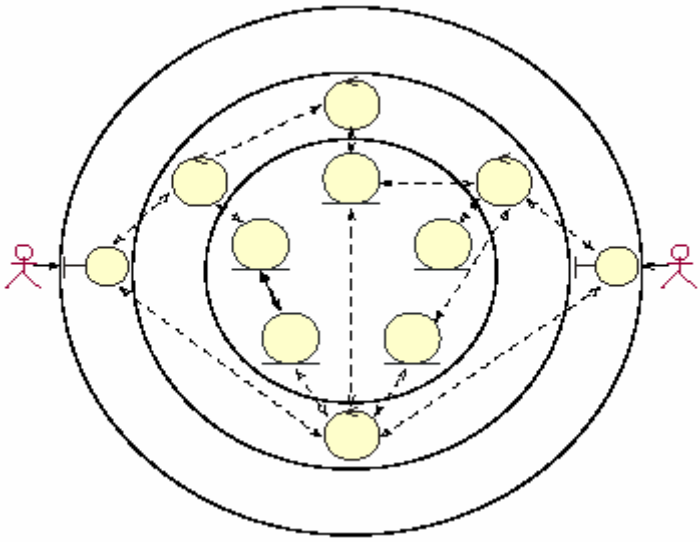


图 3：边界对象，控制对象和实体对象之间的关系

UML 将这些类别称为*构造型（stereotype）*[2]。我们注意到实体对象类将最终被持久化，而控制和边界对象类则通常不会被持久化。在第 5 节描述启发法时，实体，边界和控制对象将会扮演着重要的角色。

开发序列图的十步启发法

在本节中，我们提出开发序列图的启发法。

我们的方法基于如下思路：

- actor 仅与边界对象直接进行通信，而不能与控制或实体对象进行通信。
- 我们为每个基本，包含或扩展用例创建一个控制对象。
- 我们为 GUI 的每个弹出式屏幕创建一个边界对象。

为了容易地表达我们的论文，可将我们的方法分为两个步骤：预先步骤（pre-step）和应用步骤（application step）。预先步骤概括了应用我们的启发法所需完成的必要工作。应用步骤则展示了启发法的细节。

预先步骤：

- 开发问题陈述（problem statement）。
- 开发一张用例图。
- 为主要的基本用例开发用例描述。
- 开发问题的一张类图。
- 开发每个基本用例的前置条件(pre-condition)和后置条件(post-condition)。

我们假定以契约（contract）的方式来开发后置条件。我们使用 Larman 的分类法来规定后置条件如下[8]：

- 实例创建和删除。
- 属性修改。
- 关联形成和破坏。

开发序列图的十步启发法可如表 5 所示。我们注意到当创建一个对象时，将会完成步骤 8.2 关联的形成。因此，一个关联的形成不需要一个单独的传递消息。步骤 8.3.1-8.3.4 提供了开发者识别各种可能操作的技巧。

案例学习

在本节中，我们会演示我们的十步启发法。我们将启发法应用于一个录像租借系统，并为 *rent items（出租）* 用例开发序列图。

问题陈述

这是一个小型的本地录像租借系统（VRS）。问题将被限制为租借，返还，库存管理（添加/删除新带，更改租价等）以及产生总结各种业务活动的报表。将商店的出租项限制为录像带。输入客户 ID 号码（任意数字），电话号码或姓名的组合，用以标识客户数据以及生成订单。输入每一项目的条形码 ID 并显示库存中的录像信息。每结算一项，录像库存文件将会依次减一。当输入了所有的录像带 ID 时，系统会计算总的租借费，并进行支付处理。将会生成，打印和存储租借表单。客户签署租借表单，领取录像带并离去。返还则是通过读取返还带的条形码来进行的。任何未结算的录像租金将会显示成每张带的到期金额和总的到期金额。当新带被取走后，必须将过期金额置为零。会为新客户生成唯一的客户 ID 并输入客户信息到系统中。录像可根据它们的类别进行堆放，例如戏剧，

喜剧，动作片等等。任何在客户和计算机数据之间出现不一致的问题将由商店经理来解决。租借费可通过现金，支票或主要信用卡的任意一种来支付。报表需求包括查看客户的租借历史记录，录像的出租历史纪录，根据类别列出的标题，租借前十名，根据状态列出的项目，根据客户列出的过期录像，以及根据客户列出的未结算余额。

1. 从用例描述中选择发起 actor 和发起事件。
2. 标识实现用例所需的基本显示屏幕。称之为*基本边界对象*。
3. 创建一个*用例控制者（基本控制对象）*来处理基本边界对象和领域对象之间的通信。
4. 若用例涉及到任何的包含或扩展用例，则需要为它们每个都创建一个*次级控制对象*。
5. 标识实现用例所需的主要屏幕的数目。为每个主要屏幕创建一个*次级边界对象*，并且要为它们每个都创建一个*次级控制对象*。
6. 在类图上，通过对用例描述进行回顾可列出所有参与到用例中的*领域*类。若任何从用例描述中识别出的类没有出现在类图中，则需要将其添加至类图中。
7. 在序列图中使用那些仅标识为方框标签（列名）的类。按如下次序列出这些类：
   - 基本边界构造型
   - 基本用例控制者
   - 领域类（以访问次序列出）
   - 以访问次序为准的次级控制对象和次级边界对象。
8. 根据如下类别来识别所有解决问题的操作：
   8.1 实例创建和析构
   8.2 关联形成
   8.3 属性修改
   　　8.3.1 计算
   　　8.3.1 改变状态
   　　8.3.1 显示或报表需求
   　　8.3.1 与外部对象或系统的接口
   这些解决问题的操作可通过如下方式来识别：
   - 标识用例描述中的动词
   - 排除用以*描述*问题的动词；选择用以解决问题的动词。我们将这些动词称为*问题解决动词（PSVs）*。
   - 从问题解决动词中，选择一个表示自动操作的动词。我们称这些 PSVs 为*问题解决操作（PSOs）*，并将其用于序列图中。
9. 尽可能地根据任何已经存在的设计模式，来重新排列对象类之间消息的序列。
10. 命名每个消息并为其提供可选参数。这也可在设计阶段进行。

表 5：序列图开发的十步启发法

用例图

前面 VRS 的系统级用例图可如图 4 所示。在该用例图中，我们标识出四个 actor 以及八个基本用例和两个包含用例。在它们当中，*Customer（客户）*是一个间接 actor，其并不使用该系统。只有 actor *Staff（职员）*和 *Manager（经理）*使用该系统。actor *Credit Card System（信用卡系统）*是一个辅助 actor（supporting actor），其接受一个消息并对该消息进行回应。

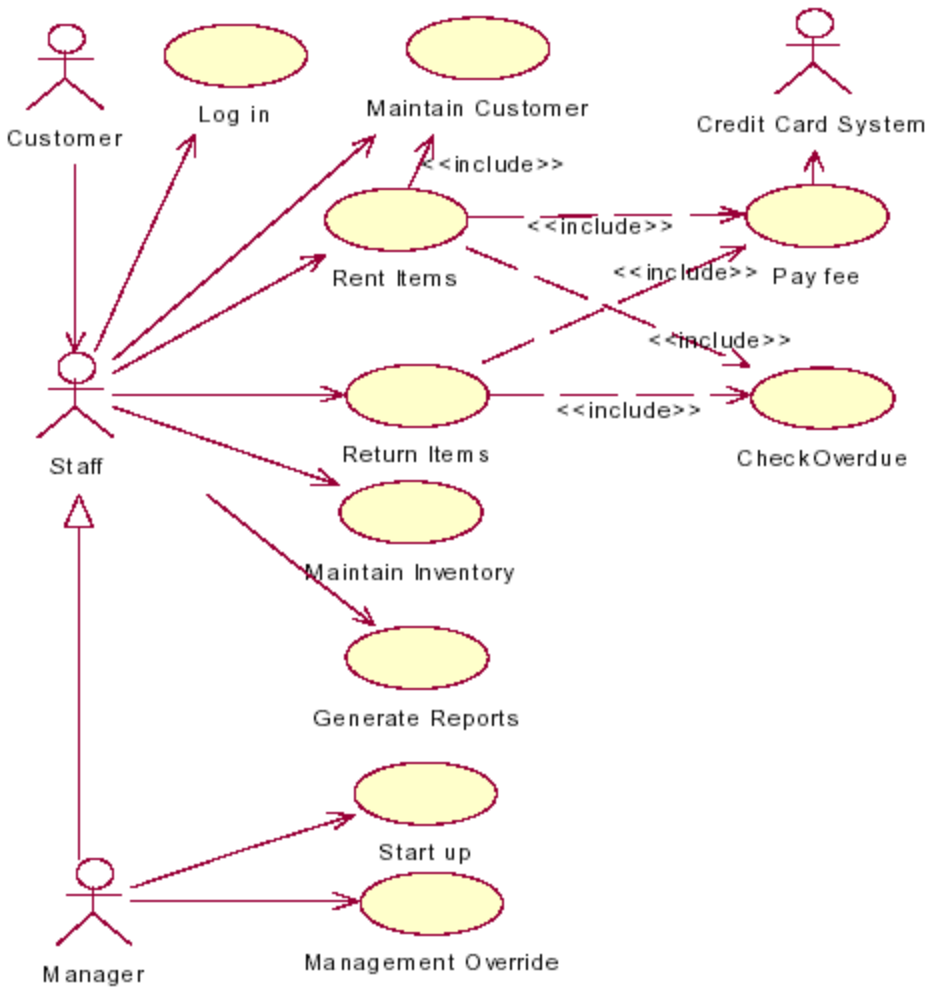


图 4：VRS 的用例图

类图

前面 VRS 的类图可如图 5 所示。注意，我们假定一个 *Item（项目）*可以与零个或多个 *RentalItem（租借项目）*相关。这是因为我们要对六个月之前的所有租借数据进行存档。

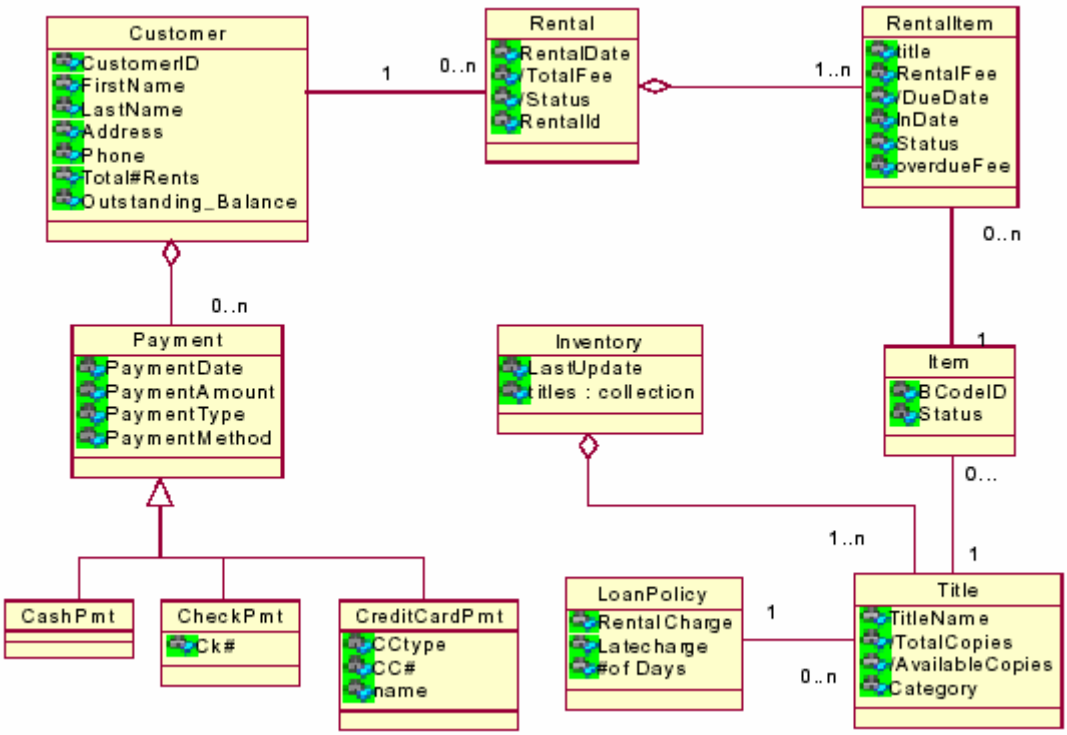


图 5：VRS 的类图

启发法应用和系统序列图

随着问题域复杂度的增加，序列图会很容易变得错综复杂。为了简化序列图的复杂度，我们使用 Larman[8] 用到的系统序列图的思想。系统序列图以黑盒的方式展示了系统 actor 和系统之间的所有系统事件。一个系统事件作为一个输入，其由系统的 actor 所产生。使用该方法，一个用例的全部序列图将被分解成一组系统事件，其将使我们能够为一个或多个系统事件开发一张序列图。图 6 显示了 VRS 的系统序列图，其具有四个系统事件。

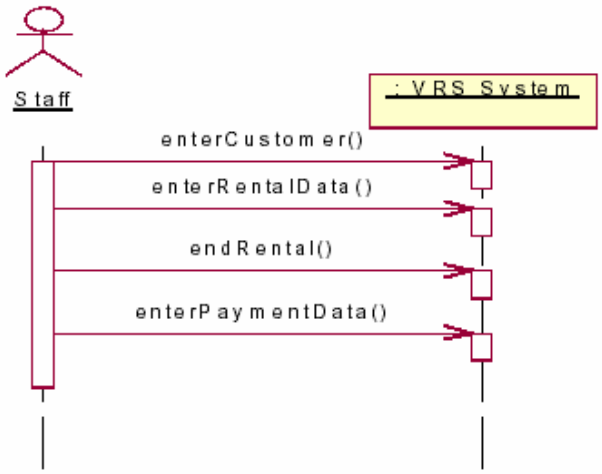


图 6：*Rent Items* 用例的系统序列图

出于空间上的限制，我们仅将我们的启发法应用于第二，三和四个系统事件：*enterRentalData()*，*endRental()* 和 *enterPaymentData()*。注意，客户数据会在系统事件 *enterCustomer()*中进行处理，以及支付对象的创建和实际的支付会在包含用例 *Payfee* 中进行处理。因此，它们将不会出现在我们下面的序列图中。

1. Actor — Staff
2. *基本边界*对象 — Rental windown
3. *用例控制者* — Rental handler
4. 次级控制对象 — Payfee
5. 次级边界对象 — 无
6. 参与的领域类 — Rental, Rental item, Item, Title, Payment, LoanPolicy
7. 方框标签 — 首先放置 Rental window 对象
然后是 Rental handler
再就是领域类
8. 主要操作

8.1 实例创建： `Rental handler, Rental, RentalItem`

8.2 关联形成： 连接 `RentalItem` 至 `Item`
连接 `RentalItem` 至 `Rental`
连接 `Rental` 至 `Customer`

8.3 属性形成：

8.3.1 计算： `calculateDueDate(),`
`calculateTax(),`
`calculateTotalRental(),`

8.3.2 改变状态： `updateItemStatus(),`
`decreaseAvailableCopies(),`
`setDueDate(),`
`setRentalDate(),`

8.3.3 显示/报表需求：

```
getItem(),
getTitle(),
getRentalData(),
getDuration(),
getFee(),
getRentalFee(),
displayTotalFee(),
```

8.3.4 与外部对象或系统之间的接口： 无

9. 序列的重新排列：本情况下无

10. 适当地命名每个消息并提供其明显的参数：可稍后在设计阶段添加

构造好的序列图可如图 7 所示。注意，我们都要为每个包含或扩展用例开发一张单独的序列图，并且使用一个次级控制对象来表现每个包含/扩展用例。因此，我们可通过一个次级控制对象 *pmtController* 来表现包含用例 *Payfee*（图 4）。基本控制对象 *RentalHandler* 可以调用次级控制对象 *pmtController*。这样，基本用例中的多个包含/扩展用例之间的连接，便能够优雅地通过控制对象之间的消息传递来表达。一张包含用例的序列图将会起始自一个次级控制对象。例如，*Payfee* 的序列图将起始自 *PmtController*。

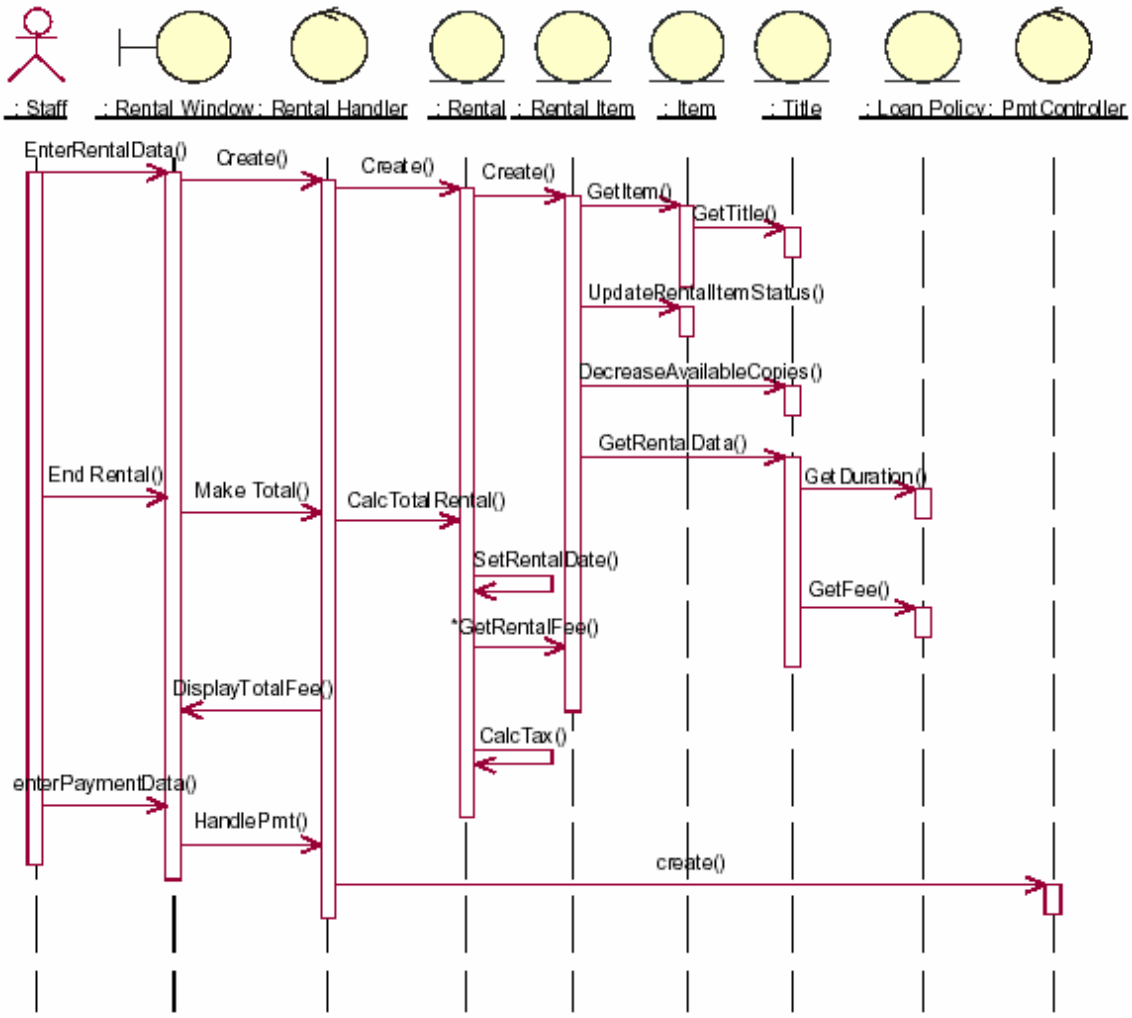


图 7：系统事件 *enterRentalData()*，*endRental()*和 *enterPaymentData()*的序列图

结论

我们注意到初学者在理解和开发序列图上会有很大的困难。大多数 UML 书只是简单地阐述了序列图的标记和语义，并介绍事先已构建好的序列图。有些作者提供了序列图开发的简单指导原则。我们发现这些简单的指导原则对于许多初学者来说是不够的。在本文中，我们提出序列图开发的十步启发法。我们并不是宣扬本文所提出的十步启发法，在所有情形下都能产生最佳的序列图。但是我们相信本方法是非常适用的，并且与问题域无关。我们也相信通过我们的启发法所产生的草图，能够很容易地针对具体的应用进行定制和优化。我们的经验表明，使用本方法的学员能够容易和快速进行序列图的开发。尽管我们假定开发者已经开发出用例建模和描述，但是在没有详细用例建模的情况下仍然能够应用我们的启发法。我们当前正开发在使用启发法开发出初始的序列图后，能够应用的优化方法。

参考文献

[1] Ambler, S. Building Object Applications That Work, SIGS Books, 1998

[2] Booch, G., Rumbaugh, J., and Jacobson, I (1999). The Unified Modeling Language: User Guide. Addison Wesley.

[3] Delcambre, L.M.L. and Eckland, E., A Behaviorally driven Approach to Object-Oriented Analysis and Design with Object-Oriented Data Modeling, In Advances in Object-Oriented Data Modeling, MIT Press, 2000, pp. 21-40.

[4] Engels, G., Groenewegen, L., and Kappel, G. Coordinated Collaboration of Objects, In Advances in Object-Oriented Data Modeling, MIT Press, 2000, pp. 308-331.

[5] Eriksson, H. and Penker, M. (1998). UML Toolkit. New York: John Wiley, Inc.

[6] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1992.

[7] Kabajunga, C and Pooley, R.. Simulating UML sequence diagrams. UK Performance Engineering Workshop, UK PEW 1998, pages 198—207, July 1998.

[8] Larman, C., Applying UML and Patterns, Prentice Hall, 1998.

[9] Li, X. and Lilius, J. Timing Analysis of UML Sequence Diagrams. UML'99, The Unified Modeling Language. Beyond the Standard. The Second International Conference, Fort Collins, CO, USA, October 28-30, 1999.

[10] Maciaszek, L. A., Requirement Analysis and System Design: Developing Information Systems with UML. Addison Wesley, 2001.

[11] Pooley, R. and Stevens, P., Using UML: Software Engineering with Objects and Components. Addison Wesley, 2000.

[12] Quantrani, T. Visual Modeling with Rational Rose and UML. Addison Wesley, 1998.

[13] Rudolph, E., Grabowski, J., and Graubmann, P. Towards a Harmonization of UMLSequence Diagrams and MSC. SDL'99 - The next Millenium, Elsevier, June 1999.

[14] Rosenberg, D. (1999). Use Case Driven Object Modeling with UML: A Practical Approach, Addison Wesley.

[15] Siau, Keng (2001). Unified Modeling Language: Systems Analysis, Design and Development Issues. Idea Publishing Group.

[16] Seemann, J. and Wvg, J. (1998). Extension of UML Sequence Diagrams for Real-Time Systems, In Proc. International UML Workshop, Mulhouse, June 1988.

[17] Song, I.-Y., Watts, P., Hassell, L., and Wong, C. "Modeling Dynamic Behavior with Object Interaction Diagrams," Proc. of 4th International Conference on Computer Science and Informatics (CSI '98), Oct. 23-28, 1998, pp. 408-412.

[18] Texel, P. and Williams C. Use Cases Combined with Booch/OMT/UML: Process and Products. Upper Saddle River: Prentice Hall PTR, 1997.

[19] Yourdon, E., Whitehead, K., Thomann, J., Oppel, K., and Nevermann, P. Mainstream Objects: An Analysis and Design Approach for Business. Upper Saddle River, NJ: Yourdon Press, 1995.

