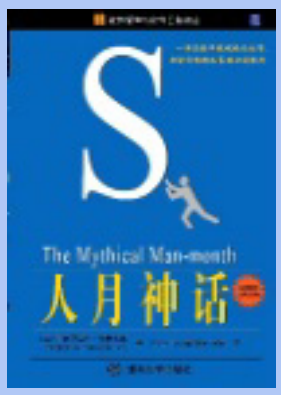
2002－12（第 20 期）



非程序员
X-Programmer
软 件 以 用 为 本

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

http://www.umlchina.com/

我在低头整理东西。

有人过来问我--还是同样的问题：“你们 UMLChina 有多少人？”

我抬头，“是‘我们 UMLChina’”

think

UMLChina 三周年，1999-2002

IBM 收购 Rational

[2002/12/6]Armonk, N.Y., and Cupertino, Calif., December 6, 2002 -- IBM (NYSE:IBM)和 Rational Software Corp. (NASDAQ:RATL)今天宣布两家公司达成最后协议，IBM 以大约 21 亿美元现金或每股 10.50 美元收购 Rational。

Rational 为开发企业应用和开发软件产品提供开放的，工业标准的工具、最佳实践和服务。通过收购 Rational，IBM 将能够提供一个全面的开发环境，从整合业务流程和软件基础架构。这次收购是 IBM 的“e-business on demand”策略的重要组成部分。

Rational 成立于 1981 年，是世界上最大的软件公司之一，在全世界范围有超过 3,400 名雇员。

（自 Rational，think 摘译，仅供学习交流，不得转载用于商业用途）

I-LOGIX 公司 BRUCE POWEL DOUGLASS 在实时设计模式领域的新书

I-Logix 授权发布

《实时设计模式：实时系统的鲁棒可伸缩架构》，I-Logix 公司 Bruce 博士这本新书引起了全世界的注意。该书为工程师们提供了一种基于 UML 构建可伸缩的、鲁棒的、高可靠应用架构的方法。该书概述了使用 I-Logix 公司的产品 Rhapsody(R)（Rhapsody(R)是针对嵌入式软件开发推出的可视化快速开发平台）构建基于 UML 的实时应用的简便性。

作者 Douglass 博士在实时软件开发方面经验丰富，在由 Addison-Wesley 出版的这本书中，他为嵌入式软件工程师们提供了应用基于 UML 在实时应用领域的设计模式成功创建鲁棒可伸缩架构的成功之路。这已经是 Douglass 博士在基于 UML 软件开发方面的第三本专著了，书中介绍了用户可以实际应用到自己的系统设计中的一些软件开发的高级技术，前两本书分别是：《实时 UML：为实时系统开发有效的对象》（Developing Efficient Objects for Embedded Systems）和《Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns》，这两本书都得到全世界读者的广泛欢迎。

“实时设计模式可以为小的单处理器系统设计鲁棒性架构，也可以适用于大的分布式控制网络”，Douglass 说，“架构可以分成几个不同的部分：子系统组织、分布、并发、内存管理、资源管理策略、安全性、可靠性，这本书里面为这些领域各自提供了一些模式，包括它们的优缺点、实现策略和例子。”

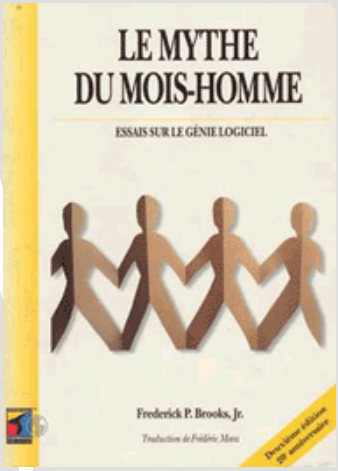
“Bruce 在实时 UML 组件的开发中仍然是一个相当关键的角色，可以说是实时系统开发领域处于领袖地位的方法学专家”，I-Logix 公司总裁及 CEO，Gene Robinson 这么认为，“这是他为实时软件开发者贡献的第三本书了，内容都是关于如何利用 UML 进行实时、嵌入式软件开发。Douglass 博士在处理器方面的丰富知识以及使用 Rhapsody 的丰富实践经验，都会帮助 I-Logix 为我们的顾客提供有效的整套解决方案。”

有着 20 多年软件开发经验的 Douglass 是实时嵌入式系统领域著名的讲师和作者，也是 I-Logix 公司的技术总宣传师（chief evangelist），他在 UML 规约的制定上和 OMG 紧密合作，同时还是（嵌入式系统大会顾问委员会）Advisory Board of the Embedded Systems Conference 的成员。他是面向对象实时领域开发和设计、软件估算和调度、交互协议、有限状态机和安全临界系统设计方面非常受欢迎的讲演者。

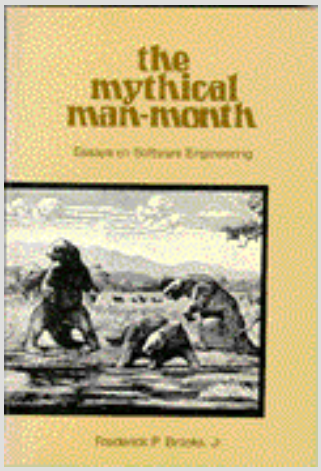
在著名出版社 Addison-Wesley 和 Amazon.com 的站点上可以购买到 Douglass 的书。

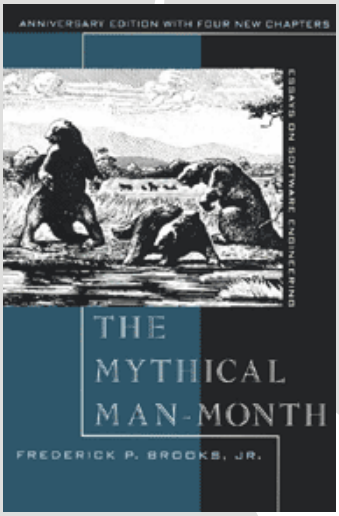
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)





