第 28 期（2003 年 8 月）

声音

X 非程序员
-Programmer
软 件 以 用 为 本

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

http://www.umlchina.com/

SteelTrace 架起需求和 UML 之间的桥梁

[2003/8/1]

总部位于爱尔兰的 SteelTrace 公司发布了其需求捕获和管理工具 Catalyze 的新版本，其中两个 UML 集成模块都提供了新的可视化合并的功能。7 月中旬，Catalyze 2.5 版本也被发布。

技术主管 Fergal McGovern 认为，为使用 IBM 的 Rational Rose 以及 Borland 的 Together 等工具的建模人员提供的集成模块。使得 Catalyze 可以自动地由非 UML 的产品中生成 UML 的 Use Case 图和活动图。他认为，在需求管理工具市场上，没有其它产品可以从其它模型中生成 use-case。

同时，软件架构师可以在 Rose 或 Together 中对活动图或者 Use Case 图进行修改，并将结果合并到 Catalyze 工具中，其中业务分析员可以决定是否接受这些修订。

“我们试图解决业务人员和 IT 人员之间的隔阂”，McGovern 说，“在 UML 和需求工具之间存在一个设计的鸿沟”

McGovern 解释，这是一个新的双向关系，它使得软件组织可以在业务需求和软件设计之间建立更好的可追溯性，为此他举出了考虑从自动取款机中提取现金的需求的例子。

“需求是这样的：从银行中提取现金的操作由一系列步骤组成，可能会给出现金，也可能会报告例外”，McGovern 说，在传统的需求管理工具中，各种需求会被记录为 use case 并在 UML 建模工具中表达出来。

但是，Catalyze 允许使用一种“step-wise”（按步骤说明）的方式来结构化地组织需求，其中的每一步，如输入 PIN、或者交付 20 的整数倍数额的现金，都可以映射为一个活动图，“通过 Catalyze，你可以创建一系列步骤，这些步骤组合起来，完成了提取现金的需求”

Catalyze 中其它的新功能包括在多个项目之间复制和重用 use case，以及根据 Catalyze 生成的 WORD 文档中的变化，反向工程，对 Catalyze 的需求模型做出相应的改动。

目前，Catalyze 2.5 按用户进行发售，McGovern 介绍，专业版本是 US$2,395/seat，企业版本是$4,895/seat，其中包括了协同的功能以及本地的数据缓冲。集成模块的售价在$800 到$1,200 之间。

（自 SDTimes，袁峰 摘译，不得转载用于商业用途）

Gentleware 双喜临门

[2003/7/15]

喜之一：Gentleware AG 获得新的投资

喜之二：Cris Kobryn 成为新的管理委员会成员

德国，汉堡，7 月 15 日 /PRNewswire/ -- Gentleware AG, 最流行的 UML 工具供应商之一，成功地结束其第二个财年。风险投资商 Neuhaus Techno Nord 博士为 Gentleware AG 注入了七位数的资金，后者已经盈利。而且，软件和系统建模领域的世界级专家 Cris Kobryn，又加入了公司新的管理委员会。

在该项交易成功后， 来自汉堡的风险投资商 Neuhaus Techno Nord 博士的资金已经开始进入这家公司，Gentleware 公司 20 月前刚刚上市的，但去年就已经开始盈利。该公司 CEO，Marko Boger 博士，指出“我们的产品在市场上颇受欢迎，现在，Neuhaus Techno Nord 博士的投资使得我们可以加速产品线 Poseidon for UML 的发展”。德国杂志《Java Magazin》最近的“2003 年读者选择”调查指出，Poseidon for UML 现在的下载数超过了 350,000，已经是软件建模领域占据第三位的 UML 工具。

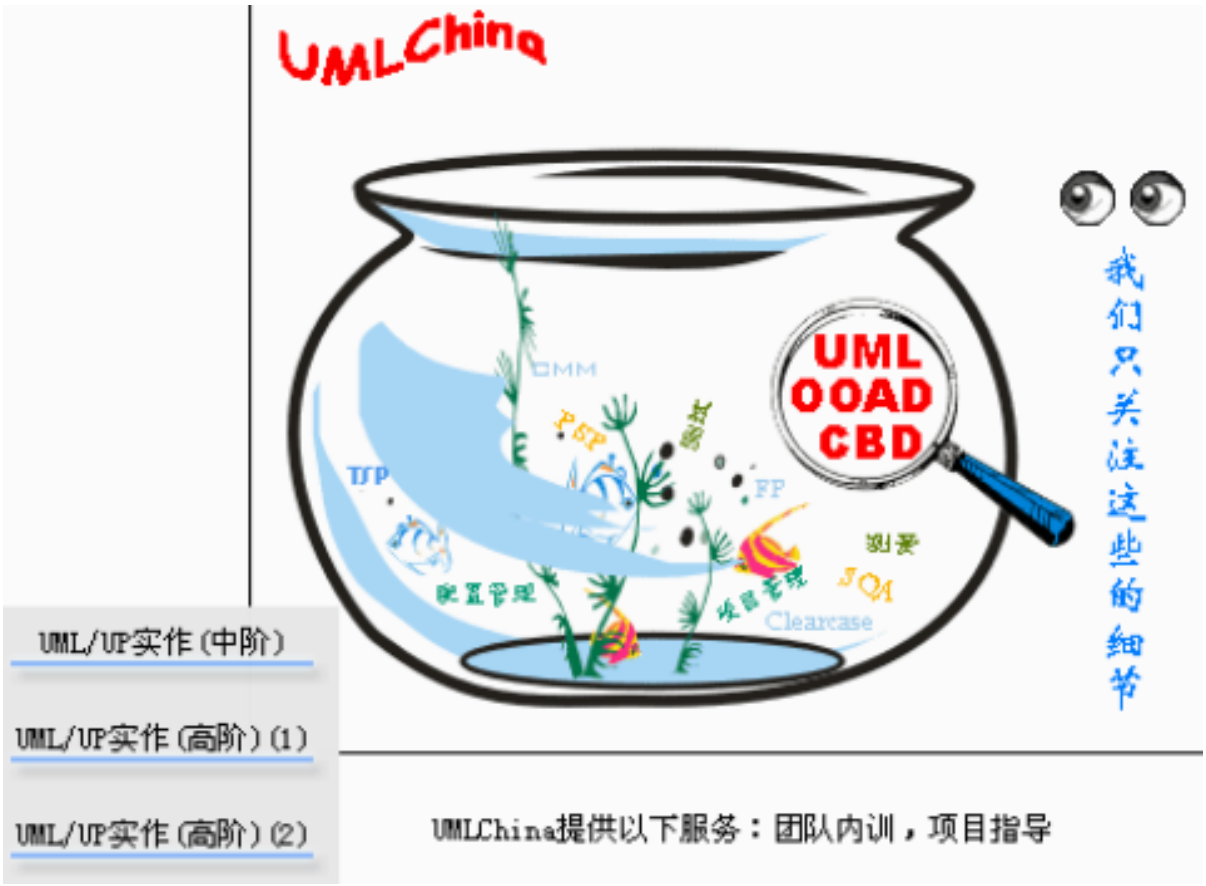
“稳定的客户群、成熟的产品方向、已经获得盈利的业务模式、优秀的管理团队，所有这些，使得我们对 Gentleware 充满信心”，Gottfried Neuhaus 博士在解释其投资决定时谈到，“在我们的资金帮助下， Gentleware 将能够稳步进入成长期，并不断稳固产品开发、市场和销售。”

随着 Cris Kobryn 成为管理委员会的一员，公司将受益于其在产品发展方面的技术知识以及他在标准化组织中表达公司利益的能力。作为 OMG4s 的分析和设计专家小组的副组长，以及成功制定 UML2.0 的 U2 Partners 提交小组的主席 ,他是软件和系统建模领域世界闻名的专家。Cris Kobryn 认为，“和其它公司所不同，Gentleware 公司强大的市场竞争力，来源于其创新的 Poseidon 产品，以及其在 UML 2.0 标准化方面的行业领先地位，公司的创新和长远的策略使得它在快速兼并的软件建模和开发工具市场中能够占据一席之地。我希望能够尽快对 Gentleware 有所建议并帮助他们实现潜能。”

关于 Gentleware AG

Gentleware AG 于 2001 年创建于汉堡。作为行业领先的基于 UML 进行软件建模的开发工具，其产品线“Poseidon for UML”已经被下载 350,000 次以上，除了免费版本之外，Gentleware 也提供了其它收费版本（标准版、专业版以及嵌入式版本），其中提供了 UML 专业使用的各种功能，其产品可以从网站 http://www.gentleware.com 下载。Poseidon4s 的主要优点在于其超级的易用性以及对各种国际标准的符合。除了工具定制之外，Gentleware 还独创性地提供了培训和咨询方面的 know-how 服务。

（袁峰 摘译，不得转载用于商业用途）



JRules4.5 增加图形表示规则新功能

[2003/7/1]

在上月中旬发布的 ILOG 公司的 JRules 4.5 新版本中，增加了图形表示规则的新功能。

Colleen McClintock，业务规则产品市场部负责人，介绍：在业务规则的创建、使用以及管理上，不同的用户扮演着不同的角色，基于这一点，ILOG 在其新版中增加很多不错的特性。

“目前的趋势是：组织们将会有基于同样业务规则的多个应用”，McClintock 补充说，2002 年 3 月发布的 JRules 4.0 已经作了很多基础工作：引入规则库，将业务规则与具体的应用分离开来。用户只需要在一个地方更新业务规则，而所有关联的应用都将自动得到更新。在 4.5 版本中，开发人员、软件架构师和业务策略管理人员可以配置 Jrules 的构造器环境，以便根据各自定制的风格来对规则进行创建和管理等工作，当然，操作的规则库是公共的。

软件架构师可以使用 UML 的类图来表示业务对象。“图形的表示可以给出规则之间关系的更多细节描述”，McClintock 介绍，同样，也可以用 UML 的活动图来表示任务和驱动任务的规则之间的联系。例如，利用新功能“RuleFlow”，可以排列一个产品中用到的规则，并对其进行验证。

对开发人员而言，JRules 4.5 提供了一个 point-and-click 的部署框架，可以生成面向 J2EE、J2SE 或者类似于 Web service 的规则服务。实际上，可以将规则引擎嵌入到一个 Web service 中，也可以创建规则来调用 Web services。McClintock 还介绍了顺序执行模式的新性能，它为规则的评价提供了更有效的方法。另外，加强的队列功能允许用户保存队列，例如，在分析某规则变化的影响时，需要重用过去保存的队列，这时候，新的功能就可以发挥作用了。

JRules 4.5 支持 JCP 即将颁布的 JSR-94 规约，后者为使用规则引擎应用提供了基于 J2EE 的标准 API，McClintock 透露，这项工作目前已经完成，但 Sun 的授权必须在公布之后才有效，她希望在 6 月底规约就可以使用。

ILOG (www.ilog.com) 声称，他们将在明年的某一时候发布使用 C#语言开发的基于.NET 的规则平台。初期，JRules 4.5 发行价格为每一个开发者 license 12,000 美元，服务端每进程 20,000 美元。

（自 SDTimes，袁峰 摘译，不得转载用于商业用途）

软件管理与软件工程译丛

50%的中国软件公司老板从不关心自己的属下。

—CSDN 调查，《程序员》2003.8

Peopleware

人件

[第2版]

[美] 汤姆·迪马可 蒂姆·李斯特 著
Tom Demarco Tim Lister
UML China翻译组 译
方春旭 叶向群

清华大学出版社

UML 相关工具一览（2003 年 8 月版）：O-R

整理

查看评论

接 27 期…

工具（最新版本）	厂商	试用允许	UML 支持								支持代码环境	XMI	平台	备注
			用例图	类图	状态图	活动图	顺序图	协作图	构件图	部署图				
ObjectGeode 4.1	Telelogic（瑞典） http://www.telelogic.com/products/additional/objectgeode/index.cfm Telelogic	有 Demo 版									C/C++			UML/SDL 实时开发，产生的代码可以在 CHORUS, Nucleus, OSE, OSEK, pSOS+, VRTXsa®, VxWorks, WIN32 等实时操作系统平台运行。
objectiF 4.7	microTOOL（德国） http://www.microtool.de/objectiF/de/index.htm microTOOL making IT better	有 Demo 版	√	√	√	√	√		√		Visual C++, JBuilder, Visual Café, IDL, SQL, Visual Basic	√	Windows	

ObjectMaker	Mark V Systems http://www.markv.com/products.html 	有试用版												支持大多数建模符号。
ObjectPlant 3.3.2	http://www.arctaedius.com/ObjectPlant/ 	共享软件	√	√	√	√	√				C++, Java, Objective-C		Mac OS X	
OCL Compiler 1.0	Cybernetic Intelligence GmbH http://www.cybernetic.org/prodocl.htm 	免费												OCL 检查工具，可以整合到 SELECT Enterprise 和 Rational Rose 中。
OCL Parse 0.3	IBM http://www-3.ibm.com/software/awdtools/library/standards/ocl.html 	开源											Java	支持语法检查和部分类型检查，无 IBM 官方支持。
OpenTool 3.2	TNI-Valiosys http://www.tni-valiosys.com/?p=industry&s=aerospace&ss=opentool&type=overview 										C++, Java, SQL			提供模型级的模拟运行。支持团队开发和需求管理。文档可以以 MIF, RTF, HTML, Interleaf, LaTex, PS 等多种形式产生。提供 OTScript 定制语言。
Oracle9i	Oracle										DDL, VB, C++			业务流程建模，数

Designer	http://www.oracle.com/ip/develop/ids/index.html?designer.html ORACLE													据建模。
PLASTIC 2003	Plastic Software（韩国） http://www.plasticsoftware.com/ pLASTiC SOFTWARE	个人版免费	√	√	√	√	√	√	√	√	Java		Windows	MDA 支持
Poseidon for UML 2.0b	Gentleware AG（德国） http://www.gentleware.com/ gentleware developing differently	Community Edition 免费，其他版本试用	√	√	√	√	√	√	√	√	Java	√	Java	基于开源项目 ArgoUML 的商业产品，支持多国语言。UML2.0 支持。XMI1.2 支持。
PowerDesigner 9.5	Sybase http://www.sybase.com/products/powerdesigner/ SYBASE	有试用版	√	√	√	√	√	√	√	√	C++, Java, C#, VB.Net,XML		Windows	把 Sybase 产品和 UML 结合起来
ProVision Enterprise Pro 4.2	Proforma http://www.proformacorp.com/provision/enterprisepro.asp proforma	有 demo 版	√	√		√					C++, Smalltalk, ERWin, VB, SQL		Windows	集成业务建模和对象建模工具。
PROSA UML 2003	Insoft Oy（芬兰） http://www.prosa.fi/		√	√	√	√	√	√	√	√	C++, Java, C#, COM, SQL		Unix, Windows	可以把状态图和活动图转变成可

														执行的 C++，Java 代码。
ProxyDesigner 1.0	ProxySource http://www.proxysource.com/Home.asp 	免费	√	√	√		√	√	√	√				支持 Patterns。而且把 ProxySource 社区集中联合起来，分享各自的模式、架构、设计。
QuickUML 1.1	Excel Software http://www.excelsoftware.com/ 		√	√			√						Windows，Linux，Mac OS X	
RAPID RMA	Tri-Pacific http://www.tripac.com/html/prod-toc.html 	有试用版												和 Rose Real-time、i-Logix Rhapsody 结合
Real-time Studio Professional 4.2	ARTiSAN  http://www.artisansw.com/products/products.asp	有试用版	√	√	√		√	√	√		C++，Java 双向工程，状态机模拟。Ada83，Ada95，C，SQL-DDL	√	Windows，Solaris（服务器端安装）	支持 UML2.0，可以和 PVCS、VSS、ClearCase、CM Synergy 结合。

reView 3.2.5	Headway http://www.headwaysoftware.com/ 	有试用版									Java, C++			
Rhapsody 4.2	I-Logix http://www.ilogix.com/products/rhapsody/index.cfm 	30 天完整试用	√	√	√	√	√	√	√	√	IDL, Java, C++, Ada, C, C166, M16C, COM	√	Linux, Windows, Solaris	实时嵌入系统开发利器，能从状态图、活动图生成代码。可以导入 Rose 模型。
ROCASE 1.0	"BABES-BOLYAI" University Computer Science Research Laboratory（罗马尼亚） http://lci.cs.ubbcluj.ro/rocase.htm 	免费	√	√	√						C++, VC++, BC++		Windows	支持 OCL，MFC
Rose 2003	IBM Rational http://www.rational.com/ 	可以试用	√	√	√	√	√	√	√	√	Java, C++, VB,Ada, IDL, Delphi, SQL, Oracle	√	Windows, Unix	目前最流行的 UML 工具
Rose Real Time 2003	IBM Rational http://www.rational.com/ 	可以试用	√	√	√	√	√	√	√	√	C++	√	Windows, Unix	支持 UML 实时扩展，支持 ROOM 方法。可以从状态图生成 C++代码。
待续…														

[OOAD*UML 谬论集之二]

点评独孤木专栏《漫谈 UML OOAD and RUP》（下）[1,2]

2003-07-06 版本 A

IT 之源 张恂

查看评论

[引言]：

有幸拜读了独孤木先生的连载文章，发觉其中的错误和问题有一大砣，而大陆历来有相似看法的人也不少，这些关于 00 缺陷的老生常谈和似是而非的抱怨指责我已经听得耳朵起茧，也反问过自己好多年，因此按捺不住，趁闲暇之余提笔写就。

对于技术性探讨，我本不是软件文坛的“嘻哈族”，罗里罗唆演绎了一通却让人茫然不知所云，只留得个风趣幽默、文笔优雅、技巧娴熟的苍白印象。我一向不太喜欢添加太多文学和/或哲学的调味料，也不懂得如何煽情，所以只好把本人意见直接加注在原文中（略有删节和调整），如有不当冒犯之处望海涵。

使用者或者是客户的信息人员，看不懂相关的文件

开发项目到底会遇到什么样的客户？其实就像是跟网友见面差不多，还没有看到真人，你永远不知道哪个每天跟你聊天分享心事的超级美女，其实是一个中年男子。就算你运气好，以前已经跟这个使用者接触过，彼此混的很熟，还是有可能会发生变化。如果以前的项目做得好，这个人有可能升官，所以他就不会做这个专案了；如果以前的项目做得不好，有可能这个人就被列入下次裁员的黑名单里，所以他也不会做这个项目。更不要提有些时候，你是跟一些从来都没有打过交道的人一起开始做一个新的项目。

[注]：所以，以不变应万变，开发商具备牢靠的软件工程实力太重要了。

既然我们在描述的对象是项目，大部分的项目，都是从需求分析开始。使用者便会提出他们的需求，系统分析师听到使用者的需求以后，就会开始把他所收集到的需求写成文件，接着会去跟使用者确认需求是否便是如此。

[1] 繁体版，Java 周刊，http://www.javatwo.net/javaweek/history/javaweek20030606.pdf
[2] 简体版，CSDN，http://www.csdn.net/news/newstopic/11/11252.shtml

采用 use case driven 的 OOA(object oriented analysis)，你会请使用者确认的文件，当然就是 use case。接着你会依据 use case，开始进行 OOD(object oriented design)。当你画好 sequence diagram, class diagram，你可能会希望客户的信息人员，可以帮忙确认，这些文件所描述的系统，是否正确。

问题是，大部分的使用者，以及客户的信息人员，其实并没有足够的能力，来确认这些文件的正确性与完整性。因为你所提供的文件，他们看不懂。通常需要你的带领，才看得懂。当他们需要靠你解释才看得懂时，这时候通常会有一些问题随之产生。他们通常可以挑出专业领域上的错误，可是他们通常会忽略掉整个系统的完整性。因为他们会觉得，你所没有描述的东西，可能写在另外的文件中。所以如果你提供的文件有错，通常是你所提供的文件可能不完整，其实要到蛮后期的时候才会发现。这时候修改的成本就会变得非常高了。

[注]：难道使用者看懂了你的文件，就不会发生遗漏了吗？如果流程错误，分析能力不够，该漏的还是会漏掉。

当然，如果看懂了（必须要看懂需求是否得到满足），发生遗漏的可能性要小很多。UML 的一个主要目的就是为了帮助使用者看懂开发商的需求文档和设计。

使用者忽略掉系统的完整性，发现不了设计文件的不完整，这与文件描述采用的形式并没有必然的因果关系。使用者和开发商的能力都可能有缺陷，这时候双方都必须根据软件工程的规范流程来做，才有可能保证内容不发生遗漏，可见正确的需求和设计方法以及软件评审的重要性。

为什么采用 use case 来描述一个系统，通常会发生遗漏呢？或许我们应该先看看 use case 是什么。

[注]："错误地采用世界上任何一种需求分析方法，通常都会发生遗漏"，这是正确的废话。

根据我的一知半解呢，use case 就是尝试着用文字来描述系统与外界之间的交互作用。对于没有看过 use case 的人来说，我在此举一个例子来说明。书上最常看到的例子呢，就是一个人用提款机在领钱。虽然我没有写过类似的程序，我可以想象一下，这个 use case 应该包含的内容。

[注]：既然您看过 use case 的书，建议您不要想象，抄一个完整点、正确点的 ATM 例子，很多呀。假如您对 use case 还一知半解，最好不要轻易下结论。

实际上，即使没有写过 ATM 程序，一般人也能写好用例，用户需求与程序如何实现的描述无关。

1.Brief Description

这个 use case 说明，怎么样透过提款机来领钱。

2.Flow of Events

这个 use case，开始于客户把卡片插入提款机后，完成身分认证，并且已经选择要提款。

2.1 Basic Flow

1. 客户输入要领取的金额。

2. 系统检查客户的金额与次数，是否超过系统中所定义每次提领金额与提领次数的上限。

3. 系统从客户的存款余额文件中扣去存款金额的资料。并产生一笔提领纪录在客户的交易文件中。

4. 如果是跨行客户，系统应该产生一笔扣除手续费的资料到信息交换中心。并且更新本行对于清算中心的应收帐款--手续费资料。

5. 进入吐钞 use case。

2.2 Alternative Flows

2.2.1 超过每次允许的提领金额

1. 如果超过每次允许的金额，系统应显示错误讯息：『你不识字吗？ 一次只能领两万！』。

2. 系统应该回到功能选择画面。

3. 回到功能选择 use case。

2.2.2 超过提领次数

1. 如果超过提领次数，系统应显示错误讯息：『你这张卡片已经刷爆了！ 赶快去补刷存折吧！』。

2. 系统应该回到功能选择画面。

3. 回到功能选择 use case。

2.2.3 客户选择取消

1. 如果客户在输入金额时，没有按下确定，却是按下取消，系统应显示 错误讯息:『不要玩我！快滚吧！』。

2. 系统应该把卡片吐出来。

3. 回到吐卡片 use case。

3. Special Requirements

无

4. Preconditions

客户要正确插入卡片，输入正确的密码，通过身分认证，提款机还有足够的钞票在里面。

5. Postconditions

进入吐钞 use case。

通常会被找到的遗漏：

1.为什么没有检查金额是否正确？台湾的提款机，只能够输入 100 的倍数。

你要领 512 元是不行的。

2.怎么没有显示要不要换百元钞？

3.怎么没有检查，机器里面的钞票是否足够？有可能没有小面额的钞票啊。

通常不会被找到的遗漏：

1. 跟金资中心如何达成联机的问题。因为这可能被 include 到另一个 use case 里面去了。

[注]：后台系统如何联机是客户应该关心的需求吗？这种细节当然不应该在这里描述。联机这件事在第 1 步和第 2 步之间应该提一下，您自己故意漏掉，这能怪谁？

2. 没有扣除机器的钞票余额档。

[注]：ATM 吐钱，自己不做记录？即使非银行雇员，让一般人写这种用例，大概也都不会遗漏。太低估大家的智商了。

3. 吐钞口要开开关关测试是否可以正常吐钱。

[注]：您大概没有读过《Writing Effective Use Cases》。

4. 如果吐钞成功的话，要扣机器本身的余额档。

[注]：说一遍不够？见上面第 2 条。

5. 如果成功的话，要把客户未登折次数加 1。

[注]：这是客户应该关心的需求吗？

…因为我没有写过 ATM 程序，只能随随便便想象可能会有的问题。

[注]：真奇怪，既然您在写用例的时候，就已经想到了可能遗漏之处，怎么还会发生遗漏呢？把这些内容补上不就行了？而且好像正是用例这种格式帮助您思考、发现“遗漏”的吧，应该感谢才是啊。

我想，用 use case 开发比较大的问题在于你其实有可能会遗漏掉一些系统该做的事情。在单一 use case 中，有可能你会有非常多的 alternative flow。每个假设，都有可能不成立。所以你得要定义如果这个假设不成立的时候，系统要响应什么。问题在于一般的使用者，他们提出规则的时候，会把预期系统的反应写在旁边。例如，如果系统没钱，就显示没钱错误讯息。

[注]：可能我比较笨，实在搞不懂这段话的逻辑。读了好多遍，也不知道您是怎么得出 use case 开发容易遗漏的结论的？

恰恰相反，这不是您所说的“用 use case 开发比较大的问题”，而正好是“比较大的优点”。您说到点子上了，用例的好处是告诉了我们有很多的 alternative flow，每个假设都有可能不成立，所以你得要定义如果这个假设不成立的时候，系统要响应什么。用例通过一种固定的格式，提醒使用者和开发商需求的复杂性，不要像过去那样，等到编码了才考虑这些问题，而是应该在需求阶段及早解决，因此不大可能遗漏掉为了满足用户，系统该做的事情。说实话，我到现在为止还不知道有比用例更好的需求分析方法。

问题是用了 use case 以后，很多这样的规则，因为你把系统的整个行为模式全部都展开出来，篇幅就会拉的非常长；如果你把共享的部分抽出来，放在 include 的 use case 中，user 又要交叉比对才可以看到对的东西。当你看到长篇大论的时候，眼睛看的久了，很容易就漏掉该写的东西。除非我先把所有的规则都写下来，整个 if then else 的决策树也画出来，不然哪记得你应该写 25 个 alternative flow 而不是 24 个？而这里就会变成是 user 还要花时间去一个一个比对，他们的 requirement 是否都被 use case cover 到了。通常使用者会把这个工作交给 SA 来做，他们

再来看结果。因为 user 通常都很忙，所以 SA 整理出来的结果他们通常也没有时间详细地 walk through。所以该遗漏的东西还是会遗漏。

[注]：估计这些都是没写过用例的人在瞎操心，杞人忧天。

Alistair、Ivar 不是告诉我们，数量和步骤太多的用例不是好用例吗？只有写得不好的用例，篇幅才会拉得很长。再复杂的用例一般也不会超过 4 页纸，1 个项目主要的用例不会超过 20 个。但针对不同的项目，“非常长”的标准恐怕也是不同的。

您了解 goal-based use case，以及用例分层、封装和隐藏的道理吗？include 用例就像调用子函数，只关心函数的输入和输出就可以了，何必交叉比对内部的细节呢？有了用例，您不必画什么 if then else 决策树，也不必一下子（其实也做不到）把所有的规则写下来，而是应该根据风险级先记录当前阶段最重要的步骤和内容。00 方法允许迭代递增地开发，所有的需求是逐步完成的，不会遗漏关键信息。

这段文字好熟悉啊。曾经有专家嫌 UML 几张图比对麻烦，就建议把 sequence diagram 和 activity diagram 并在一起，省掉一个图。好像搞结构化的人都不大晓得抽象、分层和多维视角的道理，看东西总爱用平面透视的方法，事无俱细特别操心。

用例是契约的一部分，如果客户连需求都不重视，嫌多不爱看，这个项目就岌岌可危了。使用者实在不愿看，那也没关系，想办法让他们签字，如果需求出了问题比如遗漏了什么，责任自负。这种遗漏与 use case 的长短无关。

另外一个问题，则在于有些东西，是刚好介于 use case 与 use case 之间。因此他会预期在 use case A 中发现的东西，他没看到，他就会觉得可能是写在 use case B 之中吧。当他去看 use case B 的时候，他还是没看到，这时候他不见得会记得，他还想看到什么。因为我们在 review 文件时，通常都只会看到这份文件描述的 scenario 对不对，比较少去想到底缺了什么。

[注]：看得我一头雾水。能告诉我们，介于 use case 与 use case 之间的“有些东西”到底是什么啊？

所以有时候一少，就是少掉一整组 use case。例如关于一些系统在运转时会用到的参数档，应该要有如何去维护这些参数的 use case。这就常常被 user 忽略掉。这在采用传统结构化分析画 DFD(数据流程图)的世界里，是不太可能发生的，因为每个 data，都要描述它是怎么样去 maintain，或是怎么样进入系统中。资料不是来自其它系统，就是来自使用者的输入，或是系统本身运算出来的结果。透过 DFD，资料的流向与加工会非常清楚。然而使用 use case 就没有这个好处。我觉得遗漏是 OOA 的天性，难怪得要配合 iterative 的 process。

[注]：如果这些参数档对用户是可见的，有人来维护它，那么就应该有相应的用例来表示。use case 本质上就是行为序列、交互事件流，其中用到的每个用户 data，都要描述它是怎样去 maintain，或是怎样进入系统中。既然“资料不是来自其它系统，就是来自使用者的输入，或是系统本身运算出来的结果”，如果作正确的用例分析，这些资料是不可能漏掉的。

大概您只知其一，不知其二吧。除了 text descriptions 之外，我们还可以用 UML 的 activity diagram（取代了结构化分析的 DFD）、statechart diagram、interaction diagram 从多个角度来描述用例的结构和行为，不但可以非常清楚地描述资料的流向和加工，而且比 DFD 分析能力更强。

遗漏是人的天性，为什么偏要武断地说这是 OOA 的天性呢？难道用传统结构化分析就不会遗漏了？过去有多少失败的结构化项目？您怎么这么自信呢？也许您一定有 100%不遗漏的非凡项目经验.....