征稿

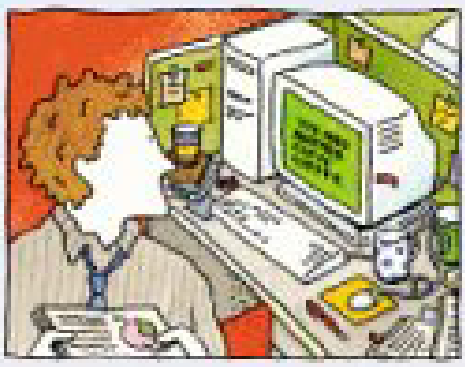
http://www.umlchina.com/xprogrammer/xprogrammer.htm

Peopleware

2nd Edition | Productive Projects and Teams

Tom DeMarco & Timothy Lister

Translated by Tomoo Matsubara & Tsuneo Yamaura



ピープルウエア

第2版

ヤル気こそプロジェクト成功の鍵

トム・デマルコ／ティモシー・リスター

松原友夫／山浦恒央訳

《人件》中译本即将发行

日経BP社

过程模式的 UML 描述

Harald Storrle 著，Happy 译

查看评论

摘要

目前，模式已经被广泛地用来描述软件设计，除了描述产品结构外，它们也可以用来描述过程结构，通过将广为人之的模式描述纲要应用于软件过程领域，可以达到这个目的。在这个纲要中，我在任何恰当的地方使用 UML 术语和符号。这样，结合优越的细化程度、精确程度和表达力，使本文有别于这个领域以前的方法。

1 简介

1.1 动机

关于软件过程的工作流已经有多种形式和风格的描述（参见 e.g.[16]的调查），如今，仍然有一些重要的需求没有被充分陈述。

理解性：

专业人员明确要求“结构良好”的过程模型是“易于理解并易于应用的”(比较[18]中的需求 2 和 3)。大多数传统过程，像“统一过程(Unified Process)”（UP,[14]），还是有一大堆文档集，需要很多经验去理解和恰当地应用。

灵活性：

很多开发人员觉得他们的创造性被过于严格的开发过程所限制了，实际上，有些人会说，这正是目的。不管怎样，实施一个过程是非常困难的管理任务，并且存在着传统（大型）过程不适用的情况，这样就出现了近来很多人感兴趣的轻量级过程，例如“极限编程（extreme programming）”。

精确性：

另一方面，一个开发过程必须足够的形式化，以允许自动执行和形式化分析，至少能选择一部分如此。理想一点来说，这将可以和灵活性相结合，使过程达到一种受控的、分级形式化程度。

分形结构：

最后，部件范例(componentware paradigm)有对整个开发过程结构的明确实现：如果产品结构是分形的，那么过程结构最好也是分形的，否则，过程和产品的映射就成为主要难题。在这里，迭代的过程模型，例如统一过程模型，就显出它的弱点。

1.2 优点

我相信，使用 UML 描述过程模式是一种新颖的方式，用它来解决这些问题会比传统方式更好一些，原因如下。首先，考虑到 UML 的影响，另一方面，UML 已经是“软件工程领域的通用语言(lingua franca)”(见[24,p.v])，并且，它的传播范围将越来越广。这意味着几乎所有的专业人员都将懂得 UML（或者不久后不得不去懂得），而且这种情况将保持很长时间。这会导致网络效应，UML 的 CASE 工具、教程和经验的数量和质量都将大大提高，过程模型的可理解性得到显著提高。当然，用 UML 为工作流建模时，有些对象还是无法表达，但这些障碍都会很容易克服（见第 2 节）。

另一方面，现在 UML 正迅速发展成一种数学上精确形式化的主体，逐步趋向于自动化和形式化，不仅对于过程模型，对产品模型也是如此。现在来看看过程模式的影响，设计模式（Design Pattern）的概念在使用者中被广泛接受，并成功地证明了它的实际适用性。模式捕捉细小的、连贯的和自充分的知识片断，允许在不同的环境中使用，特别是应用于不同的规模。使用设计模式是一种重用设计知识的形式。过程模式非常像设计模式，只是应用于过程领域。确切地说，术语“设计模式”有点误导：典型的设计模式如(MVC)模型－视图－控制器等，真正说的只是产品，那是设计的结果，而非设计的过程。所以，为了区别它们，我把它们叫作结果模式和过程模式。

设计模式的优点可以转移到过程模式上来，通过在不同环境中使用相同设计模式，面向模式的过程描述的重复性将远少于传统方式。因此，一个模式语言比传统的过程更容易理解和应用。同时，每个模式可以根据它自身的形式化程度而被公式化（即适用性标准的严密性），既可以用于灵活的、严格的过程，也适用于混合的过程。模式可以用来指导开发过程，也可以在事后为描述开发过程文档。过程中特别的补丁步骤对一个面向模式的过程来说是微不足道的。最后，模式是与规模无关的，也就是说，只要能够满足它们的应用性标准，就可以应用在任意规模的粒度和抽象层次上。使用这种过程模式语言的方式，很容易达到构建一个面向组件原则的软件：用组件来

构建系统将导致一个层次化、自相似（分形）的产品结构（例如[3,13,9,19]）。这也许是过程模式相比传统过程最重要的好处。

1.3 概要

本文讲述的方法可以归纳为：首先，我总结了 UML 过程建模的工具，并提出了一些小小的扩充。接着采用[11](和[5]非常相似，是另外两本主要的设计模式书籍)中的纲要以满足过程领域新的需求。

1.4 相关工作

对于软件过程的模式语言已经有一些研究成果，我所知道最早的是 Rolland et al.（[23]）提及的“过程块”。过程块是一个四元组，由形势、决策、行为和意见组成，在引出需求阶段产生正式的设计决策，它既没有使用 UML，也没有使用现代意义的模式。

接着，出现了一些论文，如[3,10,26]已经使用过程模式这个术语了，但仅仅停留在表面，很少的详细地描述这些模式。在任何细节中，既没有描述整体的纲要，也没有描述特定某个角度（例如分类、过程、参与者）。

在 Catalysis 中（[8]），出现了一个更为重要的方法，其中有 60 种不同的模式，每个都从三至五个角度（名称、意图、策略，有时有重要性和益处）描述，并使用自然语言和特定的草图。虽然，Catalysis 提出了第一个过程语言的真正实例，不过它也没有使用 UML。即便使用了一些符号（UML 中含糊不清的、废除的符号），也没有在过程模式中使用。既没有涉及到角色和职责，也没有需求和提供文档类型，在任何重大细节中，也没有实际设计活动的描述。

相比较上面提到的几种方法而言，我们的方法是唯一认真试图和 UML 融为一体的，而所有其他上面提到的方法在模式描述纲要上都不够细化和精确，当然 Rolland et al.是例外，因为它们根本就没有表现出一个纲要。

2 使用 UML 来描述软件过程

在软件工程领域，UML 是最广为理解的设计符号，很显然，使用它来作为过程的描述是很有帮助的，这样，从 UML 工具、知识和研究成果中都可以受益。然而，用 UML 来描述工作流仍然不是非常适合。严格意义的标准只包含活动图（和“UML 标准 Profile”之一的一些注释，见[20,4 部分]）。统一过程模型在符号和概念提供了更多，并且，虽然技术上它们不属于严格意义的标准，但也算这个范围的准标准。

统一过程模型通过两个元素扩展 UML 符号。首先，有一个双齿轮符号表示 SubactivityStates，它仅仅是一个依照句法的构造型（stereotype），另外，还有表示工人和资源的符号（见[14,pp.25,145]）。

我也增加了一个元素，虽然不是 UML 的一部分，但是一般都会明白：附上一个像狗耳朵的矩形框，用来表示文档（见图 1 的抽象示例）。文档类型可以放在矩形里面，既可以是简单文字，也可以是象形的图形。图的内容可以以不同的方式定义：文档类型表示一个特定 UML 图类型，或者 ModelElement 完全由它单独指定。表示程序的文档通过编程语言语法详细说明。其他的文档类型或许会被显式指定，或许不会，例如通过一个 XML DTD，或者平常的自然语言。概念上讲，我假设一份文档表示一个 Document，它作为一种 Classifier，例如它可以关联为一个 ObjectFlowState 的类型（见下面 3.4 节）。为了区别普通 UML 活动图和这些符号，我称它们为工作流图。请注意，在语法上，工作流图就是 UML 活动图，只是有 SubactivityState 和 Document 专有的 Presentation Element。这种图在统一过程模型中常见到，以非形式化方式使用（但是没有定义完全）。

注意，工作流图可以表示成一个粗略的版本，所有的 SubactivityState 用统一过程中双齿轮图标表示。概念上讲，它们可以被任意 ActivityGraph 细化，如果它细化得非常明确，就出现如图 1（下图）的图。请再看看，这是一个活动图。类似的，如果需要，文档类型也可以被细化（这儿省去）。

观察一下，会发现工作流图的粗略版本展示了一个和 Petri 网过程的紧密对应关系。

3 描述过程模式

过程模式可以用和结果模式相似的纲要来描述，过程模式整个描述纲要和术语从[11,p.5f]来，在下面的子章节中，我会首先介绍整个纲要，再说明它们从结果领域迁移到过程领域所作的改动（参见[3]，一个关于过程模式和结果结构模式需求对比的讨论）。

3.1 整个描述纲要

两本最著名的关于设计模式[5,11]的书使用非常相似的纲要来描述设计模式，只有术语上细小的区别。为了使我们的方法更容易接受，我尽可能采用这个描述纲要。参见表 1，是一个[5,11]和本文描述纲要的对比。

名称：

首先，每个模式有一个名称。对过程模式来说，这通常等同于该模式所支持任务的名称。

分类：

因为模式类别可能会增长至相当庞大，必须有一种系统的方法来获取它们，这可以便于这个角度的描述。这个角度在[5]中没有，在[11]中作为名称角度的一部分出现，详见 3.2 节。

意图：

模式基本原因的直观说明，它的好处和应用领域。

别名：

有时候，模式并不只有一个最好的名称，还有一些其他同样恰当的名称，在这里给出。

动机：

描述问题所在的一个场景，应用性条件和模式的目的。

应用性：

应用一个模式的先决条件，它所应用的上下文环境。在 Capsule 的范围内，这可以由所需的 Views 和确立可能的一致性条件来指定。应用性还包括前提条件。

过程：

在设计模式中，这个角度被称作“结构”，并包含了一个 OMT 类图。对于过程模式来说，“结构”等同于过程，它是活动的因果结构，参见下面 3.3 节。

参与者：

在结果模式中，这个角度指参与的类，在过程模式中，参与者是文档和资源（包括人），它们在模式中扮演一个角色。参见下面 3.4 节。

POSA[5]	GOF[11]	本文
名称	模式名称	名称
—	类别	类别
问题	意图	意图
别名	别名	别名
范例	动机	动机
环境	应用性	应用性
结构	结构	过程
动态角度	协作	—
—	参与者	参与者
含义	结论	结论
实现	实现	实现
示例方案	示例代码	示例执行
应用	已知应用	已知应用
参考	相关模式	相关模式

表 1、模式纲要比较

结论：

使用模式的优点与缺点的讨论，例如一个模式所达到的质量目标或其他目标等。

示例执行：

设计模式的实现是一个程序，它是用一种编程语言来表示。过程模式的实现不是一些计算机上的程序，而是一个组织里的程序，包括人、工具、组织的设备和一组个人及人与人之间的技巧。参见下面 3.7 节。

实现：

特定实现问题的讨论，也就是一些关于实现模式的问题。当过程模式被组织实现（见上一个角度），这个角度指明有关组织和工具的问题。

已知应用：

通过具体应用比较成功的模式示例，可以使读者易于应用某个模式。

相关模式：

和其他模式的关系，也就是模式取舍、同伴模式或是先决条件和差异讨论。

这个纲要没有包含[11]中协作的角度（在[5]中叫“动态角度”），它被用来表示纲要的行为（动态结构），这在新角度“过程”中已有描述，所以省去。

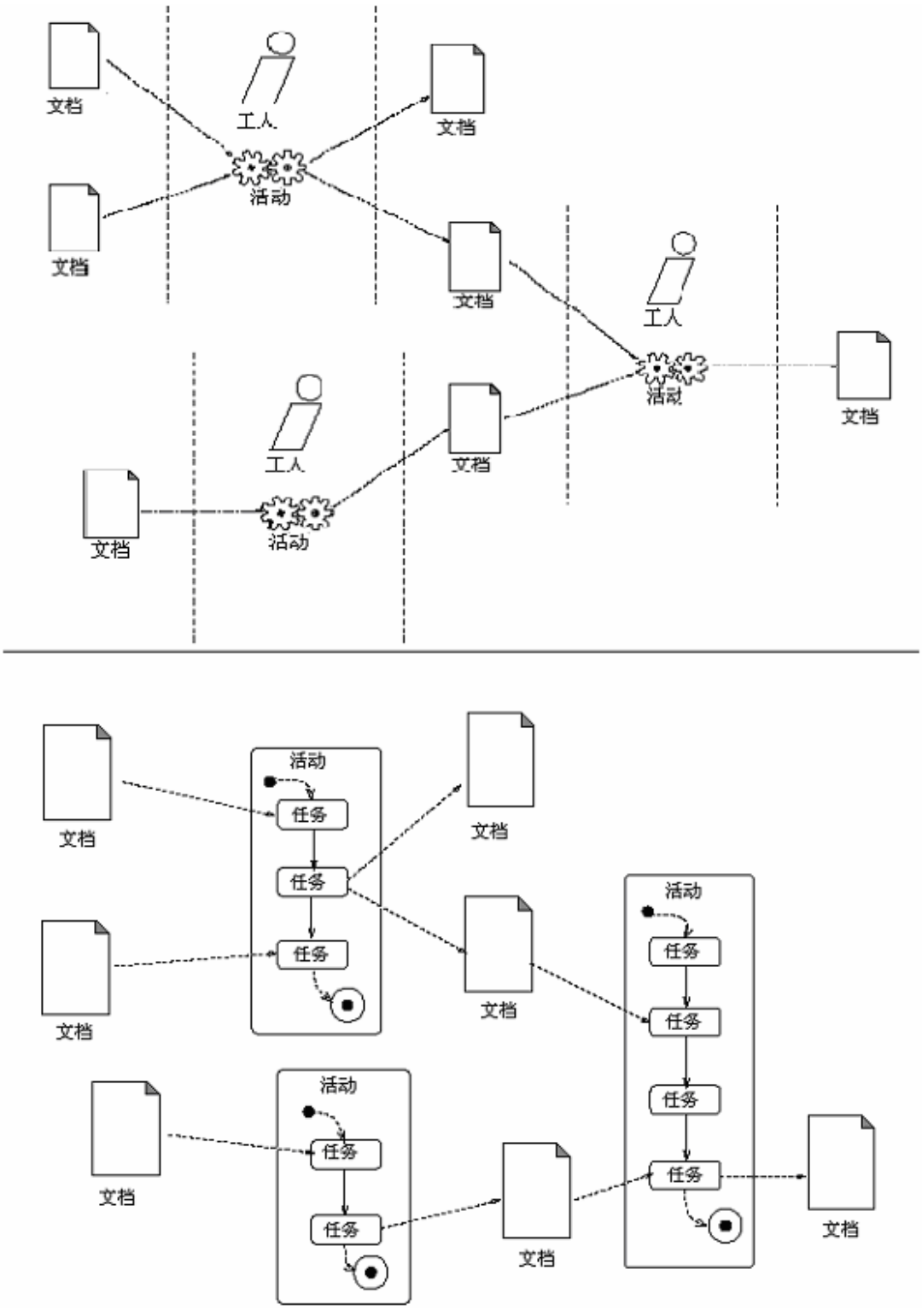


图 1、描述工作流的图，由粗（顶）到细（下）

3.2 分类角度

模式总是只作为一种模式语言的部分存在，它定义了标准。这些语言可能会变得相当庞大，所以为它们编制目录和如何检索它们成为非常重要的实际问题。这些问题已经在重用社区(见 e.g.[21])被研究过。在那儿，已经为分类的维度创造出新的术语以及它们各自的值，这些术语是面（facette）和简单类别（simple class）。对结果模式，有若干分类纲要，Buschmann et al.建议使用抽象级别和问题分类（[5,p.362ff]）。Gang-of-four 提出使用目的和范围分类([11,p.9ff])。对过程模式，这些纲要显然需要改写，都不是十分适合。我已经确定了四种分类面来代替上面的分类，抽象级别、阶段、目的和范围，下面，我将阐述这些面的简单类别。

抽象级别：

在[5]中，有三个抽象级别，习惯用法、设计模式和架构模式。类似的，过程模式可以按技巧方法、严格意义的过程模式和开发风格分类。

阶段：

设计模式归结于一个问题种类，对过程模式，相似的是开发阶段，即规格说明、设计、实现和维护，当然，还有其他一些阶段。

目的：

在[11]中，结果模式分成“创建型”、“结构性”和“行为型”，对过程模式而言，有其他的目的，例如管理、实际构建和质量保证。

范围：

最后，设计模式通过它的范围是一个对象还是类来辨别。因为这是一个面向对象技术中的术语，不大适合过程模式的情况，这些简单的类被替代成特指发生在传统过程中的实体，例如“架构风格”、“产品线体系”、“可重用组件”、“实现模块”或者“测试案例”可能比较恰当。

注意这些是确实在[5,11]中使用到的分类面，每个面给定的简单类对每个项目都不同，在这儿提到的仅仅是似是而非的缺省内容。同时，通常各个面的简单类之间也没有硬性和彻底的分界线。注意粒度对一个分形过程来说，不是一个有用的分类标准。在这些面的不同简单类之间没有硬性和彻底的区别。

3.3 过程角度

很多人都认为一个设计结果模式的结构角度（OMT 类图）就是模式，过程模式很显然没有这个意义的结构。它们指定一个（一部分）过程，因此，这个角度更名为“过程”，它由 UML 工作流图组成，而非静态的结构图。

除了活动图，还有其他的 UML 动态符号可以使用。实际应用中，也可以用其他一些描述软件过程的符号，例如 Petri 网、规则、活动树/事务图、或类编程语言符号，不过它们都没有达到 UML 被接受的程度。

3.4 参与者角度

在一个过程模式中有两种参与者，文档和资源。在统一过程模型的术语当中，资源特指工具、机器和人（“工人”，见图 1）。

文档被用作 Classifier，例如可以关联为一个 ObjectFlowState 的类型。文档既可以是过程模式的先决条件也可以是它的交付品，或两者皆是。它们可以使用典型记号来图形化表示，一个带狗耳朵符号的矩形框。这个图标可以标记上文档类型，表明是文本文档还是图形文档。用文档类型表示 UML 图类型已经完全由这个标记指定了，复合文档类型由它所包含的文档类型结构定义，其他种类的文档类型可以显式指定或不可显式指定。

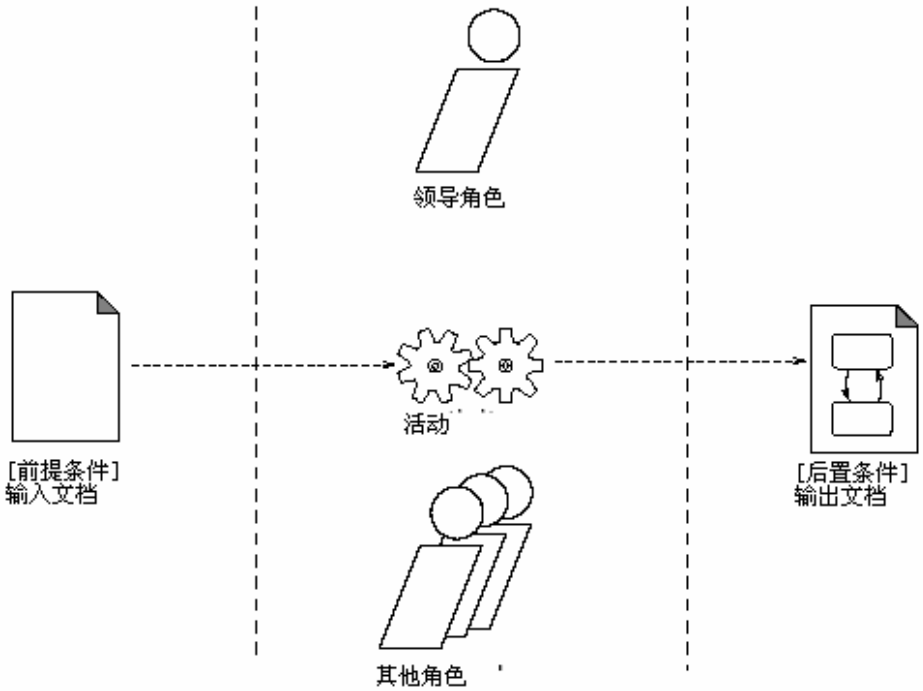


图 2、在统一过程模型的符号纲要中一个活动的示意符号，见[14,pp.25,145]

我使用统一过程模型中活动的纲要来表示模式的参与者，注意双齿轮映射到一个 SubactivityState，它可以细化成一个完整的 ActivityGraph。

3.5 应用性角度

脱离这些角度的传统内容（自然语言描述使用模式的应用条件），在我们的方法中，这个角度可以细化成形式化的参与者的前提条件。特别是对于文档，当然也可以应用于资源。这就有助于软件过程的形式化分析和自动执行了。在这一点上，说一下关于 UML 元模型的离题话题。首先注意 UML 所有的符号元素都对应一个元类，状态图对应 StateMachine 类，活动图对应 ActivityGraph，动作状态节点对应 ActionState，对象流程节点对应 ObjectFlowState 等等。我已经提到 Document 是一种特殊的 Classifier，因此，他可以有一个 StateMachine 来描述它的生命周期。在 ActivityGraph 的环境中， ObjectFlowState 有一个类型，关联到一个 Classifier 上，例如 ClassifierInState。这可以用来表示文档生命周期的当前状态。

或者，换句话说，文档和其他资源都有生命周期，在工作流图的文档（或资源）图标上可以记下它们特定的状态。举例来说，一份文档可以是“被检出”、“已测试”或“已核准”等状态，所以这些状态可以作为资源前提条件的规格。同时，其他方面的条件（包括时间约束、复杂逻辑谓词）可以使用 OCL（对象约束语言）表示。

3.6 结论角度

所有上一节说到的都可以类似地用于结论角度，只是状态附在文档和资源上是作为后置条件，而非前提条件。

3.7 示例执行角度

正如我前面说的，设计结果模式实现为一段程序，而过程模式由组织中的人、工具装备和应用的技术实现的。所以，过程模式中类似于示例代码的东西就是示例执行，它将给出一个实例，阐述各自的过程模式如何被一个组织实现，谁负责什么部分，那些技术和工具比较适当，工作的各部分如何同步和集成，使用哪些变更控制和版本控制系统等等。

例如，最初的类模型可以通过和领域专家进行纲要会谈产生，接着抽取出名词和动词，分别定义分析级的类和方法；对于分析级集成测试，使用适当的原型来进行角色扮演；对用户界面设计和确认，使用情节板，等等。在设计级，像走查，审查和自动一致性检查可能比较适当。

3.8 相关模式角度

在设计模式中，这可能只指其他设计结果模式。而在过程模式中，可以指任何类型模式，包括结果模式、其他过程模式和组织模式（[12,6]）。

4 使用过程模式

在这一节中，我将讨论如何使用过程模式。设想有一种使用上面纲要的模式语言，从某个预定义的初始状态（一组文档）出发，开发人员现在可以应用任意这些模式，当然它们的应用性要满足，所有参与者都在所需状态，所有 OCL 约束都满足。根据在参与者角度提供的纲要，应用这些模式之一会导致现有文档的修改或创建新文档。表面上看，应用一个过程模式就像图 2 中的纲要合成，再深入一点，活动的细节和文档状态都必须考虑。

使用非常相似的方式，一个传统过程可以使用适当的过程语言重构。见图 3，这里，参与者角度的纲要和右边的文档节点融合在一起。一些过程模式语言的好处可以体现在此，例如更简练的表示法、使用 UML 和控制需要什么级别的形式化需求。