(第二版)

(第二版日文译本)

家用报警器的 UML 设计

及其在 C++和 VxWorks 上的实现

M.W.Richardson 著，liuweiw 译

查看评论

论文描述了如何运用 UML（统一建模语言）设计一个简单的家用报警器，并实现到 VxWorks 操作系统上。本文分两个部分，第一部分描述了如何用 UML 设计和验证家用报警器的模型，以使其独立于特定的硬件和操作系统。第二部分则细节地描述如何将此模型实现于运行 VxWorks 操作系统的 486 硬件平台上。模型的设计、开发和验证使用了 I-Logix 公司的可视化编程环境“Rhapsody”。

第一部分：设计一个独立于操作系统的家用报警器

一个好的面向对象设计力图将不变的东西与变化的东西相互分开。在家用报警器的设计中，一个可能的变化是实际的硬件平台。本文描述了如何用 UML 设计这个模型，以使它可以容易地应用到不同的硬件结构中去。文中也表现了如何通过“Rhapsody”工具，可以由模型产生全部代码，并通过“设计级调试”进行验证。设计级调试使我们可以运用描述模型中用到的相同图示来进行调试，基本上可以让状态图和消息序列图活动起来。这样就可以使我们快速地验证所设计的模型。

需求

我们要设计的家用报警器有如下需求：

可以通过遥控器或者简单的小键盘进入戒备/取消戒备状态。当用小键盘设置戒备时，一个四位数字密码必须输入随着要按下 ON 键。如要取消戒备时，一个四位数字密码必须输入随着要按下 OFF 键。密码应该可以改变。当在警戒时检测到情况，马上会发声报警。设置警戒时，将有一个延时供户主离开。警戒状态时开门，会有一个入门延时，这时户主可以取消警戒，如果不能消除警戒警报就要响了。

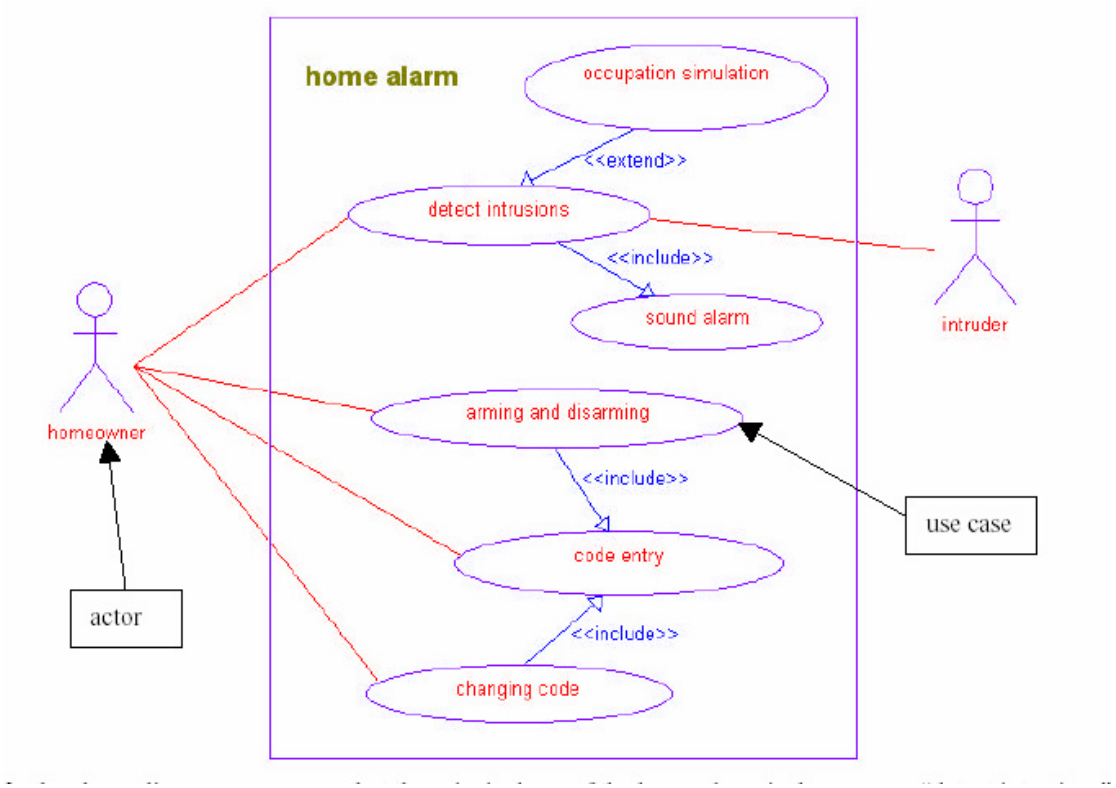
报警器要有两个 LED，一个为绿色，另一个为红色。红色的 LED 表示报警器处于激活状态，在报警器激活时或者报警器已激活且门打开时闪烁。绿色的 LED 显示报警器启动并且在密码成功修改后闪动四次。

一个未来的需求是家用报警系统应可以通过开关电灯模拟房间被闯入。

设计 UML 模型

用例图

第一步是根据需求建立用例图。下面的用例图表明了两件事：一个显示了主要用处（或用例）即试图给“家用报警器（home alarm）”建模。另一个表示谁使用家用报警器；有两个主要的角色或者用户，即户主（homeowner）和闯入者（intruder）。用例是系统功能反映给角色的可观察的结果但并不揭示系统的内部构造。