事实正好相反。我们为什么要用 OOA 而放弃 SA 呢？道理很简单，客观世界是复杂的，而 OO 世界观更贴近现实，它有很多 SA 所没有的概念和机制供我们使用，SA 在许多方面缺得太多了。虽然 OOA 也不能做到 100%不遗漏，但它显然比 SA 要强多了。

提出 iterative process 的主要目的是为了减少项目风险、适应需求变化、保证交付进度，根本不是为了解决什么遗漏问题。

特别强调要用这样的方法，另外还会衍生出来的问题是，有些客户因为看不懂这些文件，所以会坚持以他们所提供的文件当作是系统的范围。这通常就会产生非常多的事端。

[注]：这不是个问题。如果客户不理解用例，我们完全可以用客户原来熟悉的文件作为输入，然后再转换成用例驱动的 OO 开发。建议您看一下 RUP 关于这方面的介绍。

客户使用者甲：布鲁斯，你们写的这个 use case 我们研究了很久，我们看不懂。这样我们不敢在这份文件上签名。

[注]：用例写作一定要有用户参与，因为他们才是软件的最终使用者。好的用例是简明易懂的。用户看不懂自己参与写的用例，这种情况真得很罕见，除非布鲁斯的 use case 写得实在太差。事实上只有用户参与才能写出优秀的用例。

布鲁斯：你们看不懂，我可以随时来解释啊。你们一定要在这份需求文件上签名啦。我们一定要有一个基准，不然以后发生问题怎么办？

客户信息部门人员乙(帮忙打圆场)：布鲁斯，我知道 use case 这个东西是最新的方法论。可是我们的 user 就是水准还没有到这边。

[注]：用例不是什么最新方法论，差不多有 30 年的发展历史，其本质完全符合人们的直觉思考。

怎么用软件完成自己的任务，客户总应该知道吧？如果用户连用软件来干什么都不知道，还谈什么开发？知道了怎么用软件，use case 不就有了吗？这和 user 的水准没有丁点儿关系。

客户使用者甲：其实系统的范围，我们一直都写得很清楚啊。我上次寄给你的 power point 文件就把系统的功能都写的很清楚了。

[注]：客户提供的 PPT 可以作为用例分析的起点，但这还不够。什么是软件功能，就是那几幅图，几百个字的描述吗？只有细化到用例这个层次，才能具体地、无二义地告诉我们功能是如何使用的，什么是真正的功能。

布鲁斯心想，狗屎，这么不详细的东西也可以拿来算数的喔？：我是觉得那份 power point 文件是已经把系统的功能大方向都点出来了啦，可是还是有很多细微的地方没有提到。(这应该算是一次成功的防御。)

客户信息部门人员乙(帮忙打圆场)：不然，我们请 user 把他们的想法写的更明确好了，我想可能要把他们的作业流程跟需求写的更清楚一点。

客户使用者甲：好吧，我把以前提供给你们的规则写的更清楚一点，再加上我们以前的会议记录，就是我们系统应该达成的范围。

布鲁斯　：这样不行啦。我们的人都是 base on 我们这份 use case 来开发呢？

客户使用者甲：好啦，我辛苦一点，我尽量把你的 use case 看一看，挑挑看有没有问题。可是你在今天的会议记录上要写清楚喔，系统的功能应该以我以前提供给你们的规则为基准，再加上我们以前的会议记录，就是我们系统应该达成的范围。至于你们的 use case，我是不会签名的。

[注]：如果实在不行，就像前面提到的，我们可以采用客户许可的传统格式需求文件（包含各种规则、约束和要求），并把这份文件作为签字的依据，use case 可以作为非正式的沟通工具，两者并不矛盾。

布鲁斯想，看来要他们确认是很难的啦：好吧，那就只好辛苦你了。你需要多久的时间？…

过了几个月，使用者看到头一个版本后，双方再度开会。

客户使用者甲：我们在文件里面提到的功能，你们都没有做到。

布鲁斯　　：那是因为你在 review use case 时，也没有提出这一点啊。 这样啦，我们在下一个 iteration 把它纳进来。我会回头改过 use case，再让你 double check 一次。

客户使用者甲：好吧。希望下一个版本就可以看得到。

过了几个月，已经把原有预计要走的几个 iteration 全部都走完了，功能还是不如预期，所以双方再度开会。

客户使用者甲：我们已经看过多少个版本了，你们一直到这版，都还是问题百出。你们到底有没有认真去看过我们所提供的文件啊？

[注]：这就是您想象的 iteration 吗？如果功能出了问题没有达到预期目标，早就应该发现了啊，布鲁斯够愚笨的。

布鲁斯：我记得上次我们已经应你们的要求，把 requirement 跟 use case 的对应都做成 excel，一条规则一条规则让你们确认了，你们还是没有确认出来，还提出这么多 change request。我不管，这些我们得要收费。

[注]：如果以传统需求文件为依据，客户不想采用 use case，那么 requirement 跟具体 use case 的对应应该由开发商负责管理，布鲁斯没有理由责怪客户未能确认 use case。

客户使用者甲：收钱？你翻翻我们 7/5 的会议记录。虽然在我们原始文件提出的规则里面没有描述到这条规则，可是我们在会议记录里面有提到这个功能需要检查员工到职不满一年，不适用这个状况啊。这是我们在去年 6 月底检讨作业办法时修订的啊。

布鲁斯：这应该算是 change request。况且你们 review use case 已经 review 那么多次了。我记得我们在 12/14 的会议里面有提到，凡是没有列在 use case 里面的需求，都应该算是 change request。

客户使用者甲：那是你单方面的想法，谁同意啊？况且你们改过那么多次版本，我们哪有能力去看你每个版本，记得你每个版本里面到底写什么？我都跟你说我们看不懂 use case 了，是你说你们的人一定要看，其他的文件看不懂，才帮你检查的。现在问题就都在我身上？

[注]：客户说的有道理。

布鲁斯：话不是这样讲…

过了不晓得又多少个 iteration…

客户使用者甲：我下个礼拜要调到 BOS 部门去了。

布鲁斯：那我们怎么办？

客户使用者甲：我还在我们公司啊。新的承办人不错啦，我会有空多帮他的忙。

过了一个礼拜…

客户使用者丙：这个 use case 是什么东西啊？

布鲁斯：……

[注]：布鲁斯的做法是错误的。这个编撰的故事从反面说明了在客户拒绝接受 use case 的情况下，应该变通地应用 use case 来管理需求。

信息人员本身不了解 UML, OOAD 以及 RUP

其实客户不了解 UML, OOAD 以及 RUP 是很正常的事情。我除了在看新人的履历表，可以找到精通 UML，熟悉 OOAD，以及专精 RUP 的人以外，在现实生活中，大概只有在 Rational 这家公司出来的顾问中，才找得到自认为他非常熟悉这些东西的人。

[注]：您一定是孤陋寡闻了，在现实生活中至少有 3 个人——Scott Ambler、Martin Fowler、Craig Larman——非常熟悉这些东西，却不是从 Rational 出来的。而且据我了解，现在全球包括大陆实际借鉴采用 RUP 的公司已经非常多了。

大部分听过这些 term 自认了解的人其实都一知半解。(这不包括我，我是根本不了解。)可是最怕的就是不懂装懂。如果你遇到客户的信息人员不了解这些东西，却在上完短期的课程后，想要给你来些良心的建议，还是卓越的指导，你就完了。

[注]：您太谦虚了，没有说实话。您说自己根本不了解，却知道大部分自认了解的人其实都是一知半解，而且客户的信息人员也不懂装懂，不了解您所了解的东西，您才是真正大彻大悟的高人啊！

您说的千真万确，最怕的就是不懂装懂。

客户 IT 人员甲：我觉得你这个图这里画错了。这个关系，应该用实心的菱形？

布鲁斯：你误会我们想要描述的关系，其实我们在图上并没有刻意去…

过了半小时…

客户 IT 人员甲：我觉得你这个 use case 这里用『当使用者输入 email 后，系统应检查 email 正确性。』这样写不够清楚，你应该还要描述 email 格式有错时的 alternative flow。不然 programmer 怎么会知道，系统要怎么响应？

布鲁斯：我们针对这些问题…

过了一小时…

客户 IT 人员甲：你的文件我们看得差不多了，现在我们来看 RUP 的 artifact…

[注]：客户说的很有道理啊！

布鲁斯心想，杀了我吧，这种无聊的会还要开多久啊…

我遇过最狠的，是在 use case 的叙述里面挑语句是否通顺。原则上呢，就是在改作文。如果你用英文写，就是抓你第三人称是否记得加 s 之类的问题；如果你用中文写，就是嫌你作文写得太差。

[注]：一点儿没错，用例的格式有比较严格的要求，写 use case 实际上就是在写作文。客户对于需求文件挑剔是理所当然的，语句通顺是起码的要求。

并不是所有程序员都能写好用例和需求，这需要专门的训练。写程序实际上不也是在用高级编程语言作文吗？其难度并不亚于普通写作。写软件文档、编程序某种程度上都需要良好的作文技巧。如果一个人不具备合格的中英文写作能力，恐怕他/她也很难成为合格的软件工程师。

当然，客户抓你第三人称是否记得加 s 之类的问题就有点过了。

随笔提到另一个更狠的客户，这位小姐的挑错就跟 use case 没啥关系。她只是强调我们用 html 做成的 prototyping 上面所有 error message 的标点符号，要统一变成全角中文。这样 error message 才不会有的比较宽，有的比较窄。尽管我们再三解释 prototyping 的用途不在于此，她还是坚持要我们把所有的标点符号换成全角，她才愿意继续 review 下去。我们换了好几次，每次只要一有漏掉，就会被她抱怨，我们低下的作业品质，似乎为她想要找人出气的生活，带来不少乐趣与练功的靶子。

[注]：这就是用户的非功能需求啊——error message 显示时必须用全角符号，格式保持美观！

你们连这种根本没难度的小细节都不屑一顾，屡教不改，以后出来的正品还不知道会漏掉什么东西、变成什么样子。难道满足客户是句空话？说不定客户就是利用这种小事来考验你们。态度决定一切。

客户 IT 人员甲：为什么你们在 use case 里面没有描述，可以在 class diagram 里面设计出这个 class？这分明是你们分析与设计不连贯。

我：use case 与 class diagram 没有一对一的关系啊。

客户 IT 人员甲心想，你分明是在狡辩：你们没有遵照 RUP 来开发程序…

[注]：use case 并不是类的唯一来源。

后来经过我引经据典，舌战群儒，终于赢得了这场辩论。比较年少无知的我，以为在辩论上获得胜利，应该获得英雄式的肯定，客户应该要跪拜在真理面前向我膜拜。

对信徒来说，要先做 OO 的 analysis 才能进行 OO 的 design，有了 OO 的 design,，才可以找出 design pattern，才可以建立可以被 reuse 的 component。这几乎是跟先有鸡，才会有蛋一样真实；只是对我来说，我们现在所谓的 OOA 跟 OOD 之间的关系，比较像是狗跟蛋之间的关系。明明就是两个不相同的物种，怎么会有什么关系呢？我记得我小时候学习 OOP 时，class 都是从天上掉下来的礼物，跟 use case driven 的 OOA 中间有什么直接的关系呢？在我的那个年代，只要你有眼睛，学过 data structure 与 algorithm，观察现象，就可以想出 class 出来。只是这种好日子已经过去了。

[注]：你说的不知是哪里的信徒？顺序好像颠倒了。应该是有了 design problems，才可以找出适用的 design patterns，然后才有 OO design 的结果。当然，很多人也不用模式，喜欢体验自己发明设计的快感。

OOA 是问题域，OOD 是解决域，OOD 是 OOA 的实现，这就是两者的关系。而 OOD 是 OOP 的抽象，是 OOP 所要解决的问题，OOP 则是 OOD 的实现。从 OOA 到 OOD 再到 OOP 就像楼层之间天花板与地板的层次衔接关系。无论 OOD、OOP，都是为了直接或间接地满足 OOA 中的 use cases（代表了功能需求）才有存在的必要，所以可以把 use case driven 作为软件开发的轴心和主线。

由于 OO 方法带来的好处，在 OOA、OOD、OOP 之间许多类、概念和表示方法可以沿用，决不是像您所说的两个物种的关系。您大概不知道除了 C++ class、Java class 之外，还有 analysis/concept class 吧。OOA 的一个主要任务就是通过建立概念模型，帮助寻找稳定、具体的 design class，而 OOP 中的类也都可以追踪到 OOA 的 use cases 或分析类。

客户要求开发商业应用，学过data structure与algorithm就可以想出class来，扯谈？没有object-orineted problem analysis，哪来 object-oriented solution design？您说通过眼睛观察现象就可以想出 class 来，说明这些 class 其实来自现象，而现象观察就是属于 OOA 的范畴。

幸好历史总是在进步的。过去只有 OOP，OOD、OOA 是后来在 10 多年内逐步发展起来的，这反映了软件工程技术发展的成熟。如今还有哪家稍微成熟点的公司接到开发业务上来就直接编码的？没有经过认真地 OOA、OOD，OOP 将失去牢固的基础，这种“好日子”是不可想象的。拍拍脑袋就可以写出程序来，只能说明那个年代的幼稚。

做苦工做久了，就会想要偷懒，就把共享的东西拉出来。偷懒是所有程式设计师设计出超强 component 的原动力之一。又懒惰又聪明的人，才会想出一些把戏，让他可以出一张嘴，就叫计算机自己把程序写好。这就是 reuse 的由来啊。认识这些好色的强人以后，我就觉得 OOA、OOD 跟可以被 reuse 的 component

之间，根本一点因果关系都没有。难道他们是在逛色情网站的时候，在脑袋里面同时多任务去写 use case 吗？还是同时多任务去画 sequence diagram 吗？

[注]：不知道您所说的这些“强人、超人”的重用效果怎么样，公司在重用上的 return on investment 如何？Ivar 发明了软件构件工程，他曾经告诉我们，component reuse 的关键首先是 use case 的重用。没有分析、设计层面的重用，只晓得低层次的函数级或代码构件的重用是效率很低的重用。世界上先进软件企业项目的重用率可以达到 90%。

一定要搞清楚抽象与现实的关系。他们没有去写 use case，没有去画 sequence diagram，不等于 component 与外部对象的交互不存在。事实上 OO 模型总是现实世界的反映，不是一种杜撰。不管叫不叫 use case 或者 sequence diagram 的名称，对应的那个 abstract reality 一定是存在的。只不过您所谓的这些“高人”没有把这些规律性的东西清晰地写出来，而是模糊地映在脑子里罢了，这就是自发的重用与自觉的重用的区别。

可惜，我了解的超强程序设计师太少了，Ivar 算 1 个，还包括 Martin，Grady，Kent，Bob，他们竟然都是 OOAD 大师！你，我，包括大部分人都是庸人，记性不好，容易犯错误，写程序总免不了返工，所以最好多做点笔记，多画点图。

除了客户不了解 UML，OOAD 跟 RUP 以外，另外一个更糟糕的现象就是 Project team 里面的人也不懂。我预期这种情况，会随着学校教育洗脑的成功而改善。有些小朋友从来都没听过也没画过 DFD，就跟我们拿建构式数学去荼毒下一代是一样的道理。教导比较年轻的一代采用比较笨的方法，可以确保老人的竞争力。

[注]：既然小朋友们没有画过 DFD，也就没有必要硬是向他们推销旧玩具。您不必为他们担心，从来都是长江后浪推前浪，一代超过一代。

您的估计太乐观了，老人的竞争力总是比不上年轻的一代，所以永远无法确保。我们这些老人如果跟不上时代，反而自作聪明，也只有等着被淘汰的命运了。这是历史进步的必然。

在我刚开始接触 UML 的这几年，遇到的现象是 project team 自己都看不懂这些东西是什么。于是彼此之间都在摸索。有经验的老鸟，对于 UML，OOAD 一点概念也没有。可是被逼上梁山，一定得要用，所以就用自己的经验胡乱使用。没经验的菜鸟，虽然懂得 UML，可是缺乏 process 的实践经验，也不懂任何 domain knowledge，所以只能任人宰割。

[注]：经验也有新、老之分。过时的老经验妨碍了学习新知识，这种经验放弃也罢。可惜您接触 UML 这么几年，好像到现在还是没找到门。

问题是当菜鸟发现老鸟画出来的图，还是写出来的文件不怎么样的时候，除了要面对年轻人因为梦想幻灭而心生怨怼以外，还得要面对老鸟漫长的学习曲线。通常在这种情况之下呢，会面临成员间永不停止的争辩，大多数都是引经据典关于正统的辩论，无形之中，耗费了相当多开发的时间。

[注]：建议最好虚心地找一个老师或顾问带一带，把问题彻底搞清楚。如果大家都依仗着过去所谓的经验，感觉“老子天下第一”，谁也不服谁，导致问题旧拖不决，这样的团队不如解散拉倒。

接着就看到原先预设的 schedule，像是自由落体一样坠落。每次遇到这种场景，就让我不禁怀念起那个使用结构化分析的年代。一切都是那么简单、直觉与美好。没办法，每个时代都有属于它自己的流行。就像是关系型数据库一样。

[注]：结构化分析年代真的是那样简单、直觉与美好吗？还是黄粱一梦？

Relational Database

尽管 OO 的声音喊的震天响，Relational database(关系型数据库)还是在 IT 产业中扮演一个非常重要的角色。很多人一直在想，要用支持 OO 的 database。问题是在这个业界里，有太多人熟悉 SQL 以及 relational database。有太多钱花在买 Oracle, Sybase, db2, Informix, MS SQL Server 上。这让人 relational database 退休的机会变得非常小。而 relational database 的基本精神，跟 OO 就不太有关了。这让我们想要用 Object 的方式来思考，遇到 RDBMS，就 O 不下去了。你可以想象一个从悬崖边大声喊着 O，然后跳下去的人，当你听到声音越来越小，着地之前那个低沉的 o(已经变成小写了喔)，这跟你用 OO 的观念遇到 Relational Database 差不多。这个无聊的比喻虽然没什么用，可是又让我多赚几个字的稿费。

[注]：OO 和关系型数据库原本就是为了解决不同问题提出来的，两者并不矛盾，“遇到 RDBMS 就 O 不下去了”简直是自寻烦恼。澄清这个问题，应该从 OOAD 和 OODB 两个层面来看，此 OO 非彼 OO。

OOAD 用在逻辑的软件分析设计方面，与之对应的是结构化分析设计，而关系数据库主要解决信息、数据的物理存储和查询等问题，与之对应的是 OODB，过去还有网状、层次数据库等等，只不过关系模型在这个领域比较常见罢了。所以，OOAD 与 relational database 实际上是互补的，不存在谁取代谁的问题，可以最大限度发挥各自的优势。从 OOAD 设计的应用软件到 RDBMS 有多种连接方法，而许多著名的 RDBMS 本身更是利用 OOAD 开发的。

另外，OO 数据模型与关系数据模型之间可以轻松地相互转换（UML 类图是 ER 图的超集，而且还有专门用于关系数据库建模的 UML 扩展集），所以在数据设计这个层面，OO 与关系数据模型是完全兼容的。

主流的关系模型（表结构）适合描述大多数商业信息系统，而在某些情况下，比如网络管理、空间数据、接口库/服务类型库等等，关系模型就不太合适了，有时还非得用 OODB 来直接保存对象树和森林。究竟是用关系表还是用对象来存储数据，其实只是不同的存储方式而已。所以，即便在数据库领域，也不能说 OO 失败了，relational database 更不是万能的。

为什么那么多无聊的传统论者老是喜欢把 OO（到底哪个 OO？）和关系型数据库人为地对立起来呢？仅仅为了指责 OO 的硬伤，留恋昔日结构化的辉煌？找错地方了吧，这原本就不是个问题，现实就是：OO 和关系可以同时存在。

有些人是用 object 的观念，把 table 包起来。然而，在 performance 不好时，SQL statement 还是会直接写在程序里，真要没办法，还要写 stored procedure。如果已经到了要写 stored procedure 这样子的阶段，还有什么 OOD 可以对应呢？还有什么 object 可以用呢？

[注]：在程序里写 SQL statement，在数据库里写 stored procedure，难道这样就做不 OO 了吗？即使不用 object 来封装 table（这只是一种实现方案），我们也完全可以在其它访问数据库的 Java class 的方法中嵌入 SQL statement。

正如上面所说，OOD（软件设计）与 table（数据库设计）是互补共存的，根本不矛盾。真不知道您所谓的 OO 定义是什么？

在 OO 世界观中，万物都是具有属性和行为的对象，不要以为只有 Java class 和 C++ class 才是对象，database、table 甚至连 SQL statement、stored procedure 本身也可以被作为一种对象来看待。因此，在 OOD 中当然也可以对 SQL statement 和 stored procedure 进行设计，它们不就是对表、数据库的一种操作么？

这个业界有太多熟悉 SQL 的 programmer。对于这些人来说，SQL 的威力这么强大，要去抗拒直接使用 SQL statement 的诱惑，改用纯粹以 Object 的观点来解问题，手会变得很痒。遇到有些很容易透过 SQL 解决的问题时，心会变得更痒。根据研究男性心里的医学报告指出(在此再次感谢灵犬莱西的显灵)，忍耐太久是会生病的。所以只要他们一忍不住出手，通常 performance 就会有钜幅的改善。只是这就跟 OO 没啥关系了。不过，这对于所有 audit 这个项目的人来说，这算是瑕不掩瑜。睁一只眼，闭一只眼也就过去了。Project 可以结案最重要，谁还管 OOAD 啊？这多少也算是 OOAD 面临的问题之一吧。

[注]：前面已经说过，关系模型和 OO 模型是事物的两个不同方面，关系 SQL statement 解决的是关系型数据的查询和操纵问题，Object 解决数据和行为的封装问题，两者可以共存，不矛盾。

究竟是通过 OO 数据对象间接访问表/记录还是用 SQL Statement（照样可以放在 1 个类的操作中）直接读取记录数据，不过是如何合理分配对象职责的一个设计权衡问题（包括性能、可扩展性等），根本与 OOAD 面临的问题无关。

结语

念过大学，准备过考试的人都知道，文不如表，表不如图。要描述你的观念，用一张适当的图形来表示是最强而有力，也是最容易让人理解的途径。然而为什么在系统分析的这个领域里，我们会希望透过 use case 来描述这个系统，而不是透过其它的方法呢？

[注]：因为 use case 是任何系统都具有的独一无二的属性，只要有使用者就会有 use case。不掌握 use case 会使我们遗漏一些最重要的系统需求信息，增加项目的风险。

举双手赞成您说的“用一张适当的图形来表示最强而有力”，use case text 结合 UML 图形来表达效果最好。

如果你采用结构化分析(Structure Analysis)的方法，而你做的项目规模也比较小，其实使用者会有能力画出他们的作业流程图。有了流程图，其实拿给 development team 看，大概也知道要做什么了。认真一点的使用者还可以帮你确认你的 DFD(data flow diagram 数据流程图)。

画出 DFD 以后，就可以让系统的范围用一个图形化的方法加以确认。也可以认清 Data 与 process 之间的关系与流向，接着只要依据资料的流向，写出 function spec，再定义出 data dictionary，大部分的 SA 工作就已经完成了。只剩下如何跟客户确认需求了。

[注]：可惜，您不知道使用者如何来用你的软件，怎么确认需求呢？缺少了 use case 描述，不能称之为真正的 function spec，这正是传统结构化分析欠缺的地方。

如果你又采用了 RAD(快速应用程序开发 rapid application development)的工具，你就可以迅速地建立起系统的雏形(prototype)。透过雏形的展示，以及 function spec，在大部分的项目中，你可以跟使用者建立起一个共识的基础。系统的范围就可以随之确定下来。

一旦 SA 有了共识，要求使用者确认相关的文件，接着采用传统的瀑布式开发方式(SDLC, System Development Life Cycle)来开发系统。当你遇到需求变更的时候，透过 SA 文件的确认，双方也就可以在一套沟通与讨论的基础上继续下去。

接着你再去采用 OOD 的方法来设计系统，用 OOP(Object Oriented Programming)的方式来开发系统，这不是很好吗？

[注]：真得很好吗？从结构化分析到 OOD，有可能吗？不知道您学的是哪一流派的 OO 方法，抑或是您的发明？您不知道 OOAD 和结构化分析设计是两套截然不同的体系吗？有人说它们是正交的，所以似乎不存在二者对接的可能。

再说了，已经有了 OOAD 和 UML，为什么还要在一个项目里用两套方法、两套符号，不嫌麻烦吗？真不知道您是怎么想的？难道就是因为对自己钟爱的 SA、DFD 恋恋不舍，希望把聪明的方法教给小朋友吗？

还有人喜欢写 use case 吗？或许在超大型的项目，例如要一百万个人做个数百年，有个几万个 iteration 的项目里，写 use case 就会是一件很有意义的事情。我想当年盖大金字塔的建筑师，一定写过 use case。

[注]：太夸张了吧，您的想象力够丰富。

我们知道，即使在极限的 XP 项目里也可以使用 use case。

至于 RUP，我个人觉得如果仔细研究 RUP，会对于大型项目开发的流程有很清楚的了解。至于对于中小型项目来说，在你进行项目管理的时候，你可以从 RUP 里面找到许多可以参考的模板。

[注]：别说笑话了，按照您的意思，没有了 OOA、use case driven 和 iterative，再嫁接上一个 SA，这这这……这还能叫 RUP 吗？

只是 iterative 的开发方式，不怎么适用在台湾社会中。尤其是与大多数的项目互斥。如果你是开发自有产品，或许可以考虑采用这样的方式。不过我可不是开发产品的专家。出了问题，除了『你资质比较鲁钝，又缺乏经验，所以没有正确地体悟大师的讲法以及采取正确的做法，才会导致这样的结果…』之外，我可是没有其它比较好的理由喔。

[注]：瀑布型开发方式早就过时了，而且对于大多数项目（不管大小），尤其是工程项目，它往往是错误的选择，iterative 才是正确的方法，信不信由你。

[临别感言]：

有些人很奇怪，大概自我感觉很好，总是恐惧、厌恶陌生的东西吧，所以总喜欢树立一些假靶子，企图用一打假命题来证明 OO 是如何的失败，从而为内心排斥 OO、拒绝学习 OO 寻找暂时的安全感。

衷心感谢独孤木先生让我们有机会通过您的好文再次反省自己，检验自己的知识和智力水平。我并不试图说服您或其他人，可是我再一次说服了我自己……

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

中译本即将上市

样章先行提供>>

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

第一本 UMLChina 训练指定教材

清华大学出版社

用例/OOAD 随笔之一

think 著

查看评论

看下面这张用例图

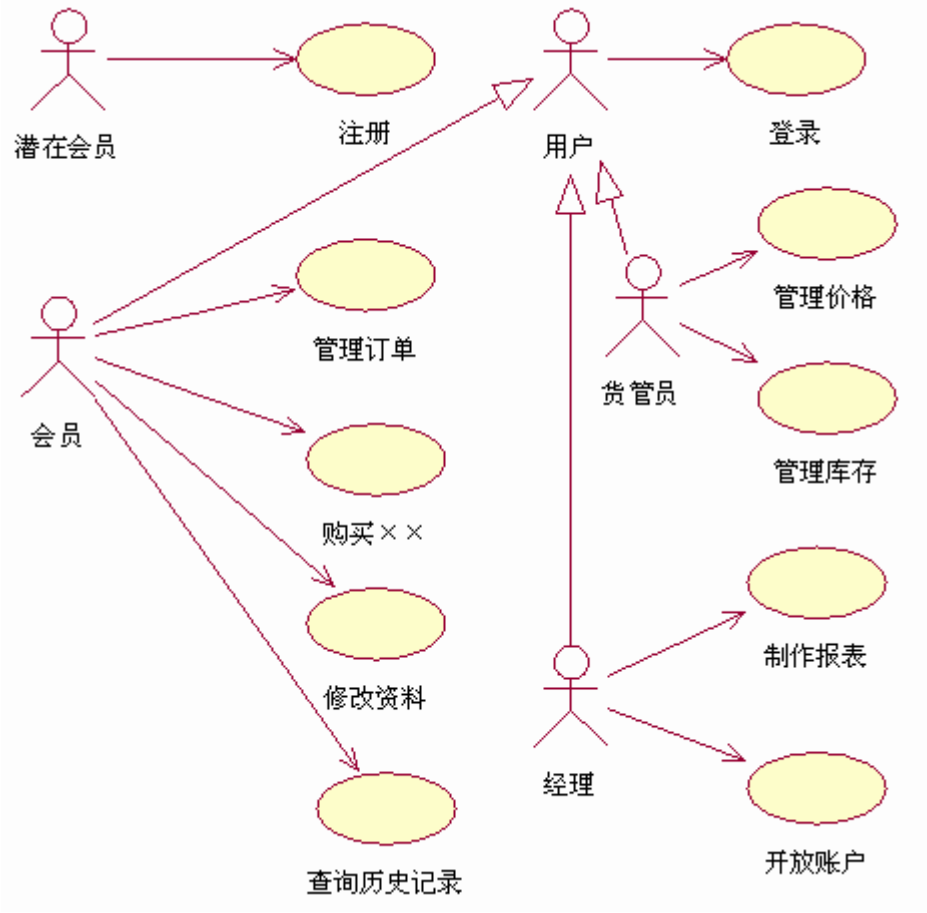


图 1 ××在线销售系统的用例图

好像还不错是不是？要是我们问一问客户，为什么要做一个系统？听一听客户——一家××代理商怎么说，“××的运费在顾客购买总费用中占相当比例，如果能想办法降低运费，顾客肯定非常乐意找我们买××，而降低运费的方法是想办法找出这样的供应商：1.顾客所要的××在它那里有足够库存；2. 由于地理位置的关系，它发货给顾客的运费低”【1】。这就是系统的愿景（Vision）。如果客户的愿景只是这一条，图 1 中的用例都是废的，这个软件的用例可能只有一个：

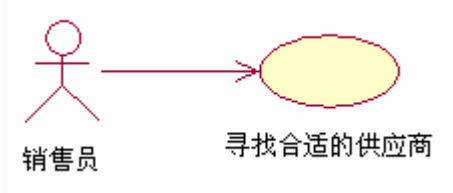


图 2 淘汰后的用例

再看下面这个简单的用例文档：

UC2：启动□□□□

执行者

操作员

前置条件

操作员已登录，□□□□处于停止状态

后置条件

□□□□已启动

涉众利益

操作员——希望操作简单

基本路径

1. 操作员请求启动□□□□
2. 系统向□□□□发送启动命令
3. 系统显示启动成功，更新界面

扩展

略…

以下是一段对话（C 代表顾问，D 代表开发人员）

C：这几步够了吗？

D（看了一会）：在 1 和 2 之间添加一步系统要求操作员确认比较好，象这样：

1. 操作员请求启动□□□□

2. 系统要求操作员确认

3. 操作员确认

4. 系统向□□□□发送启动命令

5. 系统显示启动成功，更新界面

C：为什么？操作员不是希望操作简单吗，为什么还要确认一次？这是为了谁？谁还关注这个用例？是不是漏了什么涉众？

D：为了“材料拥有者”

C：它的希望是什么？

D：安全第一，自己的材料不受损失。

C：好，把这个涉众加上：

涉众利益

操作员－－希望操作简单

材料拥有者－－希望安全第一，自己的材料不受损失

现在来看看，材料拥有者的希望在一定程度上和操作员是矛盾的，谁的更重要一些？

D：材料拥有者。

C：现在看看，我们还能不能做些什么来满足材料拥有者的利益？

D：好像没有了，这和操作员的责任心有关。

C：那么，系统还有没有办法在提高操作员的责任心上起些帮助呢，提醒确认是一种办法，还有没有…

D：系统应该记录下操作员的行为，这也是一种对操作员的约束：

1. 操作员请求启动□□□□
2. 系统要求操作员确认
3. 操作员确认
4. 系统向□□□□发送启动命令
5. 系统记录操作员信息、启动时间
6. 系统显示启动成功，更新界面

可能有人会想：这个太简单了吧，好，我们来看一个复杂一些的。

待续…..【2】

【1】文中大量出现××、□□，是为了隐去某些真实信息。

【2】“通往职业初段”本来应该写“书写用例文档”了，但 UMLChina 指定教材《有效用例模式》马上就要出版了，所以写一些重复的东西意义不大。很多东西论证已经没有难度，更多需要的是实实在在的实践。因此把一些经历写出来，供大家参考，可能更有价值。写几页东西不容易，刚写到一半，杂志就要出版了，只好待续….

软件管理与软件工程译丛

50%的中国软件公司老板从不关心自己的属下。

—CSDN 调查，《程序员》2003.8

Peopleware

人件

[第2版]

[美] 汤姆·迪马可 蒂姆·李斯特 著
Tom Demarco Tim Lister
UML China翻译组 译
方春旭 叶向群

清华大学出版社

集成 Rational 工具套件编写需求文档—Step by Step

Kirsten Denney 著，michael 译

查看评论

1、序言

1.1 目标

RequisitePro 通常是作为 Rational 统一过程（RUP）中大多数人将之与需求方面相关联的 Rational 软件工具。然而，当缺乏其他 Rational 工具的帮助时，它事实上是一个难于使用的、WORD 驱动的需求文档化过程。此外，在逻辑数据需求这个 RUP 需求工程还没有充分解决的方面，它还不是最好的工具。

这一白皮书说明了这些缺点可以通过集成使用 Rational 工具套件的不同组件来克服。它还说明了通过提供图形化的能力和对需求工件的适当管理，这些工具可以进一步增强需求工程。最后，这一白皮书也对 RUP 中提供的用来改进和扩张需求工程的工作流和最后交付提出了建议性的修改。

总而言之，这一白皮书的目标是描述利用 Rational 工具套件的全部好处，以及建立一个帮助确保为你的项目完成需求文档编制的一个全过程的需求文档编制方法。

1.2 范围

这一白皮书讨论了如何文档化需求信息；它并没有讨论如何搜集这些需求，也没有讨论在项目生命周期中如何管理它。关于如何搜集与管理需求的更多信息请参考 RUP 站点[1]。

1.3 假定

为了完成这一过程,下列工具必须正确安装，并且你必须具备如何使用它们的基本知识:

.MS WORD

.Rational Requisite Pro

.Rational ClearCase

[1] RUP—>核心工作流程—>需求—>活动概述：确定前景，获取涉众请求等。

.Rational Rose

.Rational SoDA for Word

.The Rational Untified Process website (为参考目的)



1.4 脚注说明

脚注中提供的说明用来帮助你浏览 RUP 站点和 RequisitePro。它们描述了从 RUP 站点左边的文件浏览器或从 RequisitePro2002 到参考文本的路径。如果你使用的是 RequisitePro2001，请用 Document->New 代替 File->New->Document。

2 好处

这一节概述了使用本文建议的方法来进行需求文档编制工作的好处。

2.1 捕获、维护和跟踪需求的易使用的接口

这一方法提倡使用 Rational RequisitePro(ReqPro)作为项目需要的不同类型的需求（如：涉众请求、特性等）的数据库存储池。ReqPro 提供了一个易使用的机制以方便输入和维护需求、它们的属性及其关系。

捕获需求间的关系的一个重要方面是从需求的一种类型到另一种类型的可跟踪性。可跟踪性对于系统范围的管理和确保所有系统元素已经被适当地处理是关键的。ReqPro 提供了一个非常容易使用的接口以跟踪需求类型。关于可跟踪性好处的更多信息请参考 RUP 站点。[2]

2.2 容易维护和发布最新交付的工件

大多数项目使用 Word 以一种可发布的格式来捕获需求，然后使用 ReqPro 的 New Requirement 或 Import 功能来导入到 ReqPro 中。他们通过 ReqPro 链接到 Word 文档的方式来加以维护。这一方法是笨拙的，当项目文档在管理控制之下时尤其困难。直接在 ReqPro 中输入和维护信息很容易，但是，这些信息又怎么发布出去呢？

[2] RUP—>核心工作流程—>需求—>概念—>可跟踪性。

SoDA 提供了在 Word 文档中植入报表能力，从而能自动从 RequisitePro(和其他 Rational 工具)中提取信息。例如：前景文档可以包含将要从 ReqPro 中提取的涉众、涉众请求和特性的 SoDA 标记。

2.3 强大的配置管理

配置管理工具的使用应该成为所有项目的必要条件，作为维护和跟踪项目工件的手段。Rational ClearCase 是市场上应用最广泛的 CM 产品之一，应该贯彻使用，以对你的项目的所有工件提供支持，包括需求工程中的必须元素。

2.4 以数据库为基础的需求的版本化

由于 ReqPro 所具有的数据库特征，这使得它自身不便加入配置管理软件之下。这一缺点可以通过 SoDA 生成的文档和 ClearCase 的结合来解决。SoDA 文档作为标准的项目工件，被添加到 ClearCase 存储池中，从而在他们被生成的那一刻提供了需求数据的一个历史快照。这一使用 ClearCase 将需求版本化和基线化的能力在协同工作环境下尤其重要。

2.5 在工件中包含用例模型图

在 RUP 中，来自于涉众的特性需求以用例的方式模型化。Rational Rose 为创建用例模型提供了一个容易使用的接口，这些模型然后可以通过使用 SoDA 标记拷贝到项目工件中。这些模型提供了系统用例的有用高级清单及其相互之间的关系。

2.6 集中访问完整的用例信息

一旦用例被模型化,他们需要被完整地描述(他们的流程和约束)并能够跟踪到 ReqPro 中（可跟踪性）。用例规格说明书就是描述用例的 WORD 文档。Rose 可以让你将这些文档附加到用例模型中，这样做之后，就隐含创建了一个从 Rose 到相应的 ReqPro 中用例需求的链接。从而，模型就为项目的分析和设计提供了一个方便的、集中的对全部用例文档的访问。

将文档附加到模型上同时创建了一个从 ReqPro 到用例规格说明书的直接链接。由于用例是 ReqPro 中唯一在其中维护但又没有完整描述的需求类型（关于这一点的原因在后面讨论需求类型时再解释），这一特征提供了一条简便而直接的途径来从 ReqPro 中访问用例完整信息。