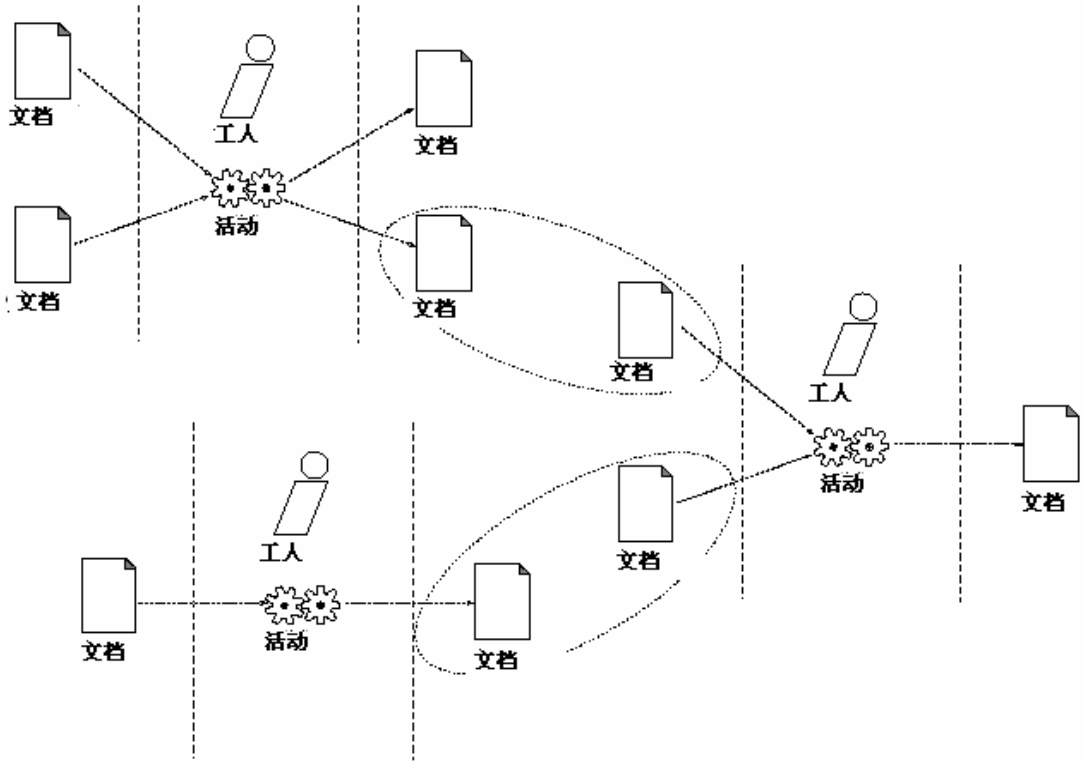


图 3、单个模式可以被合并，形成开发图

活动和面向文档的过程都可以使用过程模式描述，为完成面向活动的过程描述，文档类型和状态可以简单地保持抽象，为完成面向文档的过程描述，文档要完成必要的细节，但是活动可以保持粗略。

在一个严格的开发过程中，这些角度都要完全指定。只有过程提供的过程模式能够创建或修改文档。在一个松散的开发过程中，可以允许开发人员使用特别的方式创建和修改文档，其后，这些活动也许又放入过程模式语言的框架中，如果需要，可以作为一种注释的形式存在。

5 结论

5.1 总结

我已在本文定义了一个和用于描述设计结果模式非常类似的描述纲要。在这个纲要中，UML 用来描述一个角度，如过程、参与者、应用性和结论等。

简单来说，过程模式可以用来描述任意精度级的过程，按现在 UML 方式，能够被普遍理解。我在这里介绍的模式都能够很容易地适用于传统的和基于组件的过程模型。总之，它们提供了一个非常灵活，但是精确而且普遍被理解的描述形式。

其他的方式不够详细，而且没有使用 UML，比较起来只有很少的应用。

5.2 未来工作

我们的团队目前正在工作于一个支持过程模式使用的 CASE 系统，特别是目录、检索、自动执行和抽象设计指导方针的满意性检查等，都会被这个工具支持。使用这个工具，能够进行实际过程的模拟研究。至今为止，我只是在教室中用到了这里提及的几种模式，不过却有令人鼓舞的效果。在[25]中，我提出了一个用于架构建模的过程语言，但在其他应用范围，包括那些典型的工作流，尚未检验。

我们另一个重要的工作是为 UML ActivityGraph 定义形式化的语义，这对发布和形式化分析工作流图非常有必要。至今，仍没有现成的得到普遍认可的语义。但是，对 UML StateMachine，却有或多或少的完整形式化语义，ActivityGraph 相比较而言简化得多。在这个领域，近来的进展和我们自己的工作在不久的将来将会形成令人满意的形式化语义。这些分析方法将会应用于过程模式和传统过程的描述中。

参考

1. Thomas Allweyer and Peter Loos. Process Orientation in UML through Integration of Event-Driven Process Chains. In Pierre-Alain Muller and Jean B´ezivin, editors, *International Workshop UML '98: Beyond the Notation*, pages 183-193. Ecole Sup´erieure des Sciences Appliqu´ees pourl'Ing´enieur–Mulhouse, Universit´e de Haute-Alsace, 1998.

2. Alistair Barros, Keith Duddy, Michael Lawley, Zoran Milosevic, Kerry Raymond, and Andrew Wood. Processes, Roles and Events: UML Concepts for Enterprise Architecture. In Selic et al. [24], pages 62-77.

3. Klaus Bergner, Andreas Rausch, Marc Sihling, and Alexander Vilbig. A Componentware Development Methodology based on Process Patterns. In Joseph Yoder, editor, *Proc. 5th Annual Conf. on the Pattern Languages of Programs (PLoP)*, 1998. Available at www4.informatik.tu-muenchen.de/˜rausch/publications/.

4. Christie Bolton and Jim Davies. On Giving a Behavioural Semantics to Activity Graphs. In Reggio et al. [22], pages 17-22. Also appeared as Technical Report No. 0006 of the Ludwig-Maximilians-Universit¨at,M¨unchen, Fakult¨at f¨ur Informatik, October 2000.

5. Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture. A System of Patterns*. John Wiley & Sons Ltd., 1998. Also published in German by Addison-Weseley, 1998, under the title "Pattern-Orientierte Software-Architektur", to which all citations refer.

6. James O. Coplien. A Generative Development-Process Pattern. In Coplien and Schmidt [7], pages 183-238. Based on the proceedings of PLoP'94. Available also at www1.bell-labs.com/user/cope/Patterns/Process/index.html.

7. James O. Coplien and Douglas C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995. Based on the proceedings of PLoP'94. Available also at www1.bell-labs.com/user/cope/Patterns/Process/index.html.

8. Desmond Francis D'Souza and Alan Cameron Wills. *Objects, Components and Frameworks with UML. The Catalysis Approach*. Addison-Wesley, 1999.

9. Brian Foote. A Fractal Model of the Lifecycle of Reusable Objects. In James O. Coplien, Russel Winder, and Susan Hutz, editors, *OOPLSA'93 Workshop on Process Standards and Iteration*, 1993. Available at http://www.laputan.org/frameworks/fractal.html.

10. Brian Foote andWilliam F. Opdyke. Lifecycle and Refactoring Patterns That Support Evolution and Reuse. In Coplien and Schmidt [7], pages 239-258. Available at www.laputan.org/lifecycle/Lifecycle.html.

11. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

12. Neil B. Harrison. Organizational Patterns for Teams. Monticello, Illinois, 1995. Available via http://st-www.cs.uiuc.edu/~plop/.

13. Wolfgang Hesse. From WOON to EOS: New development methods require a new software process model. In A. Smolyaninov and A. Shestialtynow, editors, *Proc. 1st and 2nd Intl. Ws. on OO Technology (WOON'96/WOON'97)*, pages 88-101, 1997.

14. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

15. Dirk J¨ager, Ansgar Schleicher, and Bernhard Westfechtel. Using UML for Software Process Modeling. Number 1687 in LNCS, pages 91-108. ACM, 1998.

16. Marc I. Keller and H. Dieter Rombach. Comparison of Software Process Descriptions. pages 7-18, Hakodate, Japan, October 1990. IEEE Computer Society Press.

17. Alexander Knapp. *A Formal Approach to Object-Oriented Software Engineering*. PhD thesis, Ludwig-Maximilians-Universit¨at M¨unchen, Institut f¨ur Informatik, May 2000.

18. Ralf Kneuper. Requirements on Software Process Technology from the Viewpoint of Commercial Software Development. Number 1487 in LNCS, pages 111-115. Springer Verlag, 1998.

19. Meir M. Lehman. Programs, life cycles, and laws of software evolution. *IEEE Transactions on Software Engineering*, 68(9), September 1980.

20. OMG Unified Modeling Language Specification (version 1.3). Technical report, Object Management Group, June 1998. Available at uml.shl.com.

21. Ruben Prieto-Diaz. *Classification of Reusable Modules*, volume I – Concepts and Models, pages 99-124. ACM Press, 1989.

22. Gianna Reggio, Alexander Knapp, Bernhard Rumpe, Bran Selic, and Roel Wieringa, editors. *Dynamic Behavior in UML Models: Semantic Questions. Workshop Proceedings*, Oktober 2000. Also appeared as Technical Report

No. 0006 of the Ludwig-Maximilians-Universit¨at, M¨unchen, Fakult¨at f¨ur Informatik, October 2000.

23. Colette Rolland and Naveen Prakash. Reusable Process Chunks. Number 720 in LNCS, pages 655-666. Springer Verlag, 1993.

24. Bran Selic, Stuart Kent, and Andy Evans, editors. *Proc.* 3rd *Intl. Conf. UML 2000–Advancing the Standard*, number 1939 in LNCS. Springer Verlag, October 2000.

25. Harald St¨orrle. *Models of Software Architecture. Design and Analysis with UML and Petri-nets*. PhD thesis, Ludwig-Maximilians-Universit¨at M¨unchen, Institut f¨ur Informatik, November 2000. to be published.

26. Bruce Whitenack. RAPPeL: A Requirements-Analysis-Process Pattern Language. In Coplien and Schmidt [7], pages 259-292. Based on the proceedings of PLoP'94. Available also at www1.bell-labs.com/user/cope/Patterns/Process/index.html.

征稿

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

Tom DeMarco
Timothy Lister
Wien wartet auf Dich!
Peopleware
in deutscher Ausgabe
2. Auflage mit acht neuen Kapiteln
Der Faktor Mensch
im DV-Management
《人件》中译本即将发行
HANSER

小项目的 RUP 应用

金哲凡 译

查看评论

问题描述

软件工业现在名声不佳，这已经不是什么秘密了。媒体中经常有报道，一些项目被中途取消，一些项目延误几年时间，一些项目超过预算达数百万美元，等等此类。原因何在？是语言、CASE 工具、编译器等的技术还不够先进？是硬件还不足以运行开发中的软件？或者，软件工程师们不能胜任他们的工作？

不，真正的原因，是只有少数软件开发单位开发出了适合他们自己的开发过程。软件的开发并不是随意的事情，为了开发出高质量的产品，我们必须以符合逻辑的顺序遵循某些特定的步骤。这种思路在任何生产具体产品的工业里都是正确的，它没有理由在软件工业里失效。既然我们喜欢称呼自己为软件工程师，现在是不是应该开始真正用工程化的方法来做事呢？要做到这一点，我们需要过程。

其他工程规范面临的问题一般来说要比我们简单一些，它们的技术更新速度比较慢，而且时间期限普遍比较长。通常，在进行完整的产品的生产前，他们总有时间开发出产品原型，并进行充分的测试。以福特汽车公司为例，他们常常花费数百万英镑进行新汽车模型的研发，在构建产品装配线之前，为了确认一切情况良好，他们会在保密的地点制作许多原型产品来进行大量的测试。他们知道一个成功的模型将带来稳定的投资回报，因为无数的产品将被销往全世界。

软件则不同，我们通常只能把系统表示一次，福特的 R&D 过程实际上就是我们的生产模型，并且在很多环境下这还是我们唯一的机会。因此我们要做的，是第一次就做对，之后次次都对。实现这一点的唯一的办法，就是遵循一个有效的过程。SEI 和 CMU 把 CMM 2 级表述为“可重复级”并不是偶然的（1 级通常被称为“混乱”）。

媒体报导的许多项目是大规模、高度复杂、花费数百万英镑的开发项目。这些高风险的项目都遵循了某个过程，即便这个过程并不特别优秀，它也是必须的。而其他大量的项目通常是一些小规模的项目，不太容易直接匹配某个特定的过程，而这些项目的开发就有其特殊性。

最后，瀑布模型由于其先天不足而给大量的工程项目造成了负面的影响，它增加了项目的风险。由于测试被延迟到开发过程的后端，所以没有一个途径减轻项目的技术风险。延迟消除技术风险本身就是项目成功的额外风险。瀑布模型的另一个问题是编码被延迟到需求、分析和设计都文档化，在内部取得一致意见并固定下来之后，整个瀑布模型很大程度上是一个文档驱动的过程。

小项目特有的问题

进入市场的时间

今天，很多小项目是为了电子商务应用开发的，这些应用一般的生命周期为 3 个月。之所以产生这种约束，是因为客户需要抢先使网站进入运作。Amazon（亚马逊）就是从中获益的一个范例。我猜想本文的大多数读者曾用过 Amazon，而知道 BOL（Books On Line）的就要少的多。这种时间约束给了开发团队很大的压力，通常的结果是不得不在项目更早的阶段就开始进行编码。

团队规模

通常人们都认为小项目只需要小团队。很大程度上这种想法是对的，但是特定的任务仍然需要特定的技能。这意味着开发团队要么必须能拥有多项技能，要么与特定技能相关的活动不得不被取消，而后一种情况更加常见。举例来说，对项目 X，开发团队决定用用例来捕获需求，并使用 UML 对用例进行分析，以此将设计工作推进到分析模型。然而，由于缺少必要的 UML 建模技能，用例开发出来，流程定义好之后，代码就直接从用例描述中产生出来。代码直接来自用例，与过去很多项目的直接从需求进行开发并无不同。造成这种状况的原因是前述的时间约束，加上缺少合适的技能。

规范化程度

在传统上，大型工程项目的开发过程都高度规范化。而小项目的开发方式的规范化程度总是低得多。大型项目通常有更多严格的复审，而小项目在从一个阶段进入到下一个阶段时，总要或多或少地假定前一个阶段的任务已经圆满地完成了。这种状况部分也是因为前面讨论过的时间约束。

变革文化

最后一个问题是“变革文化”，这个问题在实行任何过程时都会碰到，但在小项目中尤其突出。类似语言、设计套路等技术的变更总是比较容易被接受，碰到的反对也较少，因为这些变更增强了团队的技术能力，被团队成

员看作一件好事。而过程的变更总是会受到怀疑，“原来干得很好，为什么现在要这样做？”这样的问题我们听到得太多了。过程经常被视作累赘，带来不必要的额外工作。实际当然并非如此，我们在后面将讨论到这一点。

何为解决之道？

实现一个过程

毫无疑问，生产可预期、高质量软件的唯一有效的办法，是拥有固定的开发过程。实现一个过程，就是让整个团队通过执行一套明确定义的行为来实现项目目标。

什么是过程？

SEI 在 CMM 框架中给出了如下定义：

1. 进程（Procedure）—进程是对“需要做什么”的定义
2. 过程（Process）—过程定义了谁来执行进程

根据 Rational 的观点，过程有如下定义：

一个过程定义了谁做什么，什么时候做以及如何实现一个特定的目标。

这些观点略有不同，但很明显，在通向项目目标的道路上，一个基本概念将与我们长期做伴：必须明确地定义要进行的活动、活动的输出结果和执行的人员。

我们为什么需要过程

Grady Booch 有一句精彩的话，很好地概括了为什么我们需要过程：在任何时候，好的人员加上好的过程，总是优于好的人员没有过程。

正如我们已讨论过的，在小规模的软件开发项目面临的问题中，时间约束是一个基本问题，而过程能提高生产率。因为开发团队理解了软件应该如何开发，整个团队/组织以一个标准工作，所以当团队成员需要从一个项目调到的另一个项目去时，不会出现“本项目不那样做”综合症。

运用过程时，使用标准也是一个重要的问题，软件开发的不少特征已经被广泛确认为整个行业的最佳实践，运用过程使得实行这些最佳实践变得简单。整个开发过程变得更加可预期，从而大大增加了产生高质量软件产品的可能性。

下一节我将讨论近年来发展起来的几个最重要的开发过程。这些过程发展起来都是为了一个目的：赋予开发团队产生高质量软件的能力。可是为什么我们要用它们？为什么我们不开发自己的过程呢？答案很简单，这些过程经过了长时间的进化，能与最佳实践很好配合。开发自己的过程则将花费大量的时间和金钱，其开销比培训团队/组织使用现有的过程要大得多。

可选过程

接下来我简要地介绍4种过程。我不准备过多地进入细节，不过你仍会发现他们有许多共同点。如果需要关于其他方法的信息，在http://www.infosys.tuwien.ac.at/cetus/oo_ooa_ood_methods.html有许多相关的链接。

DSDM

动态软件开发方法（Dynamic Software Development Method）是在1994、1995年发展起来的，其主要目的是消除人们对快速应用开发（Rapid Application Development）的神秘感。它把许多最佳的工业实践标准化了，其目的是保证DSDM项目产生高质量的软件产品。

该方法基于9个基本原则，它们是：

1、 必需要有用户的积极参与；

2、 DSDM团队必须有作决定的权力；

3、 开发活动的焦点是频繁的产品发布；

4、 发布的产品能否被接受的基本判断标准是是否符合业务用途；

5、 为了收敛到一个精确的业务解决方案，迭代和渐进开发是必须的；

6、 开发中的所有变更都是可逆的；

7、 高标准地制定需求基线；

8、 测试贯穿于整个生命周期；

9、 所有参与者的协作和交互至关重要。

DSDM 项目的生命周期是一个迭代的过程，它不断产生出功能和适用性递进的发布版本。并且在本质上，它是一个以用户为中心的方法，在开发过程中吸收了用户群的意见。DSDM 项目的生命周期如图 1 所示。

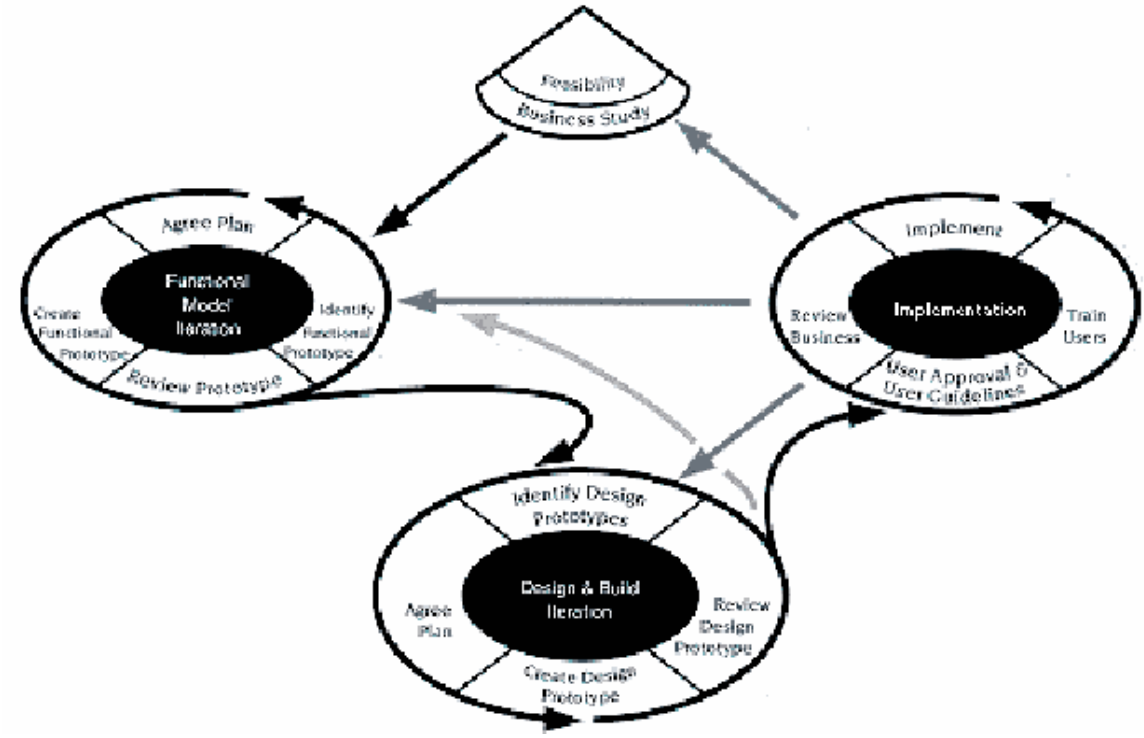


图1 DSDM开发过程

DSDM方法还用到了固定间隔的时间框来定义递进发布的日期。需求按照MoSCoW规则进行分类（必须有 Must have，应该有 Should have，能够有 Could have，想要有 Want to have）。如果某一阶段的开发时间紧张，那么“能够有”和“想要有”的需求项目可以暂时不做，以后再把它们补充进来。这保证了核心业务的正常发布。图2显示了DSDM方法相对于传统方法的灵活性。

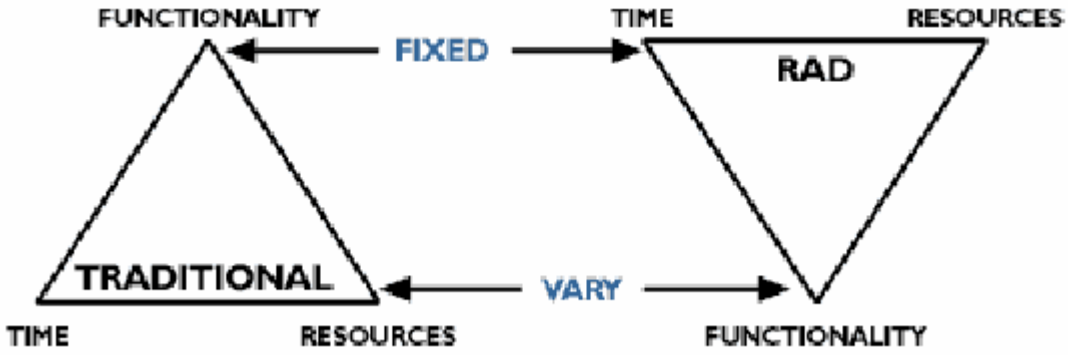


图2 DSDM开发和传统开发相比的需求灵活性

总之，DSDM过程覆盖了软件工程的许多问题。该方法的基础之一是用户群的参与，这一点的重要性在Standish Group的1995年Chaos Report中被确认。另外，DSDM过程允许在需求方面有一定的机动性，其好处在于保证了基本业务需求，但这样也可能使一些潜在的用户感到不满。如需DSDM更多的信息请访问 http://www.dsdm.org。

Open 过程

面向对象的过程、环境和符号都是 OPEN 协会开发的，它们服务于基于 OO 和组件的工程项目。这个过程被定义为一个框架，名为 Open Process Framework（OPF），因而它能按照任何开发组织的需要进行裁剪。由于它非常紧密地与 CMM 相关，故它是一个非常广泛的过程。OPEN 和 RUP 之间也有很多相似点。OPEN 过程的生命周期如图 3 所示。