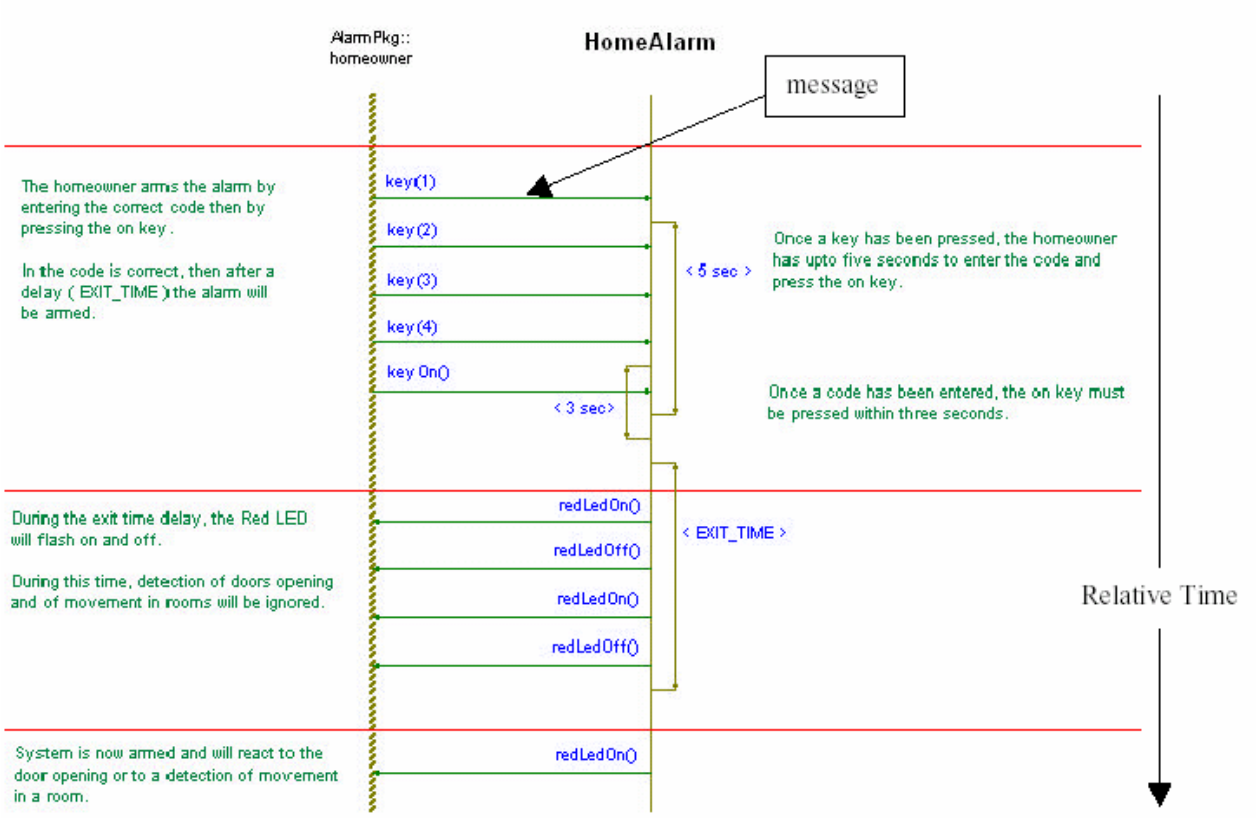


在上图中我们可以看到报警器的主要用途是“检测入侵（detect intrusions）”的用例，这时户主和闯入者都会参与。“占用模拟（occupation simulation）”用例扩展了“检测入侵（detect intrusions）”，在警戒时会通过开关灯模拟屋内有人。其他两个有户主（homeowner）参与的用例是“警戒和消除警戒（arming and disarming）”和“修改密码（changing code）”；两者都使用或包含了“密码输入（code entry）”用例中的服务。每一个用例可以加一个文本描述，并且如果需要，用例可以在另一用例图中进一步分解。

用例不隐含任何内部结构，只是一个功能视图。

黑盒场景

已经建立了用例图，下一步就是看一些黑盒场景。这些场景的目的是显示我们试图去建模的系统和角色之间的交互情况。我们把系统看为一个黑盒子。例如：对于“警戒和消除警戒”用例我们可以想到有下述场景：



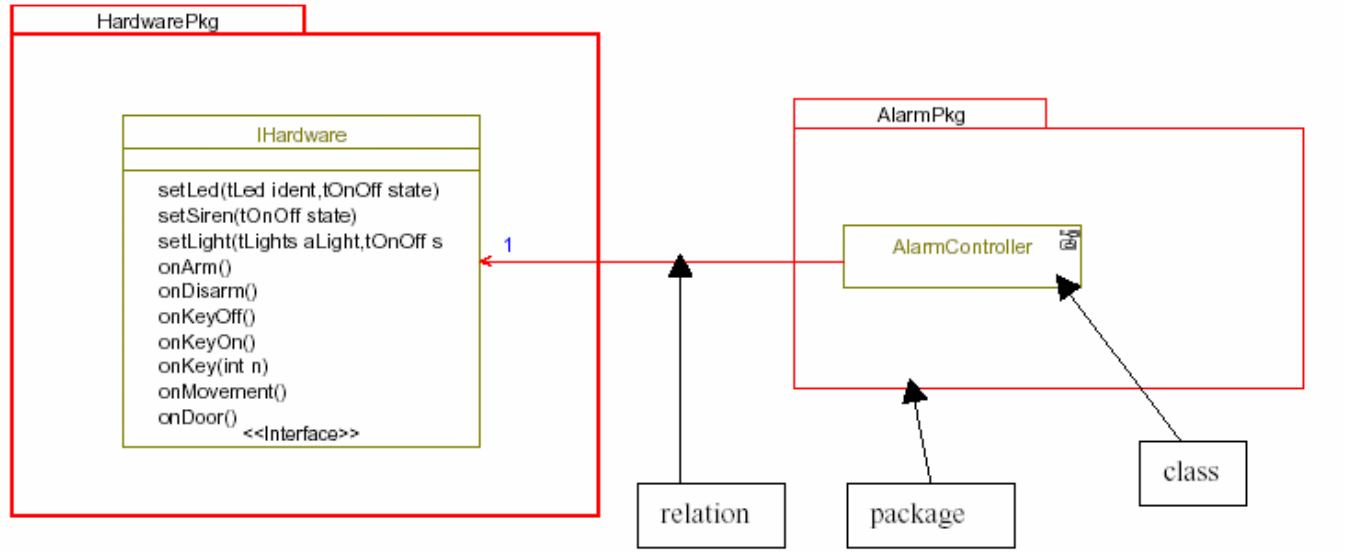
上图中的场景用消息序列图，显示了一段时间内户主（homeowner）和报警器（HomeAlarm）之间的交互。当建立这样的图时，系统的更多细节问题将被发现。例如在上述场景中，按键的时间间隔信息必须被确定。