SoDA 使得将模型和用例规格说明书直接提取到相应的项目文档成为可能（明确的说，就是软件需求规格说明书）。

2.7 包含全部的数据需求

RUP 在处理逻辑数据需求的收集方面显得不够，而这些需求对系统的设计师和分析师来说又是关键输入，这一方法帮助确保这些需求被适当地包含在需求文档中。

逻辑数据需求可以通过 Rose 中的类图（在逻辑视图中）或数据模型图加以描述，选择哪一个取决于你计划采用的分析和设计方式。每一个方法都允许你创建你的逻辑数据实体的高级视图，它为描述实体间关系的多重性和方向提供了一个理想的手段。他们也允许你以结构化的形式捕获数据的所有细节，因此他们可以作为项目的逻辑数据需求的单一存储池。

SoDA 标记用于提取模型和详细数据到相应的文档中（更具体地说，就是“数据需求”文档）。

信息昵称

信息昵称是 Alistair Cockburn[3] 用来描述一堆相关信息的术语，例如：“客户名称”可能就是一个昵称，用来描述来自客户实体的 First Name、Middle Initial 和 Last Name 的组合。客户地址和客户开单信息可能是另外一些昵称。通过使用昵称，而不是指定我们的用例中的实际数据字段，我们就能够将功能需求的设计任务和数据需求详细设计任务分开。这样做也避免了在用例中阐述而导致开发和维护实际逻辑数据需求复杂化的情形出现。

这一方法包含信息昵称的使用。

2.8 包含全部的业务规则

RUP 在强调收集完整的业务规则的重要性方面显得不够。收集业务规则的活动位于业务建模工作流程中。业务建模规程通常不被认为是一个核心工作流程，因此，它的某些方面经常没有引起应有的注意。然而，业务规则对那些主要输入为需求工件的设计师和分析师来说是关键的。

结果，虽然这一方法与 RUP 的在业务规则上必须完成的工件和活动方面没有区别，但它更强调将它作为系统需求文档编制工作的关键成分。

3 过程

这一节详述这一方法所需的实际步骤。除第一步外，它们都非常有可能在迭代中出现，甚至并行出现在迭代过程中。为简化起见，我们顺序地描述它们。

[3] Alistair Cockburn: “《书写有效的用例》” (Addison Wesley, 2001)

3.1 配置需求环境

该过程的第一步称为配置 ReqPro 并建立起你的项目模板，以供后面的步骤使用。这一步骤中的任务如下：

1、让你的 Rational 管理员配置 ReqPro 将项目文档以 Word 文档格式保存[4]。

这一步是必须的是因为通过与 SoDA 相关联创建的项目工件必须为 Word 格式。此外，从我们的观点出发，也认为使用标准 Word 文档所带来的工作简化超过了使用专用 ReqPro 所带来的好处。

2、配置需求管理计划（RMP）

RMP 将适用于这一项目的需求类型、属性和可跟踪性文档化。它同时也为 ReqPro 应该如何根据特定的项目加以配置提供了蓝图。

如果你还没有 RMP 模板，那么你可以使用 ReqPro 中提供的[5]。

在这个模板中，除了你的项目所需要的其他类型之外，还定义下列需求类型。这些需求类型的某些建议的属性可以在 RUP 前景文档和 ReqPro 默认模板中发现；不管怎样，详细的注释由这一方法所需的属性组成。此外，每一需求类型所需要的可跟踪性都被标注下来，并应加入到 RMP 中。

涉众

这一需求类型让我们可以记录和追踪所有需求的最终源头，并提供了源的信息，以便我们可以更好的满足需求。

包含一个“Actor”属性，值为 Yes/No，以说明涉众是否为 Actor(用户)。这一明确区分在前景文档中是必须的，并且当过滤根据其他元素生成的报告和过滤到其他元素的跟踪时是有用的（其他元素如用例）。

涉众请求

这一需求类型让我们跟踪涉众的不同需求。从而，这一类型的需求可以向后追溯到涉众请求类型。

[4] ReqPro—>File—>Project Administration—>Properties—>Documents tab—>不选择“Save documents in RequisitePro format”复选框。

[5] ReqPro—>File—>New—>Document—>选择“Requirements Management Plan document type”。

特性

特性是来源于涉众请求的系统的高级特性。

用来区分特性类型的属性将为以后在文档中描述这些信息变得方便。前景文档对此提供了一些指导。例如：你可能想要包含一个“类型”（Type）属性，它具有“功能”、“非功能”和“文档”值。或者你也许想要分解“非功能”为系统，标准和性能。或者你可以使用RUP的“FURPS”类型：Functionality（功能性），Usability（可用性），Reliability（可靠性），Performance（性能）和Supportability（可支持性）[6]。根据情况选择最合适的。

特性向后追踪到涉众请求，这样可以确保没有多余的特性，也帮助确保没有未处理的涉众请求。

用例

用例通过使用场景提供整个系统特性的上下文。

不象RUP，这一方法不推荐放置用例的详细说明，如前置和后置条件和事件流，将它们放入ReqPro将变得非常笨拙且没有多大好处[7]。因此，只有一个用例的清单——名字和简要描述，包含在ReqPro中，同时还伴随有其他标准属性，如优先级，状态等等。

用例向后追踪到派生出它们的特性和将要使用这一用例的用户或Actor（带有Actor属性的涉众）。

附加规格说明书

附加规格说明书是应用程序的详细的非功能需求，并来源于已被标识的应用程序的非功能特性。因此，它们也跟踪到特性。

你也许想要提供一个与特性的“类型”类似的属性，以便区分规格说明。这在以后进行文档化描述时也有用处。

[6] RUP—>Disciplines—>Requirements—>Concepts—>Requirements

[7] 这一说法可能不完全正确，取决于你如何设计和跟踪你的测试案例，在作出最后决策之前与你的测试设计师商量一下。如果你将用例的详细信息放入 ReqPro，你将需要使用 SoDA 来创建用例规格说明书，这将在后面描述。然而，这是一种较复杂的方法，你必须保证你的团队对此感到舒适。本文假定用例的详细信息没有包含在 ReqPro 中。

最后，所有的特性应该可以通过一个或多个附加需求和（或）一个或多个用例追踪到；这一检查将验证所有的系统特性已经被需求过程所处理。

你也可以选择为数据实体和（或）业务规则包含需求类型，特别是当用于处理用例到它们的跟踪时。本文档假定不存在这一情况。

3、将RMP检入ClearCase

所有的项目工件都应该常规性的检入ClearCase中，作为维护、跟踪和控制它们的手段。

4、让你的Rational管理员配置ReqPro以反映RMP中定义的需求管理计划

如果还没有定义标准的模板，那么管理员应该为项目使用复合或用例模型模板。

5、裁剪这一方法所需要的文档模板

裁剪下列描述的模板并确保他们在你的项目的开发人员工具包中被文档化[8]。

前景文档

ReqPro中提供了一个默认模板，然而，你还需要转换该模板为SoDA报表格式以便ReqPro中输入的信息可以被自动检索到报表中。

使用适当的SoDA标记替换RUP前景模板的下列段落（这些段落推荐使用表格形式）。

涉众——使用一个带有Actor=No的涉众摘要列表来代替。重命名这一段落为“涉众摘要列表——非用户”。这暗示了用户不是涉众，挑战了一个重要的RUP概念。

用户摘要——使用一个带有Actor=Yes的涉众摘要列表来代替。重命名这一段落为“涉众摘要列表——用户”；

涉众概要——用每一个Actor=No的涉众的详细信息代替。适当地重命名这一段落。

用户概要——用每一个Actor=Yes的涉众的详细信息代替。适当地重命名这一段落。

主要涉众或用户需求——用ReqPro中标识的涉众摘要列表代替

产品特性——用ReqPro中的功能类型特性代替；

[8] RUP—>工件—>环境—>开发案例

其他产品需求——用ReqPro中标识的非功能（或系统、性能、标准等）特性代替。根据需求结构化标题及按类型分组。

文档需求——用ReqPro中文档类型的特性代替。如果这一段落需要更详细化而不容易在ReqPro中捕捉，则适当地扩充这一段落。

你也可以选择包含一个段落或一个附录以捕获需求的可跟踪属性，而SoDA对此提供了标记。跟踪需求的动作以及能够使用可跟踪性来决定变更所带来的影响对每一个项目来说都是关键的；发布它们的需要也许更少一些。

用例规格说明书（UCS）

ReqPro中提供了一个默认模板，然而，我们推荐以用例的目标来代替描述。描述通常重复了UCS的事件流或标题，而目标提供了某些事物让我们清晰地测试用例中流程的成功。这帮助澄清了主要、可选和异常事件流之间的区别。

SoDA提供了自动化的选项将用例规格说明书合并到软件需求规格说明书中。我们也推荐这么做，除非你感到分别维护它们可以更好地管理用例的数量和复杂性。如果你在SRS中包含它们，就不要在单独的用例规格说明书中再单独包含标题页或目录了。

软件需求规格说明书（SRS）

在ReqPro中的标题“现代软件需求规格说明书”下（如果使用复合项目模板来创建项目）为SRS提供了一个模板；RUP也提供了一个模板。不管怎样，你将需要转换模板为SoDA报表格式以便输入到ReqPro中的用例和附加规格说明书将被自动检索到报表中。

下面是需要对这一文档作出的修改：

（1）、在用例模型纵览一节中添加一个标题为“用例模型图”的段落。使用SoDA标记来从Rose中提取用例模型。

（2）、在用例模型纵览之前添加“Actor”节。使用SoDA标记来提取ReqPro中Actor=Yes的涉众的名字和文本（描述）；

（3）、使用SoDA标记替换下列段落（以表格形式）

a、用例模型纵览（或者是用例层次，取决于模板）——用ReqPro中说明的用例的摘要列表代替；

b、用例报告或用例规格说明书——用用例规格说明书文档自身代替；

c、附加需求（或者是现代SRS格式中的适当节）——用ReqPro中的附加规格说明书代替。根据需求适当按类型分组和起副标题。

数据需求文档

在 ReqPro 或 RUP 中没有本文档的模板,然而,你可以使用其他的 RUP 文件作为开始点。包括下面的常规段落：标题页，目录表及介绍。本文档包括下面一些内容：

（1)、一个名为“逻辑数据模型”的段落（或“逻辑类图”，取决于你是使用 Data Modeler 还是逻辑视图中的类图)。在这一段落中，使用适当的 SoDA 标记从 Rose 中提取数据模型/逻辑类图。

（2)、一个名为“附加属性信息”的段落，包含位于模型、但没有显示在图中的实体的信息。这些属性使用 SoDA 标记从模型中提取出来。

（3)、一个名为“信息昵称”的段落。这一段落将映射用例中的信息昵称到逻辑数据模型中的实际实体和属性。使用表格的形式，包含以下列：

A、信息昵称；

B、映射属性隶属的逻辑实体名；

C、对应于信息昵称的属性名称。

这一段落是完全手工的，无须 SoDA 标记。

（4)、查找表段落。 查找表指出应用程序中使用的静态列表的有效值。(它们也许并不是真正静态，可以通过管理员功能或数据库接口更新)。如果使用了存在的数据库表或输入文件，指出它的名称和文件的位置而不是它的内容。

在许多项目中，部分甚至所有数据也许是从旧有系统或口头方式或其他系统中获得。在此情况下，就无须进一步分析 ReqPro 规程中的数据。取而代之，该信息的位置（例如：范例文件的位置或服务器上的数据存储）就足够了，可以将它纳入 SRS 中，数据库设计员就可以从其中获得它。

业务规则文档

RUP 中有本文档的模板[9]。取决于业务规则的数量和（或）复杂性，你也许会选择直接合并业务规则到 SRS 中。这一方法假定使用单独的业务规则文档。

6、将模板检入ClearCase中。

3.2 编制前景文档

现在，我们已经配置好了用来捕获需求信息的工具和模板。下面的主题描述如何使用这些工具和模板来捕获描述项目前景文档所需要的信息。

在 RUP 中，前景文档定义了涉众关于将要开发的产品的观点。它典型标识了涉众、涉众的需求及派生自这些需求的高级系统特性。前景文档在项目概念阶段的早期是关键工件，因为它可以让涉众验证他们的观点已经被需求收集过程所捕获，并且它给了项目团队开始计算项目计划和其他项目工件所需要的信息。

下面是完成这一步骤所需要执行的任务：

1、将需求信息加入 ReqPro

当你提取涉众信息以及标识出特性后，在 ReqPro 中捕获以下内容，使用每个需求类型的 ReqPro 属性作为指南来确保完整的描述：

（1）、涉众；

（2）、涉众请求；

（3）、特性。

2、建立可跟踪性

在 ReqPro 中，指出下列可跟踪性

（1）、涉众请求到涉众；

（2）、特性到涉众请求。

[9] RUP—>工件—>业务建模工件集—>业务规则

3、创建前景文档

完成前景文档中不自动从 ReqPro 中生成的段落（如介绍、定位、产品概览等）。完成后，生成 SoDA 报表并在结果报表中更新目录表，这将提供一个即时的前景文档。

4、将前景文档加入 ClearCase

将 SoDA 生成的前景文档的版本加入 ClearCase 库中，它将产生一个原始需求的稳定和基线化的版本。

附加规格说明书、业务规则和数据需求在这一过程中很可能不被转换，他们将在随后讨论的步骤中解决。

3.3 编制功能需求

现在,系统的特性已经在 ReqPro 和前景文档中标识出来了。接下来的挑战就是创建派生自它们的用例了。

用例以角色和系统之间的交互系列的方式描述系统表现单元。每一个用例对操作员都提供了一个可观察的有价值的结果。例如：可能存在名为“添加客户”、“添加订单”或“运输订单”的用例。用例为系统的单一特性提供了一个上下文，因而对项目复审员，分析师和设计师来说显得特别有帮助。

1、对用例建模

使用 Rose 对上一步中捕获的需求信息进行建模。关于用例建模的目的和方法参见 RUP 站点[11]。

2、创建用例规格说明书

在Rose中，在模型中的要进行详细描述的用例上右击，选择“Use Case Document—>New”，这将打开ReqPro（如果还没有打开的话），并在其中新建一个用例规格说明书文档[11]，你可以在这里详细描述用例。RUP在详细描述用例的过程方面提供了指南[12]。

当你使用ReqPro中的保存需求文档特性保存用例规格说明书后，用例的名称将自动作为用例需求添加到ReqPro中。这样，通过在ReqPro中保存每一个文档，你就将在ReqPro中创建用例清单。这一清单可用来跟踪用例到产生它们的特性上。这一功能也为用例产生了一个从Rose到ReqPro需求的直接链接。

[10] RUP—>核心工作流程—>需求—>指南概述：其中有许多相关的有用的文档资料。此外，模式的使用也会有帮助，尤其是对于大型或管理规范的项目。查看RUP—>核心工作流程—>业务建模—>概念—>业务模式。

[11] RUP—>工具向导—>AnalystStudio—>使用 Rational Rose和RequisitePro管理用例z中将提供Rose与ReqPro协作的更多信息。

[12] RUP—>核心工作流程—>需求—>指南概述：其中有许多相关的有用的文档。

当详细描述用例时，存在一些惯例[13]：

（1）、使用粗体指示被用例调用的用例的名称，这将突出用例之间的关系。

（2）、使用斜体指示被用例调用的业务规则的名称。这让你书写简单的句子，如“显示全部已审批交易”。斜体指示存在一个业务规则描述审批交易由什么组成而无须每次都要进行解释。至于规则的详细细节用户则可以参考业务规则文档。

（3）、避免在用例中包含数据的详细细节，因为数据需求是很容易受影响而改变的。取而代之，使用信息昵称映射数据需求文档中的实际逻辑数据实体（你也许想要甚至为单独的属性也创建昵称从而使得他们可以改变名称或实体而不影响调用他们的用例。）

（4）如果对说明用例有帮助的话，使用活动图模型。[14]

如果你强调使用这些惯例，那么在最高级别的文档中解释它们（或者是SRS或者是单独的用例规格说明书）。

在上面的步骤中，非功能需求在这一步骤中还没有涉及到。他们将在下面的步骤中描述。

3、在 ReqPro 中更新用例属性

由于当创建每个用例规格说明书的时候，用例需求就自动被添加到了ReqPro中，你需要在ReqPro中直接更新相应的属性。

4、建立可跟踪性

在ReqPro中说明下列可跟踪性：

（1）、从用例到特性；

（2）、从用例到角色。

要完成这一任务，你可能想要使用ReqPro中的查询功能来显示属性为角色的涉众。

[13] 你可以通过添加Word格式样式以帮助惯例的一致使用。

[14] RUP—>核心工作流程—>业务建模—>指南概述:用例模型中的活动图。

5、将用例规格说明书加入到 ClearCase 中。

现在,系统的功能需求已经在 Rose 中图形化了,在 ReqPro 中形成了清单并且在用例规格说明书中进行了详细说明。用例规格说明书和需求可以从 Rose 中或 ReqPro 中访问。

现在你可以生成 SRS 了,作为对 SoDA 模板和标记的检查,然而,为简单起见,我们将在下一步所有需要的信息已经全部包含在 SRS 中后再讨论这一点。

3.4 编制非功能需求

附加规格说明书是系统的详细的非功能需求,包括诸如性能需求、系统约束和法律限制等。它们描述了项目中已经识别出的非功能特性的详细细节。

附加规格说明书像特性一样相当简单明了的,它们被直接录入到 ReqPro 中。然而,附加规格说明书使 SRS 变得完整,因而在这一步后就可以生成 SRS 文档了。

1、将附加规格说明书加入到 ReqPro 中。

2、跟踪附加规格说明书到派生出它们的特性。

3、生成 SRS。

现在,系统中的所有的角色、用例和附加规格说明书已经进入到 ReqPro 中;确保文档中的需要的所有其他非 ReqPro 信息已经被更新了,然后生成软件需求规格说明书的 SoDA 报表并更新目录表。

如果 SRS 象开始描述的那样定义,SoDA 就将从 Rose 中提取用例模型,从 ReqPro 中提取角色和用例清单,以及用例规格说明书。

4、将 SRS 加入 Clear Case 中。

3.5 编制数据需求

数据需求通常并不在需求收集和详细描述过程中进行。因此,这一步骤很可能与开始的那些步骤并行进行。然而,良好的数据需求细节仍然需要在一旦用例被完整描述后就确定下来。

数据需求包含系统中的逻辑和概念实体的名字,如客户、订单、项;还包括实体的详细信息(属性、类型、长度、校验规则等)和实体间的关系(多重性和方向)。目标是以形式化的方法捕获这些信息,这些信息然后将用于分析和设计活动中。

1、创建逻辑数据模型或类图

Rational Rose 的 Data Modeler 和逻辑视图都具有完整的建模能力，可以描述你的系统中的逻辑数据实体。理论上，你应该在 Data Modeler 中创建逻辑数据图而数据库设计师将使用这个图来创建物理数据图。设计师将使用物理模型在逻辑视图中创建类图。然而，根据项目的大小和角色，这些步骤可能会互相部分或全部重叠。例如：你可能会只有一个数据模型和逻辑类图，或者甚至只有一个类图，使用最适合你的项目需要的工具。

一旦你决定了要采用的工具，描述项目需要的数据。使用模型或图描述实体间的多重性及关系的类型；使用规格说明书弹出菜单说明属性的约束（如类型、域、默认值等）。当描述约束时，推荐你为公共约束（如电话号码格式）使用域；为描述其它的东西使用标准语言（如 OCL）。

2、在数据需求文档的信息昵称段落，映射用例中已经识别出的信息昵称到相应的逻辑实体和属性。

3、生成数据需求文档

生成数据需求 SoDA 报表并更新目录表。

如果这一文档已经象开始描述的那样定义了，它将从 Rose 中提取逻辑数据模型及其他属性信息，还包括你在上一步中完成的信息昵称的映射。

4、将数据需求文档检入 ClearCase。

3.6 编制业务规则

业务规则声明应用程序相关的政策、计算规则和逻辑。一些常见例子如状态的确定和费用的计算。业务规则通常没有在上面步骤的并行过程中覆盖。

由于业务规则经常包含伪码，这使得它不能很好地放入 ReqPro 中，业务规则在业务规则文档中记录。然而，为实现可跟踪性，你可能希望将它们在 ReqPro 中编制一份清单。

1、直接在业务规则文档中记录业务规则

当描述由业务规则封装的逻辑时，考虑使用伪码、UML 对象约束语言（OCL）[15] 和（或）流程图，以便提供所需要实现逻辑的简练描述。

[15] RUP—>核心工作流程—>业务建模—>指南概述—>业务规则。

2、将业务规则文档检入 ClearCase

完成这一步后，你就将有一个完整的和完全可跟踪的需求的集合，可以被你的项目团队及评审员很容易地维护和发布。

4、其他考虑

ClearQuest

ClearQuest 是 Rational 工具套件中的变更请求组件。项目中已经处在开发中或已经部署的需求可以通过 ClearQuest bug 报告生成。这样，ClearQuest 就可以被明确地与 ReqPro 进行集成以确保需求可以容易地移植到 ReqPro 中，从而保证 ReqPro 能够持续提供项目需求的单一、统一的库。关于如何集成的详细信息，参看 RUP 站点。[16]

词汇表

词汇表在 RUP 中是一个需求工件。在 RUP 站点中有一个词汇表模板[17]。不管怎样，我们推荐使用表格形式，因为它更容易阅读和排序。不推荐你将词汇表存储在 ReqPro 中，除非你想要通过 SoDA 报表形式发布它们。除此以外没有其他什么可以想得到的好处。

风险列表

风险列表不被认为是 RUP 需求规程中的一部分，但是它对那些需求工程中涉及的过程以及归因于它的活动是必须的。由于需求活动在项目的概念阶段是最重要的活动，它们提供了唯一的一个机会在项目的最早阶段识别和管理风险。

RUP 中提供有风险列表的模板；然而，我们推荐使用 EXCEL 格式，因为它使用起来更简单；更重要的是这么做可以按照风险的量级排序。另一方法是使用 ReqPro 跟踪风险，因为这将使你能跟踪风险到它们所影响的用例和附加规格说明书。

[16] RUP—>工具向导—>AnalystStudio—>使用 Rational ClearQuest 和 RequisitePro 管理涉众请求。

[17] RUP—>工件—>需求工件集—>词汇表

5、总结

这篇文章令人充满希望地完成了下列内容：

（1）、为帮助需求文档的编制，对如何建立和使用一个集成和有效的 Rational 工具套件环境提供了必要指南。

（2）、为扩展标准 RUP 工件以确保一个完整的需求信息集合被传递到你的设计师和分析师手中提供了指南。

象开始陈述的那样，存在许多不同的方法解决需求文档化问题。本文介绍的方法将非常完美地为你的项目提供扎实的起点。

征稿

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

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

中译本即将上市

样章先行提供>>

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

第一本 UMLChina 训练指定教材

清华大学出版社

用户界面设计背后的理论

Mauro Marinilli 著，Sha Meng 译

查看评论

第一部分

导言

设计专业的用户界面不仅仅是有好的图形艺术家和好点子的问题。不幸的是，人们只是为了产品而创建用户界面，完全没有意识到背后的基本原理和理论。紧迫的计划、错误的观念（像“可用性是现在我们不能提供的额外的东西”）和缺乏专业性造成我们周围充满了低劣产品。本文开始可能会有一点抽象，但它的主要意图是让大家了解一些仅限于学术界和专业领域的主题。我们不去探讨诸多 UI 设计方法的细节，因为从我个人的经验，我发现为建立有效的用户界面“所做的事情”往往更吸引人们，而不是掌握一些简单的概念，在 UI 设计的过程中受到启发。

这一篇文章的主要阐述对象是开发人员或是图形设计人员，他们的目的是创建高质量的用户界面（UI）。设计用户界面看起来是整个应用程序中一个简单和不重要的方面；实际上它是，也许是，整个系统中最重要的部分。尽管我们关注于软件系统，但是在这里所做的很多考虑都需要应用于人类产品，目的在于和其他的人交互。这常常是被忽视的部分。主动、流畅的人机交互是好的设计，人们创造产品，被其它的人们使用。这是一个复杂的问题，和典型的工程任务有很大的不同，因为人们（拥有他们各自的特质，感情等等）要参与到整个设计的过程中（从设计者和开发者到最终用户，间接用户等等）。

与系统交互

让我们从基本概念开始，看看机器和人如何进行交互。

一个最简单的方法是建一个交互系统的模型来描述用户在面对系统的任务时完成动作的阶段。我们可以粗略地将普通的交互系统中典型的用户交互标识为七个步骤：形成目标和意图，具体化并执行动作，观察并解释系统状态，最后还有按语义评价交互结果（见图 1）。

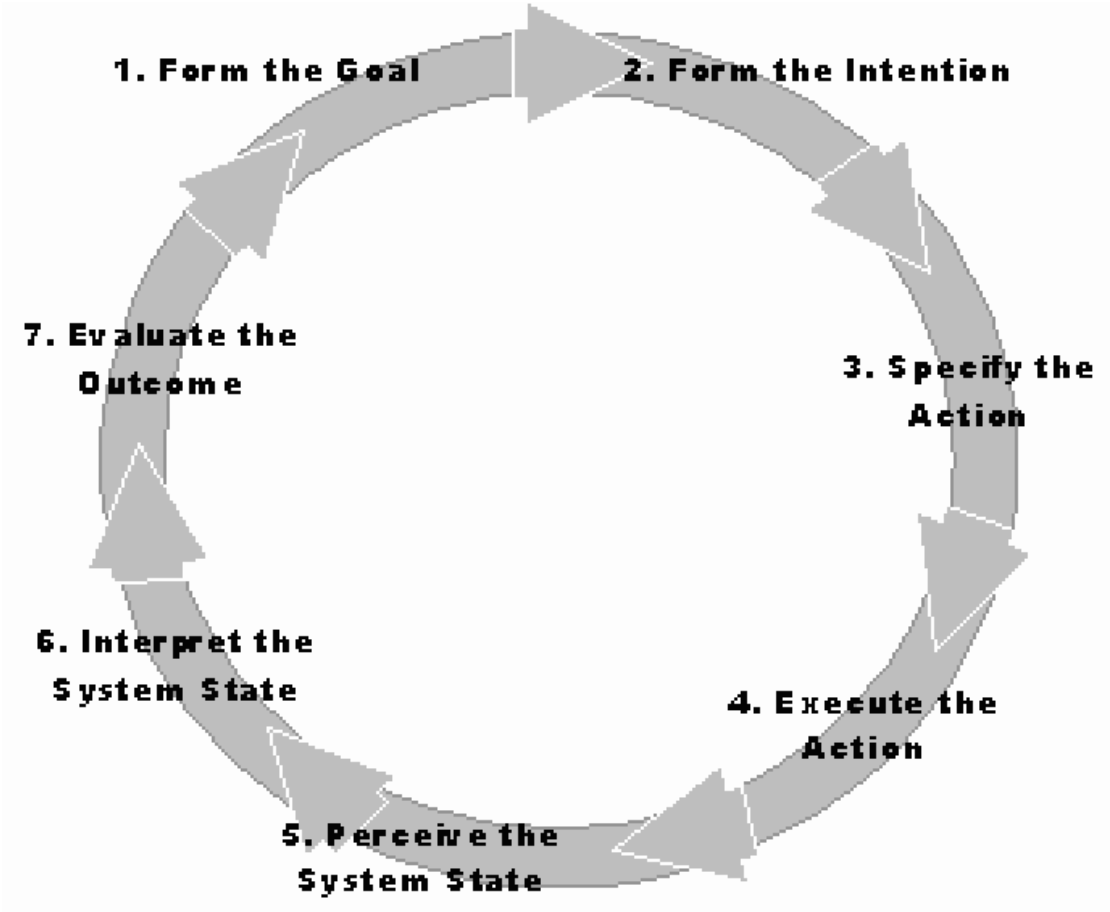


图 1 交互的七个阶段

首先，用户从她/他的目标中形成一个概念性的意图（比如，一用户想要访问一特定的项目，所在的仓库可以通过图 2 中网站来访问）。其次，她/他试着把这个意图转化成系统能够提供的命令（在我们繁琐的例子里，在开始浏览网页时需要弄清楚如何实现这个意图）并根据这些（用户观察的）命令采取行动（比如，用户试图在检索文本区域敲入信息，然后点击旁边的“检索”按钮）。接下来，用户试图理解她/他动作产生的结果（在我们的例子中，通过检查在按下“检索”键所获得的页面）。对于计算机系统而言，内部工作隐藏在哪里，用户必须从很少的提示中弄清内部状态，这一点尤为重要。最后的三个阶段是帮助用户发展她/他的思维。整个处理过程是动作与评价的循环。用户通过解释她/他动作产生的结果提炼她/他记忆中的系统模型。

当然，我们的讨论是太过简单和直接了。显然，人机交互设计背后的问题不可能如此简单和直接。但是，我们的目标是要从不同的视点来提供这些重要问题的导引。有兴趣的读者可以通过阅读专门文献更深入地了解这些主题。在文章的最后给出了一些切入点。

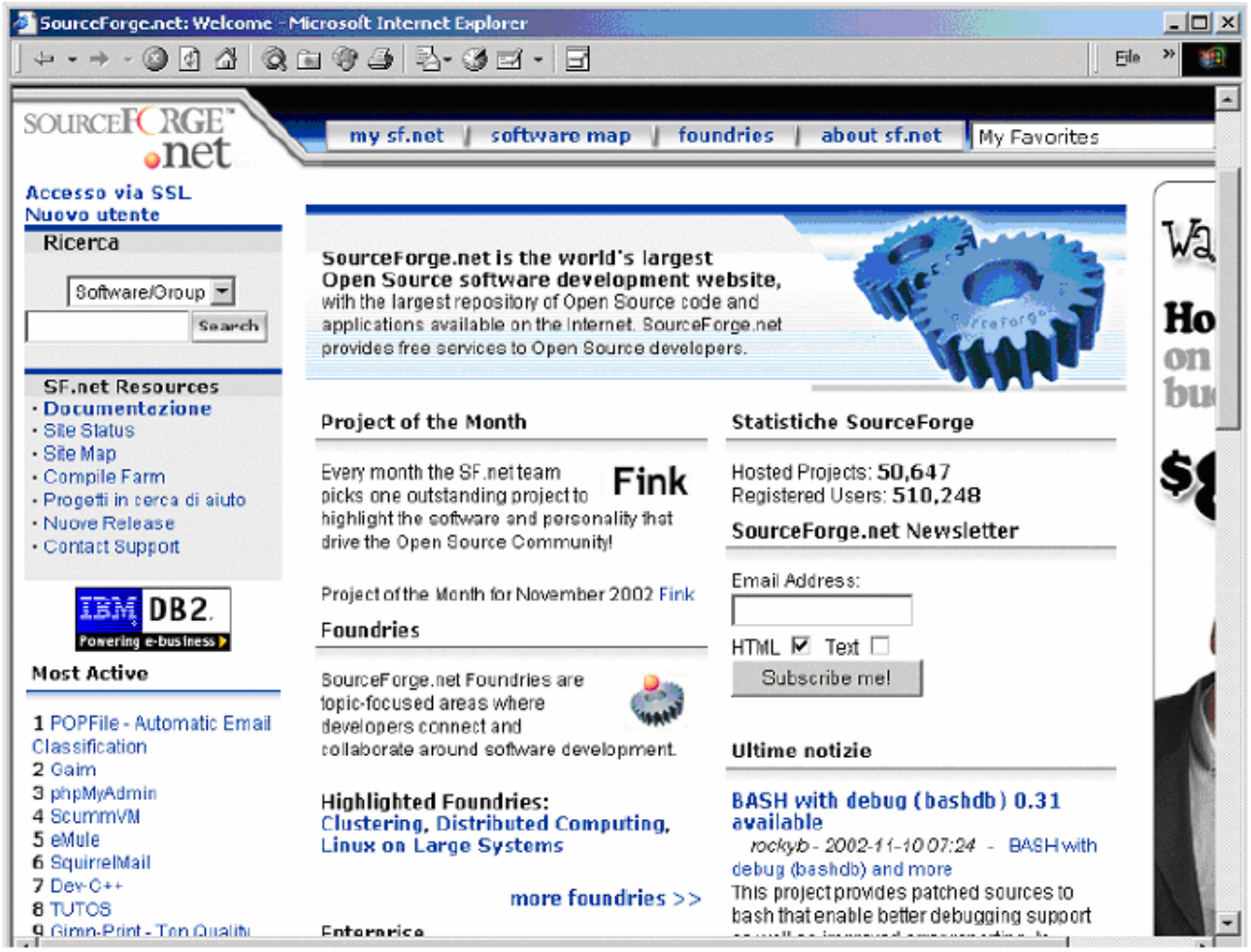


图 2 一个 Web 上的 UI 例子

用户不是设计者，设计者也不是用户

作为人，当我们和周围的世界打交道时，可以依赖的仅仅是我们现在的和过去的经验；我们也需要语义模型。比如，我们需要给我们周围的事物以定义。这就是我们为什么常常听到人们所谈论到关于电脑的经验：他们熟悉文件，鼠标动作等等的概念。但是这不意味着最终用户能够理解设计者的想法。很可能对设计团队来说简单的应用程序，对最终用户而言就是糟糕的，难以掌握的。经常有一些甚至开发人员都不能掌握的调试程序，这些程序的内部模型是隐藏的。因为最终用户需要在给予很少的，人工的提示的情况下搞清楚软件是如何工作的，这些提示必须经可能的一致；那些基本的思想，可见的项目和他们的交互，名称以及其他所有的内容都应该在设计的时候搞清楚。

当设计一个 UI 时，设计者应关注于最终用户的需要。然而，设计者常常太过于引用其他一些不错的，获奖的产品而导致开发者难以实现，最终用户产生迷惑。但是当 UI 通过开发者来设计时（经常发生在小公司，原因是没钱），场景很可能是漏洞百出的：一个由开发刚刚转成设计的人会按照他旧的程序员思维作出不实用的界面。这只是因为开发者远远不够了解好的 UI 是怎样的。然而，大公司与其他组织的专业设计团队能够制定他们的设计指导方针，传播他们，并最终把方法应用到普通软件中去。