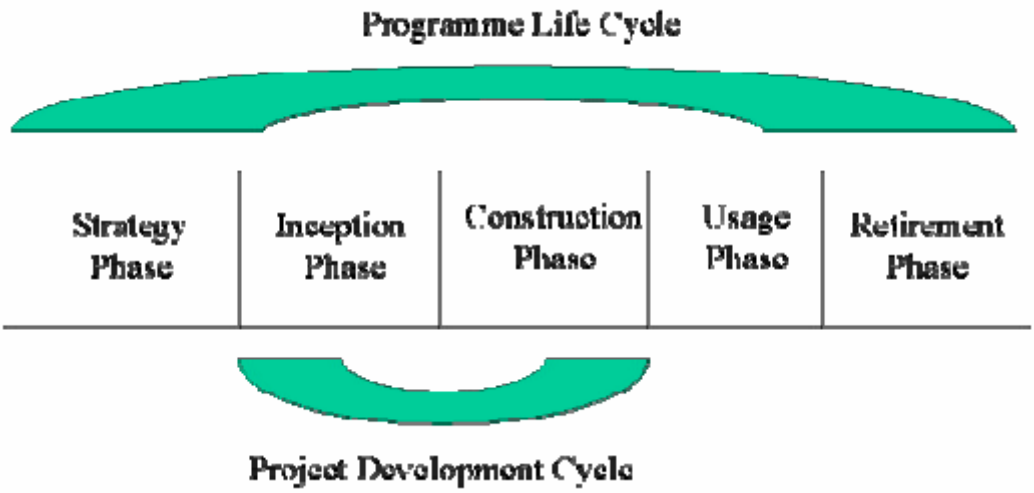


图3 OPEN过程

OPEN 过程由不同的组件组成，这一点与 RUP 很相似，对它可以进行裁剪以适应组织的需要，或者在更低一个级别上，适应个人的需要。这些组件如图 4 所示。

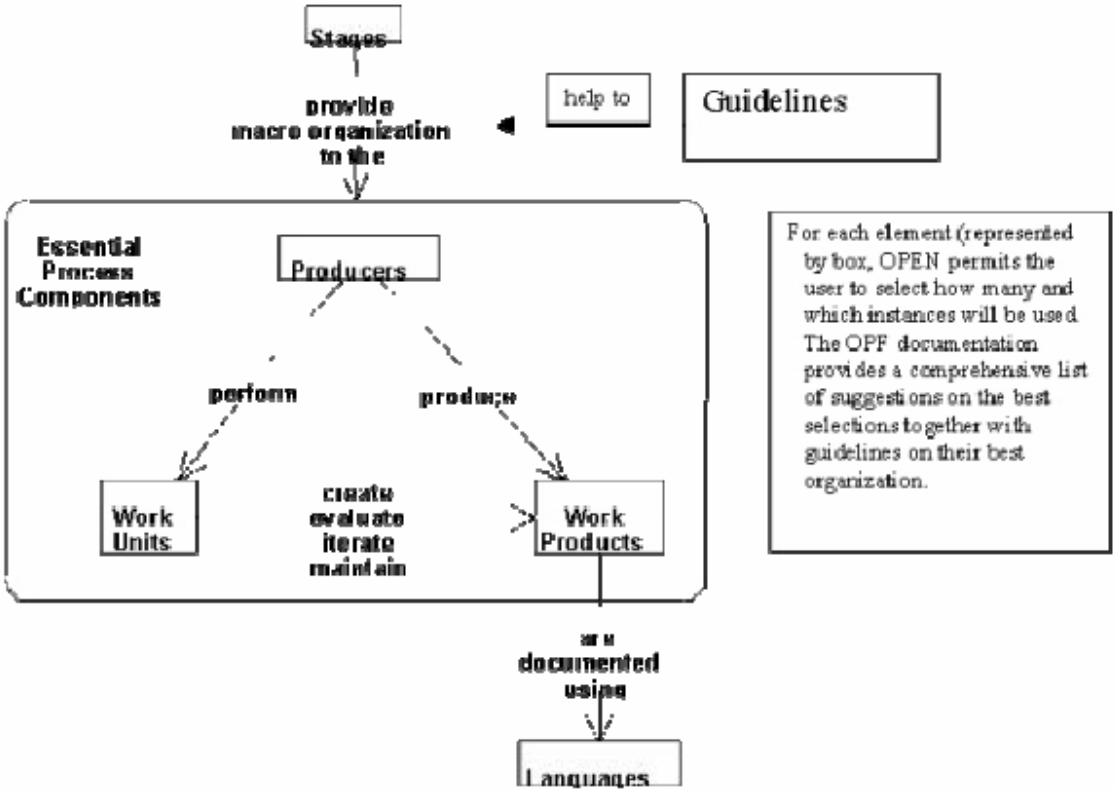


图4 OPEN过程组件

如我所言，对该过程可以进行裁剪以适应个人需要。由于策略阶段被定义为程序生命周期的一部分，所以这种裁剪在组织一级和项目一级都是必须的。项目和程序的区别在于程序的活动（工作单元，图4）对所有项目都是一样的，而项目活动与特定的开发项目密切相关。这是一个非常详细的过程，很值得进一步的研究，目前它还没有固化到任何一个CASE工具或套路中去。在http://www.open.org.au可以得到更多的信息。

OOSP

在基本的生命周期方面，面向对象的软件开发过程与 OPEN 和 RUP 没有很大的不同，不同的在于它使用过程模式来定义过程。过程模式与设计模式和结构模式非常相象，它对已被发现的开发中的问题和解答进行了定义。它针对的是在成功的软件开发中使用的技术、行为和活动，其生命周期如图 5 所示。

项目管理、质量保证、培训、风险管理、重用性管理和环境管理贯穿整个过程的始终，在各个阶段都要执行。

另外需要记住的一点是这些都是模式，它们只界定了做什么（what），而没有界定怎么做（how）。如果您还记得 CMM 的定义，这是一个进程（procedure）而不是一个过程。

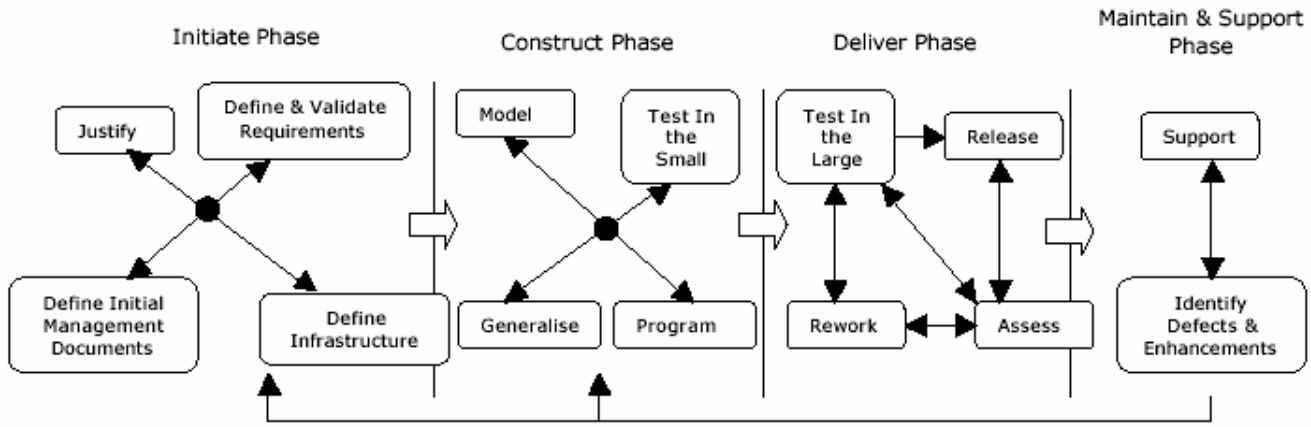


图5 OOSP生命周期

生命周期被分为四个阶段，每个阶段包含一些步骤。步骤里有一些需要进行的活动。对每个级别都有模式的定义。如同 OPEN 过程，OOSP 也定义了维护阶段，而 DSDM 和 RUP 没有。

由于它完全由模式组成，OOSP 是所有我们讨论的过程中最具灵活性的一个。而不定义 HOW 也给过程的实现带来了一些风险。因为许多开发团队配备有他们所认为的配置管理，如果只包含纯的 check in 和 check out，就不能被看作真正的配置管理。

OOSP 其实是一个目的非常明确的系统，它考虑了如下事实：重用是 OO 背后的主要驱动之一（也是构建阶段包含“泛化”步骤的原因）。它也是一个迭代的过程，步骤之间的箭头指明了如果需要的话可以重新访问哪些步骤。阶段也是可迭代的，同样有很多箭头指明了修补缺陷和增加功能应返回哪些阶段。OPEN 过程有一个缺点，它过于广泛，以至于开发者可能要花很多时间来搞清楚应该做什么，然后才能真正去做。

RUP

在本文谈及的所有过程中，Rational 软件公司的 Rational Unified Process（RUP）也许是除了 DSDM 之外应用最为广泛的一个了。这很大程度上是因为 Rational 工具集中的一些工具软件得到了很广泛的使用。

RUP 的基本原则，被称为“六种最佳实践”，它定义了任何开发过程都有的基本要素，即：

1、 管理需求

2、 迭代开发

3、 可视地建模

4、 使用基于组件的结构

5、 不断地验证质量

6、 对变更进行管理

RUP过程实现了基于如上基本原则的最佳实践，同时其生命周期是以Walker Royce在《Software Projcet Management》中定义的螺旋模型为基础设计的。RUP的生命周期有4个阶段，每个阶段包含若干迭代步骤，每个迭代过程被定义为一个小型的瀑布模型，输出一个可执行的版本（内部的或外部的）。RUP生命周期如图6所示。

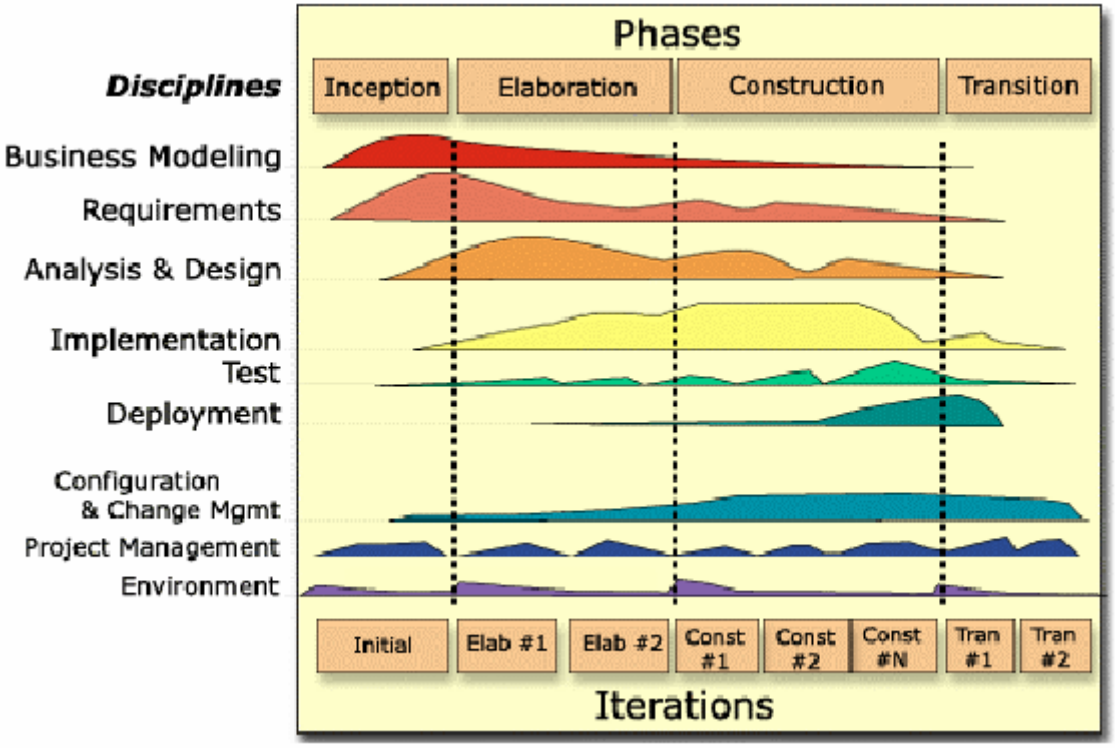


图6 RUP生命周期

根据定义，RUP过程由两个结构组成，一个沿着时间轴给出了阶段和迭代步骤，另一个在左边自上而下地包含了科目。科目是多组活动，每组活动由具有相同技能的角色来执行。活动通常都生成产品。在科目中活动的定义取决于项目当前的阶段和迭代步骤。这与构成open过程的组件很相象。

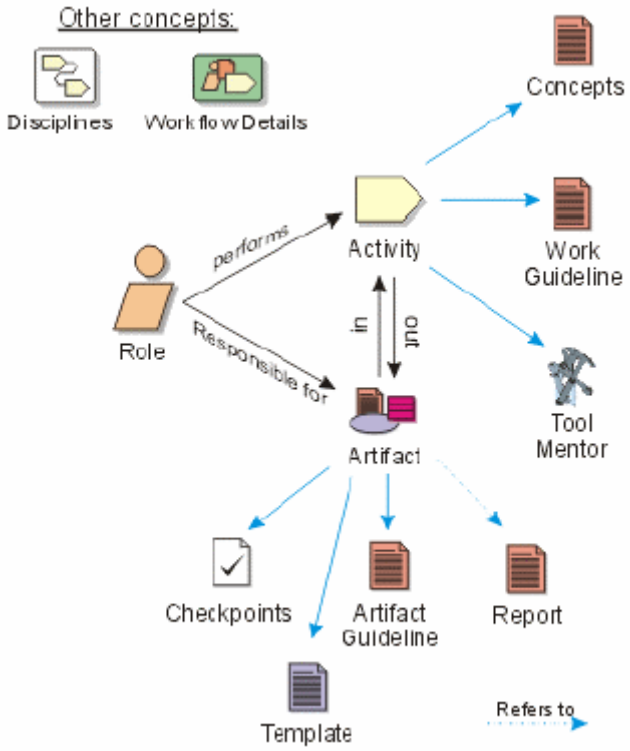


图7 RUP组件

如同DSDM，RUP是一个用户驱动的过程，需求用用例来捕捉。从这里开始在RUP过程背后有两种主要的驱动力：一、定义和验证构架，二、减少风险。通过尽早解决风险，成功的阻力戏剧性地降低了。

在某些方面RUP被简化了，它没有投标和维护阶段。在一些CMM的关键区域也有简化，例如CMM2级的子合同管理。一些人认为这样做的目的是鼓励用户使用Rational的工具集，这固然不错，它同时也正是RUP有力的地方。

RUP过程包含了一些指导方针，称为工具顾问，它带领你通过一个小型的教程，学习如何执行过程定义的活动。虽然并不特别推荐这样做，但这确实使开发团队只需最少的培训就能使用过程和工具。另一个好东西是路线图，它指导了如何根据所开发的应用的类型对RUP过程进行裁剪。

使RUP超出其竞争者的最大优点也许就是它的发布办法。RUP过程的存在形式是一个web站点，不论开发团队在何处，他们都能轻松的访问。在今日现代化的开发环境里，地域上分散的团队并不少见，因此所有团队成员访问同一个开发过程的能力是至关重要的。如果你把过程说明装订成厚厚的一叠寄出去，其结果通常是被扔到架子上积灰尘。

选择哪个过程？

虽然可以说任何过程都比完全没有过程好，但这不是一种可取的态度。当选择使用某个过程时，不管是不是上面提到过的几种（在“过程选择”提到的链接里有充分的选择），最重要的是时刻记住哪个过程最适合你的组织。

上面的每个过程都有自己的优点和缺点，有一个办法可以评估一个过程是否符合某个特定个体的利益：找到你的组织的弱点，然后选择最能弥补这些弱点的过程。当然，这样做也可能导致一些问题，因为这个过程对你的组织的长处可能有负面影响。

为什么选择 Rational Unified Process

得到开发团体最广泛接受的过程或许就是RUP，它由在业界有深厚实践基础的Rational公司提供，它们的CASE工具也是应用最广泛的。它的过程与工具配合，使得过程实现中的某些部分的工作可能比其他部分轻松一些。另外，将工具集成进来可以降低过程的开销，我相信，尽管有前面提到过的一些不足，这仍是最好的选择。本文的余下部分将讲述我们是如何在Dunstan Thomas的工程项目里实现RUP的。

RUP 实现的例子

Dunstan Thomas 的 RUP 实现

使用路线图

我们已经提到过，RUP包含一种称为路线图（Roadmap）的工件，其用途是给过程工程师（参与过程定制活动的角色）提供指导，根据所开发应用的类型对过程进行裁剪。这种裁剪用一种称为开发案（Development Case）的工件来表达，它定义了执行和不执行哪些活动，不执行某些活动的原因，活动产生的工件，工件何时产生，何时更新，如何对它们进行检查以及生产它们所需要的工具。如果组织或项目需要，还可在开发案里定义RUP不包含的角色。下面列出了已有的一些路线图：

Small Projects developed by Rational

eBusiness Solutions developed by Rational

Component Solutions developed by Applied Information Sciences Inc

Evaluating Quality developed by Rational

Usability Engineering developed by Rational

Websphere Enterprise developed by IBM

这些路线图自然无法覆盖所有的应用类型，比如开发实时解决方案的路线图就不在其中，不过由于在现实中RUP过程得到了大量公司的支持并且不断地在更新，它今后出现的可能性也很大。在Dunstan Thomas公司我们以小项目（Small Projects）路线图为基础实现了RUP。

生成开发案

以我作为一个顾问的经验，在客户实现一个过程时，开发案的生成通常是一个引发对项目的疑问和反对最多的阶段。通常RUP首先提供的是培训，这使得大家热情大增，项目主管和团队成员都开始相信，它们有了一个能解决其问题的不错的过程。之后，当我们与项目主管面对面开始讨论开发案模板时，疑问来了。原因很简单，模板覆盖了所有的可能性，它也许过于广泛了。下面表1给出了开发案模板中需求科目的工件完整列表。

Artifacts	How to Use				Review Details	Tools Used	Templates Examples
	Incep	Elab	Const	Trans			
Actor							
Boundary Class							
Glossary							
Requirements Attributes							
Requirements Management Plan							
Stakeholder Requests							
Software Requirements Specification							
Supplementary Specification							
Use Case							
Use-Case Model							
Use-Case Package							
Use-Case Storyboard							
User-Interface Prototype							
Vision							

表1 需求科目的工件

对此常见的反应是：“我们必须产生所有的工件吗？”，或“这太多了，我们没时间全都做”。事实上整个RUP、包括模板都是可定制的。下表给出了Dunstan Thomas的需求科目模板：

Artifacts	How to Use				Review Details	Tools Used	Templates Examples
	Incep	Elab	Const	Trans			
Glossary							
Supplementary Specification							
Use-Case Model							
Vision							

表2 DT的需求科目工件

可以看到，这要少的多，在需要的工作量方面给人的印象也比较好。粗一看好像很多内容被删去了，可其实角色、用例、用例包都是用例模型的一部分，把它们分开表达是多余的。

这下，看起来我们已经决定了要产生哪些工件，但你可能会想：等等，RUP的基本思想是避免文档驱动，这样做不是违背了原则吗？对这个问题的回答是：不，开发案不仅仅是工件的列表，它同时记录了与原始版本RUP中的完整科目的差异。决定采纳哪些工件，是对将执行的活动进行确认的结果，而不是其他。

定制工件

RUP的所有标准组件中最有价值的可能就是模板。除了需要Rational Rose或其他建模工具生成的模型工件之外，所有工件都有模板。模板包含了不同规模的项目产生高质量文档所需的所有信息。每一节都有注释，给出这一节的目的和需要包含哪些信息等细节。这些是Rational已提供的，而组织或项目还可以根据需要自由地对其进行定制。

合并到外部过程

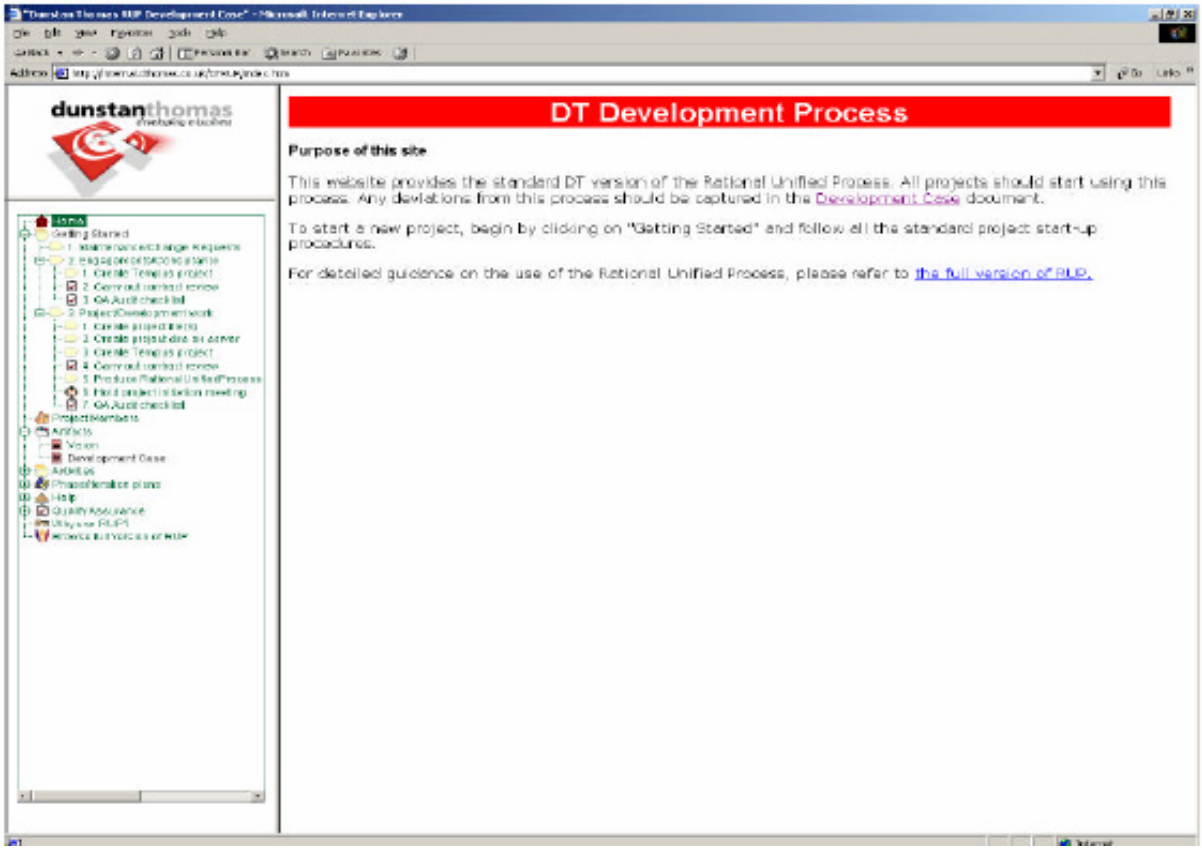
到目前为止我们只看到了项目与开发相关的一面，但在Dunstan Thomas项目开发和运行的过程中，情况要复杂得多，开发必须合并到我们的全局过程中去，一个主要的部分是开发过程影响到我们的ISO9001认证。这种影响主要来自于项目初期执行的一些活动，其次来自项目的整个生命周期。在后面的章节里你将看到我们如何将其合并到我们的全局过程。

使用项目网站例子

RUP提供了一个项目网站的框架，它同样是完全可定制的，我们用它作为Dunstan Thomas过程的实现。下面的屏幕图片给出了Dunstan Thomas项目网站的一个例子。

左边的树形浏览器包含了按照ISO标准的所需要的活动，每个项目链接到其主页，里面给出了活动的细节。还包含了辅助质量保证工作的检查列表，项目组成员的网页给出了角色和责任的细节，帮助部分则包含了对特定活动的指导。

在树形浏览器的底部，你会注意到有一个链接指向完全版本的RUP。之所以这样做，是因为将所有需要的信息合并到项目网站是不可能的，开发者仍然需要使用RUP过程来得到项目当前所在的阶段/迭代步骤的活动的细节。



总结

Dunstan Thomas 为其工程项目选择了 RUP，因为我们觉得它提供了一定的灵活性，这正是我们所有的项目都需要的。本文提到的各个过程都有其优点和缺点（RUP 也不例外），而 RUP 结合 Rational 套件的工具集，是今天的胜者。它解决了我们和其他许多开发组织在工作中面临的大量问题。最后，RUP 或其它过程都不应该被看作“银弹”，它不会，也不能解决你的所有问题。正如 Douglas Adams 在《Mostly Harmless》中说的：在一个看来完全是傻瓜都能胜任的设计工作中，人们常犯的错误就是低估了傻瓜的智能。

参考文献

Web 上的参考资料

Rational Software – http://www.rational.com

The OPEN consortium – http://www.open.org.au

The DSDM consortium - http://www.dsdm.org

常用的软件过程链接

Scott Ambler’s site (lots of great info here) – http://www.ambysoft.com

Software Engineering Institute – http://www.sei.cmu.edu

There are lots of other sites, but the following has links through to lots more

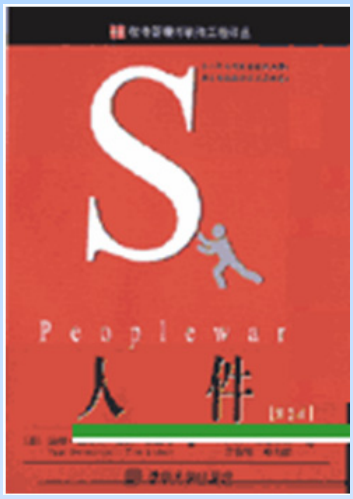
information/sites: - http://www.ispi.co.uk

参考书

Kruchten, Philippe. 2000. The Rational Unified Process, An Introduction, Second Edition. Reading, MA: Addison Wesley Longman, Inc.

Royce, Walker. 1998. Software Project Management: A Unified Framework. Reading, MA: Addison Wesley Longman, Inc.

《人件》：老板，别把开发人员当成牲口!!!