这些场景帮助我们更好地理解用户如何使用系统并且帮助我们确切理解什么是需要做的。

一般地，每个用例有几个要注意的场景要捕捉。每个场景实际是用例的一个实例。

领域图

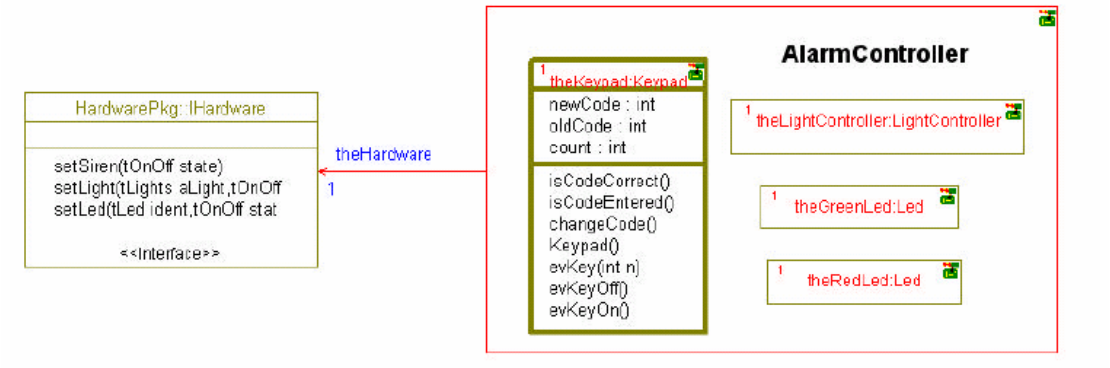
下一步要看看我们如何将模型分解为小一些的子系统和领域。领域可以在 UML 中通过包来表示，即基本上是类的集合。下图的领域图显示了家用报警器如何被分解为两个基本包：报警器领域（AlarmPkg）和硬件领域（HardwarePkg）。此处的想法是报警器领域完全独立于实际的硬件，并且任何能实现 IHardware 接口的硬件领域能与报警器领域联系。接口类 IHardware 描述了所有报警器领域中必须的操作。set 操作是报警器领域可以在硬件中调用的操作，而 on 操作则可被硬件调用。



一旦报警器领域和硬件领域之间的接口被确定了，每个相对领域的更细节的分析就可以同时和独立地进行，因为每个领域都了解和其他领域之间的接口。例如我们现在可以开始分析报警器领域而不必知道将要用到的硬件的更多的任何信息。

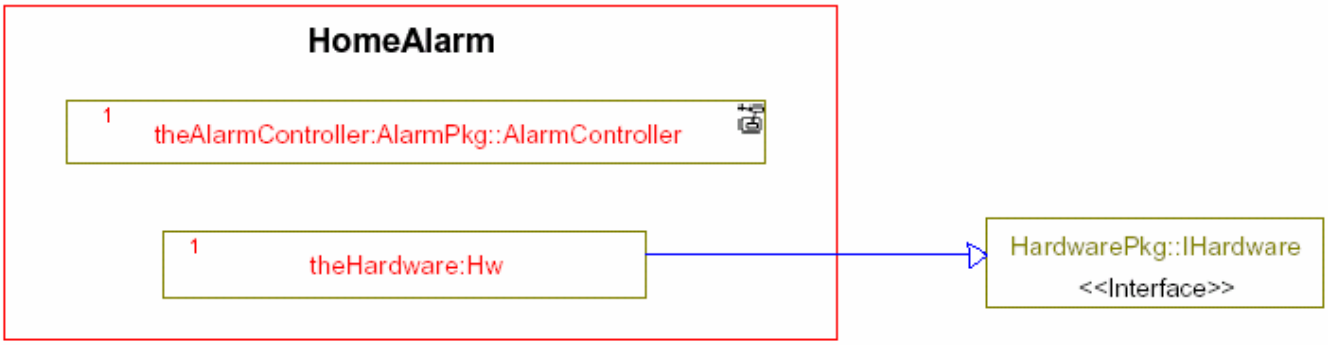
对象模型图

对于报警器领域（AlarmPkg），我们可以设想有一个 AlarmController 类，它包含 Keypad、Led 和 LightController 类。这些类如下图：