更多的概念

这一节里，我们简要地介绍一些其他来自 HCI（人机交互）领域的有趣概念。

用户模型与系统模型之间的不匹配，常用所谓的裂口来比喻。

- 执行的裂口是指用户的意图和允许的动作之间的不匹配（比如图 3 所示，在著名的用户网站里，新手用户想要访问她/他以前想要的书单，但是从页面上不能直接获得这个专题。
- 评价的裂口是指用户的预想和系统的表达之间的差异（比如，用户点击“Gold Box”图标，把它和想要的书单的图标混淆了）。

反应时间是一个重要参数，因为慢的反应是在使用程序时造成错误和用户受挫的原因。这对执行时间可能是严重瓶颈的 Web 的应用程序来说尤为正确。而且，反应时间从不同的方面影响用户。预想和过去的经验扮演了重要的角色。如果有人习惯任务在给定的时间段里完成，过长或过短的完成时间就会使用户产生混淆。此外，个人的态度也应该考虑进来。短的反应时间同样有助于更容易地开发 UI，这一行为在任何地方都得到鼓励的（用撤销动作降低错误代价等手段）

短期记忆（STM）是指充当变化数据缓冲区的有限记忆，并且被用作处理感知输入。经验研究发现，人类通常拥有记住 5 到 7 件事情能力的 STM。这些事情可以是单独的对象或是连贯的信息集。可被存储在 STM 中的非原子的信息块的大小依赖于主体的熟悉程度，但是通常信息保持不会长过 15-30 秒。你自己可以试一试：记住其中不同的颜色很容易，但是要记住七种随机挑选的西班牙文单词就不容易了（只要你不太熟悉那门语言的话）。STM 是非常易变的。注意力分散，外部的噪音或者别的任务会很快扰乱它的内容。假设你在使用图 3 所示的网站时发现了一本有趣的新书而且作者从来没听说过，然后你立刻离开这个站点并关闭它。即使在你五分钟之内回来，你也会难以记住确切的书名。STM 一般用来保持语音界面里的状态：当你通过声音或按键回答语音界面，选择菜单和选项时，你需要记住操作的上下文（你在菜单和选项链表中的“位置”）。

图 3 一个普通用户界面中系统-用户不匹配的例子

另一种记忆是所谓的长期记忆（LTM），比短期记忆更稳定且更有能力，但想获得比较慢。LTM 的主要问题是重新获得方面的困难。我们都使用记忆法帮助获得 LTM，像记住个人密码的思维关联等等。

STM 也可提高操作效率。仅用 STM 处理的操作可以比那些需要 LTM 或者外部识别帮助的操作更简单更迅速地解决。而复杂的操作需要在整个过程中维护数据的上下文，使用 STM 只会使得情况变得更糟。

STM 是良好设计界面中有价值的帮助。STM 需要专注，而且通常人们为了获得最好的表现而处在恰当的环境里。他们应该觉得应用程序容易使用，有一个如何工作的可预测的想法，不害怕造成灾难的误操作，不感到受系统的强制等等。我们当然不能干涉使用程序的最终的物理环境，但是我们可以将其考虑到我们的设计中。

设计者应当始终尝试设计让用户尽可能地利用他们的 STM 来工作的用户界面；按这一方法，他们的记忆量就会更少并且交互起来更加迅速，更加无错。Unix 命令行界面需要长期访问 LTM 或一些外部的识别辅助。对于 Unix 新手来说，要用文字笔记记住命令和语法，甚至用命令序列完成某一项任务都是难以想象的。随着图形用户界面的到来，这一状况会有所改变。现在，设计者有一套功能强大的设计表现工具，更容易用于用户界面。另一种避免用户白白花费记忆的方法是采用标准设计。用这种方法，用户可以使用从其他标准 UI 处获得的知识。

控制和自动化是又一个重要的用户界面设计问题。能够提供一些特色的自动化是很实用的。但是，这样用户将会失去控制。如果用户感到无法全面控制他们所作的工作，他们将感到受挫和紧张。提供给最终用户某种程度上的控制是很重要的。

相反，根据 UI 的定义，UI 应该提供一个高级的，易于使用的服务与数据视图，隐藏像 CPU 内部寄存器或硬盘表面的低级物理状态之类的无意义细节。成功的 UI 设计的一个重要因素是平衡自动化与用户控制，显示有意义的细节，隐藏剩余的细节，按照个别用户的要求作出适当的调整。甚至是同一个用户，当她/他从应用程序中获得信心后，可能会跳过自动化的特色而全面取得控制。UI 中能够获得不同级别的控制是很有用的。这能有助于把所提供的自动化层次（如定义宏，为常用操作提供向导等等）加到设计中去。总之，一般说来，计算机程序是一种内在的有限产品，因为它不可能考虑到所有的可能情况，而仅仅是有限的，在高级的合并集中想清楚的。

所以，平衡人的控制和自动化是 UI 设计的典型平衡。一方面，提供完全自动化的 UI 可能太过冒险，特别是当任务很重要的时候（如管理化工厂），因为很多独立的变量可能导致看不见的行为。另一方面，允许用户紧紧掌控也是很危险的。他们可能修改一些敏感的数据或用意想不到的方式使用它。

一般性原理

有一些原理需要在设计用户界面的时候牢记，记住下面的几条：

- 了解你的用户　可能这是用户界面指导中引用得最多的一条。然而，有时很难对你的用户群体做出假设。图 2 中的界面主要集中在电脑工程师和程序员，而图 3 中的界面则是给更多范围里的客户使用的（可以从颜色、术语中看到这一点）。

● 最小化用户的工作量　这意味着减少记忆量和识别量（前面讨论过的），提供信息反馈，记忆帮助和其他识别支持。保证一个工作进程被打断几分钟而不造成当前工作进度的丢失（人们只能在有限的时间内集中注意力）。当那些网页过期并且信息无法保存的地方不可能后退到刚才的地址时，应当在网站设计中考虑到。

● 保持一致性　有许多一致性的方面需要在界面设计时保持：标签，术语，图形习惯，组件，分布等等。许多指导，原理甚至是软件系统设计方法都面向一致性原则。例如，仔细查看图 1，你会发现某些语言的不一致（一个不完整的站点本土化；在这里意大利语和英语是混在一起的）。

● 保证总体的灵活度，错误恢复和定制　在与人交流的时候灵活度是必要的。人类是会犯错误的；提供一个恢复机制，允许用户去探索这个 UI，减轻他们因为一个不可挽回的错误而受陷所造成的焦虑。而且，界面应该可以被用户定制。对特定人群来说（比如残疾人），这可能是使用该应用软件唯一合适的方式。灵活性也包括为不同类型的用户提供不同的可用机制。新手可以用向导或者其他简单但是时间长的、容易交互的方式，而专家用户可以利用全部包含在 UI 中的某些形式的快捷方式。一般地，这可以通过两种单独的交互路径来完成；一个给有经验的用户，简单的功能集给没有经验的用户。

● 遵循标准　有许多交互，缩写，术语等方面的标准和指导。标准对于交叉应用的一致性和有效的实现来说是必要的。它们在减少设计工作的同时确保了质量。

● 让系统的内部状态可见　我们在上面已经讨论过这个重要的原则。例如，在敏感的数据被直接操作，哪怕是被有经验的用户操作的时候提供警告信息。Amazon 网站（参考图 3）在当前没有用户登陆的时候，通过文本显示“你好，请登陆以便得到个人建议”就是此种情况。

结论

我们在文中简要地讨论了用户界面设计背后的一些问题和基本概念。我们看到，UI 设计可以通过一些标准，如消除 UI 中可能存在的分散注意力的事务，提供用户反馈，避免错误或使之易于处理或恢复（提出一个探索性的用户界面交互模型）等来组织。

我们还看到，在用户界面设计中，基本的概念模型扮演了一个很重要的角色。一般说来，人们是按照现实的概念，有意义的表达来行动的。这些表达来自他们现在的和过去的经验。因此，同一个应用从那些设计它的人们（UI 设计者），实现它的人们（开发者）和使用它的人们（最终用户）看来有不同的思维模型。对设计者来说，重要的是要明白在用户界面的创建和后续使用中不同思维模型的表达关系。

我们所提到的这些原理和其他原理是用户满意度、低错误率和高效任务表现的基础，都是启发 UI 设计的指导和标准。

参考文献

在这一节列出了一些关于人机交互，可用性和用户界面设计内容的来源。这些主题的书籍和资源是难以计数的。我只选了很少的一部分，一般题目以简洁为主。书的评价只反映我个人的观点。

Mullet, Kevin; and Sano Darrel. 1995. Designing Visual Interfaces. Communication Oriented Techniques. Englewood Cliffs, New Jersey: Prentice Hall.

一本视觉设计方面灵活的书，重点在视觉设计和功能设计方面。

Nielsen, Jakob. 1993. Usability Engineering. San Diego: California Academic Press.

一本（过时）可用性论文的合集，仍是可用性当面的参考书。

Norman, Donald A. 1993. Things That Makes Us Smart. Defending Human Attributes in the Age of the Machine. Cambridge, Massachusetts: Perseus Books.

不像“日常用品的设计”一书那般明白，但确实值得一读。

Norman, Donald A. 1990. The Design of Everyday Things. New York: Doubleday.

经典，有趣，非专业人士可读。

Shneiderman, Ben. 1998. Designing the User Interface, Third Ed. Reading, Massachusetts: Addison Wesley.

如果你需要一本（入门型或者非入门型）关于用户界面设计的书，就看这本吧。

Preece, Jenny. 1994. Human Computer Interaction. Reading, Massachusetts: Addison Wesley.

这个领域的手册书，来自学术的观点（但有一点老了）。

Tufte, Edward R. 1990. Envisioning Information. Cheshire, Connecticut: Graphic Press.

和其他三本经典的 Tufte 教授的书一样有趣的信息可视化方面的书。

Tufte, Edward R. 1997. Visual Explanations. Cheshire, Connecticut: Graphic Press.

可能比不上那三本，不过值得研究（不仅仅是“阅读”）。

Tufte, Edward R. 2001. The Visual Display of Quantitative Information. Second Ed. Cheshire, Connecticut: Graphic Press.

Tufte 的第一本也是三本书中最好的一本,：“视觉”领域无敌的经典

Various Authors. 2001. Java Look And Feel Design Guidelines, Second Ed. Reading, Massachusetts: Addison Wesley.

看看并感觉一下 Java 平台。此类文档中制作较好的一本。

提供 Web 可用性和 GUI 设计的建议

http://www.usabilityfirst.com

提供可用性和 GUI 设计的建议

http://www.gui-designers.co.uk

GUI 设计的实范例

http://www.tworivers.com

一般且实用的 GUI 设计方面的讨论

http://www.asktog.com

Tognazzini 网站

http://www.acm.org/sigchi/

CHI 协会进展情况摘要和其它学术研究材料

http://www.useit.com

Nielsen 网站和一些有用的文章

http://www.iarchitect.com

“interface hall of shame”的主页和其它有用的材料

http://www.acm.org/~perlman/readings.html

第二部分

导言

在第二篇文章里，我们要简要地讨论一些图形用户界面中更有趣的方面。我们的方法是开放式的和发散性的，而不是过分讲究技术和细节。我们主要关注图形用户界面（GUI），对实际用户界面设计感兴趣的地方。

作为技术革命的成果，现代 GUI 设计者为设计和制造它们产品不言自明地依赖于形式指导和大量的工具。这里要讨论的主题经常被这样的工具或指导直接引用，而设计者所关心的是更具体的细节性的设计。所以，讨论这些隐藏在当前指导中的基本概念是非常重要的。

一个小的术语解释。我们在这里把术语 GUI 当作任意一种利用图形特性的用户界面。这与该术语在学术上的意义相比是有差别的。同时，我们想要传达一般的，实用的建议而不是详细的、具体的信息（有兴趣的读者可以在大量与该主题相关的文献中找到）。

显示的组织

让我们从用于组织屏幕信息的基本概念入手。

主要有两种策略来处理显示区域的层次分布：高密度策略（用于传递大量的信息）和与之对应的，我们称为有限信息分布策略（目标是减少被显示的数据量）。这些策略是互补的而且应当在每一个我们要设计的 GUI 屏幕里一起使用，或者按比例使用。取决于实际情况，两者之一将会占主要地位，但是有必要精确地混合使用它们。信息过量和含义隐讳的界面都是不好用的。在图 4 里，有一个使用高密度策略占主导的设计例子。这样的网页主要是为那些经常使用的专门用户设计的。

在图 4 里，SETI@HOME 项目的网页（http://setiathome.ssl.berkeley.edu/unix.html）展示了运用高密度组织的方法。该网页背后的意思是要在一个屏幕里立刻展示人们感兴趣的全部信息。这可能对新用户来说有一点迷茫，但是对专家和老用户来说就来得方便些。而且，根据现有状况（当前的应用领域，要完成的任务和可能的最终用户群），这也许是唯一可行的方式。

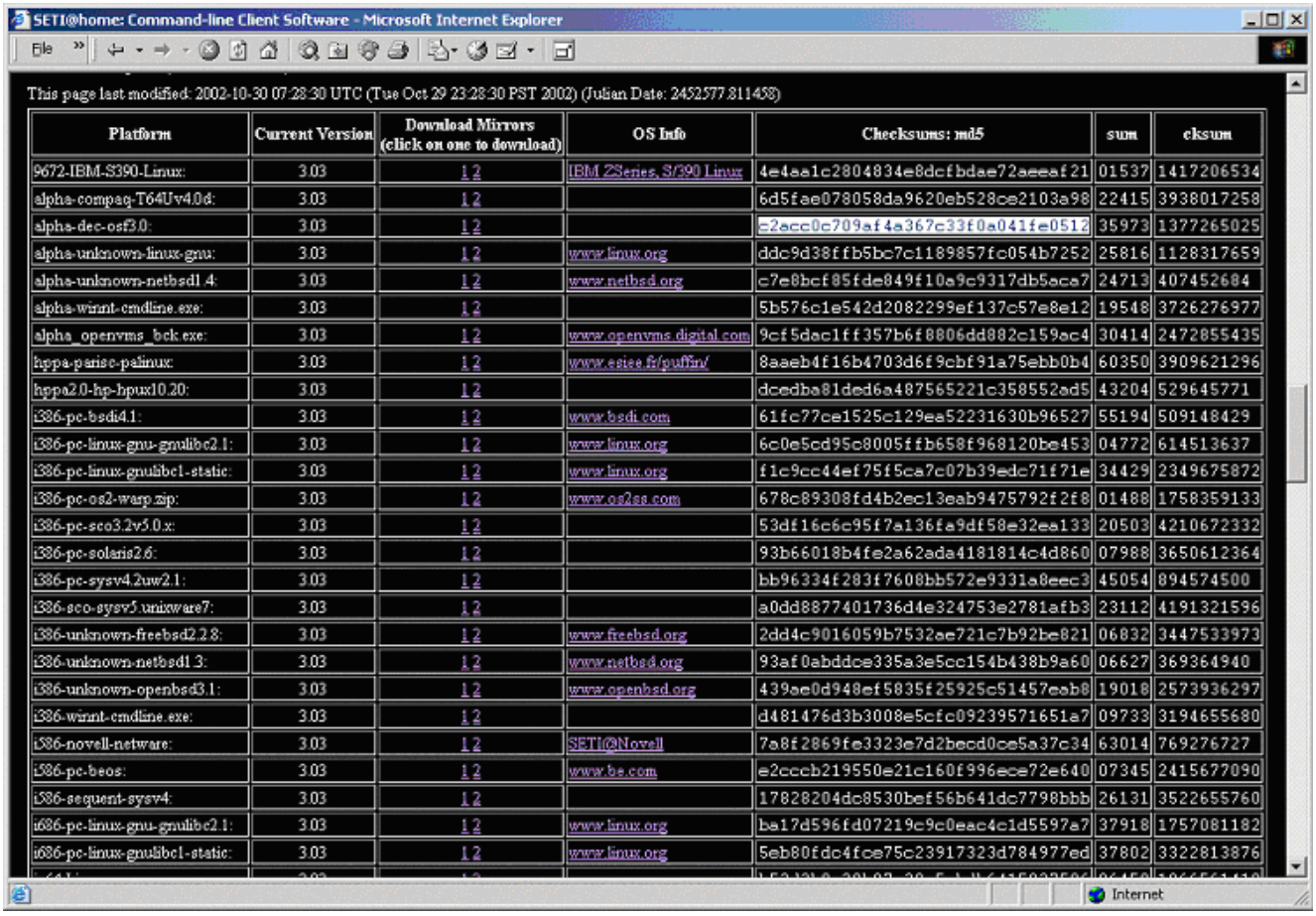


图 4 高密度的区域组织

下面简要地描述了这两种方法：

■ 高密度分布策略可以通过三个常用的方法获得：

□ 表格编排　数据通过结构化的值列表组织起来。典型的例子是空白表格程序和数据库网格。图 4 是这种常用分布策略的例子。

□ 分级组织　信息被建立成像文件系统的图形化表示那样的树状结构。

□ 图　数据用图的形式来表现，如图表或图解。

■ 有限信息分布策略，在另一方面，它的目标是最小化所显示的数据量。有几种方法可以用来控制所显示的数据：

□ Step-by-step 交互　数据被裁减一页一页显示；这一方法的经典例子是所谓的向导界面，借 Microsoft Windows 流行起来。

□ 按需提供细节　一些可选的数据可以按用户的要求显示出来。这一策略的典型例子是有些对话框带有一个“更多细节”的按钮，能够放大对话框提供进一步的信息。然而，后一种方法应该小心使用，因为用户更喜欢熟悉的，可预知的窗口，他们对那些外表改变得太大的 GUI 会感到不适应。

□ 去掉/最小化无关的信息　有很多方法能最小化数据；比如，使它变阴影。在图 5 中显示的命令菜单，一些命令有意地变灰了，表示了它们当前不可用。我们经常把这种特性想当然，但只要设想一下用户仅仅在引用一个命令就要受到“当前命令不能使用”的错误信息恐吓，那是多么令人沮丧的事情啊！

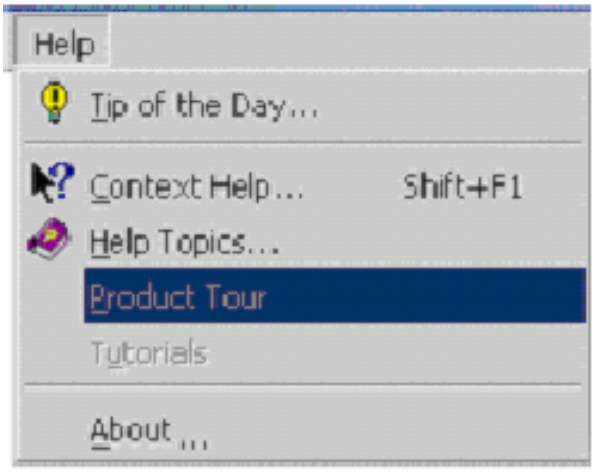


图 5 去掉菜单里不能用的信息

这一节里两个基本的分布策略是任何一个 GUI 的基础。你们自己试着用它来找出你们常用的应用软件中设计者的想法。简而言之，一般高密度方式适用于专家和经常使用的用户；而对更广泛的非专业用户，另一种方式更可取。比如 Microsoft 的 Windows 浏览器，使用的就是高密度组织标准，显示除了当前打开的目录内容之外，还有文件系统的树形结构。

美学的考虑

毫无疑问，优质的 GUI 同时也是赏心悦目的。然而，常常有种对“赏心悦目”名词的错误假设。实际上，一个 GUI 设计中重复出现的陷阱是错误地认为越多越好，停留在过分雕琢的视觉感受上。GUI 决不应该是这样子。一个成功的 GUI 是很少被注意到的，运行平滑而迅速，用户几乎注意不到它，

“过度沉溺的设计”甚至对成熟的设计人员来说也是常犯的错误。事实上，按当前软件发放的速度，最明显能加入新特性（显示新版本的成果）的地方常常就在用户界面。

但是，美学方面也是很重要的。开发者常常自以为是地设计他们的用户界面的视觉表现，做出不好用的设计。有些人觉得那些视觉细节，如按钮大小，整体视觉平衡之类的很无聊。这类开发者中大多数从实现方式出发，尽量自动化用户界面（暗中把界面设计看成是单调，不必要的活动）。不幸的是，还没有为人类设计的替代品，自动生成的窗口虽然很酷，但是却不是好的用户界面。

抓住用户注意力的技巧

抓住用户注意力的技巧在用户界面中得到广泛的使用。他们来自于经验研究，可被概括如下：

- 动画　在屏幕上闪烁的东西很容易抓住用户的注意。这种技术可能有干扰性和侵犯性的。动画通常用来表达 GUI 的内部状态，表示工作进程或一般的活动。
- 颜色　和动画一样，这一技术应该聪明地运用。太多的颜色会造成 GUI 的混乱。
- 声音　使用这种技术，如果用的好的话，可以非常的有效率。事实上，考虑到残障人士，声音信号提供了一个高效的反馈手段。
- 图形修饰（如粗体字，特别的图案等等）。如果使用灵活而连贯，这些没有干扰的图形习惯是高效的。

幸运的是，依靠专业的设计方法可以避免很多严重的错误。对于那些想吸引注意力的如闪烁标签，颜色之类的方法尤为重要。

提供系统内部状态的反馈

给系统的内部状态做标记是必要的。这个重要的特性可通过不同的技术获得。最常用的技术是下面几种：

- 改变指针形状　　在现代 GUI 中，指针的形状得到广泛的使用，用来表示应用程序的内部状态（等待指针）和当前可用的操作（如，托曳窗口的一角可以调整它的大小）。简而言之（如果没有被你公司的指导方针所陈述的话），任何一个超过三秒钟完成的操作都应该使用等待指针。

- 动画　　前面已经提到，它可被用来显示完成操作的准确进程（一般使用进度条组件），和一般的活动（如使用临时动画）。总之，进度说明应该至少每四秒钟更新一次。

- 消息　　系统能够使用消息对话框，状态条或者相似的技术传递 GUI 的内部状态给用户。这是广泛使用的技术，交互的细节在使用的风格指导中有说明。

在设计 GUI 时，一个需要仔细考虑的争议性的问题是 Mode 的使用。Mode 是影响部分用户界面行为的特别的状态。你可以把它们看成是以前的用户交互改变当前动作的上下文。依靠特别的 Mode，应用程序可以产生方式完全不同的行为。设计方针通常不鼓励 Mode 的使用，或者禁止它。标志出当前应用程序的 Mode 是很重要的。修改指针形状（当某工具被选择的时候）或者凭借开关按钮或状态条等可以做到这一点。

交互风格

看过这个领域的经典文献之后(上一篇文章的参考文献一节)，我们可以确定用户界面的几个基本的交互风格。这些风格在现实中的 GUI 里加以组合。对每个风格，我们会总结出几个设计的方面。

- 菜单选择　　无论何时有多少的选项，这种方法都可以使用。

- 表单填充　　主要用于数据的输入。用户看到一个语义相关部分的显示。它们可以用键盘检查并最终修改数据，从一个部分到另一个部分。

- 直接操作　　GUI 创造了用视觉表达的世界，用户可以直接操作它。现代的字处理器（用户可操作屏幕中的字符），视频游戏和流行的桌面隐喻（见图 9）都是这一方法的例子。

- 命令语言　这是最老的交互风格了，它特别适用于那些直接使用命令行的工具表达复杂指令的专家用户。

- 自然语言　用户与系统之间用他们自己的自然语言进行交互；比如，使用语音识别和语音合成的方法。

我们将看到最常用的几个设计方面的细节，即列表的前两个：菜单选择与表单填充。

菜单选择风格

菜单用来选择项目（通常的目的是引用命令），用一种系统的，集中的方式。注意到这里所使用的非属性词“菜单”，我们把像超文本中的链接的可选择项一样表示为下拉菜单，按钮等等。

组织下拉菜单是个重要的问题，特别是当有很多项可供选择的时候。常用的标准有：

- 任务相关的组织　这是组织菜单项里唯一最成功的策略。在设计的时候组织选项，按照语义学的标准，大大地帮助用户在运行时间之后访问到它们。但是，如何与最终用户共同制定语义标准的问题仍然存在。

- 属性结构的层次分组　等级（深度）的数目和每一级包含的选项（宽度）是目录树的特征。经验研究表明，在目录层次中（见图 6）宽度比深度的优先权要高。简而言之，目录层次的深度不能超过三级。有一些选择正确等级结构的使用规则：在根目录级，可以考虑深一点，却败选项不会重叠但能清楚辨别，考虑所有的可能性。在叶子处则考虑更宽的范围（即菜单层次中最终的选项）。

- 标准的组织　采用标准的菜单组织有助于用户很快地适应新的应用程序，把工作中的记忆量降到最小。这种技术用软件应用的命令菜单实现（如菜单中常见的“文件”，“编辑”和“帮助”）。

这些策略，如果组合起来，可以减轻用户在大菜单目录里查找一个选项时所耗费的时间。

同样，Windows XP（见图 6）中的开始菜单遵循的就是前面提到过的建议组织易于访问的大菜单。

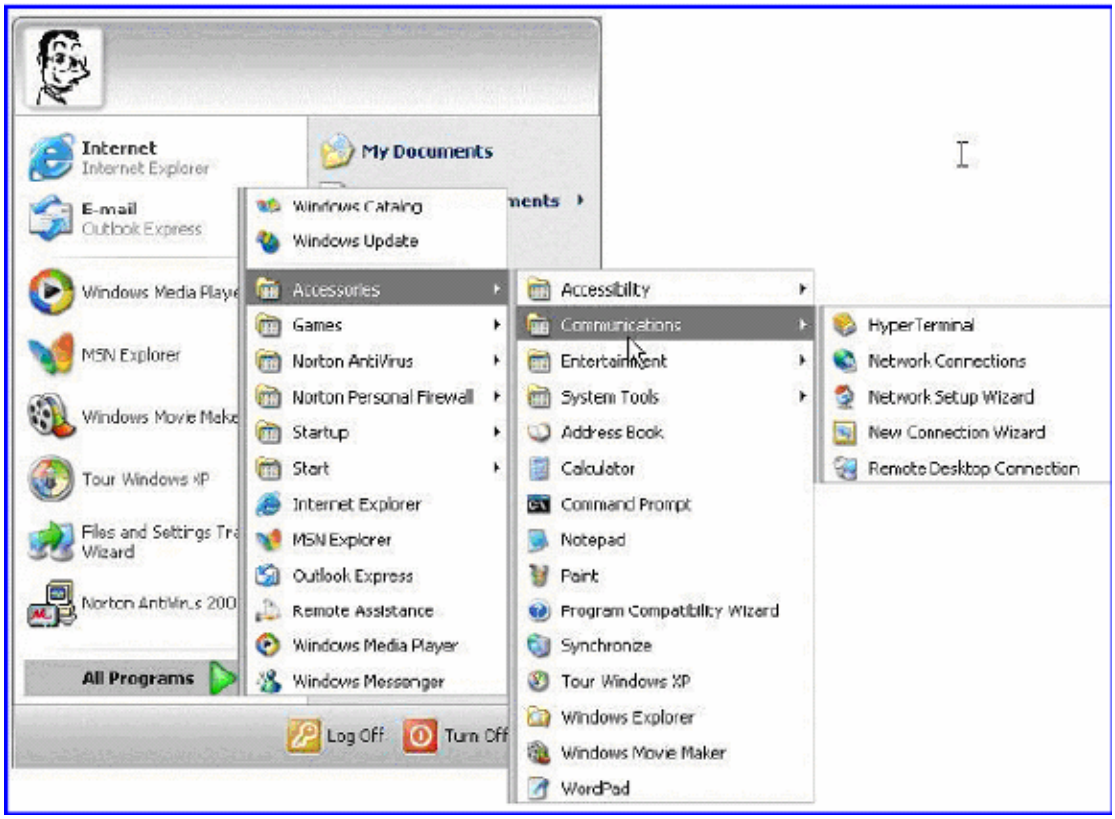


图 6 等级目录的例子：Windows XP 中的开始菜单

表单填充风格

另一个在现代图形用户界面中广泛使用的交互风格是表单填充。这是和菜单完全不同的交互方式。它主要的功能范围是允许用户阅读和输入（主要是离散的，也可以是连续的）信息到系统中。

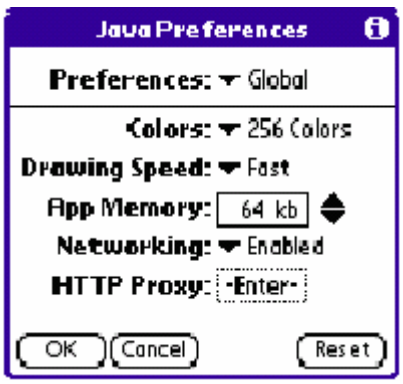


图 7 Java 的表单对话框例子：使用无线电话

表单填充风格的基本原则是能够输入基本的数据，提一下几个主要方面：

- 数据输入方法的一致性。显然，数据输入应该尽可能地做到一致，从表单的设计开始就应如此。

- 最小化用户的输入动作。用户界面应做到把操作步骤减到最小。因为使用表单填充的都是老用户，这一点是必要的。

- 最小化用户的记忆量。利用相关性的数据和短期记忆（见前文）减少交互的难度。

- 数据输入和数据显示的兼容性。这是现代基于计算机的表单常常忽视的一个属性。当我们输入密码在密码栏的时候（显示的是星号而不是实际的内容），我们完全可以依靠短期记忆成功地完成我们的输入子任务。设想一下在复杂的表单中输入完全不智能的数据是多么困难啊。

有一些基本的设计数据输入表单的原则；我们在这里仅提一下最有用的：

- 按逻辑来分组和排序　这是创建一个有用，合理的产品所必需的。

- 提供清楚地说明　包括从提供有意义的标签和表单题目到给填空处提供附加的解释说明（如通过相关的帮助文件或工具提示），再加上采用一致性的术语和缩写。这里是一般性的描述：尽可能避免有障碍的术语，而要提供尽可能贴近用户的行业领域中所熟悉的名称。

- 高效集中的导航　其他的导航机制（除了鼠标之外）也应该被考虑进来；首当其冲的是键盘。标准在于它的交互性和可用性，而不仅仅是图形方面的表现。

- 提供高效的完成标志　完成与表单相关的数据填充任务需要清楚的方式。完成的计划一般要求平台和相关的风格指导。比如无线电话，通常有一个键（用某个图表或绿色的标志表示）用于完成当前屏幕的数据。

- 表达性的文字　文本标签保持简洁并在一旁提供相关的组件，如图 8。通常，所有的主要消息（命令名称，标签，窗口标题等）都遵循所谓的标题大写原则（见图 7 和图 8）。

- 错误处理　只要有可能，GUI 都应当设计防错；如果不可能，应当显示有意义的，建设性的错误消息。

- 设计视觉外观一般要考虑采用视觉上有吸引力的布局再加上标志说明哪里是必需的，哪里是可选的，并把语义相关的部分分到一起。

创建一个高效的表单需要额外的考虑。在图 8 中有一个简单但是设计出色的表单框。导航能力的提高和交互的流畅可以归结为几个简单的细节：

- 表单的每一个部分通过使用相关的记忆键可以方便地访问（这些依赖于特定的平台习惯）。
- 靠键盘和整体的表单导航在表单之间移动，这已经被证实能加快老用户或习惯用键盘的用户的使用。
- 标准的按钮添加在 GUI 的底部。用这种方法，用户从过去的经验（其它的窗口和表单）明白怎样去除对话框（所谓的完成信号）。注意“确定”键是缺省状态下的选择。
- 小心设计视觉的表现和对话框的组件布局可以有效避免无用的图形并提供整体的舒适效果。
- 可获得的信息（适应那些感官或认知有障碍的人）也要增加。如“不可见”的特征在某些情况下可能是很重要的，应该一直使用。