Tom Demarco，Tim Lister

翻译：UMLChina 方春旭 叶向群

http://www.peoplewarecn.com

人件中文版网站

Frederick P. Brooks, Jr--《人月神话》第 19 章

近年来，软件工程领域的一个重大贡献是 DeMarco 和 Lister 在 1987 年出版的《人件》，我衷心地向我的读者推荐这本书。

Edward Yourdon

《人件》在 1987 年出版后，立即成为最畅销的作品。当人件第一版出版时，我写了一份评论，“我强烈推荐你买一份人件给你或你的老板，如果你是一个老板，那么为你部门的每一个人买一份，并给自己买一份”。这建议在 12 年后的第二版依然有效，并且更加热烈。

Mark A. Herschberg

我只学了办公环境的章节，就辞掉了原来的工作！

Joel Spolsky

我想，微软成功的原因之一就是公司里的所有经理都读过《人件》

Steve McConnell

由于本书第 1 版的赫赫声名，新版的《人件》是我不用看就会决定购买的少数几本书之一。

中文译本即将发行！

把业务对象连接到关系数据库（上）

Joseph W. Yoder、Ralph E. Johnson 等 著，Happy 译

查看评论

摘要

这些模式描述如何把业务对象映射到非面向对象的数据库中。面向对象和非面向对象这两种技术存在着阻抗不匹配，因为对象由数据和行为组成，而一个关系型数据库则是由表和它们之间的关系组成的。虽然不可能完全消除这个阻抗不匹配，你可以遵循适当的模式使之最小化。适当的模式可以向开发人员隐藏持久化细节，而让他们专注于理解域问题而不是如何将对象持久化。

简介

使用关系数据库的面向对象系统开发人员通常要花费大量的时间来将对象持久化，这是因为在两种技术间存在一个基本的阻抗不匹配。对象由数据和行为组成，通常可以继承，而关系数据库包括表、关系和基本的谓词计算函数用以返回想要的值。

为避免对象和关系之间的阻抗不匹配，一种方法是使用面向对象数据库。然而，系统通常需要将对象存入一个关系型数据库。部分原因是一个系统需要关系型理论或关系型数据库的成熟性，部分原因是公司策略就是使用关系型数据库而非面向对象数据库。无论是什么原因，一个将对象存入关系型数据库的系统需要提供一个减少这个阻抗不匹配的设计。

本文只描述了将对象映射到关系上的部分模式语言，但是它描述了我们认为在其他地方没有描述充分的模式。全部模式的概要可以参见[Keller 98-2]，其中阐述得较好的模式是关于关系型数据库设计和优化的[Brown 96][Keller 97-1,97-2,98-1]。Serializer 模式[Riehle et al. 1998]描述了如何串行化对象，让它们可以向不同后端存储和获取，例如文本文件、关系数据库和 RPC 缓冲。

我们曾使用或研究过若干持久对象系统（GemStone[GemStone 96], TopLink[TopLink 97-1,07-2]和ObjectLens[OS 95]）。另外，我们用 VisaulAge for Smalltalk 为 Illinois Department of Public Health(IDPH)实现了一个简单的持久化框架，这里介绍的模式存在于所有这些系统中。商业系统在这些模式上的使用通常比我们的框架更加彻底，我们曾更想购买一个持久化框架，但是我们的预算无法负担它们。我们这些需要使用持久框架的应用系统很简单，涉及到几十个数据表，每个应用管理一个病人的病历信息，所有应用共享病人统计信息，例如病人姓名、地址和医院、医生的信息，而每个应用各自负责某个领域，例如病人的免疫、血液检查等。虽然一个应用能够管理一个病人的大量信息，在某个时刻，它将只检查一个病人。本文的示例将向您展示在为 IDPH 开发的应用中，如何协同使用这些模式，解决持久化 Name 和 Address 对象的问题。

这些模式串在一起，手拉手地工作，来解决上面提到的阻抗不匹配问题。一个***持久层***将开发人员和实现持久化的细节隔离开，并保护开发人员不为变更所困。***持久层***是构建一个层的特例，保护您远离应用程序和数据库的变更。实现***持久层***的方式之一是通过一个 PersistentObject，另一个方法是通过一个 Broker[BMRSS 96]。

向数据库读取和写入需要基本的创建、读取、更新和删除操作，虽然每个对象能够有它们自己的访问数据库接口，但是如果您的系统向***持久层***提供一组共通操作，那么所有的对象都可以使用，这样的系统就会更易使用和维护。不管您采取哪种实现***持久层***的方式，都需要支持 ***CRUD***（创建、读取、更新和删除）操作。

CRUD 操作最终会使用 SQL 代码访问数据库，有某种 ***SQL 代码描述***来构建实际的数据库 SQL 调用是很重要的。

当从数据库取出值或者将值存回数据库，系统必须进行属性映射来映射数据库字段值和存储在对象属性中的值。阻抗不匹配的一部分就是关系型和对象系统有着不同类型的数据，映射对象和数据库的值需要转换两种技术中值的类型。

当对象属性发生改变后，将它们存入数据库是很重要的，因此，任何持久对象系统都应该使用某种***变更管理器***来跟踪哪些对象发生了变化，使得系统能够跟踪到哪些对象已经被改变，以确保根据需要保存。***变更管理器***同时也有助于减少数据库访问，因为它只为改变过的对象创建事务并保存。

因为在面向对象系统中每个对象都是唯一的，通过一个 ***OID 管理器***为每个新对象创建一个唯一标识就很重要。同时，支持事务也是非常重要的，它确保改变一个对象是一个原子操作，可以通过一个***事务管理器***回滚该操作。任何访问一个 RDBMS 的系统将通过某种***联接管理器***提供对目标数据库的联接。通过一个***表管理器***处理数据库表名、字段名也是非常有益的。

表 1 中的模式目录概括了本文中讨论的模式，它列出了每个模式的名称，以及它所解决问题的简要描述。

模式名称	描述
持久层	提供一个层，将您的对象映射到关系数据库或其他数据库上
CRUD	所有持久对象至少需要的创建、读取、更新和删除操作。
SQL 代码描述	定义实际的 SQL 代码，从关系数据库和其他数据库中取得值，被对象所用，反之亦然。它被用来生成执行 ***CRUD*** 操作的 SQL 代码。
属性映射方法	映射数据库值和对象属性值，这个模式也处理复杂对象的映射，根据数据库表的一条数据行产生对象。
类型转换	和***属性映射方法***一起使用，在数据库类型和对象类型之间转换，确保数据完整性。
变更管理器	为维护数据完整性，跟踪对象值的变化情况，由它决定是否需要写入到数据库中。
OID 管理器	在插入时为对象产生唯一的对象 ID。
事务管理器	当保存对象时提供事务处理机制。
联接管理器	得到并维护数据库联接。
表管理器	管理一个对象和数据库表、字段的映射

表 1 — 模式目录

持久层

别名

关系数据库访问层

动机

如果您构建一个大型的面向对象业务系统，而将对象保存到关系型数据库中，您可能要花很多时间去处理如何使对象持久化。如果您不够仔细，开发系统的每个程序员都不得不了解 SQL 代码以及访问库的代码，从而被数据库所约束。将您的系统从 Microsoft Access 转到 DB2 上会有大量的工作，乃至为一个对象添加很多变量。所以您需要将您的领域知识从对象是如何存储的知识中分离开，保护开发人员不会为这些变化所困。

问题

如何将对象保存到一个非面向对象的存储机制中？例如关系型数据库，而开发人员不必知道实际的实现方式。

特定约束

- 对熟悉数据库的开发人员来说，写 SQL 语句很容易；
- 设计一个优秀的持久化机制需要花费时间，但是它不直接给用户提供什么功能；
- 数据库访问代价不菲，通常需要优化；
- 开发人员应该可以不必担心如何在数据库中存取而专注于解决应用系统的业务域问题；
- 有一个使用模板方法的共通接口，使代码更易重用；
- 使用一个单一的接口将强制所有的类拥有最低程度的共同性；
- 在应用系统的生命周期中，持久存储类型有可能会改变；
- 在应用系统的生命周期中，业务模型有可能经常会变；

解决方案

提供一个持久层，可以从一个数据存储源中生成对象，并可以把数据保存到数据存储源中去。这一层向开发

人员隐藏了对象存储的细节，这实际上是构建一个层(Layer)[BMRSS96]使您自己免于变化之苦的特殊情况。所有持久对象都使用***持久层***的标准接口，如果数据存储机制改变了，只有***持久层***需要改变。例如，公司主管在项目开始使用 Oracle，到项目中期又转到 DB2 上。

系统需要知道如何存储和装载每个对象，有时候一个对象存储在多个媒介上的多个数据库中。一个对象作为另一个更复杂对象的部分，需要跟踪它是哪个对象的一部分，这叫做所有者对象。这个所有者对象的概念使编写复杂查询变得很容易，因此，***持久层***为每个对象和它的父对象提供唯一标识是很重要的，这个唯一标识符和父标识符在实现 Proxy 模式时非常有用，可以作为部件对象的占位符。

总结一下模型名称的使用，***持久层***提供必要的方法，通过构造 SQL 代码提供 ***CRUD*** 操作，提供***属性映射方法***，为对象数据值进行类型转化，访问***表管理器***，提供对***事务管理器***的访问，通过***联接管理器***联接到数据库。同时***持久层***也有助于提供适当的变更管理，并和 ***OID 管理器***协作提供唯一的对象标识。

有很多方法可以实现一个***持久层***，这儿列举一二。

1. 使用一个对象层[Keller 98]。每个域对象从一个抽象的 PersisentObject 类继承，知道如何执行必要的 ***CRUD*** 操作。本文示例使用的就是这种方式，它的主要好处是易于实现。虽然它在每个域类中都要写一些数据库相关的代码，不过这些代码是分开的，易于查找和修改，它可以在必要时进行优化，尽管一个过于优化的系统难以理解。

2. 使用一个 Broker，它可以从数据库读取域对象或将对象写入数据库，Broker 必须知道每个域对象的格式，生成 SQL 语句去读写。这种方式将数据库代码和域对象类分离开来，是一种最具伸缩性的解决方案，不过需要很多基础部件。

3. 用一组数据对象组成一个域对象，这些数据对象和数据库表具有一对一关系。这样，一旦一个域对象改变了，改变相应的数据对象，并且在域对象保存时，他们也将被保存。例如，一个 Patient 的值可以映射到 Name 和 Address 数据表，Patient 对象将拥有映射到 Name 和 Address 的数据对象。VisualWorks 中，ObjectShare 的 ObjectLens 就是采用这种方式构造数据库对象。***持久层***通过这些数据对象管理起来，它很容易实现并易于理解，尽管有些慢，并且开发人员不得不维护数据库表的一一映射关系。

注：主要的决策依据应该看对灵活性、伸缩性和可维护性的要求。

实现举例

现在很多关于正确描述构建 Broker[BMRSS96]的细节工作已经完成，而且我们拥有更多实现层对象的经验，因此，我们的例子将主要关注于这个模式的实现。其他在实现 Broker 中描述的模式在我们讨论中也将简短提到。我们所有的示例代码都将描述基于 PersistentObject 的实现。

图 1 是一个 UML 类图，表示一个***持久层***将域对象映射到关系数据库的实现。请注意在这个例子中，需要被持久化的域对象是 PersistentObject 的子类，PersistentObject 为***持久层***提供接口。PersistentObject 和***表管理器***交互，可以为 SQL 代码提供物理表名，在 SQL 代码生成时，PersistentObject 和***联接管理器***交互以提供必要的数据库联接。如果需要，当需要一个新的唯一标识时，向 ***OID 管理器***请求。这样，PersistentObject 作为一个中间的集线器，为域对象提供需要而它本身没有的任何信息。PersistentObject 为***持久层***提供标准的接口，通过和其他模式一起合作，一旦 SQL 代码准备好了，SQL 语句将被数据库部件触发，在 IBM VisualAge for Smalltalk 中，这些就是 AbtDBM*应用系统。

PersistentObject 的属性如下：

- objectIdentifier — 对象唯一的标识符，可以是数据库键值。
- isChanged — 标志对象是否被修改过，告诉***持久层***这个对象是否要写入数据库中。
- isPersisted — 标志对象是否曾写入到数据库中。
- owningObject — 标识父对象，在数据库中作为一个外键使用。注意这个外键在本对象而不是父对象中。

PersistentObject 的公共方法如下：

- save — 将对象数据写入到数据库中，它将更新或插入行；
- delete — 从数据库中删除一个对象的数据；
- load — 从数据库中返回一个类的单一实例及它的数据；
- loadAll — 从数据库返回一个类的实例集合，包含所有数据，这对为选择列表返回数据非常有用；
- loadAllLike — 从数据库返回一个类的实例集合，包含部分数据；

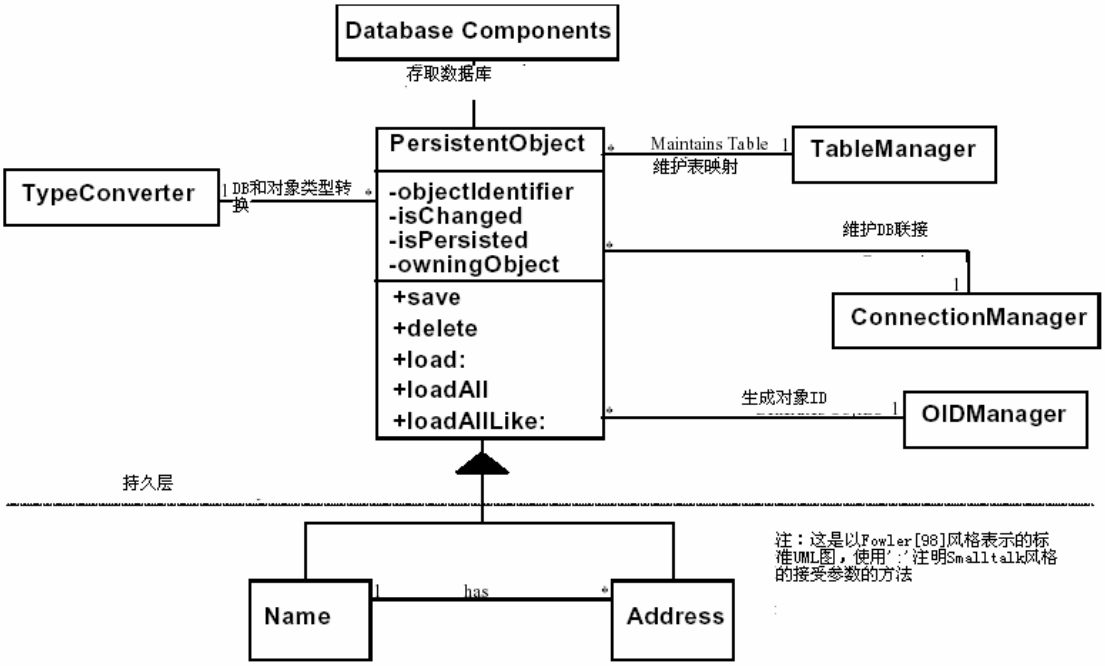


图 1 — 持久类类图

数据库记录可以以三种方式读取：

- 读取一行（PersistentObject>>load:）；
- 读取所有记录（PersistentObject>>loadAll）；
- 读取所有符合条件的记录（PersistentObject>>loadAllLike）

指定一个特定条件，创建一个对象的新实例并为它装载对应的属性集，这个功能可以通过 PersistentObject>>load:和 PersistentObject>>loadAllLike:方法实现。而当您想产生一个选择列表或是下拉列表时，PersistentObject>>loadAll 方法是非常有用的。

下面的示例代码描述了上面所说的 PersistentObject，它们是公共的接口方法，支持事务管理（后文详述）。read:和 saveAsTransaction 方法将在 ***CRUD*** 模式中详述。

Protocol for Public Interface PersistentObject (instance)

`load`

“得到匹配它自己的 PersistentObject 子类的单一实例”

```
    | oc |
    oc := self loadAllLike.
    ^oc isEmpty ifTrue: [nil] ifFalse: [oc first]
```

loadAllLike

“得到匹配它自己的 PersistentObject 子类的实例集合，selectionClause 方法为 PersistentObject 的 read 方法准备 Where 子句”

```
    ^self calss read: ( self selectionClause )
```

save

“保存他自己到数据库中，包含在一个事务当中。”

```
    self class beginTrasacction.
    self saveAsTransaction.
    self class endTransaction.
```

delete

“从数据库中删除他自己，包含在一个事务当中。”

```
    self class beginTrasaction.
    self deleteAsTransaction.
    self class endTransaction.
```

Protocol for Public Interface PersistentObject (class)

loadAll

“从数据库返回我的所有实例”

```
    ^self read: nil.
```

下面是 Name 类的示例代码，Name 有 Address，因此它将有一个需要被存储的部件，这个方法被任何需要被

存储而拥有持久化部件的域对象重载。

Protocol for Private Interface Name (instance)

saveComponentIfDirty

“验证 address 对象的存在，并验证 proxy 模式没有占据这个位置，address 所有者对象被设为当前对象，并且 address 对象的保存也是当前事务中的一部分。”

```
(self address isNil or: [self address isKindOf:
                                PPLAbstractProxy])
    ifTrue: [^nil].
self address owningObject: self objectIdentitier.
self address saveAsTransaction
```

结论

- 把应用开发人员从对象存储细节中隔离开来的另一个好处是易于实现域对象，因此，域模型变更的工作就变少了。总之通过封装对象持久机制的功能，可以有效地对开发人员隐藏对象存储的细节。
- 可以改变数据库技术，而不影响您的应用程序代码。
- 使改变对象存储到数据库的方式变的很容易，因为我们已将需要改变的地方隔离起来了。
- 用户只需要调用同样的接口去持久化对象，开发人员无需检测一条记录是否已经存在于数据库中。
- 用 SQL 代码实现起来很简单的事情，用*持久层*可能使它变的复杂而且有时候难以操作。
- *持久层*的优化很困难。程序员应对不同方式做评测以决定哪一个最适合他的实现。

相关或交互的模式

- 关系数据库访问层是一个很相似的模式，它描述了一个需要持久化的对象交互的层。
- 分层架构[Shaw 96]描述了必要的模式，将体系架构分层，以便在开发中隔离变更。
- 信息系统的分层架构讨论了开发分层系统的实现细节。
- 层[BMRSS 96]描述了分层系统架构和设计时，需要考虑的细节。

已知应用

- GemStone OODBMS 使用*持久层*隐藏一个值是一个持久对象的事实。在实际需要时使用 Broker 为*持久层*获取值，在这个例子中，存储系统不是关系型数据库。
- Caterpillar/NCSA 金融模型框架[Yoder 97]使用*持久层*，所有的值都通过一个 Query 对象存储。这个例子中，应用程序在数据库中不存储任意域对象，只是获取事务。然而，*持久层*仍隐藏了数据库和关系型数据库技术的细节。
- ObjectShare VisualWorks Smalltalk ObjectLens[OS 95]使用*持久层*映射数据对象和数据库表。
- VisualAge for Smalltalk 也使用*持久层*和它们的 AbtDbm*应用。VisualAge 提供图形化联接的 GUI 构建器，形成一个持久机制。
- Illinois Department of Public Health's TOTS 和 NewBorn Screening 项目使用一个和本节例子非常相似的实现方法。
- TopLink、MicroDoc、Sparky 和 Object Extencer[MicroDoc 98, Sparky 98, OE 98]都提供一个*持久层*来将对象映射到关系型数据库中。
- PLoP 登记系统实现了一个*持久层*，将 Java 对象存到 PostGress 数据库中[JOE PUT THE REF HERE]。

CRUD

别名

创建、读取、更新和删除

基本持久操作

动机

试想你有一个 Patient 类，有 Name 和 Address 部件，当你读取一个 Patient，必须同时读取 Name 和 Address。写入一个 Patient 到数据库中将有可能写入一个 Name 和 Address 对象。他们是否有同样的接口去读取和写入呢？也许有些对象需要不同的接口？我们能否给出完全同样的接口，如果可以，是什么?

任何被持久化的对象都要对数据库进行读取和写入，对新创建的对象，它的值也会被持久化，另外，对象也可以从持久存储中删除，因此，如果一个对象需要持久化，至少要提供最小的操作集合，他们是创建、读取、更新、删除。

问题

一个持久对象最小的操作集合是什么？

特定约束

- 保存在数据库中的所有对象需要一个装载、保存自己的机制；
- 在一个地方放置读取和写入的代码有助于对象的演进和维护；
- 如果类只实现同样而且小的接口，可以很容易将他们组成嵌套的类。

解决方案

为持久对象提供基本的 ***CRUD***（创建、读取、更新和删除）操作。其他需要的操作如 loadAllLike:或 loadAll。重要的是要提供足够多的信息以便能够从数据库实例化对象，并保存新建的或改变了的对象。

如果所有域对象都有共同的 PersistentObject 超类，那么这个超类可以定义 ***CRUD*** 操作，而所有的域对象能够从它继承，如有必要，子类可以重载他们以提高性能。

如果*持久层*是使用 Broker 实现的，那么 ***CRUD*** 操作也由 Broker 实现，不论什么情况，*持久层*必须生成 SQL 代码来读取和写入域对象。这样，每个域对象必须能够获得必要 SQL 代码的描述，来访问 ***CRUD*** 操作的数据库。***CRUD*** 和 ***SQL 代码描述***紧密合作，确保这些操作能有效持久化域对象。

示例实现

前面阐述的 PersistentObject 提供了标准接口，一组基本的操作来映射对象到数据库，保存、装载等。这些方法从 PersistentObject 继承，访问 ***CRUD*** 操作。有些 ***CRUD*** 方法需要在域对象中重写。AbtDBM*数据库部件提供了 executeSql: 方法，让数据库执行 SQL 语句并返回值。updateRowSql 和 insertRowSql 将在下面 ***SQL 代码描述***模式中详述。

Protocol for *CRUD* PersistentObject (class)

这个方法指定一个 WHERE 子句作为中介，并返回一组和 WHERE 条件相匹配的对象集合。

read: aSearchString

“从数据库返回一个对象实例集合。”

```
| aCollection |
aCollection := OrderedCollection new.
(self resultSet : aSearchString)
    do: [:aRow | aCollection add: (self new initialize: aRow) ].
^aCollection
```

Protocol for Persistence Layer PersistentObject （instance）

这些方法对数据库保存或删除对象，这些方法要基于对象的值判断执行什么 SQL 语句（insert、update 或 delete），一旦决定，SQL 语句将在数据库中执行。

saveAsTransaction

“保存自己到数据库中。”

```
self isPersisted ifTrue: [self update] isFalse: [self create].
```

```
    self makeClean
```

update

```
    “更新聚合类，然后在数据库中更新他自己”
    self saveComponentIfDirty.
    self basicUpdate
```

create

```
    “插入聚合类，然后在数据库中插入自己”
    self saveComponentIfDirty.
    self basicCreate
```

basicCreate

```
    “在数据库中触发插入 SQL 语句”
    self class executeSql: self insertRowSql.
    isPersisted := true
```

basicUpdate

```
    “在数据库中触发更新 SQL 语句”
    (self isKindOf: AbstractProxy) ifTrue: [^nil].
    isChanged ifTrue: [self class executeSql: self updateRowSql]
```

deleteAsTransaction

```
    “从数据库中删除自己”
    self isPersisted ifTrue: [self basicDelete].
    ^nil
```

basicDelete

“在数据库中触发删除 SQL 语句”

```
self class
executeSql:( 'DELETE FROM ',self class table, ' WHERE ID_OBJ=',
                        (self objectIdentifier printString)).
```

结论

- 一旦你的对象模型和数据模型分析完毕，分析结果可以用 ***CRUD*** 实现，提供一个性能优化的方案，使开发人员从性能优化的考虑中隔离开来。注：如果你的对象模型和数据模型分析完毕，你能够为你的数据库提供优化的性能方案来实现 ***CRUD*** 操作，以向应用开发者隐藏实现细节。
- 基于多少行或何种类型数据（动态、静态或介于两者之间）来获取数据的灵活性是有必要的；
- 简单的实现数据保存到数据库，应用开发人员无需决定是插入还是更新对象；
- 如果对象模型和数据模型没有很好的分析，***CRUD*** 将引起次优性能的问题。这将使开发者的工作变困难，不得不尝试其他的方法。

相关或交互模式

- ***事务管理器***为这些操作提供事务支持；
- ***CRUD*** 和 SQL 代码协作，生成必要的数据库调用。

已知应用

- Illinois Department of Public Health TOTS 和 NewBorn Screening 项目
- ObjectShare 的 VisualWorks Smalltalk ObjectLens[OS 95]使用一个 ***CRUD***，以定义如何操纵简单数据对象。VisualAge Smalltalk[VA 98]也在他们的 AbtDbm*应用系统中使用 ***CRUD***。

SQL 代码描述

别名

查询、更新、插入和删除代码定义

对象查询语言（OQL）描述

通用查询语言（CQL）描述

结构查询语言（SQL）描述

动机

有的地方，不得不编写从数据库中读取、更新、插入和删除值的 SQL 代码以保持对象值和持久存储值的一致性。再一次看看一个拥有 Name 和 Address 部件的 Patient 类，SQL 代码需要读取和写入 Patient 的值，同时也必须存储 Name 和 Address 的 SQL。一方面，你可以硬编码 SQL，来读取和写入数据库，另外你也可以在一个共同的地方存储值，开发一个存放对象到数据库的映射的结构映射，并在运行期动态生成 SQL。