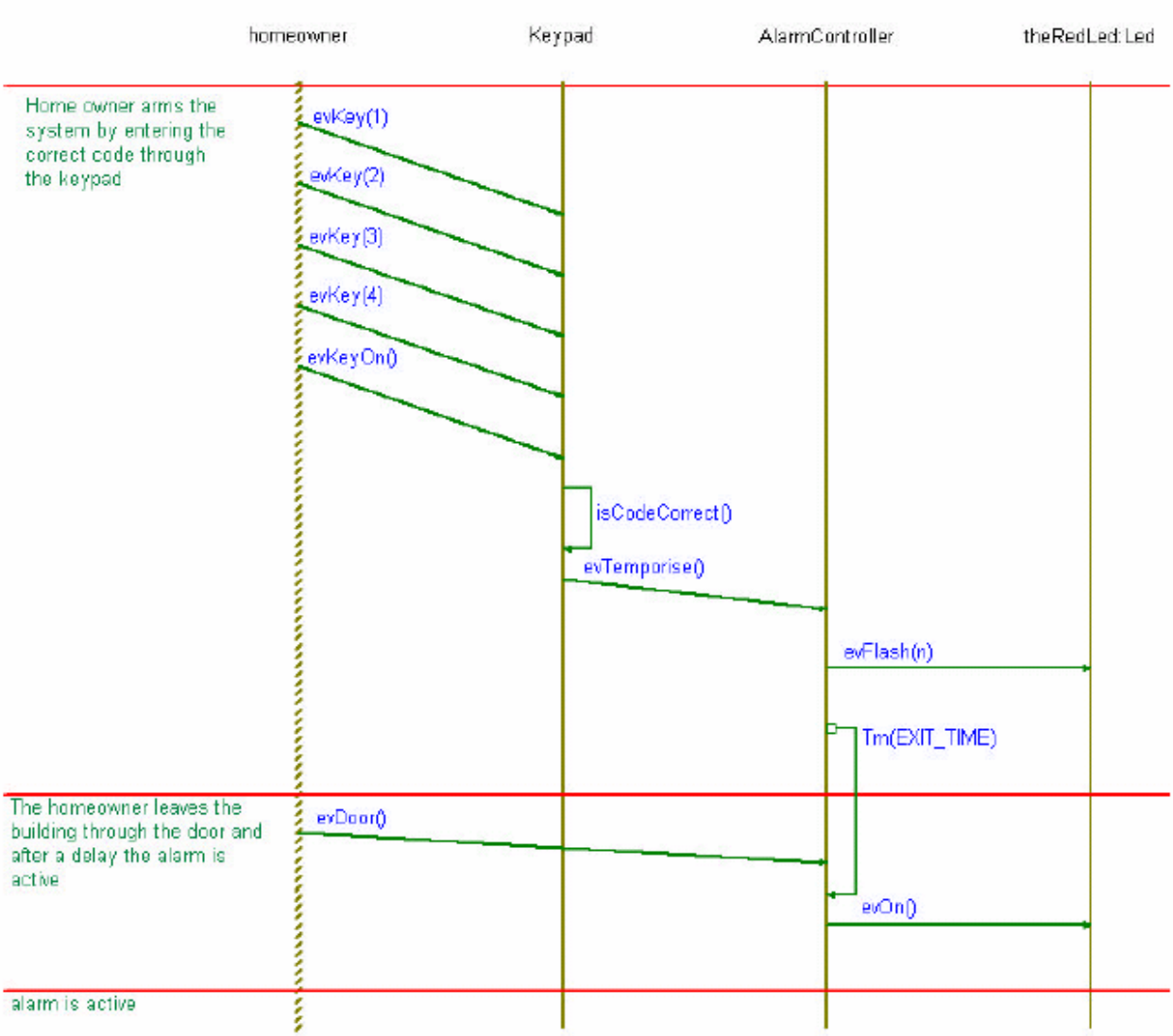
图中显示 AlarmController 是一个复合类，包含了其他类的实例。我们也能看到选择显示了 Keypad 类的操作和属性。复合类是一个显示强聚合关系的有效方式。上例中如果一个 AlarmController 类的实例生成，就会生成一个叫做 theKeypad 的 Keypad 的实例，两个 Led 的实例分别叫 theRedLed 和 theGreenLed，也生成 LightController 的一个叫 theLightController 的实例。如果 AlarmController 的实例被删除则其包含的实例也要被删除。

IHardware 类有一些纯虚操作，为了测试 AlarmController 必须忽略这些操作。下图中显示了 HomeAlarm 包含了 AlarmController 类的一个实例和 Hw 类（从 IHardware 类继承且忽略了一些操作）的一个实例。



白盒场景

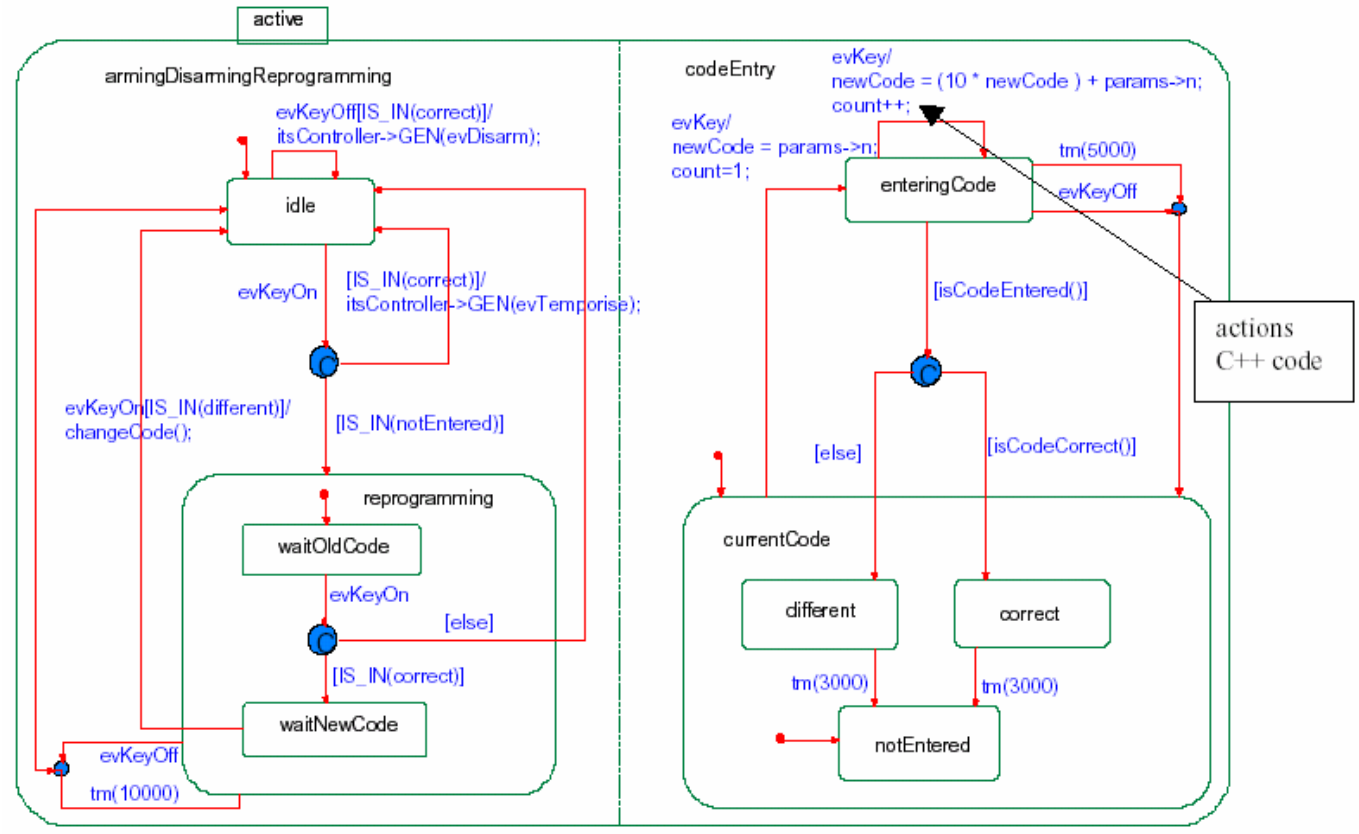
现在我们发现了几个类，可以开始将每个黑盒场景扩展，生成白盒场景。消息序列图再一次被用来捕捉上述类的实例在不同场景中的交互情况。例如下图中描述了户主设置警戒后离家的场景。