这些小的细节大大地提高了整体的可用性，特别是对那些老用户而言。

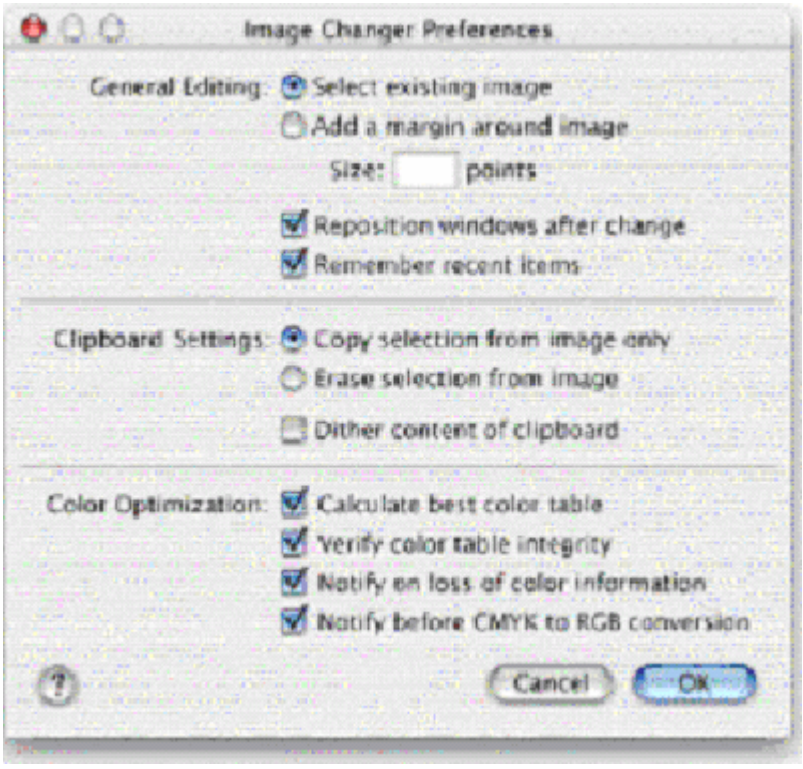


图 8 Mac Os-X 表单对话框的例子

直接操作

图 9 中有一个直接操作交互机制的经典例子，Mac System 1.0。用户界面上的元素可以被拖曳，编辑和删除，操作按照一致且“直接”的方式进行。

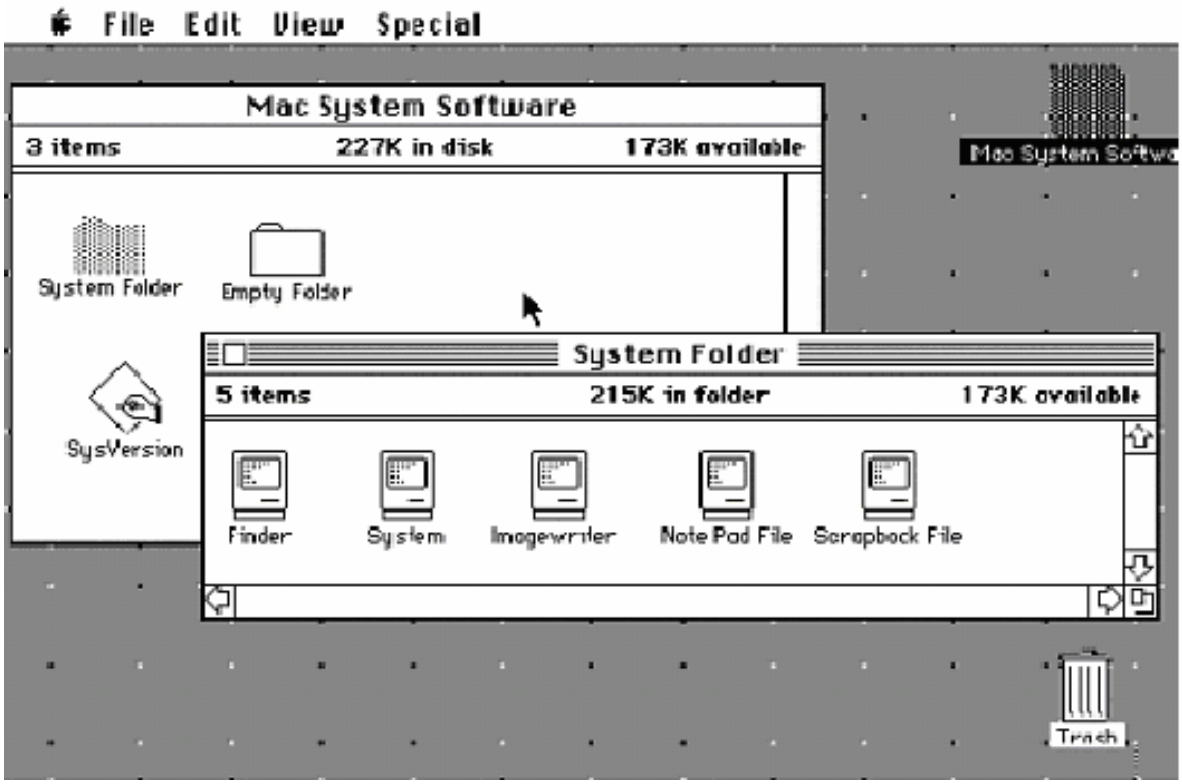


图 9 直接操作交互风格的例子（Mac 1.0, 1984）

幸运的是，现在的直接操作是良好的设计界面的缺省操作风格，这种风格将面向广泛且多样的用户群体。

命令语言

历史上，这是交互式计算机的第一种交互风格。

早期的现代计算机拥有很有限的资源，几乎不能提供用户界面的部分。用户一般是专业人士，且没有形成大的市场来推动更实用的界面。显然，在资源有限且市场狭小的状况下只能推出简单的，面向功能实现的交互风格。图 10 是第一代所谓的“家庭”计算机的用户界面的截图（注意空闲内存的数量）。图 9 中操作系统软件的内存空间要比图 10 中操作系统提供的 ROM 空间要多一个数量级。

图 10　命令行交互风格的例子（Vic20, 1983）

总而言之，命令行用户界面远远没有消亡的迹象。相反它们仍然存在，因为它们为专家用户提供了功能强大且灵活的表达方式。图 11 是 Linux 操作系统终端窗口的快照。

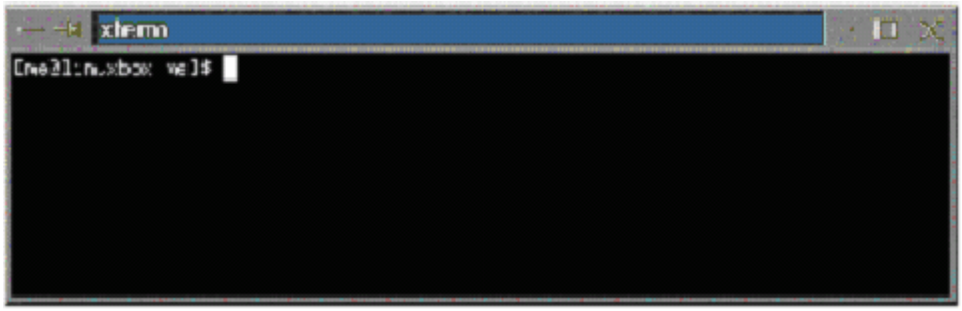


图 11　现在的命令行界面的例子（Linux, 2002）

命令行用户界面一般需要初期的培训，而且与其它的交互风格相比错误率一般较高。

自然语言

虽然实现起来非常复杂，但是自然语言（包括文字和语音识别）在某种情况下是实用的。

许多商业应用软件的例子经常在市场中出现。一些分析家预测它将是未来占主导地位的用户界面。但是，自然语言处理（NLP）的理论难题远远没有解决，而且这一有前途的交互方式还仅限于少数有限的应用软件。

结论

在这篇文章中，我们涉及了一些图形用户界面设计中有趣的问题。我们采取实用的方式试着凭借一些普通的、现实世界中的例子支持其中的理论。关于参考文献，感兴趣的读者可以参考上一篇文章开出的书单。

软件管理与软件工程译丛

50%的中国软件公司老板从不关心自己的属下。

—CSDN 调查，《程序员》2003.8

Peopleware

人件

[第2版]

[美] 汤姆·迪马可 蒂姆·李斯特 著
Tom Demarco Tim Lister
UML China翻译组 译
方春旭 叶向群

清华大学出版社

一种请求及分配有限资源的分析模式

Fei Dai、Eduardo B. Fernandez 著，wnb 译

查看评论

1 目的

如何公平有效地为多个请求分配有限数量的非可重用资源？

2 内容

在许多情况中，可用资源的数量远远小于请求的数量。例如，流行体育竞赛的门票在比赛开始前就已经全部告罄；公司中重要的职务往往有多个角逐者。在中国，原始股票由于其价格低廉往往首次提供就全部卖出。从系统的观点来看，这些资源都是不可重用资源。不存在资源的返回和重新分配的问题。与之相对应的是可重用资源，如交通工具、宾馆房间、飞机座位，这些可重用资源在使用后返回并可在未来重新分配使用。

由于其操作可选则的多样性和复合性，对有限不可重用资源的分配非常复杂。通常，需要预先请求（发送请求并等待分配），但在某些特殊情况下，客户可能没有预定或等待而直接需要资源，比如当资源接近有效期限时。例如当重要足球比赛球票的请求处理完成后，人们仍然可以在比赛开始前通过到售票处直接购买球票（直接请求）。通常分配在某一特定日期或事件到来后将被放弃，例如等待机票的人们在飞机即将出发时，当确定不会再有预定机票的乘客到来时，可以获得机票。其它情况还包括，请求处理完成后分配也就完成了（如足球票的例子）。

当然也还存在其它的分配资源的方式：可以采用先来先服务的方式，通过比较优先级，通过抽签、谈判或其它策略。本文假定资源相互之间是不可区分的，它们都属于特定的类型。例如，某一请求声明了一些指定价格的票，或一定数量的某类股票。

3 问题

分析模式应该满足下列基本用例：

请求资源：客户请求获得某一类型的固定数量的资源。这些资源相互之间是不可区分的。请求放入队列中等待资源分配。客户可能是人、机构或计算机实体（进程）。

分配资源：在经过一定时间后，或者有足够的资源（还没有分配），从请求等待队列中检索出请求并将资源分配给该请求。

获取资源：在请求被接受后，客户需要立即获得分配的资源或者在一定时间后获得该资源。客户对资源的请求也可能得到部分的满足，例如，某一客户需要 10 张球票被分得 5 张。

撤消请求：客户撤消其请求并释放已经分配得到的资源

下列因素将对可能获得的解决方案产生影响

对请求的处理应该是公平的（对所有请求采用同样的处理方式），尽管某些分配策略可能带有不公平的色彩。例如优先级分配。

解决方案仅仅包含基本的语义，代表最小的应用，包含一些用例。这是使得该模式成为语义分析模型的基础[Fre00]。这类模式有助于建立概念模型。

对实体的有效使用。没有冗余和关联的实体。

许多应用需要纸质文档。标准文档应该清楚地体现在模型中。例如，一系列预留球票的发售可能包含一个发售文档标明价格、日期等。这使得该模型更加直观并且方便了这些文档的建立。

4 解决方案

使用抽象队列作为分析模型的核心。根据分配策略将请求放入队列中。例如，基于先来先服务策略将会把一个新请求放入队列尾；基于优先级策略将新的请求按照其优先级进行放置。每个请求必须放入队列中。资源只能从队列前端的请求开始进行分配。直接分配将被视为特殊的情况，在此情况下，请求将被放入队列前端。当一个已经分配的请求撤消时，其所占用的资源将立即成为可用资源。客户可以选择是否接受对其请求的部分分配。

4.1 类图

图 1 显示了一些必要的类。客户对一些类型的资源发出请求。对每种类型的资源有一个队列，分配由 **AcquisitionRecord** 描述。**AllocationStrategy** 用于分配资源。注意请求需要几个类型的资源而实际的分配可能只有其中的部分。属性 *acceptPartialAssignment* 指示客户是否同意接受更小的分配。

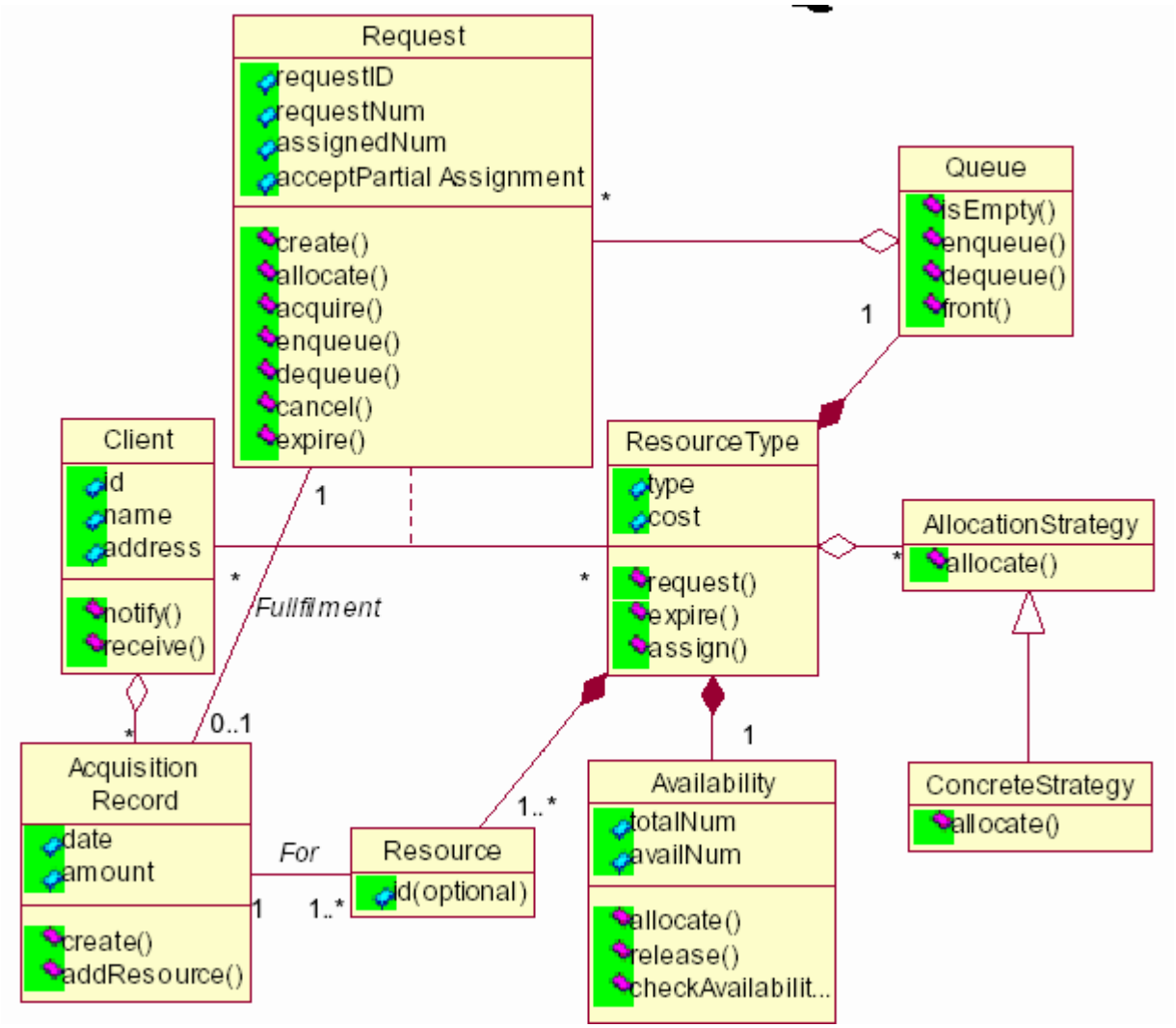


图 1 模式的类图

4.2 动态性

图 2 展示了请求对象的状态图。请求最先建立，加入队列中，然后将资源进行分配。任何时候都可以撤消请求或者在某一时间限定后终止。图 3 显示了资源请求、分配及获取的序列图。首先，请求对象被建立并填加进队列中。前端队列的请求经过检验，将资源分配给它们并作出通知。建立了请求资源的客户以及 AcquisitionRecord。

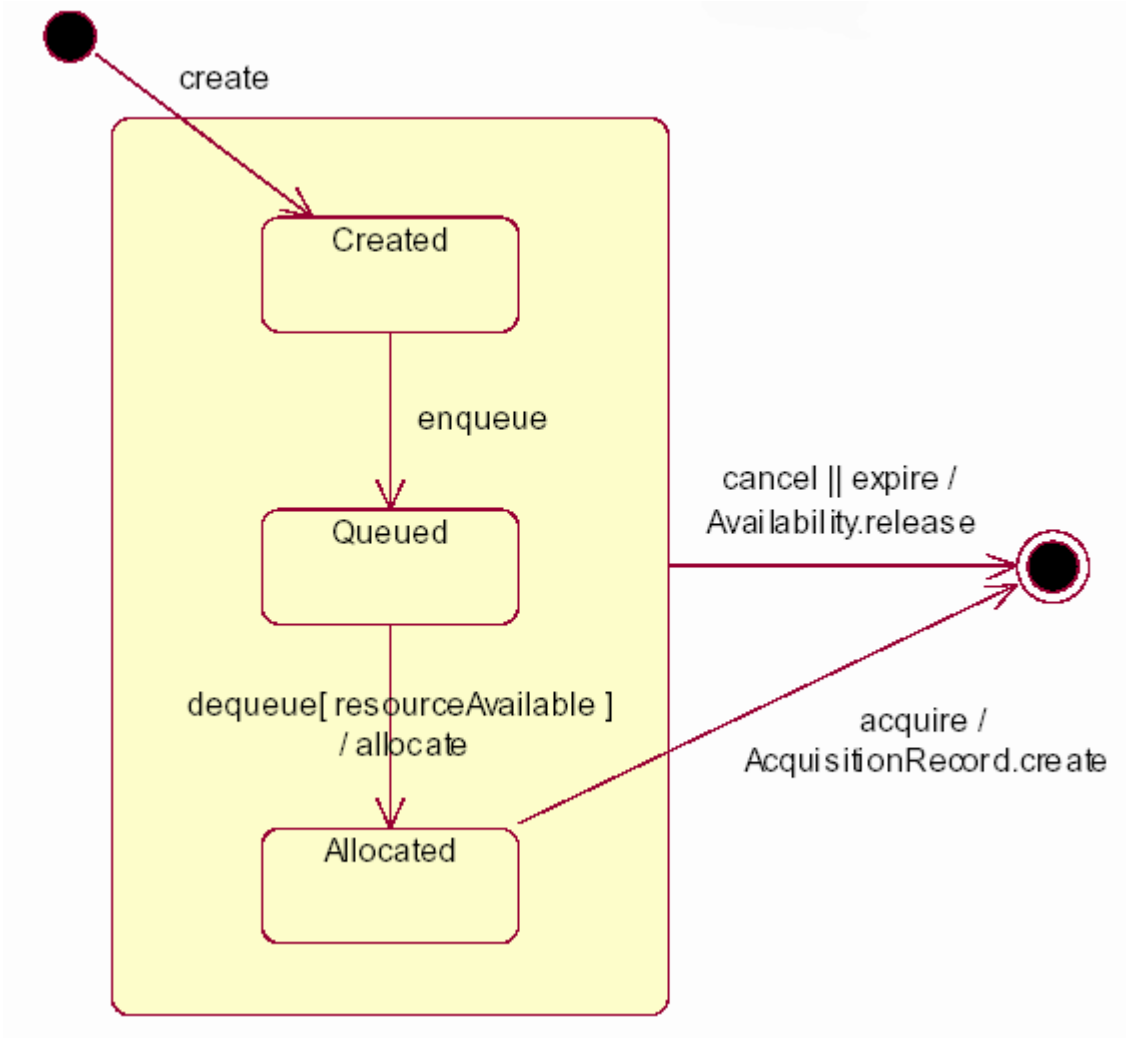

create
Created
enqueue
Queued
dequeue[resourceAvailable]
/ allocate
Allocated
cancel || expire /
Availability.release
acquire /
AcquisitionRecord.create

图 2 请求对象的状态图

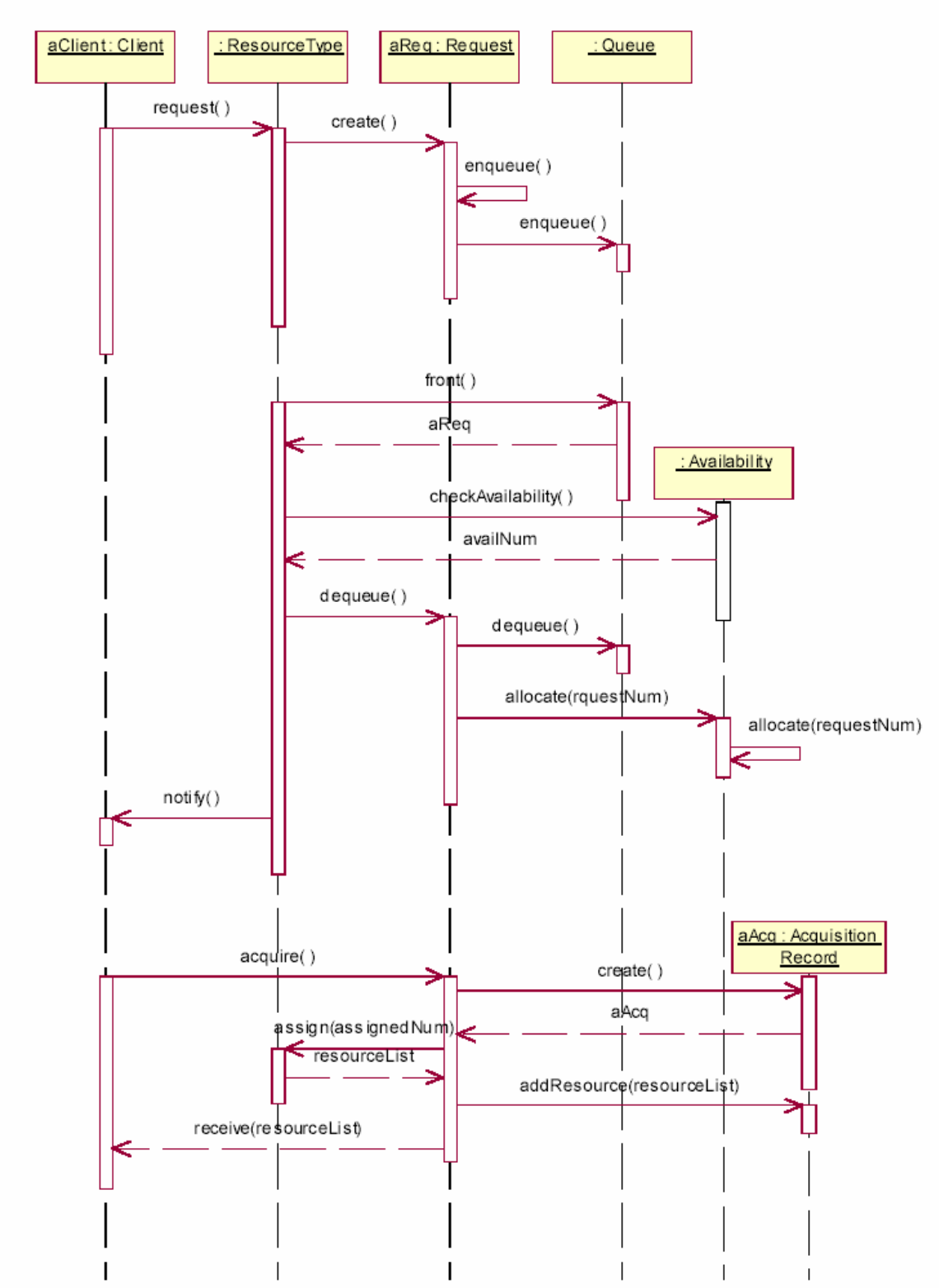


图 3 请求、分配、获取资源的序列图

5 解决的实例：中国股票市场的股票发售问题

在中国，当公司在市场上发行新的股票时，他们将其直接销售给市场的参与者，包括大投资商（基金、投资银行等）和为数众多的个体投资人。由于发售时价格比市场价格低，因此购买数量往往超过有效的股票数量。为了给每个投资者提供良好的机会，管理层采用了彩票系统来分配购买操作。典型的股票发售周期有以下阶段：

公司公告将发售的股票代码、名称与价格。在下面的预申请阶段（通常 1-3 天），每个投资人可以提出包含请求数量在内的请求。请求者在其帐户中需要有购买其申购数量相当的资金，对应数量的资金将被冻结直至第二阶段结束。在冻结期间的利息由发行股票的公司获得。

每个请求被赋予一个有效的抽奖号码，抽奖号码是连续的顺序号。这些号码按照比例分配给各个请求。最后，通过随机抽取末尾号码确定中奖的顺序号。中奖的号码的频度按照整个请求的数量和能够提供的股票数量确定。有效的顺序号代表购买股票的权利，将发送给相应的投资人，这一阶段通常需要 1 周时间完成。

在第三阶段（通常是一天时间），获得购买权的投资人使用其购买权益购买股票。当然，他们也可以放弃购买权利。如果在结束时仍然有股票剩余，则这些股票将卖给预先确定的投资银行。

图 4 显示了该使用模式的类图。说明如下：

去除了资源类，因为同一公司的股票是不可区分的。只要知道每笔交易的股票的数量；

只有一个 LotteryStrategy 对应所有的股票。发布策略是唯一的并且对所有投资人都一样。

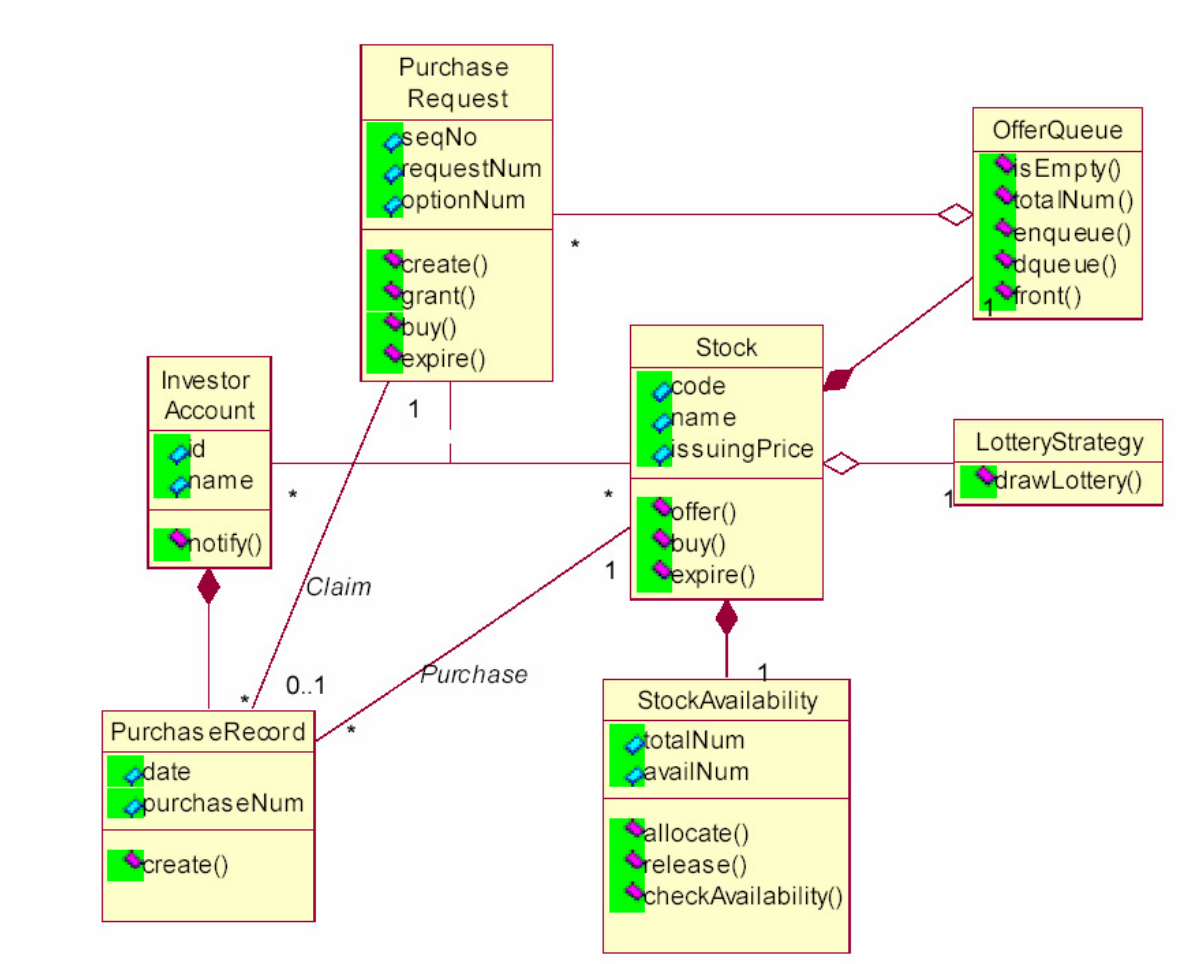


图 4 购买股票应用的类图

6 应用案例

使用该模式的应用包括：

体育比赛球票预定及购买（例如，2006 年世界杯）；

工作申请，多个申请者申请几个工作；

中国股票申购问题，投资人首先通过彩票系统获取购买权，并按照其所获得的购买权申购对应数量的股票；

由政府提供的廉价公寓的销售问题（例如，中国深圳）；

为产品分配组件。

7 结论

模式满足下列要求

队列机制为处理请求提供了良好的处理机制，每个希望获得资源分配的请求加入到队列中。

正如在应用部分所展示的，该模式应用范围宽广：球票、股票、工作申请等。

请求的每个方面都进行了表达，没有冗余的类产生。

该模式为请求和 AcquisitionRecord 文档产生了清晰的类。

该模式可以通过定义变量实现定制。在股票的例子中，我们没有考虑资源类，因为没有区分购买的股票的要求。

为了使该模式更加一般化，本文省略了一些细节

可能存在不同的需要区别对待的客户类型；

在分配时可能存在一些需要进行交互的协商。实现该机制需要在设计阶段建立一些额外的类以定义用于协商的用户接口；

某些系统可以对每类资源使用多个请求。例如，在航空港，对不同类的乘客应该采用不同的队列。

8 相关的模式

该模式对 Reservation and Use pattern 的某些方面进行了扩展。同时，该模式没有考虑资源的使用问题，仅仅考虑到资源获取以前的情况。其它不同的地方包括：资源被视为不可重用的，请求在一定的时间条件后将被取消。Order and Shipment pattern 模式是对该模式的补充。提供了占有资源的细节，即分配的资源的支付与发出。另外该模式还包含以下两个子模式：

Strategy pattern 模式。用在此处为使用不同策略进行资源分配提供可能。

Reservation pattern 模式。为 Reservation and Use pattern 的子模式。对应类：**Client**, **Request**, **ResourceType**。请求与预定方式类似，等待完成，其差别是，实际的预定方式包含确认，在此没有表达。

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

中译本即将上市

样章先行提供>>

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

第一本 UMLChina 训练指定教材

清华大学出版社

中国“人件”的声音

UMLChina 整理

查看评论

以下文字整理自 csdn.net 和 china-pub.com

godhate，2003-05-19

已经做过两年比牲口都不如的工作！

两年加起来发到我手里的工资、奖金及所有一切，不超过两万元！

最可气愤的就是我有一次因出差所去单位离市区较远，找不到住的地方，（这个单位没有招待所），不得已，和烧锅炉的师傅一起睡在锅炉房里（如果谁有失眠症状，到那里睡一个星期保证痊愈），每天早晨耳边都伴随着充斥着除渣机和水泵上水时巨大的轰鸣声起床，（本人有幸在此地住了两个月），回到公司，公司却以住宿没有花钱为由，（顺便说一句，我的公司每天出差补助 70，实行包干制）要扣掉我每天出差补助中的 50 元，出去折腾了两个月，一分钱没挣到，到搭进去不少，简直就是自费出差，还遭到公司在背后下黑手！

值得庆幸的是：从此以后真的再也没有睡不着觉的时候了！

wltsui，2003-3-25

现在提起程序员就是加班、熬夜的代名词，如果你是一个程序员，有一天下班正点到家了，你的朋友就会问一句“今天这么早回来了”！　可是你的亲人却是在等这一天等好久了！　难道程序员就是必须要加班吗？？

Jukyy，2003-3-25

开除老板。有些老板自己花钱大手大脚，铺张浪费，对员工却小抠得很，给这样的人干活没有意思，也没有什么意义。

zdx1026，2003-5-10

气愤：讨论一下老板心理-----(欢迎讨论)(分不够，再开贴！)

01 年在一家公司，辛辛苦苦，夜以继日，通宵的加班，谈好的工资从来都少发几百；没有计较怎么多，最后为了安排老板一个兄弟，竟然以我没有按时打卡为由(全公司都没有打)要扣半月工资，我性格刚烈，毅然辞职，还按照公司规定提前一个月递交了辞职书，这下更惨，两个月工资都没有了!!我哭啊!!!

02 年，去外地搞实施，也是客户点名要我去，结果公司撑不下去了，等过了两个月回来，发现公司徒有四壁，摆出一付架势：我没有钱！后来我实施的钱收回来了，发给了某些员工人事、文员，却没有我们工程师的!!我苦啊，后来和几个同时走上了劳动仲裁和法庭，我们赢了，却谁也找不到了。有小道消息说，公司和黑社会洗黑钱有关，就不要争了.....

现在的公司，每次都要推后几天发工资，我已经不想搭理那些道貌岸然的家伙了!!我已经太失望了，太失望了!!据说有些城市政府都强制命令当月发工资，为什么这个城市不实行呢？据我所知，这种现象太经常了....

朋友，你们还好吧？

Mostice，2003-05-07

你的老板这样矛盾吗？

新公司，老板第一次自己做老板。什么都想做，真做好的没几样。什么任务都想让一个员工完成。却不给手下哪怕多一点点的薪水。你怎么对付上面的 BOSS。

Atstudy，2003-1-11

老板毕竟是老板，打工的始终是打工的!!!