问题

在什么地方存储用来生成 ***CRUD*** 操作所需的必要 SQL 语句的实际描述？

特定约束

- 当访问一个关系型数据库时，对数据库访问的 SQL 代码必定在某处出现；
- 当域模型增加时，SQL 代码的数量也随之增加；
- 编写有效的 SQL 代码需要你对数据模型和数据库有很深的了解；
- 域模型有可能在一个应用的生命周期内频繁变动；
- SQL 代码可以放置在数据库访问需要的任何地方；
- 重复的相似 SQL 代码可能引起维护的问题；
- 从元数据生成 SQL 代码能够隐藏一个对象访问开发者框架的细节，但是有一个性能和维护之间的平衡。

解决方案

提供一个开发者描述 SQL 代码的地方，以维护对象和持久存储之间的一致性。至少域对象需要知道如何执行 ***CRUD*** 操作（创建、读取、更新和删除）。必要的处理 ***CRUD*** 操作的 SQL 代码需要在某个地方定义。

维护对象值和持久存储值的一致性非常重要，同时，提供一个手段，让饱受煎熬的程序员尽可能不会在修改完一个域对象后忘了提交一个 Update 语句也是同样重要的。

这个模式可以以多种方式实现，但要点是 SQL 语句是封装起来的并且要很容易和持久对象关联起来。SQL 代码和域对象紧密关联，那么开发者修改了一个域对象而忘了更新 SQL 语句的可能性就不大了。

实现二者紧密关联的一个方式就是实实在在地为每个操作编写完全的 SQL 代码，然后让***持久层***从域对象读取 SQL 代码，创建数据库联接并执行数据库调用，Broker 和对象都能够从域对象生成 SQL 代码。另一个方式是提供一个对象查询语言（OQL），它是对 ***CRUD*** 所需必要操作的描述，OQL 将被翻译成对数据库必要的调用。另外，还可以使用元数据(Metadata)[Foote&Yoder 1998]来描述 ***CRUD*** 操作，一个 ***CRUD*** 操作翻译元数据来构建适当的 SQL,大多数商业性框架都使用这种方式，他们通过某种 Schema Map[Foote&Yoder 1998]构建这个结构，这些商业性框架大多提供一种可视化语言来构建和操纵这些查询。使用元数据和 Schema Maps 构建的实现和维护会变得复杂些，并且会产生一些非优化的查询，但是他们能使开发者易于描述域对象和数据库之间的映射，特别是在有可视化语言的辅助时。

如果你正实现一个更大规模的系统，你拥有上千行的 SQL 代码和动态的 SQL，他们不够快，你也许想替代他们或修改他们，例如调用存储过程、预编译的 SQL、实现推拉技术或是高速缓存技术，这叫做优化查询模式[Keller 97-2]。

示例实现

当你使用一个数据库，你必须编写一些 SQL 语句来获取、插入、更新和删除记录，例如最简单的形式，如下：

```
SELECT * FROM table_name.

INSERT INTO table_name (column_name) VALUES (values)

UPDATE table_name SET column_name = xyz WHERE key_value

DELETE FROM table_name
```

这看起来很简单，但是如何从一个对象得到值放置到这些语句中呢？

你能够使用一个字符流来放置语句中固定部分和对象的属性，或者，那样可以定义你想获取的列。

```
aStream nextPutAll: 'SELECT';
        nextPutAll:  column_names;
        nextPutAll:  'FROM ';
        nextPutAll:  table_name.
```

或者

```
aStream nextPutAll: 'INSERT INTO ';
        nextPutAll: table_name;
        nextPutAll: (column_names);
        nextPutAll: 'VALUES ';
        nextPutAll: (values).
```

下面是一 Name 类实例方法的例子，Name 类在我们的例子中是一个域对象，名称和地址是管理类应用系统中大多都有的。这是用一个非常小的复杂对象演示***持久层***的简单例子，如上所述，SQL 语句并不非得硬编码不可，如果你的数据库和你的对象模型非常相似，那么，可以从一个数据库 schema 来生成这些语句。

Protocol for SQLCODE （instance）

这些方法为***持久层***提供实际的 SQL 语句，他们将被送到数据库中执行，这个 SQL 语句构成一个字符流并传入到***持久层***执行。这个例子演示了***类型转换***器和***表管理器***的使用，他们将在本文的后面章节详述。

这些方法构建实际被送到数据库的 SQL 语句，为了性能考虑，使用了写入字符流而不是字符拼接。

insertRowSql

“返回一个插入 SQL 语句，语句中包含了从对象得来的值。”

```
| aStream |
```

```
aStream := WriteStream on:(String new).
aStream nextPutAll: 'INSERT INTO ';
nextPutAll: self class table;
     nextPutAll:' (ID_OBJ,
            ID_OBJ_OWN,
            NAM_FST,
            NAM_LST,
            NAM_MID,
            EML_ADR,
            ORG_NAM,
            NUM_PHO)
          VALUES (';
     nextPutAll: (self typeConverter prepForSql:
            (objectIdentifier:= (self getKeyValue)));
     nextPut: $,;
     nextPutAll: (self typeConverter prepForSql:
                        self owningObject);
     nextPut: $,;
     nextPutAll: (self typeConverter prepForSql:
                     (self first asUppercase));
     nextPut: $,;
```

```
        nextPutAll: (self typeConverter prepForSql:
                        (self last asUppercase));
        nextPut: $,;
        nextPutAll: (self typeConverter prepForSql:
                       (self middle asUppercase));
        nextPut: $,;
        nextPutAll: (self typeConverter prepForSql:(self email));
        nextPut: $,;
        nextPutAll: (self typeConverter prepForSql:
                       (self organization));
        nextPut: $,;
        nextPutAll: (self typeConverter prepForSql:(self phone));
        nextPutAll: ')'.
^aStream contents.
```

updateRowSql

“返回一个更新 SQL 语句，语句中包含从对象得来的值”

```
| aStream |
aStream := WriteStream on:(String new).
aStream nextPutAll: 'UPDATE ';
        nextPutAll: self class table;
        nextPutAll: ' SET NAM_FST=';
```

```
        nextPutAll: (self typeConverter prepForSql:
                        (self first asUppercase));
        nextPutAll: ', NAM_LST=';
        nextPutAll: (self typeConverter prepForSql:
                        (self last asUppercase));
        nextPutAll: ', NAM_MID=';
        nextPutAll: (self typeConverter prepForSql:
                        (self middle asUppercase));
        nextPutAll: ', EML_ADR=';
        nextPutAll: (self typeConverter prepForSql:(self email));
        nextPutAll: ', ORG_NAM=';
        nextPutAll: (self typeConverter prepForSql:
                        (self organization));
        nextPutAll: ', NUM_PHO=';
        nextPutAll: (self typeConverter prepForSql:(self phone));
        nextPutAll: ' WHERE ID_OBJ=';
        nextPutAll: (self typeConverter prepForSql:
                        self objectIdentifier).
    ^aStream contents.
```

这个方法为 SQL 的 select 语句提供 where 子句，每个类都有这个方法来决定 select 语句中 where 子句可以使用什么。

selectionClause

“得到一个 where 子句的字符串表示”

```
| aStream app|
aStream:= WriteStream on:(String new).
( self objectIdentifier isNil )
ifFalse: [ aStream nextPutAll: 'ID_OBJ=';
          nextPutAll: (self class typeConverter prepForSql:
                    self objectIdentifier).
   ^aStream contents ].
( self owningObject isNil )
ifFalse: [ aStream nextPutAll: 'ID_OBJ_OWN= ';
          nextPutAll: (self typeConverter prepForSql: self owningObject)].
^aStream contents.
```

Protocol for SQLCODE (class)

这些方法为上面的方法提供表名，并定义在上面产生的 where 子句中，哪些列将从数据库返回。

table

“从*表管理器*返回表名。”

```
   ^TableManager gettable: 'EXAMPLE'
```

buildSqlStatement: aString

“为对象返回读取的 SQL 语句。”

```
| aStream |
```

```
    aStream := WriteStream on:(String new).
    aStream nextPutAll: 'SELECT
              ID_OBJ,
              ID_OBJ_OWN,
              NAM_FST,
              NAM_LST,
              NAM_MID,
              EML_ADR,
              ORG_NAM,
              NUM_PHO FROM ';
       nextPutAll: self table.
    ((aString isNil) or:[ aString trimBlanks isEmpty])
    ifFalse:[aStream setToEnd;
       nextPutAll: ' WHERE ';
       nextPutAll: aString].
    ^aStream contents.
```

Protocol for SQL Code PersistentObject （instance）

每个对象需要向***持久层***提供***SQL 代码描述***，如果对象不需要部分 SQL 代码，它应该返回“shouldNotImplement”。

insertRowSql

```
    ^self subclassResponsibility
```

selectionClause

```
    ^self subclassResponsibility
```

updateRowSql

```
    ^self subclassResponsibility
```

结论

- 灵活性，只返回需要的集合，同时，SQL 语句可以被替换成其他的形式，如存储过程或预编译查询等；
- SQL 语句的性能可以很容易使用数据库工具来判断。

相关或交互模式

- ***SQL 代码描述***使用翻译器模式[GHJV 95]生成数据库语句；
- ***SQL 代码描述***使用构建器模式[GHJV 95]为不同的对象提供相同过程；
- ***SQL 代码描述***能够使用元数据[Foote&Yoder 98]来生成 SQL 语句；
- ***SQL 代码描述***需要知道 Schema[Foote&Yoder 98]以生成正确的语句；
- ***SQL 代码描述***为*持久层*的 ***CRUD*** 操作生成代码；
- ***SQL 代码描述***使用从*属性映射方法*得来的值生成。

已知应用

- Illinois Department of Public Health TOTS 和 NewBorn Screening 项目；
- ObjectShare 的 VisualWorks Smalltalk[OS 95]使用 ***SQL 代码描述***，用来定义如何为简单数据对象执行 ***CRUD*** 操作。VisualAge Smalltalk 也在他们的 AbtDbm*应用程序使用 ***SQL 代码描述***；
- 在 GemStone GemConnect[GemConn 98]，使用 ***SQL 代码描述***，向一个关系数据库读取和写入对象值。

属性映射方法

别名

映射数据库到对象

映射对象到数据库

动机

当从数据库中得到一行记录，每个列的值必须映射到对象的一个属性或一组属性上，同样，当将值存入到数据库中，一个对象的属性必须以某种方式映射到数据库的字段上。试想一下病人的例子，一个 Patient 对象有病人的姓名和性别相关联，它们可以从数据库的病人表中读取，同时 Patient 对象还有一个相关联的地址，这个值也许有一个外键，参照另一个对象例如 Address 对象，这样，当读取一个 Patient 对象，属性映射必须分别将数据库中放置姓名、性别值的字段映射到 Patient 对象的 name 和 sex 属性上，同时，需要创建一个 Address 对象，并映射到 Patient 对象的 address 属性。

问题

开发者在哪儿、如何描述数据库值和对象属性之间的映射？

特定约束

- 对象在属性变量中存放值，而数据库在字段中存放值；
- 一个对象的属性到数据库表字段的映射并不总是一对一的；
- 有些对象需要从多个数据库、多个数据库表中得到值，它们是复杂对象；
- 非面向对象数据库并不能很好地表示层次结构和对象类型；

解决方案

对每个需要持久化的对象，编写一个映射数据库值到对象属性的方法和一个映射对象属性到数据库值的方法。*持久层*将使用第一个方法把从数据库返回的值存储到相应的对象属性中。同样，当 PersistentObject 存储时，*持久层*将使用第二个方法把对象的值送到数据库中。当 PersistentObject 生成 SQL 代码时，它将映射这些数据。

这些方法从数据库返回一行并填入到相应对象属性中，有时若干字段会映射到一个属性上，这些方法必须也能得到一个对象的属性值，通过一个数据库写入例程映射到数据库字段上，通常情况下，一个属性映射到一个或多个数据库字段，但有时也会是多个属性映射到一个数据库字段上。还有些情况，属性值可能从不同平台的不同数据库产生，对这种情况，聚合类将从别的数据库装载，返回对象赋予给当前的属性。

通常至少有两组***属性映射方法***，一个用来从数据库读取值，一是将值写回到数据库。当值已被映射到数据库上，***属性映射方法***需要提供对***类型转换***的调用。

元数据可以用来定义属性映射，可以和 Schema 一起使用，也可以不用它。在这种情况下，翻译器将用来生成属性映射。可视化语言也可以用来描述映射，但这种方法通常很难实现和维护，然而一旦实现了，开发者很容易映射一个对象的属性。

当一个属性被映射到另一个域对象，通常使用 Broker 来延迟初始化，例如上面提到的 Patient 例子，因为很少需要一个病人的地址信息，一个 Broker 可以用来初始化 address 属性，以后，无论何时需要访问病人的地址，病人的地址信息将从数据库读出来并创建 Address 对象，在这个例子中，address 属性也被更新成指向新创建的 Address 对象。

实例实现

一旦数据从数据库取出，他们将从返回的行移到对象属性中去，返回行（VisualAge 中）是一个字典结构，可以按下面方式访问：

```
aRow at: keyValue.
```

接着，你要将值赋予给属性：

```
attribute := (aRow at: keyValue).
```

或者，如果属性要包含一个其他类实例：

```
attribute := ((Class new) owningObject:
                objectIdentifier; youself) load.
```

将从数据库装载一个 Address 类的实例并赋予给属性，如果数据库包含的地址信息在不同平台的不同数据库中，Address 类将从其他数据库装载。

下面是 Name 类的***属性映射方法***，展示了如何映射数据库行到对象以及应用***类型转换***的过程。***SQL 代码描述***模式展示了如何将对象映射回数据库中。

Protocol for Map Attributes (instance)

这些方法从 PersistentObject>>read:方法接受一行 aRow（一条记录），数据表的每行被传入到一个新的实例（被 PersistentObject 初始化），行中的每个元素在赋给属性前，进行必要的***类型转换***，对于复杂对象的情况，属性值通过发送一个 PersistentObject>>load:（如果是集合的话，发送 PersistentObject>>loadAllLike:）到属性类类型，这个方法使用一个带条件语句***SQL 代码描述***实例。例如，在 Name 类中，你将发送 loadAllLike:消息到 Address 类，并以 objectIdentifier 作为参数，这将装载所有的 owningObject 为同一个 Name 的地址。

initialize: aRow (Name class)

“从一个数据行初始化一个对象实例。”

```
objectIdentifier := self typeConverter convertToNumber:
                (aRow at: 'ID_OBJ').
owningObject := aRow at: 'ID_OBJ_OWN'.
isPersisted := true.
first := self typeConverter convertToUpperString:
                (aRow at: 'NAM_FST').
middle := self typeConverter convertToUpperString:
                (aRow at: 'NAM_MID').
last := self typeConverter convertToUpperString:
                (aRow at: 'NAM_LST').
email := self typeConverter convertToString:
                (aRow at: 'EML_ADR').
organization := self typeConverter convertToUpperString:
```

```
                (aRow at: 'ORG_NAM').
phone := self typConverter convertToNumber:
                (aRow at: 'NUM_PHO').
address := (( Address New)owningObject:
                    objectIdentifier;yourself) load
```

结论

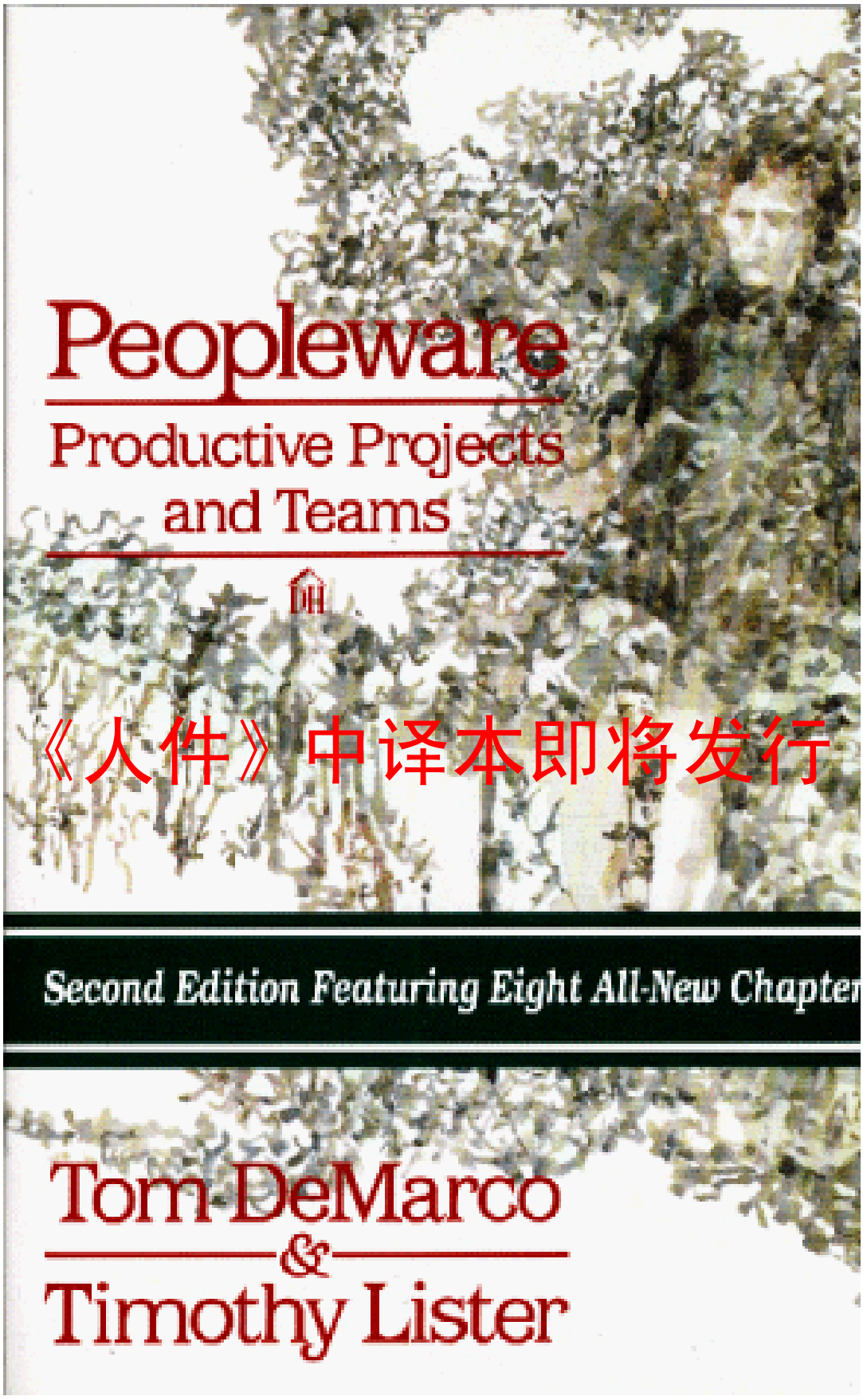
- 当数据库结构改变了，只要改变一个地方；
- 属性名和数据库表字段名无需一致；
- 新进入项目的开发人员很容易对应库表字段名和属性名；
- 随着数据库和域对象演进，***属性映射方法***需要维护。

相关或交互模式

- 在映射值到数据库和从数据库映射值都需要进行***类型转换***；
- 当向***持久层***请求一个调用来读取和写入数据库时，要生成***属性映射方法***所需的 SQL 代码；
- 元数据能够和一个 Schema 一起，用来定义属性映射，或者有可能使用一个翻译器来生成实际的映射。

已知应用

- Illinois Department of Public Health TOTS 和 NewBorn Screening 项目；
- ParcPlace 的 VisualWorks Smalltalk[OS 95]使用***属性映射方法***，用来定义如何处理属性和数据对象之间的映射。VisualAge Smalltalk[VA 98]也在他们的 AbtDbm*应用系统中使用***属性映射方法***。
- JDBC 中的 ResultSet 类是一条数据行，他的方法如 getInt(), getString(),getDate()为属性映射工作，同时也是读取器的***类型转换***。在 JDBC 中映射回数据库，PerparedStatement 类构造 SQL 语句，并以问号‘？’作为参数的占位符，并有一组 setInt(), setString(), setDate()等等。
- GemStone GemConnect 将数据库字段映射到对象属性中。


Peopleware
Productive Projects
and Teams
《人件》中译本即将发行
Second Edition Featuring Eight All-New Chapter
Tom DeMarco
&
Timothy Lister

现实世界电子商务系统的数据库设计

Il-Yeol Song 著，herman 译

查看评论

摘要

本文探讨现实中电子商务系统数据库的结构与组件。我们首先介绍一种电子商务系统所需的包括8个环节的完整价值链，和链中每个环节的相关数据。接着我们讨论典型电子商务数据库系统的逻辑结构。最后，我们描述电子商务交易过程系统的详细设计，以及对电子商务数据库系统几个设计细节的评论，比如说主键、外键、外部连接、弱实体的使用和模型分隔。理解电子商务数据库系统有助于高效的数据库设计开发和电子商务系统维护。

1 介绍

在本文中，我们介绍现实世界电子商务系统的结构及组件。一般来说，可以用下面两种方式之一建立电子商务系统。第一种是定制方式，使用套装工具，如IBM WebSphere 商务套装[Shur99]。举例来说，商务套装提供建立电子购物中心的基础工具，包括目录模版，注册，购物车，订购和付款处理，以及常规数据库。第二种是由使用系统的公司的专家们在机构内部由下至上独立开发。这种情况下，开发者用整合(mix-and-match)工具手工建立电子购物中心。此外，支持电子商务系统工作的数据库也需要手工开发。

无论是用定制方式或是由下至上方式，理解电子商务数据库系统都能帮助数据库设计人员更有效的开发和维护系统。本文的基础是我们在几个不同领域的实际经验，如在线购物中心，在线配送服务和工程申请。

电子商务环境下的数据库设计主要难点有 [BD98, CFP99, KM00,LS98, SL99]：

-处理多媒体和半结构化数据(没有严格结构及类型定义的数据)

-把纸上的商品目录变成统一标准使格式并整理数据

-支持数据库级用户接口(例如对浏览器，商店布局，超链接)

-模型演变(如，两个目录合并，商品类别，售完商品，新商品)

-数据演变(如，改变规格说明和描述，命名，价格)

-处理元数据(meta data)

-为个性化服务获取数据，如浏览器内的相关数据。

在第二部分中，我们关注在电子商务系统所需的价值链，以及链中每个环节的相关数据。第三部分开始讨论电子商务数据库系统的逻辑构件。接着我们介绍电子商务交易进程系统的详细数据库模型，讨论几种电子商务系统特有的数据库设计。 第四部分以电子商务数据库系统的作用和将来发展作为本文的结束。

2 电子商务价值链和数据需求

电子商务价值链是表现在线购物者与电子商务系统交互性的一系列业务过程。价值链帮助我们理解电子商务系统的业务过程，确定建立可操作数据库系统的数据要求。 Treese和Stewart展示了一个四步价值链，由“ 吸引”、“交互”、“操作”和“反应”组成。“吸引”指获取并保持顾客的兴趣。“交互”把兴趣变为定单。“操作”管理定单；作出“反应”服务顾客。四环价值链可以被视为最小的电子商务系统模型。

在本文中，我们描述一种更详细的八环节价值链。新的价值链环节如“个性化”，通常情况下由分离的附加产品履行。图1是8业务过程的完整电子商务价值链，它们的目标与数据要求。我们称价值链的每一环节为业务过程，在其中从每一环节本身来说是重要而且复杂的。为达到不同的目的，每一业务过程都包含一系列在线购物者与电子商务系统的交互。基于电子商务系统的基本业务模型，每一业务过程都有不同的数据要求。举例来说，对于B-to-B，B-to-C，拍卖，或是兑换，电子商务系统的商品、服务与用户都不相同。数据库设计人员必需完全理解每一种业务过程，确定支持每一过程的数据需求。