消息的斜率显示消息是同步（水平线）的或者是异步（斜线）的。从上图中我们可以看到这些类中的许多有行为，它们处理超时，对其他类操作调用也接收事件。在 UML 里，这个行为能用状态图规定。

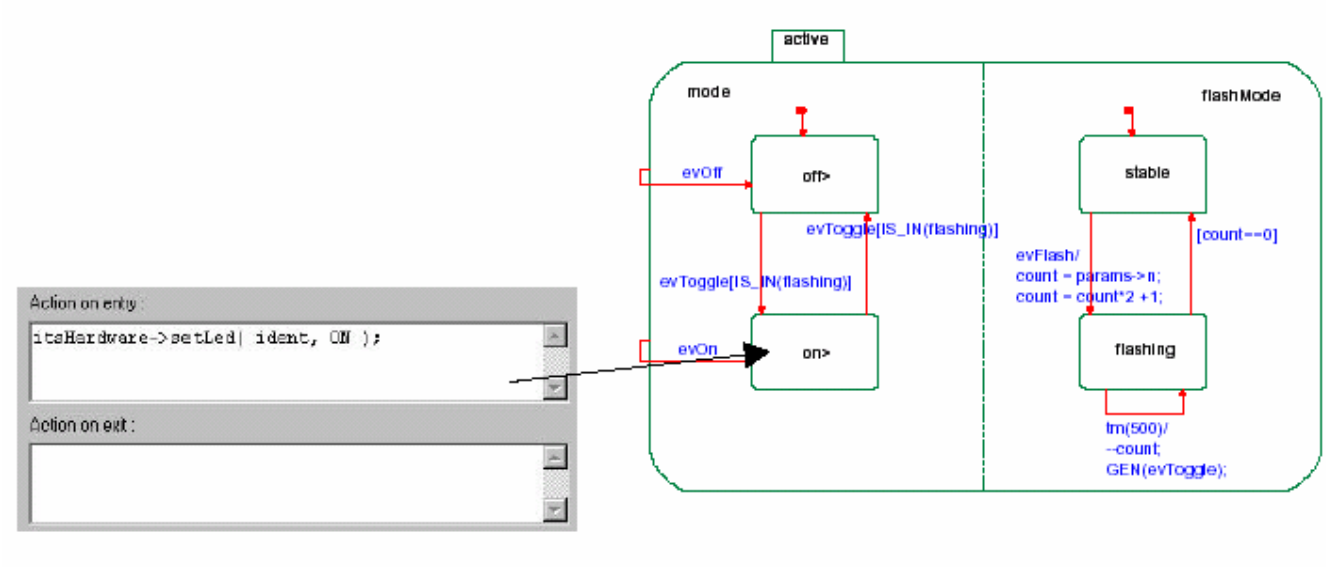
小键盘的状态图

下面的状态图描述了小键盘的行为：经常在问题复杂时，会被分为几个更简单的问题。这就是对小键盘做的事情。并发状态可被使用，所以“密码输入（codeEntry）”能用一个并发状态图解决，而高一级的功能“戒备，解除戒备和重编程（armingDisarmingReprogramming）”可通过另一个并发状态图解决。可以看到在状态图里可以输入动作的 C++代码，也可将不同的操作叫做例如像 isCodeEntered()，changeCode()，…也可用到各种属性如 count 和 newCode。