真 TMD，项目失败怪在老子的头上了。项目后期维护，没有提供用户的业务流程，没有程序流程，只有代码和表结构就这么让我成立个小组做，如果你程序写得不错，处理方法也得当也不至于后期给我埋下这么多地雷啊。没办法是老板开发的，这种擦屁股的事怎么让我摊上了，倒霉。

wudid007，2003-04-25

他不是搞计算机的，在进行维修中，当你在思考时，他在旁问你想怎么做；当你在测试时他问你在做什么。

Kongxiangli，2003-4-16

老板看我重新买手机就说：手机都买了，还不给我把笔记本赔来？

笔记本丢了，郁闷，还有手机？P4-IBM 使用不到半年，公司让我赔一个相当的，家里来小偷了，TMD，有什么线索请提供？老板看我重新买手机就说：手机都买了，还不给我把笔记本赔来？寒心啊？为他干活时总是 TMD 看不见！

ChocoboY，2003-3-21

现在我还在大四，跟一家小公司签了约，不过感觉老板有点。。。也不和我们商量就直接在外面租了一套房子说要搞封闭式开发，我们几个程序员都不愿意。

homerocker，2003-3-21

不管吃住在公司，不能上网，不能接电话，也不能打电话，就像坐牢一样

BoyPlusPlus，2003-3-21

就是变相软禁。光提这个名字，就是对人权的一种践踏。不过这是以美国人观点来看的。至于中国，就无所谓了。老板就是天。

NullGate，2003-3-21

说白了,公司想省点钱。吃喝拉都在那里。这样的公司千万别去,小弟。都什么年代了,还相信什么闭门造车。

zb_china，2003-3-22

脑力工作需要 15 分钟进入状态，白天在办公室平均每 20 分钟我就要受到一次干扰（电话、问事等必须中断思考的事情），基本上没法编程，实际上我的程序都是晚上完成的，有时候为了编程就白天在家里工作了。

CProFan，2003-2-12

在我的长时间消极怠工下，老板终于找我协商离职了，早就想走人了，可如果我提出就会算我违约，这回是公司提出的，所以可以得到违约金啊！

iPDA

今天中午我和经理聊了一通，查点吵起来，大致如下：

我：经理，整天加班，我受不了了

经：你们不加班作什么？年轻人就要拼体力

我：但是我还要学习、洗衣服、作饭啊

经：我看你是想在家里偷偷做私活吧？

我：可是钱太少了

经：你一个小小的程序员，还想要多少钱？听说过软件蓝领吗？过几年还要降。

我：那咱们以后在谈吧（挂电话）

我：伤心 ing……

WnEunfn

我突然发现，我只是一台被老板百般盘剥的编程机器罢了……

老板对你说：操你妈，你们这些猪，全他妈的是贱种，老子不会 VC 也不会 Delphi，但老子有的是钱，老子养了他妈的一大群程序员，他们都他妈的

Nirvana0，2003-5-20

唉，部门经理开始用权利维护他落后的思路了，郁闷，放分。

比如：用 java 写控制逻辑,调用不同类的时候不用反射，强调要用 switch,还说以后这套设计可以用到 C 设计的系统上，等等就不全说了，免得大家鄙视我们公司。。。真郁闷。

uuuuu，2003-4-21

有没有因为在技术上与项目经理有不同意见而说出来被炒的！我可能是第一个。

写了封 email，提出这种做法我认为可以产生的后果，再说明是个人意见，及完全服从经理工作安排！但问题是这封信抄送给他的上级项目总负责，因为我知道光和他说是一点用也没的，之所以发给他，只是尊重他是我的上级，这种事有必要让他知道，可是。。。。。。

人人都说软件开发是扁平的组织架构，软件开发是团体性很强的工作，团体中的成员都要为项目考虑，所以我们才不停的加班，没有一天是 10 点前到家的，还是没有加班费。

我很郁闷，离开，我一点也不伤心，因为这样的团体没有发展前途，前几天才被挤走的一位，没几天就在竞争对手公司当了经理（绝对没有靠出卖公司），不说了，可能我倒霉进了这公司！我想不会每家都一样吧！

gary_shi 2003-4-13

我讨厌工作狂，他们使得工作效率不高，想准时下班的人不敢准时下班，即使我干的活比工作狂多的多。

Zouwenyan 2003-4-14

真的不希望再看到谁用贬义的口气来说"工作狂"了，你看到别人下班不加不一定就是在赶项目进度，下班谁都是自由的，别人不回你也不回吗，你就没有自己的主见吗？下班后可能别人只是在进行自己的学习，做开发上班的时间你要赶项目进度，你要写各种计划与文档，你要协调组员工作，控制代码质量，这些时间占去了所有的上班时间，自己的技术能不能向上提一层，新出来的东西要不要了解一下，论坛上别人的意见要不要看一下，如果你家里没有电脑没有宽带，留在公司多呆几个小时，学习工作提高自己，也要遭到别人的非议和贬义吗

wltsui 2003-4-14

对于在家没有条件的人来说在下班不走留下来学习，那是因为在上班时间一般很少时间能让你自己去学一些你想学的东西，

newdongkui 2003-4-14

禁止工作狂，这是危险和短视的，长期工作会让人失去思考的能力。公司重视人才，是长期的行为，培训应该系统和考核的。

Zouwenyan 2003-04-12

没有人比我们的办公环境更惨的，我们的小格子长 1.5M，宽 1.5M（好像还不到呢），面积大概是 1.5 个平方

gary_shi 2003-04-11

窗户是有，可 TMD 我的位子是面对窗户的，背对着门，因为老板想一进来就看到我们在干什么。

Qingrun 2003-04-11

没办法，我们只有日光灯光，上班的时候根本不知到外面的天气情况，除非午餐和晚餐的就餐时间。呵呵

wltsui 2003-04-11

看来都一样，日光灯光是不能没有的，自然光是没有的！

webcat

有阳光的地方被老总占了！

Banping 2003-04-11

他妈的，窗户都给隔开了！给老总坐了！

tj_dns 2003-04-11

有窗户，而且外面还有花和草。工作休息时可以看一看外面的风景。

shblx2000(可乐) 2003-04-11

没有。奶奶的。郁闷

zouwenyan(珠穆朗玛) 2003-04-12

我们这有窗，右边的墙全部是窗

但是前后是二个工厂，离我们大概有 50 米，一到上班时间，就有前后两台机器开始响，不会有什么环境比我们这里更糟糕了，我们真正是在厂房里做软件开发的

没有空调，大厅里人很多，天一热，所的窗都是开着的，机器声音很大且又长又闷。。。

slmengcn

2003 年最让我沮丧的事情之一就是看了 3 月份的财富杂志，那一期的主题是世界一百最佳雇主。

bomb_hero 2002.12.03

我以前一直想做 A 类，但发现老板根本看不到你。每个 Project 的奖金也被 TEAM LEADER 拿了大头。我也要吃饭，供房子，赡养父母，让妻子不需要在菜场里为了几分钱和小贩吵吵闹闹。我觉得在中国想靠你有很好的技术别想加薪水。渐渐的，我也学会了拍马屁，在客户面前依靠打太极把死人说活。我终于慢慢的变成了 B 类，开始做 Team leader，开始剥削我的手下。

Connonrocket 2002.11.30

中国的企业，也就这个样子了，我来公司一年多了，宽带从开始来的时候就申请，一直到现在也没有申请上，原因很简单，因为一年需要一万二的钱!! TMD BOSS!

Jxzproer 2003-04-11

空间很大，但工作设备太差。

freeagle 2003-07-19

闲话“人件”：判断软件公司是否尊重员工的四条标准

最近《人件》这本书很热，作为一个技术人员也谈谈自己的感受和想法。

判断一个软件公司是否尊重和关心员工，我提四条简单的标准（当然我这个不是什么权威的原则，希望抛砖引玉）：

第一，是否配置液晶显示器。

第二，是否不合理限制上网。

第三，员工是否有提出建议的渠道。

第四，对于员工提出的建议是否重视和及时地答复和解释。

当我看到我的同事用电脑吃力地看电子书的时候，我只好摇摇头。很多收费的人员都在使用液晶显示器的时候，而程序员却没有这个福气,可悲啊。我想要求单独的办公室也许还不合国情，要求一个液晶显示器应该不过分吧。

我想技术人员的权利要自己去合理地争取，不要奢望贪得无厌而又鼠目寸光的老板会大发慈悲，争取的手段就是先沟通，如果无效就跳槽，只有大部分（过 51%）的人有了意识并行动起来，才可能改变这丑陋的行规。我刚被公司炒了鱿鱼，不过我有一种解脱的感觉而不是痛苦。

公司老板关心提高生产力，而软件公司的生产力主要与三个因素有关：人员素质，过程和技术。而这三者中人是核心。这两年比较流行 SEI 提出的 CMM,而 SEI 的 P-CMM（关于人员的）却谈论的人不多，国外对此早已认识到（这些都是 80 年代的东西）。

当然，老板也很关心成本，并且拼命地压缩成本，于是员工就成了牺牲品。

真正聪明的老板一定会重视提高生产力，真正重视员工，合理地控制成本，比尔•盖茨就是好榜样。

《人件》里提到一种管理办法叫“赶驴”，我想这样的领导迟早会被扔进垃圾堆，当然前提是驴醒悟了。

下面我再谈谈对公司的期望：

基本的要求：配置液晶显示器，不限制上网（聊天，闲逛除外），有提出建议的渠道和及时的答复。

当然如果能达到下面的要求那就太理想了：阅览室、健身房、培训深造机会、弹性工作制。

我的 blog:http://chenge.blogbus.com

email:chenge99@263.net

bbcallen(无心居士) 2003-07-19

中国啥都缺，就是不缺人

因为缺，所以压缩成本

因为不缺，所以从人开刀

.....

gary_shi(Gary Shi) 2003-07-20

我觉得楼上提到的那条才是中国软件业最值得反省的地方之一。

人力成本不是靠压就压得下来的，另一个方面给员工配备相对好的电脑才是节约人力成本的有效办法。

songyl(天狼) 2003-07-20

问题是中国的经理老总们不知道是看不到，还是认为没有必要，对软件开发人员还是和一般的体力密集型工人一般看，孰不知，让员工心情舒服一点儿，脑力的工作效率会大好多倍，这才真正会节约人力资源。

例如：公司严格的打卡制度，我经历过好几个公司，原来不打卡，公司员工在下班时一般都不急着回家，而且经常加挺长时间的班 ，把手头的活干一段落才走，而公司开始严格打卡后，首先大家都会有一定的情绪，一般都会到点就走，才不管手里的活到什么程度，反正是你要求我现在下班的，第二天继续昨天的工作时经常想半天才能继续。

我们公司尤其可笑的是居然贴出一个公告，说打卡时间是早晨几点的半个小时和晚上几点的一个小时，过时不候，这不是明显提醒大家，不要早到，也不要晚走吗？没见过!!!

Schlemiel(维特根斯坦的扇子) 2003-07-20

液晶屏我倒不太在乎，反正我面对电脑的时间最多只有工作时间的一半而已。另外三点，我们公司做得都还算不错。整体来说，我们公司虽然有些不足，对员工还是算好了。

但是还有一点应该补充的：是否有家具警察。家具警察做两件事：（1）在办公室有限的空间里安排尽量多的人；（2）不允许家具乱放。现在我们的办公室里坐得很挤，连抽烟的地方都没有，让我很不爽。

原先我总是在身边放一把空椅子，谁到我这里都可以坐下来聊聊天，很多工作的问题就在交谈中解决了，还可以随时了解别人的工作情况。结果某一天，一个家具警察（我们的研发秘书）过来把空椅子拖走了——她并不是要用这把椅子，只是不喜欢看我“乱放”。结果那两天，没人陪我聊天了，我解决问题的速度明显下降，而且团队的笑声也减少了。

过了几天，我又找了一把没人坐的空椅子，摆在座位旁边。团队的同事去打水的时候都会从我身边经过，端着水杯就到我这里坐一坐。我的工作效率又恢复了原来的高水准。

Flynner(狂猫) 2003-07-21

《人件》之联想

《人件》关注以人为本的管理。书中所述虽然并非金科玉律，但凡通读过一遍的人必然会有豁然开朗的感觉。从繁杂的软件开发事务中总结出来工程化和标准化的管理法则和度量标准固然不易，但是能够从人的角度全面描述行业的现状同时给出解决问题的建议，我认为是更加难能可贵的。

本来有这样的印象，美国的企业管理总会体现出或多或少的人性化的一面来，以为这是美国人通有的个性所致，在我曾经兼职的公司里，来自 US 的胖老板给我的感觉就不是生产力至上和严重关注成本管理者，相反，他营造的事很宽松的环境，每周五下班之前，他毫无例外的会自掏腰包买足够的啤酒请客所有员工。但是，看了《人件》我有一点新的认识，原来以人为本的管理精神在美国许多的企业里还是很缺乏。人们总是觉得处理技术上的难题比处理人的问题要容易得多。作者费尽唇舌讲明白了这样的道理，花在人性化管理上面的成本绝对会带来更大的效益！

自然会联想到国内的情况，国内的软件业有书中提到的许多同样的问题，比较普遍的情况有，员工的个人工作面积很小、隔板式的工作间既无法阻隔噪音又影响小组合作、把员工（程序员）作为生产零件的工人来调度等等，这些问题既是由于行业的年轻程度和大环境的不景气所造成的，更是企业管理者没有认识到行业的独特性而形成了恶性循环。许多成熟的行业也许在某些方面会显得更加成熟，但是真正实现了以人为本的企业到底有多少或者说有是否存在，我可不知道。在其实稍加联想，在整个的社会系统中，体制上的以人为本的缺失同样可以很容易的从书中的某些例子中联想得到。

每天，我们几乎都会从媒体中了解到许多反面的新闻，许多人首先会以体制不健全、立法不完善来对其加以品评，“有关部门”也会很快的作出这样那样的法规和标准来照应这样的案例。可是这些“事件”无一例外均关乎个人、全体，不考虑人的因素，单单关注于体制岂不是舍本逐末？许多群体的关系都是一种管理，只有对参与其中的人的因素施以更多的关注，这些关系-尽管可能并非是一种追逐效率和利润的关系-必然会正确、快速的发展，否则也会形成许多“恶性循环”。这些“群体的关系”包括交通管理、师生的教学关系、行政管理等等。

中国是一个内敛、关注于实用性的民族，现在正是处在经济发展的阶段，人与人之间沟通交流的失败以及对人的因素的极少关注等等类似的问题是很容易发生的危机，这个问题的解决不仅仅需要社会高层的意识转变来达到，同时也需要所有我们这些普通人来尽力为自己为别人营造良好的环境。我想，《人件》所述应该不仅仅限于软件限于企业吧…

作为一个学生，水平有限，所写只能达到陈述问题的境界，也无触类旁通之功力，只能算作抛砖引玉啦。

yushi124 2003-07-21

在软件开发中人当然是很重要的因素。

在国内有很多的软件公司没有重视到这点，但是也有公司重视了这点，例如我们公司。

我觉得很多小的软件公司其实他的目标只是赚钱盈利，并没有想到去做更大更高的目标。只要有钱赚就 ok 了，当然不会关心更多的事情。而那些有更大更高目标的公司他看到的不是眼前的这一点利益，他有更大的理想，而要实现，人是重要，他看得更长更远。

还有一些公司要面临“吃饱饭活下去”的问题，有些公司是员工加班加点，没有加班费，工资不高，员工抱怨；但是有些公司却能让大家心凝聚在一起，自愿的为公司做些事情，携手共同度过难关。

Schlemiel(维特根斯坦的扇子) 2003-07-20

对了，其实上网闲逛和聊天也是很没必要管的。就拿我来说吧，每天我都会花一到两个小时的时间在网上闲逛，MSN 更是随时在线。但是我们的领导很容易看到：我“闲逛”的网站主要是 CSDN、TheServerside、Java.net、SourceForge、Apache……，我“聊天”的对象主要是 myan 之类的老流氓（公司同事除外）。如果真要深究，我还真不知道闲逛和聊天是不是也提高了我的工作效率。

但是，提醒大家不要晚走我觉得是很有必要的。现在我只是一个小兵，所以我能做的就是一到下班时间就收拾东西、关电脑，然后呼朋唤友，把尽量多的同事拉起来跟我一起下班回家。如果（呵呵，“如果”）我做了部门主管或者项目经理，我会硬性规定手下人不得加班，至少是不到项目上线的紧要关头不得加班。制度性加班是最损害团队生产率的，没有什么比制度性加班更具破坏性的了。

我觉得，光是规定打卡时间还不够，因为总有人会先打卡然后回去加班——真的有很多人喜欢加班，真的。如果我有足够的权力，我会要求每天下班后半小时全部办公室断电，需要加班的部门必须特别申请，由 CEO 或者 CTO 批准才能提供电力。只需要简单观察一下就会发现，不加班的人工作效率更高（不是每小时的工作效率，而是每天甚至每周的效率）。如果你不信，可以自己去观察一下。

songyl(天狼) 2003-07-21

我也认为强制性加班和经常性加班长期来看对员工的工作效率是有负面影响的，但是偶尔的员工的自愿性加班对工作效率还是有好处的，可以把工作告一段落，最重要的是自愿和不能长时间，我认为对加班也不能一概而论，不能认为任何的加班都没有好处，至于下班强行断电之说不敢苟同。

Freeagle 2003-07-21

今天买了 Humphrey(cmm 的领导人)写的《技术人员管理》,书中写道：“改善组织的第一步便是改善组织对待员工的方式”。（希望正在搞 CMM 的朋友看看这本书）

选录几条 p-cmm2 级目标：

为所支持的工作创造良好的环境。

提供必需的资源。

确立双向交流制度，以便增强信任，分享信息。

p-cmm 是对人件的很好的支持。

developerly(阿阳) 2003-07-22

刚刚读完这本书。

关于工作环境，的确很重要，不过，好像就目前中国的现状来说，似乎比较困难，作为一个公司来说，它的目的就是尽量的减小成本。不过在我们公司来说，状况正在有所改变，从上到下，逐渐的都开始换液晶显示器了。虽然缓慢，但也是一种姿态吧。至于办公空间，由于北京的地价实在是太高，所以，让每个人都有充足的空间，似乎是一种比较难的事情。，上地那边情况要好点，可是，又有多少员工愿意去现在来说各项配套设施都不太完善的地方工作呢？现实就是比较残酷~可改变性不大

但是，我发现阅读这本书的很多人，都只看到工作环境这一部分

他们是不是都忽略了这本书所着重强调的另一个内容：胶冻团队的形成。我觉得，这部分才是本书中真正令人振奋的内容。想想，你和你所在的团队，象体育团队一样，或者象合唱团一样，为着统一的目标，为着一种精神，而奋斗，那是多么愉快的经历，甚至可能是会让你一辈子都不能忘却的，模仿《钢铁是怎样炼成的》里面的一句话：“我希望，在我老去的日子里，当我还想着有什么可以回味的话，那就是我和我的团队，为改变这个数字化的世界，做出了应有的贡献”。现在的我，只是一个小小的担当，但是我热爱自己的团队，我愿意让自己成为胶水，凝聚整个团队的力量。

也许这才是每个程序员都能很快做到的事情。

对工作环境，对开会，对 CMM 我依然怀疑（关于大方法论那章，我觉得写的尤为精彩）

对团队，对周围的伙伴，我不怀疑，我真的希望大家能溶为一起，为整个数字化的世界，做点什么，——也许是惊涛骇浪的事情。

carrotmin(carrotmin) 2003-07-22

同意楼主所说的几点

还包括一些细节，如免费提供的茶叶、咖啡

宽松的员工聊天的环境（其实员工聊天多半也是聊工作）

行政人员帮助处理一些杂务…等等

caton2 (听雨弹剑)　2003-7-11

这一段时间，看了三本经典的软件工程方面的书籍。《人件》，《人月神话》，《最后期限》。最先看的是《人月神话》，对我的思维冲击很大。我将其称为是继《设计模式》后对我冲击最大的书。后面的两本也都是经典之作。

几本书都讲的是如果管理好一个项目。《人月》应该说最全面，也比较权威。其中关于计划，人员配置，增加人力，质量等有全面的描述。可以作为“项目经理速成”来看。《人件》让我们从开发人员心理学方面来看问题，把调动积极性，团队建设作为一个很重要的因素。记得我看完《人月》后，一个网友推荐我去看《人件》，二者相辅相成，的确很好。《最后期限》则独辟蹊径，以讲故事入手，讲如何控制好一个项目。不过，感觉这个项目还是有些理想化：他有充足的人力，包括大量可调配的程序员，以及各种优秀的专业人才。更加过分的是：他的项目都不需要需求分析，也不需要和客户的交流。哈哈，这样的项目恐怕很少吧。不过，的确也是一本好书。

相信其他人吧，一本可以被大家认可为经典的书，一定可以给你收获。

看《人月》，我的感想，包括对精英团队的渴望，对质量控制的更加重视，对增加人手方面的思考。而《人件》，是对每个个体的充分尊重，人并不仅仅为了钱而工作，成就感，舒适。建议所有的老板都去看这本书。还有，听音乐编程，呵呵，编码的时候可以听，但作设计的时候不能听。而《最后期限》，不要给程序员太大的压力，那并不能改善工作。还有设计时期少数人，随着项目发展阶段性的增加人手。

还是把几本书作为一个整体来分析吧，因为它们在我的脑袋中，已经互相融合了，呵呵。加上以前我对 XP 的一些认识，总结几条心得吧：

1， 质量是第一位的，不要为了赶进度而降低质量。

2， 作为项目经理，可能唯一的权利就是决定不做哪些功能（scope）.

3， 迭代式开发，高频率内部发布。

4， 不到项目的最后阶段，绝对不要加班。

5， Play to win,not to avoid loss.

6， 较松散的 Pair work,一个人基本了解另外一个人的工作。真正的 Pair Work 成本还是有些高。

7， 自动单元测试。（保证质量）

8， 不仅仅要做 Code Review，在开始一个功能点的开发前，一定要进行设计评审，以免设计有缺陷或与其他部分有冲突。

9， 团队，交流，共同承担责任，不能指责某一个人。

10， 如果一个男生为主的团队，有一个 PPMM 来调节气氛也很不错，呵呵。这一条是我为了凑构整数加上去的。

其实，这一段时间，我看的最主要的书是《Refactoring》，不过现在还没有看完。上周我的业余时间都贡献给了唐浩明的《曾国藩》，觉得曾国藩那么辛苦，终于位极人臣，也善始善终，可他还是很不爽，一辈子快乐得意的日子没有几天。反而不如胡一刀，我前一段也抽时间重新看了一遍《雪山飞狐》，觉得胡一刀夫妇的真挚感情与江湖豪情真是令人神往。我这辈子是没有希望了，但愿我的儿子能出类拔萃，如胡一刀一般潇洒任物，不要像曾国藩那样为功名所累。我的儿子不知道多少年后才出现。:）

“小事以速办见长，大事往往因草率而致误”。这是曾国藩中给我印象最深的一句话。对 IT 业也适用，我们的领导们在启动一个项目时可以想想这句话。

笔随心动，不知所云。

yuechun(加密狗) 2003-7-12

我看了这几本书最大的感觉就是，书中讲的所有问题 90%以上都可以在我们公司找到原型，我曾经给高层推荐了这基本书，也写了一些读书笔记给他们，但是都基本是泥牛入海，他们说，按书中所讲的东西做太理想化了，其实我的意思并不是照抄，而是借鉴书中表达的思想。公司的我现在所做的项目充其量是个程序，根本不是软件，因为自始至终就是写代码，基本没有需求文档，设计什么的，项目管理人员基本不知道什么是黑盒测试，什么是白盒测试，不知道单元测试的重要（根本不知道什么是单元测试,XP 中可能最看中测试了），项目管理人员的目标是如何骗客户，因为他们基本没有责任心，完全就是为了在短时间内如何实现自我利益的最大化，难道中国所有做软件的都是这样？不是用心去做一个东西。不知大家的公司怎么样，谈谈好吗？

plainj(与 SARS 斗争到底!) 2003-07-22

《人月神话》《最后期限》《人件》三本我都有，被这三本书深深感染。

我三年的工作经验里，就能找到很多的相关例子。《人件》上个月我才买到，那个月正好经历了一家软件公司，它是一个活的《人件》的反面教材。

Yeawa

这是我从别的地方看到的一篇文章，希望你看完了之后能做出正确地判断！

行进中开火

时不时，总有一阵儿，我什么事也干不了。

我也去办公室，东瞄瞄，西看看，每十秒钟查一次电子邮件，网上逛一圈。也许干点儿象付运通卡账单之类不需要大脑的事。不过要回去哗啦哗啦写程序，可没门儿。

这种不出活的状态，一般通常会持续一两天。在我的软件开发生涯中也有过几个星期干不了活的时候。就像他们说的，我不在状态，我进入不了情况，我找不到组织。

人人都有情绪波动，有的人温和一些，有的响动大点儿，也有的可以整个乱套。但不管怎么着，那段不出活期似乎总是跟忧郁有点儿关系。

我不由得联想到那些专家说，人们基本上控制不了自己吃什么。任何节食计划都长不了。大家总是悠回各自的正常体重。也许作为一个软件工程师，我也不能控制什么时候最能出活。我唯一希望的就是发呆那段能被哗哗干活那段扯平，最终还能混碗饭吃。

自从我干上软件开发这一行起，我平均每天只有两三个的高效时间。这真让我头大。我在微软实习的时候，另外一个实习生告诉我，他每天 12 点上班，5 点下班。5 个钟头还包括午餐时间，但他的同事还对他特别满意。因为他干的活比一般人都多。其实我也一样。我每天只有两三个小时的高效时间。看着别人那么卖力的干，还有点不好意思。不过呢，我总是组里出活最多的。由此可见，“人件理论”和极限编程都坚持不加班，每周只干 40 小时，还是有点道理的。他们都清楚这么做不会降低一个小组的生产能力。

madgod(空指针·null) 2003-07-24

我们的传统政治习惯就不是基于民主的，“人”从来就是“羔羊”的同义词。

在封建时代况且如此，在生产力高度发达的现代社会，恐怕程序员作为时下最吃香的技术工人会受到更惨重的压迫。

在国外 IT 领航者的身后亦步亦趋已经多年，可是大家仔细看看，发展的最快的是日本的软件外包项目。为什么呢？原因很简单，日本软件企业不需要很多的程序员，只需要很多劳力。

曾经、正在、或即将作为“人件”的程序员，是否考虑过为什么会这样？

不过是成熟的经济学加上成熟的封建政治造就了这一切。

企业不需要太多的优秀设计人员，因为成本高且难于管理--他们总是有很多新的想法，很喜欢改变成熟和稳定的管理结构和管理流程。

可是多数 PM 不会喜欢的--他只想项目按照要求结束。至于程序员在项目完毕以后会得到什么？愉快的工作经历还是对软件设计的新的感悟？这些项目经理都不会关心的，因为他不是程序员；老板也不是。

下面对所谓的项目管理中的名词做个解释：

文档---保证某个“人件”在卸载掉并且更换另外的一块“人件”以后不会造成项目进度的改变，也就是使“人件”做到既插既用，某种意义上，优秀的程序员只是一块高级显卡，但是老板只需要它（就是用的这个“它”，没错）能够在“文档”这个公版驱动下面用就可以了。

士气---严格的来讲，这相当于“人件”的超频指数，士气越高，“人件”在“加班”频率下运行时烧毁的可能性就越小。而烧毁会导致公司花额外的成本另外购置一块，这样老板会心疼的。

培训---刷一下 ROM，干什么用的大家都知道。

想累了，我文笔不好，还有好多想法写不出来，大家补充一下。

另外，提交这帖子以前我环顾四周，我觉得拿板子隔成一间一间的办公室看起来好象……主板一样。

不是这样的吗？人件们？

madgod(空指针·null) 2003-07-24

最后一句说掉了一点，应该是：

办公室像主板，每个人的位置像插槽，10 针的！~（十个手指头就是 10 针）

developerly(阿阳) 2003-07-24

你的比喻做得太好了。

IAmProgrammer(暗黑) 2003-07-24

这个比喻很有趣，而且很贴切

smartcar(历史) 2003-08-06

举个不尊重开发人员的例子：且说说我们公司吧！我们公司是个网络公司，mail 服务是最出名的（大家不要乱猜，这部重要）。我们在 X 楼工作（开发人员大多在这一层楼），白天气温达到 31 摄氏度。可是什么业务，什么*资源的人在 x-1 , x -2 楼工作，舒服得很！

开发，本身也只是工作，也不要什么好条件，但至少，最起码，公平一点点！

qbaby 2003-7-2

现在看来,熟练的国外程序员出卖的是经验，熟练的中国程序员出卖的是体力，看后心寒不已。

openlinux 2003-7-14

一本烂书!烂就烂在不知道这本书是给什么人看的？而且极不适合国情。

在我看来起码不是给开发人员看的。那么是否是给管理人员看的呢？ 看看译者是怎么说的吧："如果您是一位经理，想要寻找“管理”开发人员的牧羊之术，不要往《人件》里找，中国的史书就是一个最大的宝库－－《资治通鉴》里的整人智慧博大精深"。说的太对了。作为管理人员，对开发人员就是牧羊，难道你想让管理人员把开发人员当爹妈养着吗？

peopleware1 2003-7-14

如果您认为世界是两极的：开发人员不是爹妈就是羊，那就去看《厚黑学》吧。

《人件》第 10 章也说道：“很明显，必须抑制诸如此类的异端推理...（老板，）在其他人看到这本书之前烧毁它。

征稿

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

软件管理与软件工程译丛

50%的中国软件公司老板从不关心自己的属下。

—CSDN 调查，《程序员》2003.8

Peopleware

人件

[第2版]

[美] 汤姆·迪马可 蒂姆·李斯特 著
Tom Demarco Tim Lister
UML China翻译组 译
方春旭 叶向群

清华大学出版社

建立完美公司的模式

Linda Rising 等 著，funwave 译

查看评论

简介

本文介绍的多种模式有助于铸造一个完美的公司。我们衷心希望在应用这些模式对公司改造之后，您的公司不再只是一个供人定点上下班的地方，而是一个可以让置身其中的人有不一样感觉的场所，一个可以让人释放活力，成就自我的舞台。

Enron 和 WorldCom 公司丑闻的曝光，让我们有机会看到一种困扰人的领导模式和它所导致的后果，在这样的模式下，以希望、诚信和完美为基础的完整意义的一个公司是见不到的，而惟利是图的公司却处处可见。我们深信，要想赢利， 你必须绝对信得过自己的产品或服务，并且关爱你的顾客或客户。这不是刻板的教条，这是活生生的商业准则。一个真切关注自己产品和产品使用者的公司是能够创造奇迹的。Tom Peters 讲过一个有关麦当劳创始人 Ray Kroc 的故事：有人问 Ray Kroc“你成功的秘诀是什么？”，他答道， “你得有鉴赏汉堡包美丽的能力”。看到这里，你可能先是一笑，稍加思索之后，你得承认他的话很有道理。

完美的公司尊重个人，不论顾客还是雇员，这样的组织透明、协作，坚信良好的人际关系是所有商业活动成功的基础。其组织结构像是一个网络，而不是自上而下等级严明。这样的公司，机会均等，不会见利忘义。[Hefferman02]

列夫托尔斯泰在《安娜·卡列尼娜》一书中写道：幸福的家庭是相似的，不幸的家庭各有各的不幸”。我们进行了一系列采访，都以“你有一个完美的公司吗？”开始发问。其他提问则围绕着“对你而言，完美的公司意味着什么”展开。最后，我们找到了一家完美的公司所应具有的共性，被访者的回答就是答案。

Arie de Geus [deGeus97] 把公司分为两种。第一种，他称之为经济型公司，纯粹为经济目标而运营，在这样的公司里，人也是资产，公司的目标就是以最小投入获得最大产出。经济型公司意识不到它对整个社会负有怎样的责任，只是一台为少数经理人和投资者盈利的法人机器。在第二种公司，投资回报当然重要，但这种回报是为了服务于人。为了盈利和生存，企业要进行如下活动：定义角色及分工，建立共同的价值取向，招聘人员，职业培训，资历评估，履行合同，处理与外界、承包人的关系，制定完善的离职政策。公司的每个成员都明白到

个人的价值与公司的价值同生共存。De Geus 把这种组织称为“活的公司”（Living Company）。我们则称之为“完美的公司”。

年仅十岁的 Lilah Polewka 有一家养老鼠的小公司，这个公司每个月可以卖出几十只品种珍奇的老鼠给本地的宠物店。Lilah 荣获 2002 年度青年成功企业家（Junior Achievement Entrepreneur）大奖。她对其他企业家的建议是：享受乐趣，不断学习，与企业共同成长[Jr02]。说得好！

其实每个人都具有企业家的潜质：人人都想要解决问题， 迎接挑战， 发掘自身的潜力。[Carbonara97] 换句话说，我们都愿意做些把可能变成现实的事情，而这正是一个企业家应该具有的素质。

开着手做这件事情时，我们买了很多书来看，想尽可能掌握在这方面的知识。这些书与我们展开了单方向的对话，他们说， “看看我们在 Lucent， 或 IBM，或 GE，或 HP 的所作所为。”但书中的故事可能令你我汗颜，而且并不一定适合你。而我们这本书面对正是刚刚开创了一家中小型企业的企业家们，市面上已有的图书大都忽略了这部分读者。

尽管坚信书中提及的诸多模式可应用于任何规模的企业，我们还是主要针对小型企业，所以，对于一般情况下小型企业主操心不着的问题，本书不直接涉及，比如与股东相关的问题。

对于赢利和成功这类任何企业都关注的话题，本书也不直接阐述。我们深信：不论一个企业的产品是什么，属于何种行业，也不管企业家个人对成功的定义是什么，这些模式都是达到你所期望目标的钥匙。在某种意义上，主要遵循了这些模式，盈利和产品的成功指日可待。

大多数人以为自己所面临的问题是绝无仅有的，而我们发现小企业主们面临的很多问题都是类似的，区别不在于产品是什么，他们面临的都是人的问题。

在模式领域有丰富经验的 Linda Rising 和 Daniel May，与商业教练 Caroline King 和 Steve Sanchez 一同研究了如何运营小型企业的问题。Caroline 有 20 年以上的担任小型企业企业顾问的经验。Steve 于 1990 年在亚利桑那州的菲尼克斯（Phoenix）创办了自己的企业 Master Marble， 从事商业教练和顾问也有超过七年的历史。

Linda 在 2001 年的一个圣诞晚会上遇到了 Steve，对模式和公司管理的一番深入交谈之后，他们决定写一本书。不久， Steve 又把 Caroline 介绍来，他们一直在从事模式的研究，而 Linda 一直在研究小型企业。Daniel 编写过两套模式，对模式和完美的公司有丰富的经验，他远在丹麦，是本书中唯一不住在菲尼克斯的作者。

致谢

感谢 PloP’02 的项目负责人 Bruce Whitenack 的鼓励和宝贵意见，感谢 VikingPLoP’02 项目的负责人 Klaus Marquardt 启发我们从全球的角度观察问题和对我们的模式的帮助。

模式语言的结构

完美的公司

你想要你的公司与众不同，制定一个完美的目标，发挥卓越的领导才能，创建优美的环境，聘请一流的人才，吸引一流的顾客和友好的企业外界人士。

完美的目标

通过制定目标为你完美的公司指明方向，不是去描述一项产品，而是更深层的东西的描述。

卓越的领导才能

只有用心，你的领导才能才会有所长进，为你的完美公司植入一颗灵魂，这一切从了解你自己开始。

世界很小

有一颗开放的心，寻找分享兴趣的机会，结识更多的人。

知道自己的短处

在一个小团体内，你以为自己必须无所不知，其实知道没有人无所不能更重要。

合适的教练

当你进退维谷，不知如何继续时，找一个好的教练吧。

优美的环境

创建一个宾至如归的工作场所，人们可以有安全感，并能自由成长。

组织的诚信

优美的环境依赖于盈利和道德之间的平衡。在这样优美环境中的你的公司和你，同样适用于这条原则。

一流的人才

想要网罗到一流的人才并充分发挥他们的才能，就要找到最适合的人。欣赏各人不同的才能，保证他们的才能可以发挥到极致，不要担心有一天他们会离开。

合适的人做合适的工作

想要找到一流的人才，就要广而告之你的要求。世界很小，相信你的直觉，不要只相信简历，试用一段时间。

试用

想要知道你准备录用的人是否适合，试用一段时间看一下。

各种才能

各种不同的一流人才发挥他们不同的才能才有公司的美丽。

才能最大化

想要你的职员发挥出他们的最大才能，就要以对待志愿者的方式对待他们。

从容辞职

公司由人构成，而人总是处在不断的发展变化之中，要知道天下没有不散的宴席。

改变谈话方式

与价值观不同的人打交道，要改变你谈话的方式。勿以怒制怒，让对方感觉到你的真诚。

完美的顾客

以诚待人并期望你的顾客同样真诚。

友好的外部人士

与公司外的人打交道时，要把话说明，并遵守承诺。

完美的公司

留心一下，你会发现一家完美的公司的不同部分之间协调运转；开会时气氛平和；各个部门都高效运作；工人、管理人员、用户和合伙人，都心态良好并有团队意识；管理人员、工人以及雇员之间相互信任，活力充沛地为共同目标奋斗着。而一家不完美的公司，雇员惶恐不安，管理人员苛刻、无情、官僚。工人觉得自己在进行一次走向死亡的行军。--Bruce Whitenack