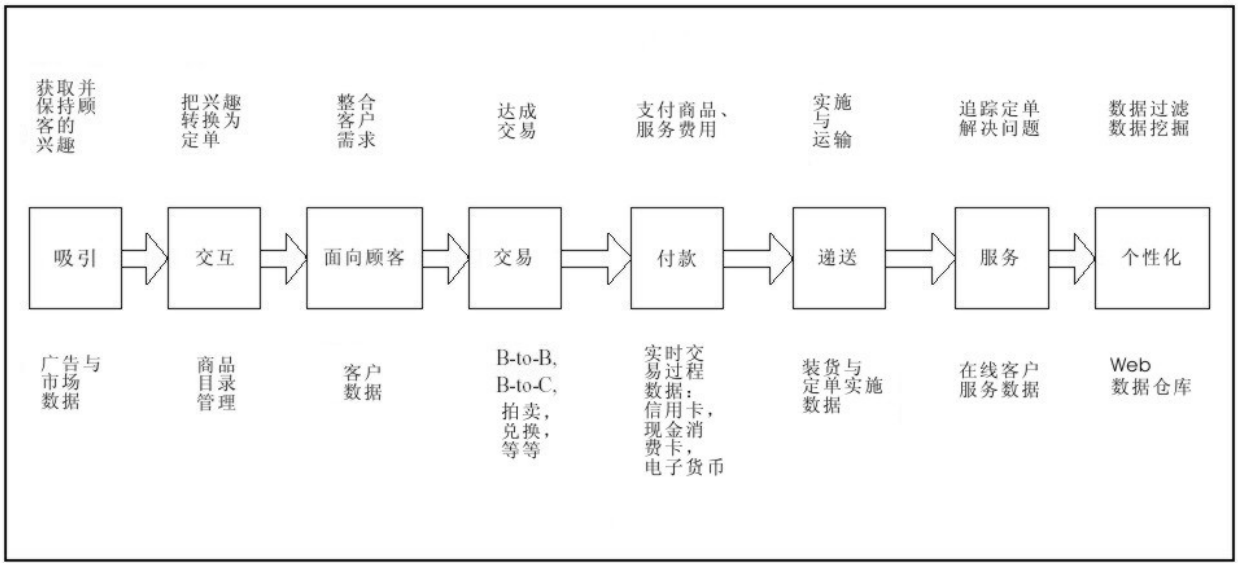


图1：8环节电子商务价值链

3 电子商务系统的数据库模型

3.1 高层逻辑构件

现实世界电子商务系统数据库模型相当复杂。图2是包图，展示了典型电子商务数据库的逻辑构件。图2使用在UML[BRJ99]中使用的包表示方法。在UML中包的概念是一组相关建模元素。图2中，每一个包含有一张或更多的数据表。

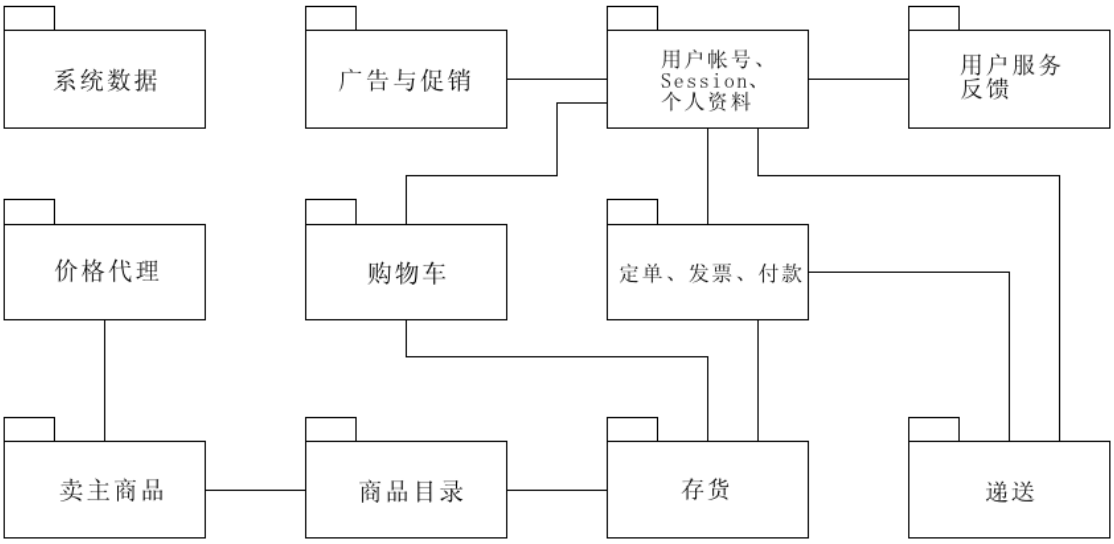


图2：电子商务系统逻辑构件包图

“帐户、Session、个人资料”包的记录有：登陆身份、密码、国家人口统计数据(可能是身份证号)、信用卡、用户个人资料和总计数据(如总购买次数、付款总额、总申报量)。这个包处理顾客数据，如特殊项目显示，总计数据显示，数据描述次序，个性化数据(如用户采购行为、统计数据、报表数据)。扩展这个包可以包含点击流量分析模块。UASP(*User Accounts, Sessions, and Profiles*/帐户、Session、个人资料)包也针对不同种类的用户，如个人顾客、小零售用户、用户群组、采购人、出售人、以及不同种类的会员，就看系统支持哪些业务模型。

“广告、促销”包括与广告、促销、优惠券相关的数据表。这个包追踪与sessions相关的促销和在sessions中播放的广告。

“价格代理”包括每一次商品的竞争信息。它允许系统搜索其他销售同一商品的在线网站，并把网站名称按价格排列。因此，如果在同一领域内没有竞争网站，这个包就很可能不存在。

“购物车”模拟购物车与车内商品项。注意放在购物车内的物品可能被购买，也可能不被购买。因此，各项的不同状态必须明了。

“定单、发票与付款”*Order, Invoice, and Payment (OIP)*明确从购物车转化而来的实际定单。它明确定购清单、折扣、售价、佣金、税、作废、以及其它定单状态更改。这个包也包括发票历史、付款历史、信用卡过程模块。

“递送”包括运输地址、运输方式和收费。

“存货”、“商品目录”、“卖主商品”三个包关联紧密。“卖主商品”包含卖主数据与商品。“商品目录”是定型模型，它把不同的卖主商品整合到同一个领域中。由于一个卖主可能用不同的术语和格式定义相同的商品，“商品目录”在“卖主商品”与“存货”之前建立缓存区，保护实际的业务模型不受“卖主商品”改变的影响。“商品目录”包同时也需要动态目录管理模块来适应事先没有考虑到的新卖主和新商品。因此，程序在实际执行中比较“商品目录”与存货。以我们的经验来说，开发合适的目录模型是最耗时也最复杂的工作。

“系统数据”包括不同的系统参数，如服务器设置和数据存贮控制参数。

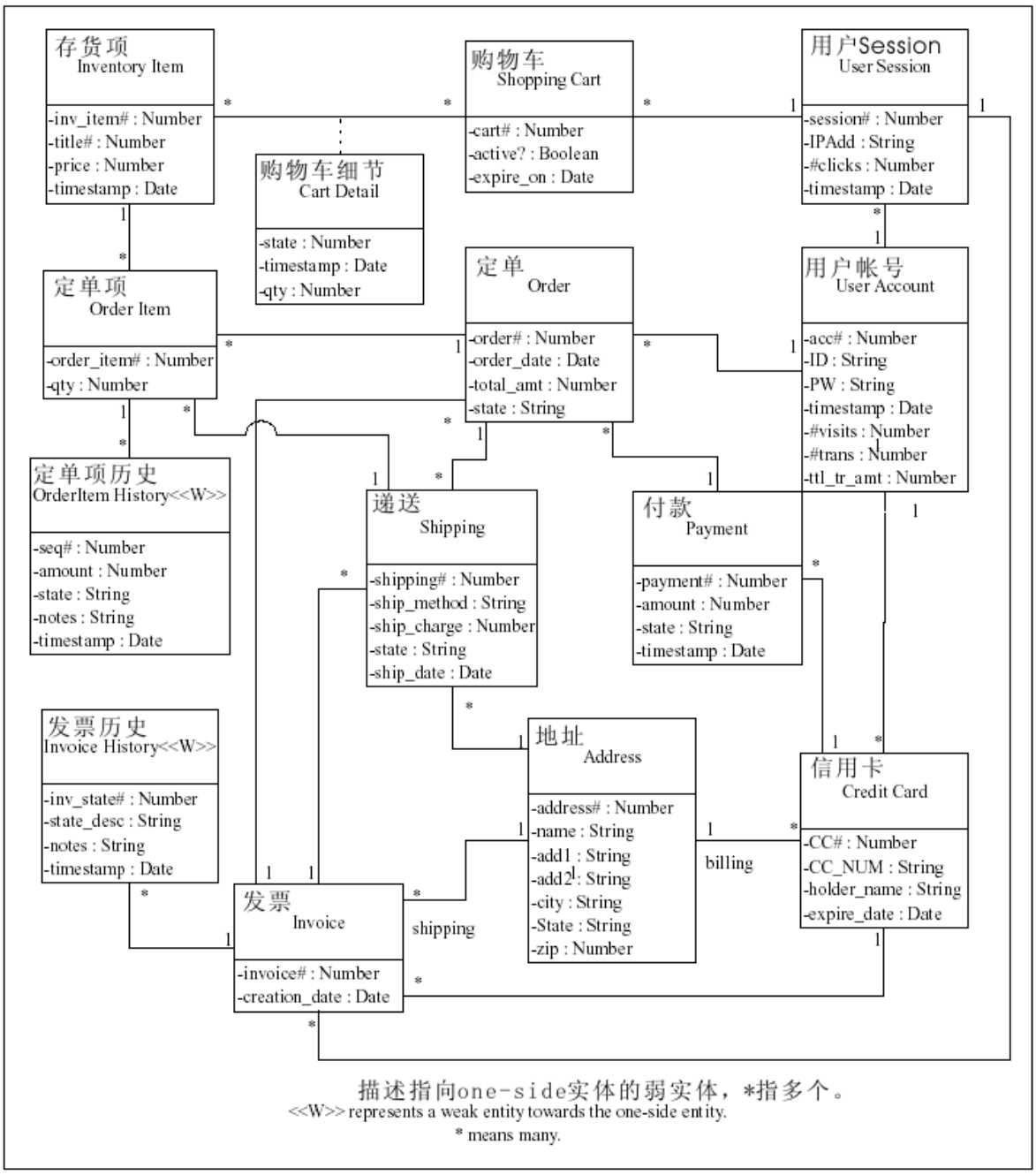
3.2 电子商务处理过程系统模型(ECTP)

在这一部分，我们描述卖出存货的电子商务处理过程系统(ECTH)的数据库模型。图3是ECTP模型，包含部分UASP(*User Accounts, Sessions, and Profiles*/帐户、Session、个人资料)、“定单-发票-付款”、“购物车”、“库存”和“递送”。尽管现实情况比图3所示的更加复杂，这一模型清楚解释了电子商务环境中的几个有趣设计思路。网站不同的业务规则和支持模块造成不同的模型细节。图3使用了UML表示法。为了建立图3中的关系模型，我们需把一对一的主键添加为一对多的外键。

合适地选择主键十分重要。对于绝大部分数据表来说，我们使用系统形成的主键(代理主键)来避免关键数据改变。有含义的主键对数据形成了依赖。因此，改变那些数据将导致主键的改变。图3中所有的表都使用代理主键。

注意为了不丢失任何个人过程和报表生成产生的数据，有些没有声明的项是另一个表的主键，在此作为外键出现。比如说，数据表ORDER_ITEM(定单项)使用INVENTORY_ITEM(存货项)表的INV_ITEM#数据项保持关联结构。但是我们并不声明外键属性。如果在INVENTORY_ITEM表国INV_ITEM#项被删除，整个联系将被破坏。这时如果我们联接ORDER_ITEM和INVENTORY_ITEM两个数据表，就变成了外联接，而不是自然连接。这样联接后的定单项中就包含了已被删除的商品。

在图3中，定单(Order)与定单项(Order Item)之间的联系就是一对多。在典型的数据模型状态中，“定单项”是“定单”的弱实体。因此，“定单项”的主键是Order#项的结合，一个序列的定单项具有同一“定单”序号。无论如何，在这张图中，我们建立Order_Item#作为“定单项(Order Item)”的主键。Order#在“定单项”中仅作为外键，而不是主键。这样电子商务系统就可以轻易、灵活地处理单独的定单项，而无需依靠“定单”表。如果你要统计定单项在定单中出现了多少次，就算定单表本身被删除，还是可以使用这一方法做到。



另一方面，“定单项历史”在建模中被设定成为“定单项”的弱实体。在“定单项历史”中，每一定单项的历史记录都可分离。每一可追踪的实体的状态都有可能变化，必需有一个弱实体记录其状态变化的历史轨迹。它使得最底层的各种顾客支持更加简单。图3中的可追踪实体包括“用户帐号”、“定单”、“定单项”、“发票”、“递送”和“付款”。我们在图3中仅仅把“定单项历史”和“发票历史”作为弱实体。“发票历史”是“发票”的弱实体，保存“发票”的状态变化。

“发票”与“用户Session”之间的关联允许系统计算购买率——由Session转变为发票的数目，然后转为实际定单。“定单项”与“递送”之间的关联允许系统把独立的定单项送往不同的运输地址。用户帐号有多个状态属性。通常他们被放在单独的表中。为节省空间，图中我们把它放到“用户帐号”表中。

因为某些原因，典型的电子商务系统现实世界数据库模型分成多个数据库，在不同的数据库中运行。比如说，静态模型如商品目录可以与更具动态的定单过程分开。这种多样化构造方式需要连接各部分，当过程需要时，还要求各模型之间的数据传送。因此，这种方式稍微减慢了运行速度。不管怎么说，它有许多优点，如更稳定、易维护、负载平衡而且安全。

4 结论

本文讨论了8环节的电子商务价值链、电子商务数据库逻辑构件、电子商务处理过程系统模型。绝大部分的现实世界电子商务数据库模型与本文中所提到的结构相似。以往的经验告诉我们，电子商务工具能提高开发速度，但是缺乏某些功能，如偏好订购、拖欠交货、回复以及Email通知用户。所以，无论采用哪种方式，理解电子商务数据库系统的结构都有助于开发和维护系统。

从数据库设计观点来看，一个有趣的研究问题是：什么样的数据库结构能最有效的支持定制和独立开发。打个比方，建立网上仓库我们需要捕捉哪些数据？怎么获得？接下来，我们怎样实现用户与系统的沟通？独立开发过程要求明确的点击数据和用户行为范本来源。电子商务环境中的典型的OLAP(online analytical process 在线分析过程)与数据仓库设计可以阅读以下资料。

参考文献

[BRJ99] Booch, G, Rumbaugh, J., and Jacobson, I, UML User’s Guide, Addison Wesley, 1999.

[BD98] Buchner,A. and Mulvenna, M, Discovering Internet Market Intelligence through Online Analytical Web Usage Mining, SIGMOD Record, Vol. 27, No.4, December 1998, pp. 54-61.

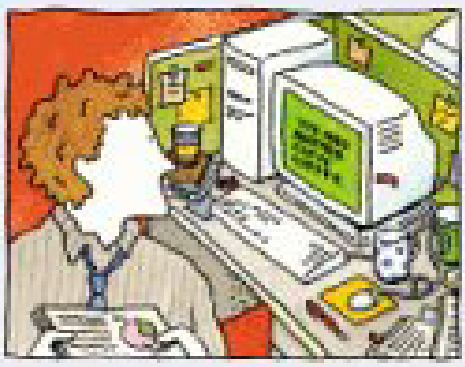
[CFP99] Ceri, S., Fraternalim P., and Paraboschi, S, Design Principles for Data-Intensive Web Sites," SIGMOD Record, Vol. 28, No.1, March 1999, pp. 84-89.

Peopleware

2nd Edition | Productive Projects and Teams

Tom DeMarco & Timothy Lister

Translated by Tomoo Matsubara & Tsuneo Yamaura



ピープルウエア

第2版

ヤル気こそプロジェクト成功の鍵

トム・デマルコ／ティモシー・リスター

松原友夫／山浦恒央訳

《人件》中译本即将发行

日経BP社

人的问题：关于《人件》

Jacques Lebrun 著

查看评论

Il nome della rosa （玫瑰之名）

名称，神秘主义者们认为，与事物之间具有内在而非偶然的联系。在“现实”中很容易对此做出归谬——叫 Mark 的不一定都好战，叫 Emma 的不一定都爱提建议。对于书籍而言，就更其如此：奥维德和卡夫卡的名著除了类似的书名（Metamorphoses 和 Die Verwandlung 在中文里碰巧都是“变形记”）之外，之间并没有太多可以分享。

但神秘主义者仍可以将一本书引为例证：《Peopleware》（中译《人件》）的名称似乎就孕育了该书其他的部分，像是沙粒里蕴藏着整个世界。当然，我指的不单单是标题里同时包含了“人”和“软件”二者这回事，虽然也许据此我们就能推断出书中的核心论断：软件开发中，归根结蒂还是 “人的因素”占主导地位，因此，项目成败的成败、企业的存亡也更多的不在于技术，而在于“人”，或者说在于对于人的因素的重视程度、组织方式等等。换言之，本书的主旨固然可以从这个标题中“分析地” （取这个词在德国人那里的意思）得出，但我指的，更多的是该标题中包含的一种矛盾，而在我看来正是这一矛盾有效地推动、同时又内在地颠覆了该书的论述。

在我书房的一角，《Peopleware》尴尬地和其他原版“技术书籍”分享着位置。尴尬，首先就从书名中显露出来，这个自造的表达方式置身众多的“Patterns”“Models”和“Components”之间显得可疑、孤单、缺乏一目了然的“技术特征”。People 和 software 之间这个狡诈而轻快的拼搭，带有明显的“头脑风暴（brainstorm）”的痕迹（你还能在书中找到其他的例子，比如“teamcide（队杀？）”）。通常来说，类似的表达方式应该由一种人（“市场人员”）生产，另一种人（“管理阶层”）消费，与“技术”了无关系。——且慢，再仔细看看副标题：“Productive Projects and Teams”——这似乎又与“我们”有些牵涉？无论如何，“项目”和“生产率”在软件业中一直被认为是与“技术”有关的，类似的实践在词汇光谱中更偏近于“技术”一端而不是“行政”或广义上的“管理”。

叶饰中的人脸

而如果你打开本书，从随便的地方开始读上几页，你的反应也许就能证明你是哪一种人。可能对任何人而言，这本书读起来简直都太“容易”了，轻滑绵软，入口即化。但是就像止咳糖浆对一些人是良药，对大多数人就是

受罪一样，我预料不是所有人都能顺利地接收以这种方式传递的这样的信息：图表都是手绘的；不时穿插着《读者文摘》风格的小故事；显而易见的煽动性表述和故作惊人之笔；最后，是技术细节的缺乏和常识错误（e.g. 第186 页上提到了“…building PERL applets at Yahoo”）。这是一份加料过火版的呆伯特（Dilbert）读物（不幸地缺乏Dilbert 的冷隽和细微）。如果你（像我初读时那样）相信这是一本“软件工程的必读书”，尤其是指望在其中找到具有可操作性的方法，那也许你会在读后大呼上当。

这种风格（并不像我曾以为的那样）能用作者们的技术背景加以解释。两个作者中，虽然 Lister 简历上没有留下任何软件开发的痕迹，但 DeMarco 却确实曾任职于贝尔实验室，甚至还写过一本关于结构化软件分析的书（Structured Analysis and System Specification，1979）。据说，从前（面向对象被引入以前）脍炙人口的“数据流图（DFD）”就是那本书的发明。因此上面谈到的甜腻风格未必是某种缺憾（知识或能力上的，比如说）的结果，而是作者们有意为之：这是他们处心积虑的对推销员风格的模仿，而目标的消费群，像或许你已经猜到的那样，正是所谓的“管理阶层”。我读到的一篇（显然是有倾向性的）书评阴险地说道“我强烈推荐这本书给每一个人，从低级工程师到 CEO”,另一个有害的推荐说：“我强烈推荐你买一份给你或你的老板，如果你是一个老板，那么为你部门的每一个人买一份，并给自己买一份”，似乎这说的是什么营养品或特效药。在此我（谦卑地）提出背道而驰的推荐：如果你的老板还没有读过，你也一定不要读，除非你碰巧想换工作（原因内详）。

然而上面的区分仍然是过于幼稚的，甚至，本书自身就抵制这种区分。书中正确的指出，软件开发的管理人员，往往原本也是“技术”人员。因此他们似乎倾向于技术性地，而非“人性地”考虑问题：无论是一个项目，一个团队，还是一个企业。

以上论断，应该说是本书的主要成就，它描绘了（尽管是漫画式的，尽管面目模糊）一个角色，介乎“管理阶层”与“技术人员”、“行政官僚”与“狂人程序员”之间，正在“蜕变”但又旧习难改，非此非彼而对二者又都若即若离的一个人群。它试图捕捉这个飘忽的影子，给这个此前无人谈及的角色定位，并签发一张（在我看来是可疑的）诊断证明。如果手上没有原书，你可以去亚马逊网站看看该书的封面：白底上散乱的一幅绿色水粉画，大量的模糊叶饰中，隐约显出一张人脸。我的（不失独断的）假设是：这个面目难辨的人，就是上面提到的“软件开发管理人员”的肖像。书中的不少观点，原本的读者群可能是最上层的决策者，可是由于提出时的语气，每每不由自主地降落在我们谈到的这个含混的目的地。

但也许对于年轻的软件业而言，这张脸后面隐藏着的形象，还是一个过于不可测度的来客，很少有人能够在它面前能够感到自在，要对它开口说话，更是一项专门的学问。就个人而言，我感到本书的作者们就没有把握好语调：他们似乎努力显得生动、不教条，但结果却可悲地滑向了另一种风格，在德语（以及后来的英语）里，人们称之为 kitsch，这指的是一种由于故作高雅而带来的俗气——在讨论“办公空间”的部分援引《建筑的永恒之道》的一些段落加重了这种效果。然而如果仅仅针对各种觊觎或已陷入软件开发行业的“商务人士”，本书却确实提供了一个近乎理想的起点。上面提到的种种特性保证了它对匆忙而好奇的眼睛具有难得的捕捉作用。

软件企业的悖论

在我看来，本书（源自书名）的张力由两组近乎正交的对立构成。首先是上面提到的“软件开发管理人员”的身份矛盾。其次，则是“软件企业”自身隐含的悖论。

根据本书，软件企业似乎特别倾向于错误地理解自身。比如说（似乎是出于软件的组件化结构），管理者总认为开发者都是匀质的、可替换的；再比如，管理者也乐于认为开发项目的组织更多的是一个技术层面的问题，而与“人事”方面无关。以上两种认识，都似乎认为软件业脱离了各种传统企业的运作模式，只需要采用机械的，或者至少是技术的管理思路就可以成功的组织。针对于此，作者们论证道：软件生产，作为一种人类活动，面临主要问题并不是技术问题而是社会学（sociology）问题。软件开发者也并非像机械部件那样可替换的无面目的劳力，而是血肉之躯。我们尤其不应被所谓的“高科技”外表所迷惑，错以为高科技的介入就能使普通的管理模式在软件开发这里失效。因为（据作者们说）各种技术的发明过程确实是属于那种人迹罕至，需要绝世天才的高科技领域，但是软件企业中多数的情况只是在“应用”他人的成果而已。这种应用，本身与其他人类实践一般无二。这样，似乎软件企业也将需要与其他行业同样的管理科学和组织技巧，需要与大量“人”的问题打交道。那么本书的特殊处又如何体现呢？为什么不在完成以上论证后结束，并推荐一部《管理学引论》呢？

这里作者们又引入了另外的观点：软件企业终归有其特质，尤其是软件开发过程本身有其特质。管理者们往往又容易陷入轻信的圈套。他们（错误地）相信，开发者身上压力越大越好，加班越多越好，开发者都像他们自己一样不在乎电话打扰，开发者只配与其他员工（比如营销、客服人员）一样坐在开放的、由格子分割的办公空间中工作，开发者自身并不在乎产品质量高低等等。而（作者们提出的）事实是：压力和加班只能破坏项目进度；由于软件开发工作的特质，应该尽量保证开发人员不受打扰并拥有足够空间；开发者最关注产品的质量，甚至视其为荣誉攸关的大事。

如果以上对软件开发人员的报告全部属实，那么我们似乎又要重新建立关于软件开发的一种神话，软件业似乎又与所有其他行业重新天差地别、判若云泥。但是，就像摄影中正片和负片传达的信息量几乎相同，把一种完全错误的实践颠倒过来也不一定就是正确的。比如我就看不出，作者们鼓吹的软件业的各项“独有特征”，究竟有哪一种不能适用于其它行业，比如说，建筑师事务所。如果这个论点正确的话，也许可以把同一文本按照特定的词汇表作全局替换，然后推出一本新书《Peopletecture》？

也许一直以来，软件业对自身的理解仍停留在一个幼稚的阶段。究竟在哪些区域，哪些实践中，软件业有别于“传统行业”，而在哪些方面我们仍应保持对“人类活动”的忠诚？这种同一/差异间的对立，加上上文提到的“谁在管理”的困惑，形成了本书中推动而又吞噬着主题的隐形漩涡。作为牺牲者，作者们轻佻的风格没有允许他们看到位于漩涡中心的（可怖而壮丽的）图景。

Menschliches, allzumenschliches（人性的，太人性的）

恐怕我已经给人一种“本书的敌人”的印象。作为挽回和补偿，也许我应该表明，我赞同书中绝大部分具体结论。我身上幸存的开发者的良知告诉我，作者们关于加班、电话、空间和离职等问题的想法都是对的。

我尤其喜欢其中对开发者“沉思（immersion）”状态的强调：任何程序员想要进入富有“生产率”的状态，都有一个缓慢的过程（通常在15分钟以上）。进入该状态之后，他/她会感到工作特别上手，劳动效率极高。因此作为领导，出于利润最大化的考虑，管理者也应该尽量促进、而不是破坏这一状态。尽量少打扰开发者；尽量将同一项目组安排在封闭的、有足够空间的办公室里；尽量避免电话对这样的办公室的打扰（采用电子邮件，甚至语音信箱）；尽量职责明确（这一点可能是我加的，因为同一人有不同的职责意味着他要在多种沉思状态下切换）。虽然这里的“沉思”概念听上去类似于梦游，但这些并不意味着神话化软件开发过程，因为我们当然可以对建筑师事务所说同样的话。而如果我们忽视这种创造性劳动者（或者，用一个著名的矛盾修辞，“脑力劳动者”？）所必需的沉思，那么8小时的劳动时间往往会被侵蚀得所剩无几（考虑一下，如果你刚要睡着就被人唤醒，连续几次，一晚的睡眠就毁了）。这样就将带来两个问题：管理者在办公空间等细节上千方百计地节省开支，却忽略了他们所支付的最大开支（劳动报酬）的回报率；比这更坏的是，企业中流传着“在9点到5点之间什么都开不了”的说法，而这将导致加班—离职的恶性循环。

类似的精彩论断还有很多，如果你是一个管理者，尤其是，如果你本人并不熟悉软件开发过程，那么确实有必要认真体味这些珍贵的思考。我想，核心的意思是要打消一个错觉：如果你的下属喜欢他们的工作，那并不表示你的工作出了问题或是企业的效率降低了；恰恰相反，这往往是企业进入良心循环的标志。而如果为了节省（多少无知的举动以此名义而行）或其他荒谬的原因（书里的例子是“这显得不专业”）而给员工造成干扰、妨害、限制以至挫伤，那么最终的结果只能是士气低落、效率低下、离职率高，简言之：灾难。

是的，人性的，太人性的。所有的建议，包括提出的语气，都包含了这种“人性的”考虑。无怪乎一些书评称本书为“人道主义的（humanism）”。在这里人性本善，麻烦只是因为庸人自扰或是生性吝啬的经理们造成的（你见过Dilbert的上司头上的角吧？）劳动和资本的利益原本是一致的，只要实行黄老之道，员工的积极性就将被调动，一切都会进入良性循环。

书中对于团队精神（teamwork）和质量的论述尤其体现了这种人道的思想。根据作者高见，提高生产率的最重要方式就是形成有效的团队。但是，团队的形成，包括高质量的达到，都无需也无法通过明显的行政干预实现。比如，四处张贴“我们要的就是高质量”的口号无助于真正达到这一目标。据说开发者们心目中原本就追求团队合作，追求高质量，所以与其用外力加压，不如潜移默化地促使团队“成长”起来，而如果工期合理，开发者的骄傲感决不会允许任何低于自身质量标准（往往比经理们的要高）的产品出炉。

类似的乐园化论述不绝于耳。如果你认为这难以置信，作者们会拿出一系列“硬数据”说话：他们一直经营着一个面向实际企业的编码演习（“coding war game”）测试。上述观点，均已被这些测试证实。作者们先前的另一文章（Programmer Performance and the Effects of the Workspace，1985）系统地分析了这些测试，并得出了更严肃的结论。

虽然我愿意同情所有这样的论述，但是，正像上文提到的摄影术比喻一样，正片并不比负片包含更多的信息。“恶魔式”的老板的失败，也未必证明了“天使式”的老板就一定成功。简单地颠倒前者的实践，带来的多半还是灾难。比如说，较之于对个人感受和安逸的强调，团队的形成更依赖于好的交流，无私的互助以及核心成员的性格因素。单纯注重“人性”，未必就能形成团队，就像单单是领导的吝啬颟顸也不一定就会毁了团队一样。

我对类似言论的反对（或者称为“疑问”更好）也许更多地是“形而上学的”而不是“理智上”的。“人”，这个“人道主义者们”诉诸的终极对象，终归仍是一个被寄予了太多感情因素的抽象物。我们听到它被作者们用动情而不乏得意的声调多次提及，但是它真的像他们设想的那样，是软件开发的核心吗？人不也受到它卷入的某个特定的进程定义吗（就像恋人们被爱情定义，团队成员被团队定义）？就“核心”而言，“人”不应更谦卑地让位于此吗？除非我们满足于把一次思考等同于一项“商业操作”（那里，片面、粗疏甚至轻微的自我陶醉都是必要的策略），否则仅仅一个“人”字并不意味着最终的答案。

我的感觉是，与《人月神话》相比，本书在很多细节上有所不及，未能给出有效的出路（比如，《人月神话》中的“外科手术式队伍”就比本书中民主乐园式的团队更有可操作性），这可能仍然与本书摇摆不定的读者定位有关：再说一遍，这是为那些钟情于“头脑风暴”甚于具体操作的人准备的书。它致力于指认某些幻觉、扭转某些态度。如果你恰好处于作者们设想的位置、并也走在相应的方向上，作为一个殷勤的侍者，它会为你打开一扇特定的门。但似乎不应指望，它能领你一路回家。

参考资料

- The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition) by Frederick P. Brooks, 1995 Addison-Wesley
- Peopleware: Productive Projects and Teams, 2nd Edition, by Tom DeMarco and Timothy Lister, 1999, Dorset House, 245pp

本文首发于《程序员》杂志2002年第12期

Tom DeMarco
Timothy Lister
Wien wartet auf Dich!
Peopleware
in deutscher Ausgabe
2. Auflage mit acht neuen Kapiteln
Der Faktor Mensch
im DV-Management
《人件》中译本即将发行
HANSER

开发人员是人吗？

Dr. Jakob Nielsen 著，Windy J 译

查看评论

由于正在为一个开发人员站点写这篇文章，我想我将挑战这个古老的问题：开发人员是不是人？我确实希望能从广大读者的血样中来个 DNA 测试，证明他们属于人类这个物种，但那并不是我关心的，相反，这里的问题是开发人员和其他人类是不是一样，或者，到底差了多少。

是的：开发人员是人。

在经典作品《人件》中，Tom DeMarco 和 Tim Lister 介绍了对程序员生产率和他们工作环境特征之间关系的研究，结论很清楚：人性因素对程序员们影响很大。

DeMarco 和 Lister 比较了程序员们在编程生产率上不同的表现级别，生产率最高的程序员们在拥有安静的工作空间方面得分很高，在拥有电话转移能力上得分更高，并在他们被打断的频率上得分相当低。相反，生产率最低的程序员们在安静工作空间方面得分比较低，在电话转移能力方面得分非常低，并在被打断频率方面得分很高。

很明显，（工作）被打断是程序员们生产率低的最主要原因。为什么？问题不是需要处理打断所需要的时间，而是被打断以后回到编程问题所需要的时间。任何人，不管做什么，在被打断以后回到原来的工作都会面对再适应时间的问题。当你正在阅读一篇期刊文章时，如果转去寻找某个问题的答案，然后阅读下一段，这样将比你连续阅读需要花更多的时间。

程序员们处境堪忧。他们需要在大脑里建立复杂的模型，对编程问题，对不同变量的状态，等等。这就是为什么大多数人不能成为好程序员的原因，但就算是最好的程序员在被打断时也会陷入困境。嘭的一声，大脑里刚才小心建立的模型就塌了下来，而回到刚才高效编程的状态很容易就要花掉 15 分钟，因此，一个两分钟的电话就会毁掉大约 17 分钟的生产率。

许多其他著名的发现证明，人性因素对开发人员非常重要：

Fred Brooks 的著名论断说，“给一个已经延期的项目增加人手只会使它延期得更厉害。”（《人月神话》）

好的程序员和差的程序员之间显著的差别：

通常在生产率方面有 10 倍或 20 倍的差距（同样，生产率更高的程序员通常提交最好的代码。）

在其他领域，人类的效率差别小得多：我过去常能在 2 倍世界记录时间内跑完 100 米。

要改进公司的软件工程水平，请遵循以下七个步骤：

- 只雇用最好的程序员，哪怕他们更昂贵。一个好的程序员胜过 10 个差的程序员。
- 为每个程序员准备一间关上门的私人办公室（不是小隔间）。
- 为电话提供秘书接听服务（或者至少提供一个转换系统允许将电话转换成语言邮件而不直接响铃）。
- 禁止邮件（或更实际一点，建立一种文化，在繁重编程的时候允许一两个小时不回复邮件）。
- 给每个程序员一台 21 英寸甚至更好的显示器（大屏幕可以更好地纵览复杂数据）
- 200dpi 的显示器一上市，就为程序员们配备（任何时候花在阅读屏幕上的时间都可以自动增加 20%的生产率）。
- 把所有程序员送去参加指法练习班。

当然，好的软件工程方法学一样很重要，但人性的角度经常被人们忽视。

不：开发人员不是人。

假设你属于这个开发人员站点预期访问者，作为我的一个未来读者，你可能属于那 1%最了解计算机的人之一。你很聪明，有着非常优秀的抽象推理技巧，你甚至能写一个用在搜索引擎上的布尔查询。

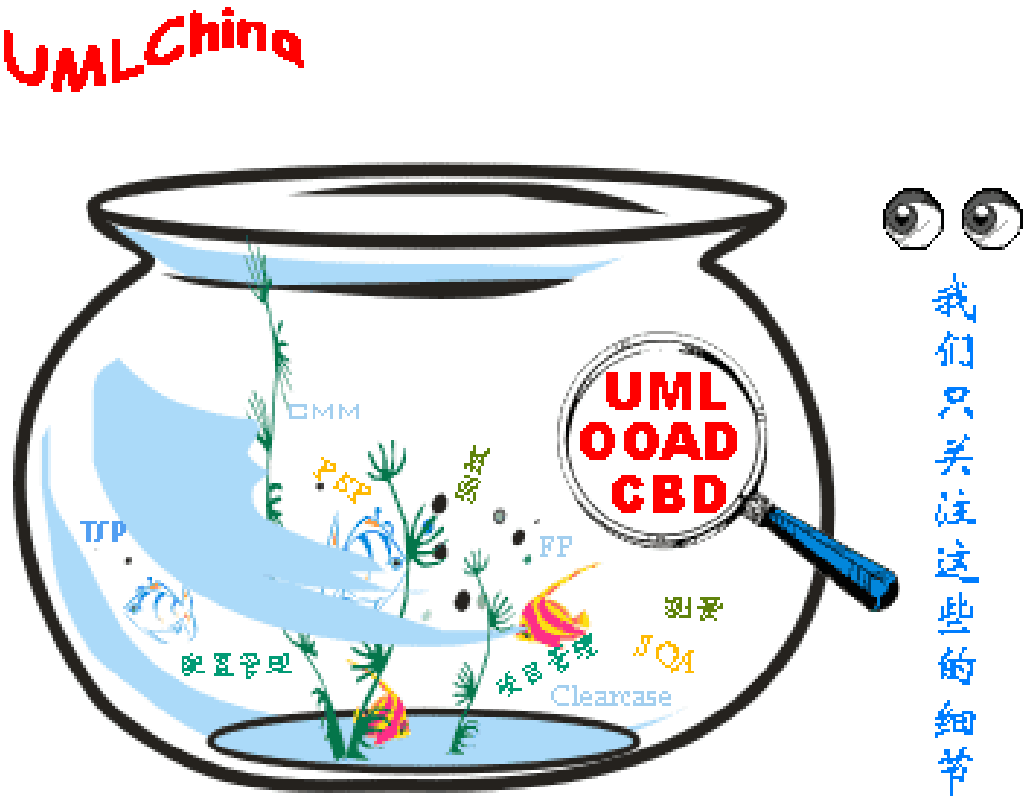
在恭喜你自己得到这么高的评价之前，我不得不指出，这些因素在评估你作为一个普通用户的经验时，恰恰是绝对的不合格。从定义上来说，绝大多数人属于那 99%计算机知识比你少的人。他们也非常聪明，但他们通常有着跟计算机不同的思维方式。而且，如果你给他们提供一个有着太多高级特征的搜索引擎，他们将找不到想要的东西。

不好意思，对搜索引擎说了这么多，但我非常讨厌的是这种倾向：用一些只有两种人才能正确使用的特征对搜索界面进行过度精化，而这两种人就是搜索引擎工程师和经过四年教育的图书管理员。象 Google 那样的简单搜索框就好多了，然后适当提供经过选择的高级特征，不要让用户输入一个有着成百上千个空格的页面，那些空格之间的相互影响也许需要大学教育程度才能理解。

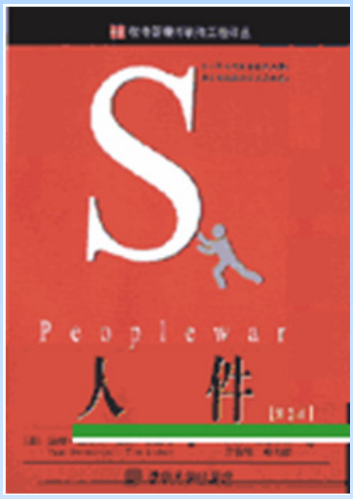
幸运的是，也有一些方法允许开发人员绕过他们自己的才华并象普通人一样来感受他们的产品。在前一篇文章里我讨论了可用性生命周期，也建议了许多不同的阶段去主动采取措施收集用户数据。

现在，让我介绍一种证明很有用的简单方法：用户测试，找来普通客户，让他们使用软件，当他们觉得有问题时，应有的反应不能是“这个用户一定很笨”，而应该是“开发人员犯了错误”。那没问题，所有经验证明，没有人能第一次就设计出完美的用户界面，所以真正的错误是，因为你个人认为容易处理就不去改正可用性问题。

总之：你不是用户。



《人件》：老板，别把开发人员当成牲口!!!



Tom Demarco，Tim Lister

翻译：UMLChina 方春旭 叶向群

http://www.peoplewarecn.com

人件中文版网站

Frederick P. Brooks, Jr--《人月神话》第 19 章

近年来，软件工程领域的一个重大贡献是 DeMarco 和 Lister 在 1987 年出版的《人件》，我衷心地向我的读者推荐这本书。

Edward Yourdon

《人件》在 1987 年出版后，立即成为最畅销的作品。当人件第一版出版时，我写了一份评论，“我强烈推荐你买一份人件给你或你的老板，如果你是一个老板，那么为你部门的每一个人买一份，并给自己买一份”。这建议在 12 年后的第二版依然有效，并且更加热烈。

Mark A. Herschberg

我只学了办公环境的章节，就辞掉了原来的工作！

Joel Spolsky

我想，微软成功的原因之一就是公司里的所有经理都读过《人件》

Steve McConnell

由于本书第 1 版的赫赫声名，新版的《人件》是我不用看就会决定购买的少数几本书之一。

中文译本即将发行！

办公室空间，下一场革命

Diane Pettus 著，Windy J 译

查看评论

现在，许多三角区域的公司打算评估他们的办公空间需要，并决定搬迁他们的办公室，或者对他们现在的工作空间作一些改变。促使他们作决定的因素将会是营业间接成本，生产力方面的关注，和/或员工的方便性。

但是在这个抢座位的游戏中，那些作出明智选择（这些选择基于如何使员工工作最好的持续研究和这方面的知识）的企业主人数，将远远少于那些把临时解决方案拼凑起来的企业主人数，而他们的那些临时方案极少从雇员那里征求意见，并把布置的费用作为底线。

在他们开创性的作品《人件》中，Tom DeMarco 和 Timothy Lister，说这些“拼凑者们”将会吃惊地看到实际的成本：将立即导致生产力的损失和雇员的挫折感。

《人件》详细说明了精心设计的办公室空间将带来的成本节约，一是专业的气氛将使工作完成得更快更有效率，而且员工在这样的空间将感到自豪和被尊重。作者们主张企业必须对这些严重损害生产力的因素引起重视：拥挤，不合理的工作空间只会导致干扰，噪声，私人空间和会议空间的缺乏。

DeMarco 和 Lister 支持这样的观点，那就是企业主们可以通过合理规划的办公室环境，对他们企业的人月计算公式——期望每个员工在 30 天的工作周期中完成的工作——产生极大的影响。

设计能用来工作的空间

一个仔细考虑了办公室空间设计的真实例子是 Interface Technologies, Inc. (ITI)，一个定制软件开发公司，去年十二月刚刚搬到罗利（Raleigh）Six Forks 大道 Forum I 大厦一个大概 11000 平方英尺的单位。他们的总裁 Kelly Campbell 说他搬迁的目标是要建立一个能象加满了润滑油的机器一样运转的地方，一个因为尊重开发人员的日常工作模式，因为认识到客户们对效率的期望，从而不断增加能量的地方。

因为 ITI 的程序员们在项目开发中既要单独工作又要进行团队合作，它的大部分工作空间是大型私人办公室和配备完全的测试实验室的结合。

Campbell 说：“我们的目标是把干扰最小化。我了解到开发人员被打扰后甚至要花 15 分钟才能回到原来的工作。在我们的工作线上，当项目以小时计算成本时，我们需要开发人员能够集中注意力，能够找到一个私人的空间，关上门，关掉电话，思考问题，并且完成他们的工作。我们不希望员工因为在白天什么事也不能做而在这里呆到很晚。”

ITI 还有三个会议室，一个大的开放式多功能房，里面有舒服的家具，可以自由拼凑的桌子和椅子，这个多功能房里头还包括一个厨房，一个休息室，一个器材室，和一个正式的接待室。

“因为在我们的空间里，我们同样需要以团队的方式工作，既有公司内部的团队，也有同客户组织的团队合作，我们需要用来将开发人员和测试人员集合在一起的实验室环境”，Campbell 强调，“我们需要脑力风暴区域，阅读/研究区域，还有工作完成时集会和社交的空间。”

洞穴和草原

在《人件》里，DeMarco 和 Lister 通过报道软件开发人员如何度过他们工作时间的有趣发现进一步补充了 Campbell 的观察结果，根据作者们的说法，开发人员 30％的时间用于独立工作，50％的时间需要跟另一个人合作，还有 20％的时间跟两个或两个以上的人合作。

对上面的分析更进一步，他们基于空间员工密度画出了噪声和干扰程度，不出意料，DeMarco 和 Lister 发现噪声和密度直接成比例。作者们建议，既然 30 平方英尺空间的噪声比 100 平方英尺空间噪声的三倍还大，一个软件公司应该在产品开发时适当调高这方面的风险，因为不同的开发团队协作时产生的嘈杂将不断带来相互干扰和阻挠。

ITI 对此做出的回应是，把它的办公室配置成一个空间规划师所说的那种“洞穴和草原”。它的员工们，根据他们的任务，相对各自隔离（在洞穴）或分布在一个开放空间（草原），在这两种情况下，始终进行着有意义的工作。

所有的 ITI 办公室都“在玻璃上”，意味着每个办公室都有窗户，由两个开发人员共享。这种制度允许开发人员在新项目需要的时候可以容易更换办公室伙伴而不必感到自己在作出不公平的牺牲。四分之三的墙营造出 10×14 平方英尺的实验室空间，非承重隔栏形成草原空间，在办公室创造性地使用玻璃延伸了户外光线的影响，而墙上布满的白板给每个空间增加了多功能性。

遵循下列要素

ITI 的正式空间随着时间扩大到两个办公室套间，通过一个 5300 平方英尺的大厅相连。套间象一个布满独立办公室的迷宫，外面是实验室，以及有着等待列表的两个会议区域，在 11000 平方英尺的可用空间里，ITI 的 20 个员工组不再互相干扰了，而且，Campbell 在这次搬迁时着眼于未来，他从经验中知道，公司的发展可以破坏哪怕是最好的格局布置计划。

为了这个新的办公空间，LLC 理查德商业资产（公司）的房地产代理商 Olivia Moore，和 ITI 一起致力于找到一个地方，让它不但可以适应公司的发展和多功能性，而且对员工们来说，也将是一个方便的地点，于是最终因为它的配备和接近交通干道的模式而选中了 Forum I，好处还有室内停车场，现场的食物供应服务，靠近大型购物中心，这些都是考虑时的加分因素。

“当一个公司租用场地时，它付出的不止是每平方英尺的成本，” Campbell 承认，“在决定把我们的钱花到哪里的时候，我们决定听取我们员工的需要，以及客户的要求。我们成功地给他们传达了两个明确的信息：首先，你们对 ITI 真的很重要，其次，我们有一个专业的办公环境，在这里可以完成很多工作。”

从 Campbell 开始跟地产代理提起办公空间的话题到 ITI 正式搬入 Forum I 花了大约 6 个月的时间，在这段时间里，一个月用来跟建筑师商量布局，另外三个月是总承包人完成工程建设的时间。尽管精心布局和看似复杂，ITI 的办公空间却一点也不浪费，“我们花钱来影响我们公司，我认为这样已经达到跟我们经营公司时同样的专业水准。” Campbell 补充道。

底线

ITI 的新办公环境正在达到预期的效果，开发人员之间的沟通改善了，团队工作更有效率，而且 Campbell 说，当他的客户们看到一个“能工作的地方”时常会啧啧称赞。

ITI 的内部设计没有多余的装饰，但是 McGee 设计室可以通过颜色，织物，家具，艺术品和大大的标志，以及接待室的陈列物品来展现公司的外观和感觉。Campbell 反映说，“当内部设计师开始谈到某个展览设计师，我说，OK，先生们，你们必须给我看一些数字，因为我不知道我们是不是支付得起。让人惊喜的是，每项价格都很合理，他们在预算内工作，以求达到最好的效果。”

对 ITI 来说，一个意外的好处是公司已经在进行额外空间扩展，数年以来，公司一直在网站上为开发者们提供免费教程，ITI 可以在这个外延的空间里主持现场研讨会，例如最近为项目管理研究院当地分部组织的"Water Cooler"活动，就吸引了将近 90 名参与者。

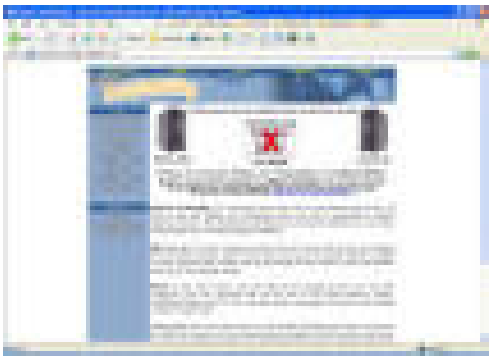
环境，文化，形象

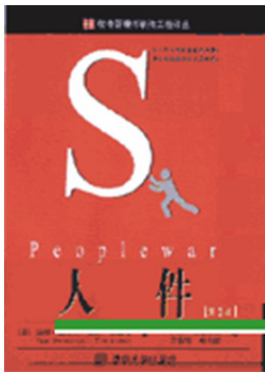
在每个行业中，都有用来定义生产率和员工满意度的可测量工作模式。这些模式已经成为企业文化和它最终外界形象的同义词。用一个不合理的空间设计破坏工作过程是企业每平方英尺办公空间上最大的隐形成本。

作为一个在节约客户开支的同时优先考虑巩固员工努力的企业主，Campbell 提供了一种终极思考方式，“设计办公室空间时，考虑你正在传达的讯息；作决定的时候，不要只询问会计师，研究一下团队生产力，尽量理解公司的工作方式；不要低估了良好环境将带来的收益。”

要得到关于工作空间生产力方面更多的信息，请参阅：

《人件》第二版 Second Edition，Tom DeMarco 和 Timothy Lister 合著, Dorset House Publishing Company, 1999.





http://www.peoplewarecn.com

人件中文版网站