你新开创了一家小企业或是一家正在运营的小企业的管理者人之一。

你不想另开一家公司；你希望你的公司有特色、完美。

书店里有多种多样的关于如何创办一家企业的自助类书籍，读者挑花了眼，不知到底该信奉哪家哪派。这世界变化快，想要学习的时候，你不用担心找不到被不断改写的新规则。基本原理和基础在更新，成功的定义也在变，就算是好的设想也需要有人来指导。如何开始？如何继续？需要帮助时找不到帮助，于是我们自然而然回到老路上去。

一家目光远大的公司的层出不穷的优质产品和优良服务的根源正是这家优秀的公司本身，而不是其他。不管多么曾经多么辉煌的产品、服务和构想，终将成为历史，而一家有远见的公司，只要它能领先于现有的产品生命周期，不断去创新，发展，那它被淘汰的概率就会减小。 [Collins+94]

因此：

制定一个完美的目标，发挥卓越的领导才能，创造优美的环境，聘请一流的人才，吸引一流的顾客和友好的企业外界人员。

在 2001 年 1 月我在北亚利桑那州与朋友滑雪的时候，摔伤了膝盖，于是不得不去看整形外科医生戴夫，分手时我们似乎已经变成了多年的老朋友。他为我做了两次膝盖手术，并一直悉心照料我。后来戴夫医生成了我们的顾客，他为他在山里的房子购买了花岗岩的厨房工作台面。从设计到完工，我和他的妻子参与了所有细节。直到完工 2 个月以后，戴夫博士才有机会见到整个工程。我约他见个面。走进检查室时他说“我有些东西要给你”，转身离开房间，回来时候交给我一张支票，已签字，但金额一栏空白。他说“我不知道需要多少钱，所以你来填

吧”。回到办公室以后，我把这张支票给员工们看，同时留心他们的反应。大家都明白了所谓完美的公司究竟是什么样子。--Steve Sanchez

完美的目标

我在 1990 的秋天创办了 Master Marble，亚利桑那州菲尼克斯的一个天然石加工企业，当时仅有两个雇员。我自认为颇具创造力，可以从无到有，从少到多生产出产品来，但事实上我的特长是产生构想，与他人分享，然后把蓝图变成现实。--Steve Sanchez

你意识到成就一家完美的公司不仅仅意味着生产一项产品、为客户服务和赚钱。

没有地图你如何确定前进？

企业家有时会陷在细节里而看不清大局，这种情形下要找到问题所在并不容易。人们还是做那种老式的烘肉卷——也许，他们曾设想过要做得更快或是更便宜，然而还是走不出自己的老套路：还是用那些料，却希望做出不同的口味。可能吗？

公司要有自己的立场，要有无可争辩的最低标准，必要时他们可以说：“违背标准的工作我们不会接受，因为它不合乎我们的宗旨”。[Webber02]19 世纪的哲学家托马斯卡莱尔（Thomas Carlyle）说，“人活着得有原则，不能活在讨价还价中。”有原则的人的生活会很简单，他只需照章办事而不必浪费时间和精力去讨价还价。[Marriott+97]

人贵有自知之明，一个公司和每个员工也应当明白“我们是谁”，这并非无足轻重的小事。可是企业的自知之明并不那么容易达到，那么降低一下难度，企业至少应该知道“我们不是谁”，当后者都达不到时，企业就无法对某个方案做出类似“这不符合我们旨”的判断，这样的公司显然缺乏判断力，而判断力意味着远见。必须有个人从骨子里明确“我们”是什么、不是什么，这种“明确“来不得半点虚假。狗可以嗅出恐惧一样，当一个公司缺少“判断力”时，员工们怎会感觉不到？[DeMarco01]

“创办企业之初，如果不制定一个众人都认可的目标，索性卷铺盖回家。就像“我们是美国的公民， 为了缔造一个更完美的联邦……”中的“目标”一样，一个公司的目标必须是对公司性质的准确描述。[Waldrop96]

因此：

制定你的目标。不是指定义一项产品进行，而是对更深层的一些东西的描述。

如果有人问起你的公司做些什么，一个完美的目标会让你发出源自内心的微笑。完美的目标能够点燃、激发、打动和创造人心灵深处美好的感情，使人感觉充实，有安全感，并且总是很兴奋。

确定并且描述你的目标并不容易，你需要时间，正确的指导有助于你完成对目标的指定。请先回答以下问题：

- 作为一个公司，“我们”是谁？
- 我们的目标是什么？
- 我们提供什么产品 / 服务？
- 我们想成为什么？想以什么闻名？
- 为什么我们是特别的？
- 我们以何为荣？
- 我们的产品 / 服务有什么不同？
- 我们在乎什么？

大卫·帕卡德（David Packard）提出了下列问题：“首先，我想讨论一个公司为什么要存在，换句话说， 我们在这里是为了什么？我想许多人错误地以为公司存在就是为了赚钱。而实际上，赚钱只是一个公司的存在的重要结果之一，所以我们不得不更深入地探寻公司存在的真正理由。”正确回答下面这个问题有助于你找到不论是现在还是 100 年以后都站得住脚的答案：“为什么不关掉公司，兑换现金，甩卖资产?” [Collins+94]

不要只是鼓吹这些理论，着手建立的具体的机制让变化和进步成为现实。[Collins+94] 一旦知道了目标，你就明确了方向，现在就制定你的规划和任务，把你的蓝图变成现实吧。发挥卓越的领导才能，创造优美的环境，聘请一流的人才。

Century Roofing 有限公司，亚利桑那州的一家在屋顶行业有着超过 30 年经验的公司，长久以来，本着友谊和诚信的原则，我们与客户建立了长期的良好关系，并以极具竞争力的价格提供最优品质的房顶系统。

身为行业的领军企业的Master Marble有限公司，是一家出品美观耐用天然石材的公司。Master Marble代表了优质的客户服务，精湛的技艺和创新精神。在双赢的环境中，我们繁荣壮大。

Ginger L. Price, D.D.S. 一个专业的牙齿美容团队，我们承诺以最贴心的方式为珍视自己健康的人群提供最全面周到的高品质服务。

小型的服务性企业唯有遵从以下规则才能生存于激烈的竞争中：

- *出色地完成工作，并把本企业的创新之处作为宣传企业的要点。*
- *吸引令人激动的人 – 而不是那些与众不同的人。*
- *一日三省吾身，不要自满，今天的桂冠不过是明天的粪堆。*
- *那些有意或无心离开的人，确保他们在这里有所收获，有过特殊的经历，并且交到了朋友。他们会帮你扩大企业的知名度！*
- *通常企业都存在的复杂的派系争斗，我们的企业里没有这种现象，有的是一个协作的、充满朝气的，幽默的，充满欢笑的团队，人们互相支持。*
- *不要让怀疑和谣言蒙蔽了我们的良知。*
- *注重小节，及时接听电话，发送正确的发票，勿忘错误总是发生在细节之中。*
- *与令人兴奋的顾客和搭档一起工作，他们会带给我们好心情，从他们那里我们可以学到东西，享受与人合作的乐趣，当然他们还得是准时付帐的人。*
- *量入为出，支出包括中上水平的员工薪水和未来较高水平的投资。*
- *在为顾客提供优质服务的过程中成长壮大，成长不是为了什么好处。*

卓越的领导才能

企业的领导着好比一艘船的船长。不是每一个船员都有机会跟船长直接打交道，但每个人都在关注着船长的一言一行。领导者的所作所为会对企业文化产生潜移默化的影响。当然不是所有的领导者都称职。我见过这样一些领导：他们安于现状，人云亦云，但求稳定，大权紧握，隐于幕后。这样的领导是不是更像保管员和一般的行政人员？显然，领导该做的事远远多于这些。一个船长要善于领导，迎接挑战并鼓舞士气。她做出危险的决定，

但是事事四平八稳，那么我想她根本不是在做领导工作。我想大多数雇员都明白：每一个决定都完美、正确，那是不可能的，他们期望的是船长展现出他卓越的领导才能。

你是一个企业主，想建立一家完美的公司，你已经为公司制定了一个完美的目标。

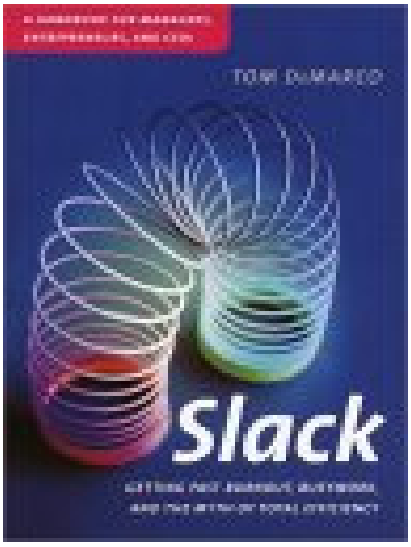
如何营造并保持一种既富有挑战性、又能培养锻炼人的公司环境？

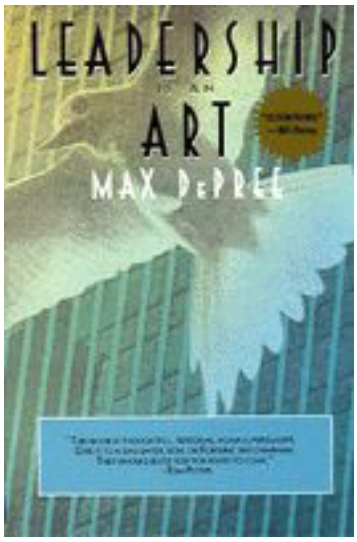
我不由地想起那些我景仰的领导者们，他们都虚怀若谷，而这正是领导者所应具有的最重要的品质。可以用多种不同的方式去营造这样一种气氛：人们之间可以正常地交流。把员工变成以优质工作为荣的人；向员工灌输这样的意识：在某种程度上，我们的团队，是精英，是世界上最优秀的队伍。让他们思考“诚信”的意义。无论是什么，总要有种东西能把大家凝聚在一起，我认为这种东西就是团队的“灵魂”。人，生来就渴望成为某个集体的一部分，作为集体一份子的人，能做出比单独的个人更好的工作。如今的现代化社会，并不存在那么多的团队可以让人加入，对于我们中的许多人而言，最顺理成章可以参与的团队就是我们工作的团队。因此，在组建一个团队的时候，一定要把“灵魂”注入这个团队。“灵魂”好比种子，种下“灵魂”，团体才能开始成长。[DeMarco97]

以人性化的方式让员工释放其聪明才智用最有效手段的完成该完成工作，这就是领导的艺术。领导要为员工排除工作的障碍，真正的好领导能使手下意识到自己的潜能。要做到这一点，领导是个明白人，他们必须考虑一些非常重要的问题，比如人的本性，组织的定位，业绩的考核等等。领导人必须有自信鼓励别人说出不同的意见，这些意见是保持企业活力的重要源泉。真正好的领导是一个倾听者，能够听取别人的想法，需要，抱负以及手下的愿望，然后运用自己完善的理念体系给予恰当的回答。这就是为什么领导必须有自知之明，为什么需要倾听别人的意见。[DePree89]

领导者要做的第一件事是描述现状，最后一件是致谢。在这中间，领导是其部下的服务员和债权人。这样三个步骤周而复始，一个优秀的领导就成长起来了。领导能力的学习需要时间，信念和心血，是一门艺术，而不只是完成一堆任务。[DePree89]

人们需要被领导，这个领导必须与他们的理念相吻合，并具有对现实事物的洞察力。当一个组织随波逐流如汪洋中的一条船，突然有人愿意掌舵，心底石头落地的声音每个人都听得到。因为人们想要被领导，所以领导存在。[DeMarco01]





今天我们在考虑人和组织关系的时候，存在着一个严重的漏洞：人们少有耐心进行一番思想动员，而这是一场意义重大的变革的不可或缺的前奏之一。只想着如何达到目的，反而达不到目的，所以我们左右为难，迷失方向，动弹不得。这就是问题所在，答案也在这里。林林总总的咨询公司、书籍、杂志和培训课程全都是为了解决这个问题。为什么标题里带有数字的书和杂志卖得好，原因也在这里。如果总是避而不谈那些基本的，底层的问题，我们如何能在两难中做出取舍？这时，单纯的技术并无用武之地。若想有所作为，另两样东西必不可少。首先，要进一步了解自己，从哲学的角度讲，你得明白作为一个人存在于这个世界上意味着什么。其次，要改变思维习惯：如何思考，重视什么，怎样工作，怎样与人打交道，怎样学习，对生活的期盼，以及怎样面对挫折。这些改变标志着向智慧的转变，标志着从非领导者到领导者的转变。[LaBarre00]

因此：

用心领导并将“灵魂”注入公司，一切从了解你自己开始。

己所不欲，勿施于人。[Waldrop96]好的领导愿意让大家分享自己的理念，激发群众的信心，激励着大家朝着共同目标努力。领导能力意味着让人各司其职。领导的安排可以让人们情愿为了更长久的发展而牺牲一时的利益。因为大多数人都是缺乏远见的，所以我们更需要有长远眼光的人。[DeMarco01]

真正的领导需要做到如下几点：

- 明确目标。
- 承认一时利益的丧失。
- 行动
- 行动
- 还是行动。

说话时天花乱坠，谈到具体措施就哑口无言的话，这充其量是个做秀的摆设。 [DeMarco01]行动当然应该包括为达到预定目标的而采取的相关举动。

世界很小，你要从每一个认识的人那里寻求帮助。知道自己的不足并且能够找到正确的指导，所有的拦路虎都将是纸老虎。

世界真小

我一直都吃惊于这么多的机会和问题都是在一种很偶然的方式下被解决的。当和人们谈论在公司里我们正面对或是正在处理的一些问题时，结果，总是有人会说，“我知道这个问题”或者是“我或许知道谁能帮你。”在这种方式下，问题被频频解决，新的思想，联系和机会被创造出来，这实在太离奇了。以为柳暗无路走，结果却是花明又一村，你会学到新的有价值的东西或是遇到有意思的人，或者你的新合作伙伴或者你的雇员。有人曾经说过公司实际上是个展开讨论的地方，我觉得这话有一定道理。--Daniel May

你是一个企业主，想建立一家完美的公司，你正在锤炼你的领导能力。

如愿以偿，获取新知，结识新人的捷径是什么？

各种各样团体、组织，跟人的圈子一样，有交流才有运作。[Senge+99]

1967 年出版的《现代心理学》的第一版中，Stanley Milgram 描述了他的“小世界”试验。随机抽取世界上的两个人，需要多少个熟人才能把这两个人联系起来？比如从内布拉斯加州开始，测试对象向位于波士顿的目标对象邮寄文件。内布拉斯加州的发件人要认识收件人才可以，然后收件人再向跟目标对象关系更近的熟人寄邮件，以此类推。难以置信的是，Milgram 声称只需 5 个人，6 个环节就能联络到一个随机的陌生人。但是，Milgram 这个令人震惊的结论由于缺乏证据而被世人冷落。这种“6 步到达”的思想事实上有一个明显的错误：在理论上等同于一个城市神话。抛开 Milgram 的实验不谈，他的有些证据还是可取的。当说我们说“世界太小”时，指的并

不是随机两个人相关联的几率，而是指遇到一个能帮我们达到目标的人的几率。一生中，这样的几率是很高的，特别是在受过教育的同行中。当一个不太可能的联系发生时，无论是否有科学证据，你都会感慨世界真是太小了。[Kleinfield02]

因此:

保有一颗开放的心，寻求分享兴趣的机会，寻找联系。

那些你每天都见面的人其实是帮助你达到目标的捷径。如果你正在失业中，他们能帮助你找到一份工作。大多数的工作都是通过朋友或是私人关系而找到的。如果你想招人，私人关系则能帮你找到最佳人选。

当与他人建立了联系时，要注意保持。让朋友和熟人了解你的兴趣和需要，这没什么好害羞的。他们或许暂时帮不上你，但他们会向别人传达你的需求或是为你引荐其他人。

我们签了一份租房合同，是新房子，但是楼下的租户封了房子—所以我们不得不放弃这个计划。因为已经告诉房东我们走了，所以从 8 月到 11 月我们就没有工作室了。我先给所有的熟人打电话，告诉了他们这件事。一个朋友说她的办公室周五空闲。另一个朋友让我周二去工作，让我的伙伴周三工作，然后我们周五一起在另一个工作室工作。这真是一场考验，如果没有这些关系或者朋友之间没有联络，那么我们的顾客很可能已经去了别家。关于我们是谁，朋友的作用，我想这件事说明了很多。[Price02]

了解自己的能力

最初几年，我事无巨细，紧抓不放，总是担心自己做得不够好，这是我所无法容忍的，我也不能容忍公司别人做出的一些我看不上的事。我试图强迫每一个人按照我的方式做事，花费了许多精力确保他们按照我的思路做事，比花在结果上的精力多得多。我害怕自己领导下的产品和公司出现不好的事情，那样会显得我太无能！所以对待属下，我挺疯狂，好迫使他们跟我一样。我就是这样过来的，这种状态持续了10年。我从不说我聪明。--Steve Sanchez

你是一个企业主，想建立一家完美的公司，你正在锤炼你的领导能力。

在一个小团体里面，你觉得自己必须事事精通。

作为一名企业家，你认为自己什么都知道，害怕放权。不过要运作一个公司，你就得撒手让底下人去做。有的领导不甚掌握具体业务，所以担心自己无法更好地领导工作。其实任何工作都牵扯许多细节，不可能人人都懂，所以根本无需担心。你能坐到这个位置是因为你有这个能力，当然也不要无视你的短处。成功人士清楚地知道自己的长处，并且把自己的长处发挥到极致，同时积极寻求机会历练自己其他方面的才能。发挥自己的长处，业务方面的缺陷不会成为你的绊脚石。[Kruger99]

好的领导必然有自知之明，我就是错在这里。我对自己的知识太自信，以至于视而不见那些证明我所知为错的证据。[DeMarco97]

当试图掩饰知识上的不足时，你会觉得向别人请教是一桩别扭的事，因你觉得自己早该知道。告诉你一个秘密：没有人知道所有的答案，没有人。无知并不代表你无能，不懂得请教才是。如果不理解别人给你的答案，追问下去，直到你明白。提问不仅有助于你了解下属每天的日常工作，同时也有助于你把握下属的劳动强度。最后，提问也可以帮助下属反思自己的工作，从而改进。[Hendrickson02]

接受这个事实：你不可能什么都知道。你也许觉得对于某些事无知是件丢人的事，其实不然，可以一分为二来看待。意识到自己保有一颗开放的心会使你感觉到自由和力量，与你不知答案时的感觉恰恰相反。

为弥补“不够完美”而迫使自己加班加点其实不划算。也许一段时间内你的产出会提高25%-30%，获得1/3人年的工作量。但是如果你能创造条件让10个员工每人提高10%，你获得的将是1个人年的工作量。[deGeus97]

因此：

了解自己的不足。没有谁是无所不能的。

首先要做的是承认自己有所不知，夸大自己的能力只会弄巧成拙。可以学习一下下属的工作，但不必精通到可以替班的程度，能预见下属的需求，可辨别其言语的重要性即可。如果你不知从何学起，问问你的下属吧。不通晓专业领域的知识不是个大问题，明白如何领导别人才是真正重要的。[Hendrickson02]

一条检测你不足之处的线索：抉择某事花费很长时间。当相关知识完备时，作出一个决定会很快。长时间的考虑往往意味着信息的欠缺。去寻求帮助吧。[Manns+02]

承认有所不能并委派专家打理，你将摆脱诸多限制，仍然对产品负责，你的不足不是障碍，完全可以继续前进。

“虽然Bob很聪明，但我喜欢他是因为他的自知之明。傲慢得不去请教他人，Bob才不会这样做呢。因为他自信，所以他勇于学习别人的长处。”[Fishman01]

我不可能具备足够的知识去做任何事，但我坚信只要认识到这一点，我就会找到会做的人，然后把事情交给他来做。我仍然对结果负责，并且可以摆脱自身的不足。正因为这样所以现在我可以管理我的公司。--Steve Sanchez

这有一些企业家们的经验教训：你不可能一个人做全部的事情。我会列出自己的长处和不足，以确定哪些事情应当我自己做，哪些我需要雇人来做。[Kurtzig91]

我的经理不知道什么是“宏”，但他知道客户需要什么，知道如何激励开发者去满足客户的需求。他帮助我们理解什么是客户需要的东西并且改善了销售与开发部门之间的关系。[Hendrickson02]

正确的指导

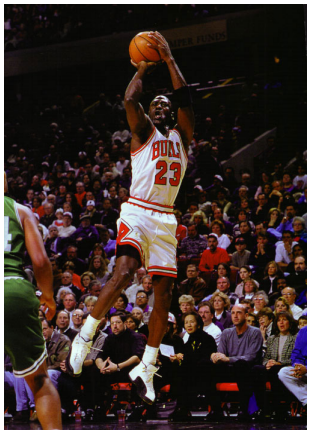
创业之初，我总是担心自己的经营方式是否正确。我听说过 SCORE（Service Corps of Retired Executives），退休主管服务公司， 是美国小型企业管理部（U.S. Small Business Administration）的智囊搭档。他们给我配备了一位顾问，这个顾问曾是一家钢铁加工企业的所有人。他比较了我公司的产品，但效果并不理想，因为我们只是就事论事，围绕着产品展开讨论，6 个月之后双方都失去了热心。之后一个朋友把我介绍给卡罗琳。她问的都是不一样的问题，你想成为哪种人，在做什么，没做什么。首先，她转变了我，然后，她把我的转变带入了企业。她仍在指导我 ——甚至在 7 年以后，而我仍有新的突破。现在我明白：生产什么无关紧要，组织体系，管理和环境才是决定因素，而重中之重，是人。--Steve Sanchez

你是一个企业主，想建立一家完美的公司，你正在锤炼你的领导能力。

你不知道如何才能进步，不知自己的所作所为是否正确。

转变的实现可分三步。第一个阶段只能掌握皮毛。（这倒也符合逻辑，有了想法，接下来就要试一下。）第二个是实践阶段，这个阶段会给你带来压力和焦虑。（我还不能完全吃透，不知自己做得对不对。）第三个阶段会使你去反思，而反思是痛苦的，能坚持下去的话，质变就会发生。（变化的不仅仅是组织，而是我。）这第三个阶段是艰难的，因为这个阶段的关键就是你。这个阶段最难实现，然而孕育着胜利。当你的公司面对挑战而你是其中一员时，最容易发生变化的一定是你，虽然这也很难！[Pascale+00] 企业家们容易一叶障目，不见泰山。由于缺乏经验和企业运营的技能，他们的公司往往会遇到问题，到一定规模之后，便很难再发展壮大。

而顾问就象一面镜子，没有了镜子，人们也能梳洗打扮，但却不一定适度得体。泰戈·伍兹（Tiger Woods）就有好几个顾问。迈克尔·乔丹(Michael Jordan)也说，如果没有了教练，他就不打球了。而帕瓦罗蒂（Pavarotti）则有一位表演教练，一位声音教练，一位音乐教练，一位语言教练，和一位健康教练。身穿 50-100 磅的衣服， 演唱 2 个小时， 没有扩音器，却要让 5，000 个人欣赏到他完美的声音，而且在演绎某些难度大的角色时，不能使用他的母语意大利语。从一个有着好嗓子的人到歌剧界的传奇人物，教练的地位举足轻重。教练能够发现你的问题所在，表现如何，并指导你做新的尝试，从而取得进步。


因此:

寻找一位好的商业顾问。

为了你和你的企业，可以向其他公司的人征求意见，多会见几位顾问并从中找出最合适的人选。可以试用一段时间，签订半年到一年的短期协议，设定几个战略目标，看看试用期内能否实现预定目标。

Master Marble 派生出了六家互相竞争的企业，这些企业可是来者不善。六家企业都没有聘请顾问，结果五家企业在 2 年内倒闭，唯一幸存者曾在 Master Marble 从业多年，并且在他是个普通职员的时候受过培训。Steve Sanchez

Caroline King 曾经为某位顾客做了 3-6 月的时间的顾问，后来这位顾客负不起费用所以不再聘请她。结果她离开后六个月，他的企业就倒闭了，只好去给别人打工。他的失败不在于企业是否赢利， 只不过他不愿去做使企业成功的事。有些企业一旦没有了顾问，很快衰落。顾问工作的实质是教导你进行自我对抗，只有不断战胜自己，超越自己，个人才会进步，企业才会进步。Caroline King

创业时，我感到自己内心涌动着一些东西，但是抓不住。现在我意识到了，它们就是那些使你的公司完美的东西。如果你的内心有这样的东西，抓住它，否则，你只是盲目地忙碌着，永远也到达不到目标。企业就是要赢利，要领先，要前进。开始时，我做不到在商言商，但有了顾问以后，我可以用企业家的头脑去思考了。起初的挣扎不可避免，之后就会发现这种转变正是你所期望的。一旦接纳了它，它就会开始影响你企业的方方面面。我也考虑过是否喜欢自己所做的事情，审视你所提供的服务，必须确定自己是否热爱它。要不停地探索，不停地学习，不停地变化。发现了新的兴趣所在，就要与新的兴趣相结合，而顾问能帮助你实现这一切。[Rike02]

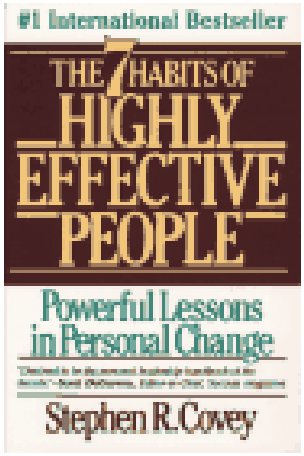
改变谈话方式

我给我的客户，一位承包商，提供了一份报价单。这位承包商又把合同转呈给他的客户，一位业主，业主接受了这份合同。第二天，那位业主打来电话要压价，我不肯让步，她于是不快。她吹毛求疵地要在这份价值$13，000 的合约中减少$250。如果她说打个折，我肯定早就同意了，但她口出不逊，所以我不会让步。以其人之道还至其人之身。过后重新思量了一番，我又给承包商做出了$350 的让步。我致电给业主：“我为自己上次的粗鲁无礼深感抱歉，希望项目完成之后你会喜欢我这个人，我的公司和我们的产品。”她说“谢谢。”，看来她也意识到自己的不妥了。之后我们又有数次关于项目的谈话，都很愉快。我放弃了以怒制怒的方式，我明白我的公司将准时、经济地为客户提供多样化的优质服务，今后各种各样的顾客还多着呢。Steve Sanchez

你试图创建一家完美的公司，希望网罗到一流的人才，遇到好的客户和外部人士，可希望只是希望。

有时你不得不与那些和你志不同道不合的人打交道。

“洗心革面”意味着从自我开始，甚至是从一个人最基础、最深层的地方开始改变：范例、性格、动机。若想拥有美满的婚姻，就得做一个态度积极的人，而摆脱消极的情绪，更不能让消极的情绪控制自己。若想你十来岁的孩子友善，协作，你得是一个充满理解力的、言行一致、充满爱心的父母。想在工作中拥有更多自由，那就成为一个有责任心、愿意帮助别人、有更多贡献的雇员吧。若想被别人信任，就得做一个值得信赖的人。[Covey89]



因此：

改变你谈话的态度，与其以怒制怒，不如让她感受到你的真诚。

改变你看待事物的态度，积极一些。每个人都有优点，不要苛求别人，那些挤压、挑战、逼迫我们的人恰恰给了我们一次从不同的角度看待事物的机会，这也是很好的一课。

你可以说：“一个与在智力、能力、责任上不相上下的人不同意你的观点，那他自然有你还不曾了解的原因，不妨去了解一下他的观点和依据。”当有人不同意你的观点，你可以说：“很好，你有不同的思维方式。”不必非要同意他的观点，你可以去肯定他，尝试着理解他。[Covey89]

别指望征服所有的人。有时最好的结局就是让你的对手自动放弃。

在模式讨论会上，我们几个人对几位作者的文章进行讨论。我们有一套严格的流程，试过你就会发现这套流程的合理之处。介绍之后，先要说出文章中值得肯定的地方。我有过数次这样的经历：一些拙劣的文章很难找到可圈可点的地方，当勉强凑出几句赞美之词时，我总是会对别人的评论感到惊讶，他们总能找到我视而不见的一些精妙之处，而一旦被他们捅破，我就会发现那的确有值得称道的地方，进而使我发现新的精妙之处。评价因人而异，所以会比较客观。虽然这是规定的流程，却总能带给我意外和惊喜。Linda Rising

我们全家都在同一家公司，但是我却强迫我的弟弟离开，我们总是意见相左。我告诉他他别在销售部门干了，如果他还想继续呆在公司里，就到加工车间去吧。他选择离开。他离开的时候我不知会发生什么。他总是压价往外卖东西，我担心公司因此要垮掉。我权利应对他签下的那些订单。电话响起，我发出不耐烦的“你好！”，是 Caroline 打来的。我担心自己的情绪不够好，的确不好。难道她不知道我都快崩溃了吗？她的话使我恢复了常态。她说如果我认为自己已经崩溃、事情无可挽救，那好，都会变成现实。她建议我对自己说：“我遇到了一些困难，但是我可以处理。”这就不一样了。如果你想要更好的肉馅糕，你必须改变一下调料。Steve Sanchez

完美的环境

在咨询公司里，作为顾客，难免没有安全感，总怀疑自己被某人注视着，这种感觉可不好。我喜欢回到办公室，看到熟悉的面孔。在会议室里喝点什么放松一下，或是插入别人的谈话中，讲讲战争故事，再或者和别人讨论技术难题或是某个刁钻的客户。我们的小卧室里有白色的书写板，我们可以像顾问那样画个图表达自己的想法，而开放的办公空间可以让你看清谁来了。有一个图书馆，可以在那查阅资料，或是和某人密谈。很明显，这不仅仅是一个工作的环境或是一间办公室，你还可以在这里见到你想见的人。每次我走进大门，我都会想：今天会发生什么？而每一天都令人兴奋。Daniel May

你是一个想创建一家完美公司的企业家，已经制定了一个完美的目标，开始发挥卓越的领导才能并且准备开始招人。

怎样才能吸引并留住那些一流的人才，友好的顾客，和完美的外界人士呢？

营造一种有益于解决问题的工作环境。[Rothman01]环境应该能够或者激发人，使他们达到最好的状态，不断地提高技能，增强竞争力，融洽人际关系，灌输主人翁意识，畅通无阻地交流。环境应该是个实现潜能的地方，是个人们常来的地方，在这里可以有效地、人性化地联系他人，或是交流技术。[DePree89]

因此：

营造一个宾至如归的空间，让人安心，自由地成长。

环境的好坏取决于组织的诚信。

我们尝试着让工作变得更有趣：下班后步行去打棒球、保龄球或是篮球。我们希望工作不再乏味、不再那么机械化，而是多些社交的乐趣。人们不喜欢自己变成机器人，所以如果心情愉快，他们就会认真地对待差错，友善地对待他人，工作质量就会提高。因为，他们知道那个人就是和他一起玩排球的伙伴。[Hammonds02]

在牙科诊所里，人们都会有恐惧心理，而我们的病人告诉我们，他们进入我们的办公室，感觉很好，这里就像一场聚会，每个人都叫得出你的名字。因为我们一直尝试与病人建立起良好的关系。病人初次就诊的时候，组员会多寒暄两句让病人觉得舒适，这样无形中就减弱了矛盾。[Price02]

组织的诚信

一家小型软件公司与一家银行签订了一份为期 5 年的软件合同。行长对这份非常满意合同，下属员工却不这么认为。一个月之后上任的一位新行长，他对用银行业务的信息化不是那么热衷，于是软件公司就陷入了财政危机。公司总裁清楚：依据法律，他们可以强制履行合同。但他更清楚诚信对于一家企业的长远意义。他告诉行长："我们签过合同，我公司的产品和服务会帮助贵银行转向新的应用领域。但也我知道你对这份合同不满意，我们答应你们撤销合同、撤回定金。但贵银行以后有什么软件问题需要解决，欢迎与我们合作。"解除这份合同无疑将使公司陷入财政危机（$84，000 的合同损失），而公司总裁看重的是长远利益。3 个月后，行长打来电话："我想改善一下我们的数据处理系统，希望能与你们合作。"这次的合同价值高达$240，000。[Covey89]

如何在不违背道德的前提下牟取商业利益？

专业人员必须学会判断和理解，还要知道如何将人们的需求与自己的产品和服务联系起来。在某些时候，他还得有勇气承认：我们的服务不能满足这种需求。一个有坚定的意志，明确的价值取向的人能够勇于对某些事物说"不!"。而当你不想做某件事时，一种独立的意愿会使你有力量去做。这种力量来源于你的价值观而非一时的冲动和渴望。这种力量叫做诚信。[Covey89]

处境越是艰难，诚信越是重要。五分之四的雇员认为诚信是他们留在一个企业的重要原因。如果领导层遵守道德规范，那么员工中明知故犯的行为就会减少，而满意度则会提高。[Bentley]

企业内的各种因素相互作用决定了企业的未来。研究表明人们会把工作中其他人的所作所为当做自身行为的参考。能记住的并不一定是丰功伟绩，反倒一些微不足道的小事更容易让人留心。人们愿意相信亲眼所见的，对一些说道没做到的地方会更是印象深刻，他们会说："哈！我知道经理刚才在吸烟，原来他也不相信自己编造的那些鬼话。" [Collins+94]

因此:

大到一个公司，小到一个人，活着得有原则。

人们或者诚信或者不诚信。一个人在的平常生活中能够保持诚信，那么在工作中也很可能如此。然而即使有诚信之心，有时，在我们的工作中或是生活里，诚信也会缺席。为了利益，我们抛弃了道德，面对比如一份大合同，或是一条不那么光明正大的捷径，诚信丢失了。如果想继续保持诚信，我们应当知错就改。就算一家公司颇有诚信：产品质量好，价格公道，客服优质，注重品质，诚信纳税，保持诚信也是一桩上上下下时刻不可轻视的事情，不论高层的主管，CEO，还是低层的收发室和保安！

当你开始接管一家公司时，任何的不诚信都会显现。你得善于辨别，哪些人是诚实的，哪些人不是：取不义之财的，急于摆脱的，到处搬动文件的，收集支票的。好的教练能帮你分辨哪些奏效，哪些没用，谁在做事，谁在混饭。当缺乏诚信的人发现自己难以掩饰的时候，他们就回在被揭露前主动辞职。不诚实的人知道自己会露馅，不等你发现真相解雇他们，自己就会卷铺盖走人。

组织的诚信会有以下效果：

- 公司的好名声能吸引、招募和留住高层主管。
- 公司增强目的性，突出领导者的能力。
- 通过企业文化，公司能从所负责的决策中得到提升和受益。
- 一些冒险性的因素降低，公司的地位会提升。
- 拥有高的客户忠诚度。

某公司的某小组在公司的某次见面会议上认为他们的美德应当包括诚信。有人问:“对内诚信还是对客户也诚信？”有人说“当然不能对客户也诚信。”之后，大家都沉默了。在他们这个行业，卖主明知道无法按期交货仍会一口答应，大家都这样。讨论了 3 小时之后，最后得出结论:“如果我们把诚信作为一种美德，那么就应在各个方面保持诚信。可是，那时他们恰好处在进退两难的境况：如果诚信，就不能按时交货，就会无生意可做。所以他们采取了一个策略。他们对公司的主要客户说:”这个行业充斥着虚假的承诺，我们大家都知道。没有人喜欢这样，但是我们都有被欺骗的感觉。我们想转变这种情况，就从对您讲真话开始。“从那以后。他们按时交货。一年后，他们的生意迅速增长，利润更是突飞猛进。[senge+94]

分享想法，分享想法，分享目标，诚信，质量，拥护和关爱----这些就是赫曼·米勒（Herman Miller）的价值体系的基础！[depree98]

完美的关键就是掌握，你不能忽略任何一个步骤。有一个入门的步骤，每步都是下一步的基础。如果自以为可以能跳过某步，被忽略的那步却不会放过你。你可以伪装诚实，但是它会重新回来萦绕在你的周围，你不会这样做，如果你没有掌握当前步骤，你就无法进入下一步。[Rike02]

一流的人才

如果一定要说出什么使得我们公司如此完美，那答案就是公正。公正，是我们的待人之道，我们尊重每一个人。买卖双方经常就价格或时间问题争执不休，而我身不由己在其中时，一般情况下，我选择维护自己人，不是为了强词夺理战胜别人，只是不想自己人丢脸，哪怕错在自己人。[Rike02]

你是一个企业主，想建立一家完美的公司，你已经为公司制定了一个完美的目标，你正在锤炼你的领导能力，创造优美的环境，聘请一流的对工作充满激情的人，你知道：人，是你最为宝贵的财富。

你知道社会上有你想要的人，但不知如何发现他们，也不知怎样才能人尽其用。很多的公司把他们的雇员看作“牲口”，而你不想这么做。

做生意不是来回拨弄几个数字，也不是反复整理组织框架图，更不是生搬硬套商学院的最新公式。商务的核心是人。[Brown85]也许在一堂管理课上不会说到“人”，可是工作场所处处是人。[deGeus97]

健康的公司能留住人。很多管理者认为，留住人意味着让公司的骨干人员满意，这种看法让我不安。事实上，不止骨干重要，每个人都很重要。任何人辞职，公司都会蒙受损失。[DeMarco01]

推己及人，是待人接物的黄金法则。表面意思是：想人家怎样对你，你也要怎样待人。而我认为其深层含义是把人作为一个个鲜活的个体去理解，像你希望别人明白你（与别人不一样），并按你对他们的理解去对待他们。一位教子有方的父母说：因人而异是我一直以来的教育孩子的方式。[Covey89]

雇用和提拔人才的标准：第一是诚信；第二是雄心；第三，才能；第四，理解力；第五，知识；最后、最不重要的就是：经验。没有诚信的雄心是危险的野心；没有雄心，才能被大大削弱；没有才能的理解力作用有限；没有理解力，知识毫无意义；没有知识，经验无用武之地。其他条件都具备的人才，经验很容易得来，也能用对地方。[Waldrop96]

因此：

雇用适宜的人来做某项工作。充分利用他们的各项才能。确认你和你的雇员为离职做好了准备。

把你栽培的人托到你的肩膀上，他们会更有创造力，更有效率，更成功，因为除了从你这里学到的东西，他们还有自己的聪明才智。和那些等着别人吩咐才干活的人不同。你的目标是能和你一同起舞的人。

加强凝聚力和团队精神。让雇员知道你在背后支持他们，他们会做得最好。如果一定要预言一个公司的成败，我不会去考察公司产品如何，我愿意见见公司里的人。团队决定了产品的好坏。好的团队懂得倾听顾客的意见，互相学习，怎样、何时取舍。一个行业中的领军企业未必每战必胜，只要比他的竞争对手赢得更多就够了。而且胜利的次数越多，企业就越能吸引人才。同最佳人选共事，企业会不断走向胜利。[Kurtzig91]

一次一位潜在的顾客和他的总承包人到我们公司，为我们演示他们为一所大宅订制的设计方案和说明书。看的时候，不时有职员穿梭：接待员来传达信息并做拷贝，生产协调员给客人介绍我们的生产时间表和技术。客人要离开时，我把父亲介绍给了他们，父亲从公司创建时就在公司。一同出去的时候，总承包人对我说，你知道吗，这里每个人都很好。好一会儿，我才想清楚他给了我公司多么高的评价。Steve Sanchez

我已经把 Herman Miller 的员工当成了我的家人。[DePree89]

为什么说我们拥有一家完美的公司？不同之处在于我们待人处事的方式。我们尊重雇员和顾客。尊重，这是关键，推己待人，是黄金法则。我们一视同仁。[Price02]

HP 公司是个好雇主。他们自始自终一直是最好的。HP 让员工感觉到自己是产品生产链上不可或缺的一环，HP 鼓励主动、创新，所以一直在不断壮大，并且一直处于领先地位。更重要的是，HP 让职员觉得自己是家庭中的一份子。[Kurtzig91]

选择合适的人做最适合他们的工作

人言的力量不可小觑。由内部员工或合作伙伴推荐来的员工，在公司干得都不错。这些推荐者知道公司的需求，为确保举荐的人适合该项工作，他们会先行筛选潜在的雇员，因此不会随便拉个人来走过场。而那些潜在的雇员也会多方打听雇主的信息以助选择。好名声当然会吸引人，恶名远扬，就没人愿意来。

你是一个想要雇到一流的人才的企业家，你明白自己想要什么样的人。

你知道有这样的人存在，但是不知如何找到他们。

语言仍然是人们交流的最主要的方式。许多广告经理坚信：在这个充斥着各种记号的年代，语言是唯一能够打动人心的方式。一项研究表明大多数工作的是通过熟人或者关系并非十分亲密的朋友，而不是密友，得来的。这种非亲密关系的当事人明白自己在说什么。不是甲传乙，乙传丙。而是有一位“话匣子”的话，消息就一传十，十传百了。[Gladwell00]

单凭履历决定是否录用一个人是错误的方式。履历会使雇佣双方把精力集中在学历、证书和工作年限这些误导人的东西上。是否录用应该看一个人的知识、经验，梦想、热情、勇气和好奇心，换句话说，是那些无法被量化和列举的东西。[Dauten02b]

因此：

打广告——但不是通常意义上的广告。别忘了，这个世界很小。让每一个人都知道你想要的人。当最终遇到一个潜在的候选者时，听信你的直觉。

不要在草堆里去找一根针，变成一个磁石去吸引针。对你想要的人有一个清晰的概念，那样你就可以跟别人描述他。想象你有一根魔棒，可以立刻把你想要的人变出来。明白地描述出来，让每个人都清楚你的想法。当明确了你想要的和你所能提供的，达成协议的几率就会增大。如果新雇员的期望与你列的条件不符合，他将很快被淘汰。把精力放在面试前和面试后。

当你最终遇到了一个潜在的候选者，听信你的直觉。让你大叫“啊！”的那个人就是你要找的，而这个人可能并不是那个从字面上看起来最合适的人，或者是那个有一堆证书的给人深刻印象的人。

要为潜在雇员的离开做准备，没有人会永远呆在一个公司。当有人要离开，尤其是那些有突出贡献的人，总不免让人担心没有他们我们怎么继续。办法是他们在刚进来时就探讨这个话题。告诉他们不要担心离开的问题。如果你承认你的小公司是一个让人成长的地方，那么当有人发现自己没有成长的空间时，或者提升他们，或者让他们离开，或许他们打算自己开创一片天空呢。

请你的潜在雇员考虑一下他辞职的问题，请他描述一下他将如何从容离开。答题的要点大致如下：不会在一怒之下离职，培训接班人，或者是找到替代者。这种方式的辞职将会使离职的损失减到最小，让一切井然有序，或是共同推举一位接班人。

这种方法会消除因某人的离职给大家带来的担忧。公司不会想当然觉得他们拥有某个人，所以当雇员心生去意的时候也不会担心。雇员也不担心离职的问题，因为答案是现成的。

有人问我如何吸纳一流人才，我从来不打广告，先是祈祷一番，然后告诉每一个人。我清楚自己需要什么样的人。老Jim会认真地进行面试，而小Jim会有的放矢地问一些问题，然后到一个安静的地方，让直觉给出答案。[Rike02]

是否雇佣一个人，证书多少并不重要，重要的是那个人的态度。人们可以忽略很多事，除了态度。我们有一个水平很高的技术人员，但当他跟顾客打交道时，技术没派上什么用处。如果态度激怒了顾客，再多的技能也无法弥补。Steve Sanchez

意愿。你一心要等那个最合适的人出现，我觉得你心里也应该有杆秤，这个人才有可能被你等到。这是桩两相情愿的事，双方都有要求，而你也满足了对方的需要。即使你勉强雇佣了某个人，你仍可以虚位以待直到最合适的那个人出现。[Price02]

在Master Marble我们正在寻找最优秀的接待员。过去我们使用“肉馅糕”方法。以下是我们开给“肉馅糕接待员”的处方。

——给招聘公司的的工作描述尽量详细，这是必需的，上次不详细的描述使我们没有找到合适的人。

——用当前时髦的词汇写一个聪明的分类广告。

——对每一个可能的候选对象或其他公司的人进行多次不同级别的面试。

——雇佣最符合我们要求的职员。

——用中等压力在Master Marble这个烤箱中工作4-6个月。

——重新考虑，重新加载，每年重复一次，持续十年，直到对肉馅糕恶心。

我们注意到，我们已经吃够了肉馅糕。这次我们要发明一种新的食物。保留工作描述、有时髦词汇的聪明广告，然后加上一个“魔棒”。“在这个职位上如果让你选择一样东西，你希望得到什么？”人们会说：

——应该相信任何事都是可能的。

——要足智多谋。

——要机智灵活。

——配合好客户的工作。

——有幽默感。不妨在面试的时候让他们讲个笑话？

——透过他的行为看他的品质。

——当然，还有其他的，工作历史，成就业绩，等等。

如果被试者有这些品质，我们如何能够发现呢？可以现场发问，并且验证他们的答案。用比较有趣的方式得到答案。我们不能总用同样的调料做肉馅糕，想做出口味不同的食品，却懒得改变做事的方式是不可能的。Steve Sanchez

试用

一次面试不可能全面了解一个人。有很多课程，顾问或是书籍可以教你如何进行一次成功的面试。好像是在实验室做实验：或许在人工的环境下运行良好，但真实的世界变化莫测，所以最好也能在真实的世界里试验一下。我们可以通过一个人的行为、言语，或是处理问题时的态度了解一个人。他们能否与周围的人和谐相处？他们了解公司的宗旨么？或许，这个新人觉得这项工作不适合他。没有人可以通过一次面试了解这么多东西。通常我们会给新职员一个月的试用期，在这期间我们可以让他们离开公司，并且确实有人被要求离开。Daniel May

仅凭一次面试来断定你所雇用的人是否胜任工作是很困难的。

管理工作中最难的是什么？人，让合适的人干合适的工作。这是优秀管理者和庸碌管理者的本质区别。选对了人，哪怕你把全部事情都搞砸了，不要紧，这个人会帮你。这就是管理。选拔人员，分配任务，激励，编队——管理的四个最基本的要素。剩下的不过是重复性的琐碎工作。[DeMarco97]

因此：

如果你决定雇用某个人，让他工作一段时间以验证你的选择是否正确。

试用期内，你和新雇员可以互相了解。试用时间的长短和工作取决于具体工作岗位。短则一天，长则数月。

哥伦比亚咨询公司是一个专门为小商店作咨询的公司。这个公司只雇用有8年以上经验的高级咨询者。在与某校合作过程时，它创立了一个面向高学历毕业生的为期一年的实习项目。在这一年里，这些人将用他们的聪明才智解决现实生活中的问题，之后，他们回校完成学业。通过这个项目，哥伦比亚咨询公司可以充分了解这些毕业生的信息，而这些信息可以被他们用来进行咨询或招聘。Daniel May

在决定雇用一个人的时候，Master Marble会让他们在商店里工作一天，以此判断他们的技能。由商店的领班来评价总结这些人的表现。Steve Sanchez

我是一家小软件公司的顾问，新职员进入时，我们会为他分配一位指导员和一项任务，用几个月的时间来考察他是否胜任。这样，被雇用者有一个评价公司的机会，队伍的其它成员也有机会考察这个程序员。Linda Rising

不同的才能

面试时，我喜欢问未来的职员：你有什么特长？你想成为怎样的人？我与他们分享我对完美公司的设想，并且要问，“你能为公司贡献什么？”这个问题很有用，雇员可以借机思考自己能否胜任，并去思考一些他可能从未想过的东西。Steve Sanchez

你正在努力寻找一流的人才。

你如何依靠各类即将成为你的雇员的那些一流人才成功地建立一个完美的公司？

每个人都有自己的特长，每个人的特长各不相同。真正的参与和高明的领导者懂得让这些不同的特长在不同的时间，通过不同的方式释放出来。[DePree89]在一个充满活力的公司，凝聚力和多样性同时存在。公司当然是一个具有同一性的整体，但公司里的人和部门却有各有各的不同。无需为整体的利益而要求各个部门同一化，正相反，多样性有其价值所在。发现职员的优势并帮助他最大限度地发挥出来，这是一个组织的管理者的职责所在。[deGeus97]

因此：

了解并充分利用雇员的才能。

这将使你用一种全新的方式去发现别人的长处。不要只凭大学学位来评判一个人，让所有的职员来评判。理解并接受差异意味着每个人都感觉到被需要和被接受。对多样性的认识有助于我们集不同人的才能完成工作。多样性让我们以特有的方式在共同的努力中贡献我们不同的才能。[DePree89]

对多样性的认识有助于我们理解在工作场合我们对机会，公平和同一性的需要。对多样性的认识给可以让我们去体味意义，职责和目的，这不仅仅与个人的私人生活相关，比如爱，美化，和喜悦。[DePree89]

确保这些不同的才能从一开始便能在你的公司里存在。时间久了，它们就会被你公司的文化同化，你会发现他们开始像公司的其它任何一个人一样思考问题。

我的父亲是Herman Miller的创建者。在二十世纪二十年代的家具行业，大多数的工厂的机器都是由中心驱动杆带动的滑轮工作，而这个驱动杆的动力来源是蒸汽机。蒸汽机锅炉燃料又来自于车间的锯屑和其它废弃物，一个很好的循环。技工监管这个循环，他是一个很关键的人。一天这位技工去世了，那时，我的父亲——一个年轻的管理者，认为他该去慰问一下死者的亲属。死者的夫人问我父亲：她是否可以大声朗读几首诗。父亲自然同意。她到另一个房间拿出来几本书并用几分钟的时间朗读了她选出的几篇优美的诗歌。读毕，父亲称赞诗歌的优美并询问诗歌的作者。她说是他丈夫，那个技工，一个诗人。这件事让我知道：领导者的基本素质就是要承认人的价值，而这一切要从了解人的不同特长、才能和技术开始。[DePree89]

才能最大化

你想尽力雇用一流的人才来建立完美的公司。

怎样才能让雇员的才能最大化地发挥出来？

别指望控制别人，人们从来不曾“惟命是从”。为了薪水，他们会听从老板的指令，但他们不会100%地服从，再多的钱也没有用。许多管理者以为他们可以无所不管——这是他们的工作。可是他们未曾想到管理还可以有不同的方式。为了更好地管理这些组成组织的活生生的人，你不得不有所不为。你要吝啬地行使你的权力，让人不觉得你在管理。不要让所有的权利都集中在你的手中，而是让它分散到各个机构，你只要用四两之力就可拨动千斤。那些省下的力气对于一个健全的团队而言，意义深远。[DeMarco01]

你的最好的员工有些像志愿者：他们能在别处找到不错的工作，但他们选择为你工作，其主要原因一定不是薪水和职位。[DePree89]

作为一个管理者，一旦你发现他们，你就要同他们一起工作。你的任务就是营造一个能使员工心甘情愿地全力以赴的环境。[deGeus97]

管理就是要和他人一起并借助他人之力达到预定的目标。太多的管理者陷入了这样的误区：他们以为员工就是来听候他们使唤的，而事实上，员工工作是为实现他们自己的目标。通过交流，我们要让员工明白我们要的不是奴仆，我们工作在一种可让双方都实现自己目标的双向关系当中。卓有成效的领导会使雇员自发地与你合作，而不是你在监控你的员工。[Brown85]

因此:

以对待志愿者的方式对待你的员工，就像你把顾客当作志愿者一样，事实上他们就是。他们把最好的奉献出来，他们的心和才智。[Covey89]

把员工的才能与尊重和信任归为一类，它们都需你的努力才能得到。把控制的想法扔到一边去吧，允许你的员工自己规划自己，而你则在他们需要的时候给予支持和鼓励。把公司凝聚为一体的共同目标和文化会激励你的员工发自内心地去工作。

合十礼

我还记得 Mark 初到我们公司时的情景：他不太自信，衣衫不整又不得体。他的身体有些毛病，这使得他的自信不足，而这是他的第一份工作。不过 Mark 的技术很好。我们帮他找了个医生治病，指导他工作和着装。眼前我们办公室里这个年轻的技术人员和初到公司的 Mark 简直判若两人。在有些公司，Mark 可能会被斥责，当然也可能是用一种委婉的方式，命令他尊从某些规则。单纯告诉某人他不符合要求和帮助他做到最好是不一样的。你要做的是让他说出心里的想法，鼓励他达到要求，看着他真正过关。Daniel May

你想尽力建立一个完美的公司并且雇用的一流的人才。

有时人是戴着面具的，他们的才能被隐藏起来，就连他们自己也看不到。

当你带着发现别人未被挖掘潜能的想法与人交往时，你须动用你的想象力而不是记忆力。不要把一个人看死。与他们在一起时，我们可以用一种全新的方式看待他们。我们可以帮助他们成为独立的、有抱负的、能够与他人建立和谐、有益、有效人际关系的人。歌德说过，“以现有的方式对待某人，此人还是此人；以他能够和应成为的方式对待他，他就会变成你期望的那个人。”[Covey89]

以我的经验：你希望他人怎样，他们就真的会那样！如果你期望他们干得很好，他们就会表现很好；相反，你期望他们表现糟糕，他们就会砸锅。我相信，一个中等水平的职员，如果知道你对他抱有很高的期望，他的表现会超过比那些自我期望很低的中上水平的职员。激发你的职员去动用那 90%未被挖掘的潜能，他们的业绩会突飞猛进！[Brown85]

想成为一个领导者，你必须明白管理者不是上帝。一个管理者不能凭自己的意愿来改造员工。要接受他们本来的面目，上帝创造的面目，并且与这样的他们一同工作。[deGeus97]

当今世界上的许多地方，人们见面时不握手，比如印度教教徒打招呼的方式是“合十礼”，即双手合十，置于胸前，低头。合十礼表达了人们心灵深处对神灵的崇敬，意思是：“我心中的神问候你心中的神，即便在这场人间的聚会，我也记挂神灵在心间”[Hindu]

因此：

帮助你的员工认识到他们的潜能。

每个人都有多面性。你对待他们的方式决定了他们以何种面目对待你，最好的或者是最坏的。

每一种宗教都强调人类的进步，爱，尊重他人，分担他人的痛苦。在这方面所有的宗教都或多或少持有同样的观点和目的。The Dalai Lama

从容离职

某公司的审计员算错了一笔帐，老板把她训斥了一顿之后很快忘记了这件事。此后审计员以为老板不再信任她，决心辞职。她向老板递交了辞呈，但没作什么解释。这时，老板参加了我的一个训练课程，对审计员的决定，他意外又沮丧，但也没想到要找她谈谈。我找来双方鼓励他们开诚布公地谈一下。审计员回想起了她的父亲，当她犯错时，父亲总是抑制住他的爱，从不与她把问题说开，于是她决定宁愿离开家也不要跟父亲讨论这件事。老板在发火后还是很信任她，但并没告诉她自己已经原谅了她，而当他一旦说出来时，审计员立刻撤销了辞呈，继续留在了公司里。若不是教练促使他们好好谈谈这件事，老板怎么也不会想到自己的勃然大怒会给审计员心理上造成如此影响，以至于她觉得既然老板对自己的表现不满意，还是走了算了。而审计员也不知道老板已经原谅了她，如果不是教练，她自己绝对不会想到要跟老板交流一下，既然每个人都会犯错，那么这些“鸡毛蒜皮”的小事就根本不值一提。Caroline King

你正在构建一家完美的公司，并正寻找一流的雇员。

组织由人构成，而人总是处在不断的变化之中。

同职员工作好比在一起跳舞，当有人决定离开，每个人都能感受得到。这个时候我就要出场召集双方坐下来谈谈。有的人会有所改变，有的决定离开。[Price02]

雇佣关系以失败告终决不是因为面试时的失误或者是面试者弄虚作假，而是因为事易时移：婚姻、恋爱、家庭问题，其重心不再是工作，其结果就是雇佣关系失败。

我父亲是医生，他就有一些始终追随他的雇员。我想有时你就得放手，他很忠诚，但他们有时辜负了这一点。员工们会对你的思想有所反应，当事情发生变化时，他们不自觉地就会自行决定不再追随你。去年我去旧金山参加化妆品培训。回来时就发现有些人打算离开：他们不能适应这种变化。[Price02]

当你头一回开始进行完美公司的训练时，你把公司定义在诚信的基础之上。有些员工达不到这条要求不得不离开，而剩下的人不离开的原因也只是时机未到。

情人眼里出西施。员工都觉得好，公司才会完美。随着时间的变化，个人的目标和组织的目的都会发生变化，当二者的变化不一致时，美，就消失了。

因此：

追查一下为什么会产生不一致，不是要追究谁对谁错，而是从中吸取经验教训。

好的教练应该帮助双方发现问题所在。大多数问题都是可以解决的，只要双方把话说开，就没有什么不可解决的矛盾，若是有，当初也就不会发生雇佣关系。试着找找问题，是误会？压抑？没有好好交流？还是有人的期望落了空？是职员不再衷心还是老板没实现诺言？有人伤了心或者觉得别人不再在乎？辞退一个人之前，这些问题必须要想清楚。

也许有时确实要结束雇佣关系，这时双方应该明确这样做对双方都最有利。互相信任，关系的解除就不会那么痛苦。过了河也别拆桥，双赢才是最好的结果。

当公司准备雇用一个人时，就要想到这个人会离开，有了这层心里准备，整个雇佣关系就不一样了，作为一个承诺，它为通常情况下可能是最坏的一种情况建立了诚信的基础。辞职时都能以诚相待，平日工作时的关系自然就会更好。

友好的顾客

我的一个顾客是个建筑商，他承建的项目价值数百万美元。对于他的工作，我们不敢高高在上，指手划脚，相反，一切都协商处理。他提出某个预算，我们就计算在此预算内如何达到他的目标。不用担心，他从未选中过报价最低的方案。Steve Sanchez

你是一个企业主，想建立一家完美的公司，你正在锤炼你的领导能力，已经聘请到了一流的人才。

怎样才能留住友好的顾客？

在生意场上人人都期望与这样的商家打交道：让你忘记了杀价，相信他所提供的说明书肯定没错，我们期望给顾客留下这样的印象。而我们愿意交往顾客是这样的：让人心情舒畅，快乐，可以让我们成长，按时付帐。因此：诚信待客并期望你的顾客也诚信。如何做到这一点？一诺千金。比如我会告诉我们的顾客：合同一签，您就什么都不用操心了，所有的细节我们都会考虑周全。然后我们说到做到。这种情形下才会有友好的顾客。因为他们信任你，所以即使你犯了错，顾客也只不过说一句，这里不对，但没关系。因为他们知道我们会努力到他满意为止。而我们也决不会扔下顾客不管的。Steve Sanchez

这样的故事一直在我的办公室里上演。人们会说，我开始还挺担心，但现在好了。在过去的二十多年里，我一直在一个大笔记本上保留着顾客的意见。我把这个笔记本称为“感谢办公室”。[Price02]

友好的外界人士

我们的供货商全都是我们的朋友，他们甚至比我们的病人更重要。凯文十六年来一直向我们提供设备，而所有的设备我们都从他那儿购买，他也负责我们所有设备的维修，我们彼此信任。我从不跟他讨价还价，也不必核对他的发票。当我不信任一个人时，我不会跟这个我不喜欢的人做生意。我享受彼此欣赏的感觉，对于朋友似的供货商，我会按时付帐，关心他们和他们的家人。无论发生什么，我们照顾彼此的生意，任何争执都能有效地解决。[Price02]

你是一个企业主，想建立一家完美的公司，你已经为公司制定了一个完美的目标。你雇佣了一流人才在你的公司里任职，同时你也要与外界的人和公司的建立联系。

如何与你的供货商、合伙人、顾问和项目承包人打交道？

他们与你的公司密切相关，但又并非你的雇员。冷战最紧张的时期，美国总统罗纳德.里根在谈及如何控制核武器时，经常会引用这样的一句俄国谚语：“信任，但也要确认”。当被问及如何保证苏联遵守裁减军备的协议时，他说他会信任他们但也要监督确认。对苏联的不信任迫使他制定了严格的措施用以监督苏联方面是否遵守协议。

信任不是结果，它是一个过程，一个没有没有结尾的，艰辛的，无需言谢的过程。我们不会注意到它，除非一些事情不对劲了。

因此：

一旦商讨达成某种协议之后，就要把它保持下去。

这些协议是符合你的长远利益的。想要什么，你只需通知你的老关系。知道自己在寻找什么的话，见到时，就会认出它。他们喜欢就会与你做生意，不喜欢就会和别人做。

你想跟专业的，有趣的，准时付帐的公司做生意，他们能够发展壮大你的公司并把你当作事业上的伙伴。如果违背了协议，务必让他们知道出了问题，并且问是否责任在你。再问他们是否想要改变协议，是的话，就商讨一份新协议。记住你想要什么，针锋相对让他们也做出相应的妥协。如果关系继续恶化，你得意识到，你仅仅与你实力最弱的合伙人水平相当，要采取措施保证履行你对你的客户的承诺。这时你需要更换你的供货商。一定要时常查看你的协议，使它保持简洁、专业。

一段时间以后，在信任的基础上，你和外部合作者建立了一种稳定的合作关系。打过几次交道之后，彼此间不再那么警惕，恐惧减少。这是一家完美的公司的构成要素。你知道它就要实现。

被你更换的供货商，可以考虑再给他们一次机会，当然要依据他们与其他公司打交道时的可查的信誉记录，但是还要保持警惕。

有一家开发大型软件的公司有多个开发小组，为便于协同工作，共享数据和访问程序的小组之间定义了一些接口文件。在这样的开发过程中，我们要时刻小心，因为总有一些小组会无视接口文件的存在，开发他自己的一套东西。有时我们得定期与那些过分自由的小组开个会，因为别无选择，我们必须和他们一起工作！Linda Rising

Chris，一个搞计算机的小伙子，6-7 年来一直负责我们公司计算机方面的事情。我们安装了这样一套新的软件系统：对软件的任何修改，都需改变软件的设置。我们的网络和新的服务器有些不匹配，Chris 花费了许多时间重新配置它。他给我寄了一张支票，又打来电话说他会降低价钱，因为他觉得自己有责任。无需多言，这就是你想要的那种关系。 [Price02]

参考文献

[Bentley] Bentley College Center for Business Ethics http://ecampus.bentley.edu/dept/cbe/newresearch/2.html

[Brand99] Brand, S., *The Clock of the Long Now*, Basic Books, 1999.

[Brown85] Brown, W.S., *13 Fatal Errors Managers Make and How You Can Avoid Them*, Berkley Books, NY, 1985.

[Carbonara97] Carbonara, P., "Wealth and Poverty," *Fast Company*, December 1997, 60.

[Collins+94] Collins, J.C. and J.I. Porras, *Built to Last: Successful Habits of Visionary Companies*, HarperBusiness, 1994.

[Dauten02a] Dauten, D., "The Best Companies are Best to Employees," *The Arizona Republic*, February 12, 2002.

[Dauten02b] Dauten, D., "Devil curses resume dependents, falsifiers," *The Arizona Republic*, Tuesday, August 13, 2002, D2.

[Davenport+98] Davenport, T.H. and L. Prusak, *Working Knowledge: How Organizations Manage What They Know*, Harvard Business School Press, 1998.

[deGeus97] deGeus, A. *The Living Company*, Harvard Business School Press, 1997.

[DeMarco97] DeMarco, T., *The Deadline*, Dorset House Publishing, 1997.

[DeMarco01] DeMarco, T., *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency*, Broadway Books, 2001.

[DePree89] DePree, M., *Leadership is an Art*, Dell, 1989.

[Edler95] Edler, R., "*If I Knew Then What I Know Now: CEOs and other smart executives share wisdom they wish they'd been told 25 years ago*," G. P. Putnam's Sons, 1995.

[Gilbreath93] Gilbreath, R.D., *Escape from Management Hell*, Berrett-Koehler Publishers, 1993.

[Gladwell00] Gladwell, M., *The Tipping Point*, Little, Brown and Company, 2000.

[Hammonds02] Hammods, K.H., "Handle With Care," *Fast Company*, August 2002, 102-107.

[Hefferman02] Hefferman, M., "The Female CEO," *Fast Company*, August 2002, 58-66.

[Hendrickson02] Hendrickson, E., "Managing Technical People (When You're No Techie),*STQE*, May/June 2002, 58-60.

[Hindu] Himalayan Academy http://www.flex.com/~jai/articles/namaste1.html

[Jr02] *The Arizona Republic*, Friday, June 28, 2002, D2.

[Kleinfeld02] Kleinfeld, J.S., "Six Degrees of Separation: An Urban Myth?" *Psychology Today*,Mar/Apr2002. http://www.uaf.edu/northern/six_degrees.html

[Kurtzig91] Kurtzig, S.L. with T. Parker, *CEO: Building a $400 million company from the ground up*, Harvard Business School Press, 1991.

[LaBarre00] LaBarre, P., "Do You Have the Will to Lead?" *Fast Company*, March 2000, 222.

[LaBarre01] LaBarre, P., "Who's Fast 2002: Feargal Quinn," *Fast Company*, November 2001,88-94.

[Manns+02] Manns, M.L. and L. Rising, *Patterns for Introducing Patterns (or any Innovation) into Organizations*, Addison-Wesley, in press.

[Mariott+97] Marriott, J.W., Jr. and K.A. Brown, *The Spirit to Serve: Marriott's Way*, Harper Business, 1997.

[McMahon02] McMahon, J.T., "Enron's leaders still don't get it," *Houston Chronicle*, February 1,2002. http://www.chron.com/cs/CDA/story.hts/editorial/outlook/1236695

[Overholt02] Overholt, A., "True or False: You're Hiring the Right People," *Fast Company*,February 2002, 110-114.

[Pascale+00] Pascale, R.T., M. Millemann, and L. Gioja, *Surfing the Edge of Chaos*, Crown Business, 2000.

[Price02] Interview by Caroline King, Linda Rising, and Steve Sanchez of Ginger Price, D.D.S., July 30, 2002.

[Rike02] Interview by Caroline King, Linda Rising, and Steve Sanchez of Jim Rike, President, CalibeR Construction, Inc., June 26, 2002.

[Rothman01] Rothman, J., "Other People's Problems," *Software Development*, September 2001, Project and Process Management Column, 49-50.

[Senge+94] Senge, P. and A. Kleiner, C. Roberts, R.B. Ross, and B.J. Smith, *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization*, Doubleday, 1994.

[Senge+99] Senge, P., Kleiner, A., Roberts, C., Ross, R., Roth, G. Smith, B. *The Dance of Change: The Challenges to Sustaining Momentum in Learning Organizations*, Doubleday, 1999.

[Waldrop96] Waldrop, M.M., "Dee Hock on Management," *Fast Company*, October 1996, 79.

[Webber02] Webber, A.M., "Are All Consultants Corrupt?" *Fast Company*, May 2002, 130-134.