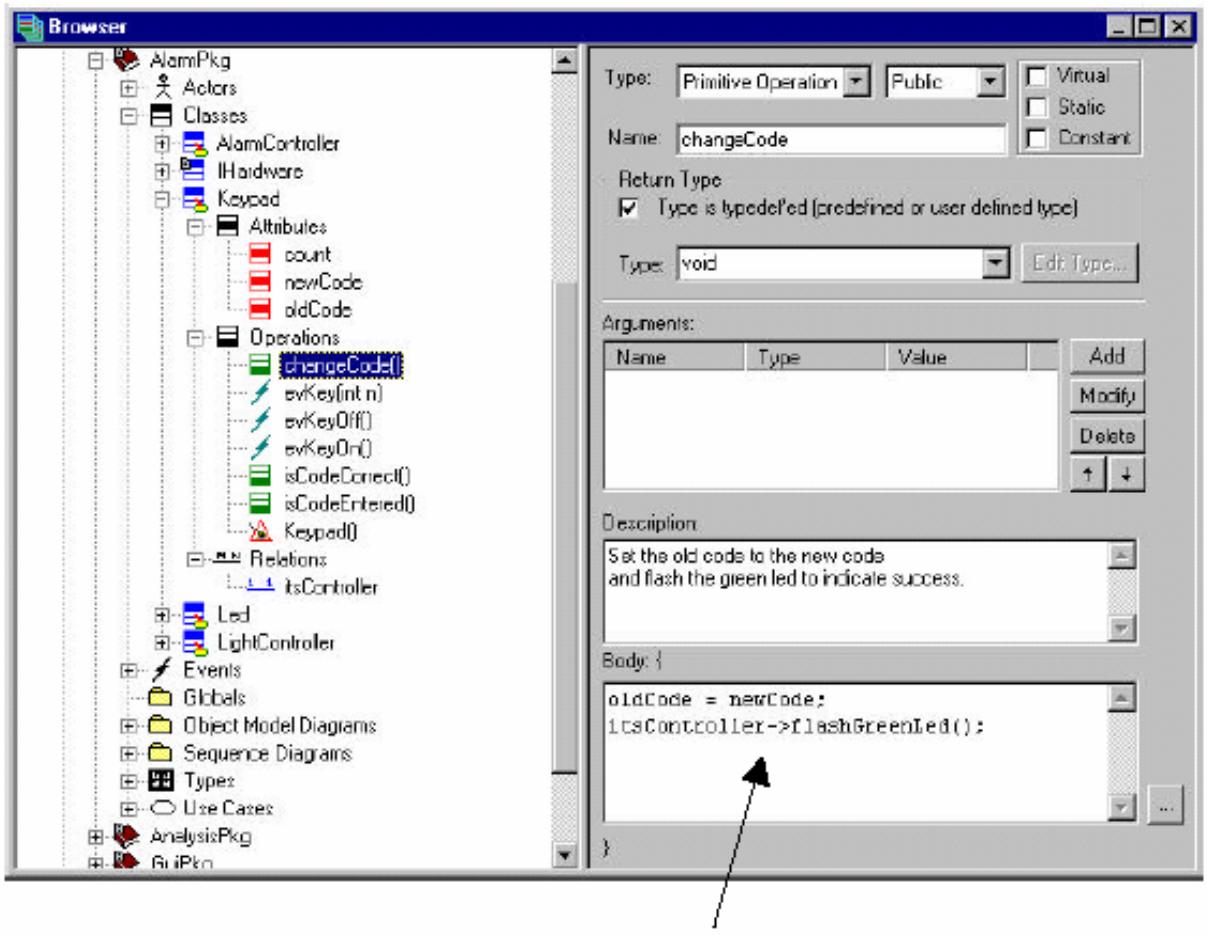
LED 的状态图

下图中描述了 LED 类的行为。同样，在这个状态图中用到复合状态。在 on 和 off 状态中，符号>显示这里有进入/退出动作。例如在 on 状态有如下的动作：



操作和属性

我们可以看到许多类有属性和操作，这些可以规定如下，注意实际的 C++代码包含在模型中。

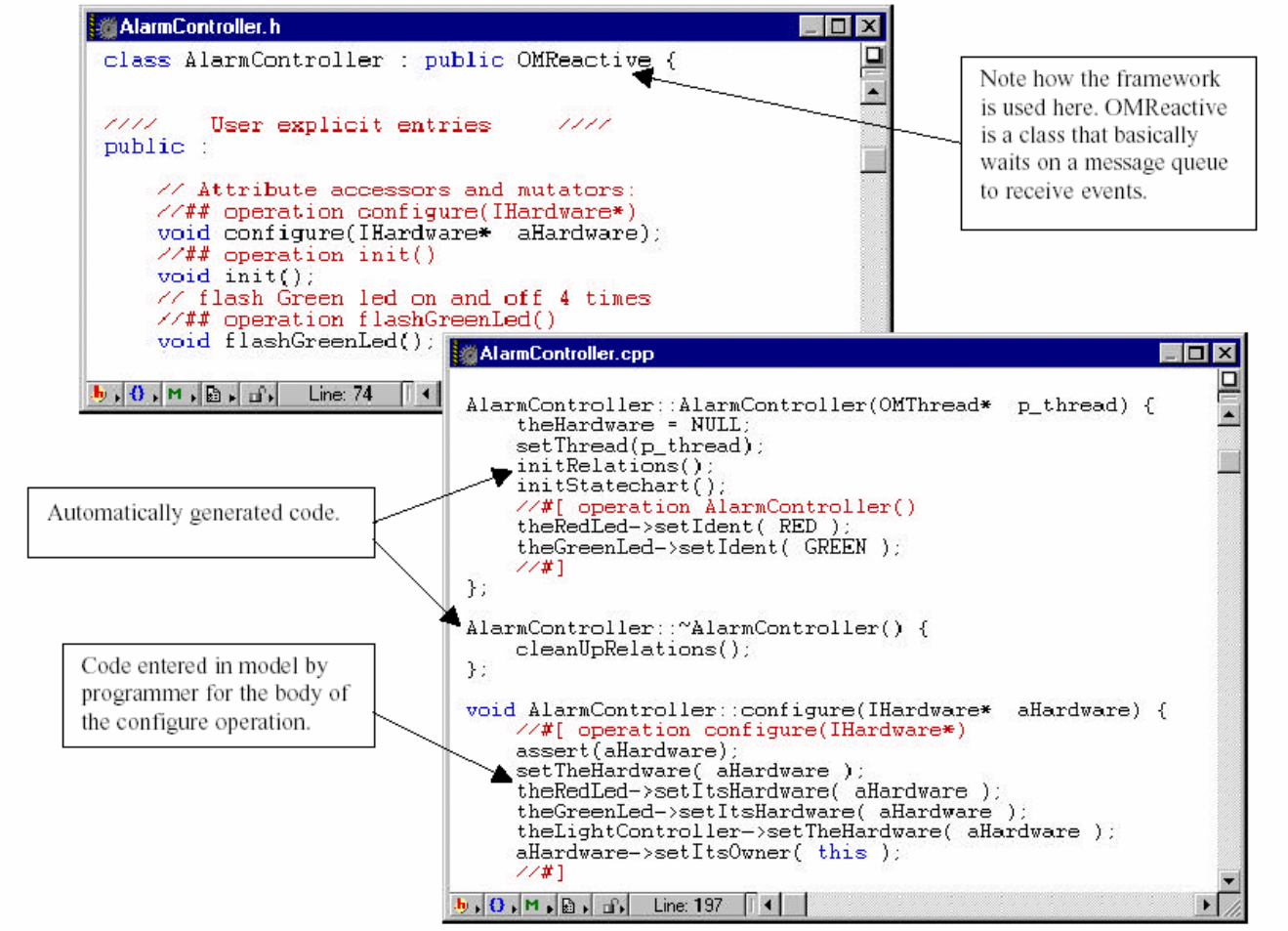


像 Rhapsody 这样的工具可以根据 UML 模型自动生成代码，当然经常是其中一些代码要手工修改，如类的操作和状态图的动作，这些可以在工具中完成（如上图）。

产生代码

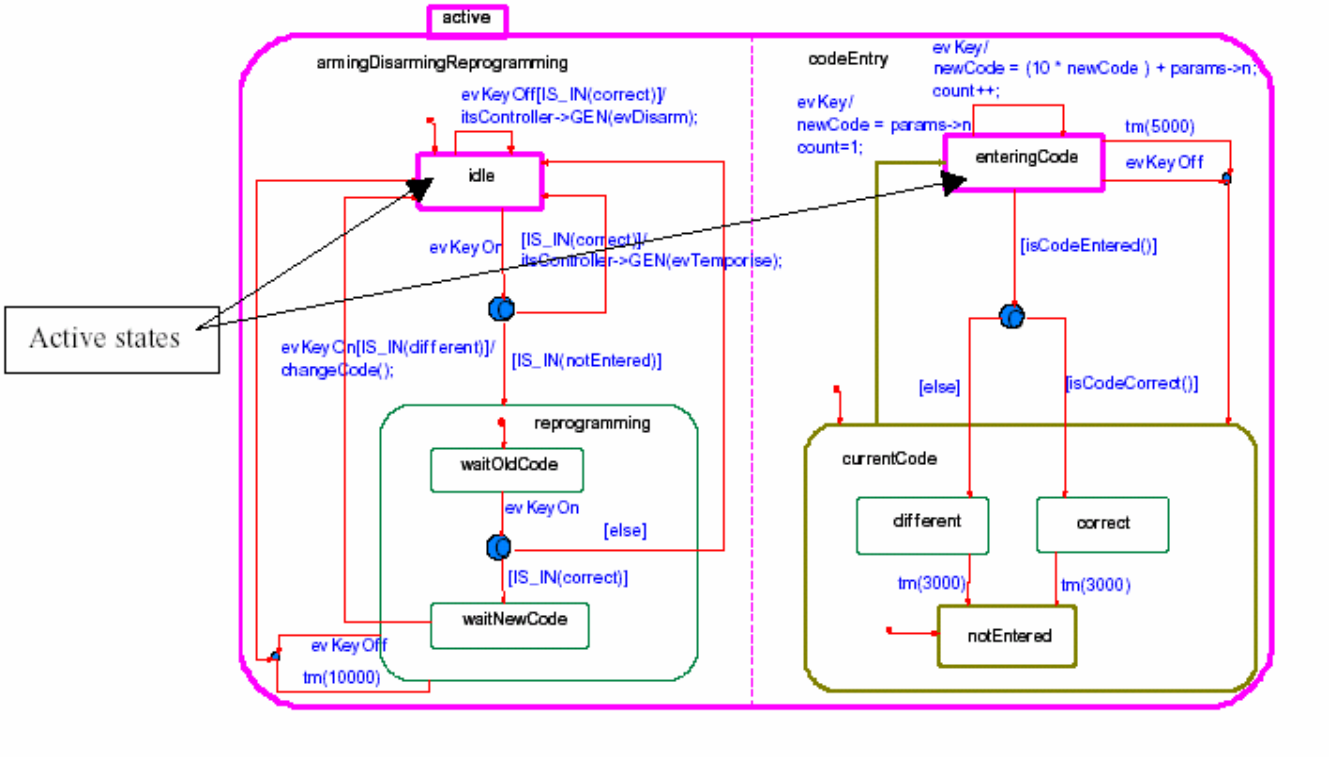
UML 是严格定义的，足以从状态图和对象模型图自动地生成代码。每次做面向对象的项目时，需要建立某些类型的框架。这些框架需要可能实现一个处理状态图的机制，可能有与 OS 接口的机制，也要描述如何实现关系，不同线程上的类间如何通信，如何处理超时，… 这个框架可以提供实现应用的大约全部代码量的百分之六十到九十。试想开发一个 Windows 程序不用 MFC 框架的情况。Rhapsody 提供了一个框架（为嵌入实时项目设计和优化）使嵌入实时系统程序员可以专注于应用而不是底层的实现。当使用了这个框架后，自动生成代码变得高效和相对直接。Rhapsody 框架也包含了一个抽象的操作系统接口使模型可独立于实际使用的 OS。

一旦其余的图画完，并且不同的操作代码体也完成了，我们就可以生成代码和测试模型。Rhapsody 里我们需要建立一个组件和配置告诉 Rhapsody 从什么类中生成代码和应用在什么操作环境中，通常刚开始时，我们使用 Microsoft 环境（Windows 操作系统和 Visual C++编译器）。那时就可以生成代码并在 Rhapsody 中编译成可执行文件。下图中是 AlarmController 类生成的代码的一个摘要：



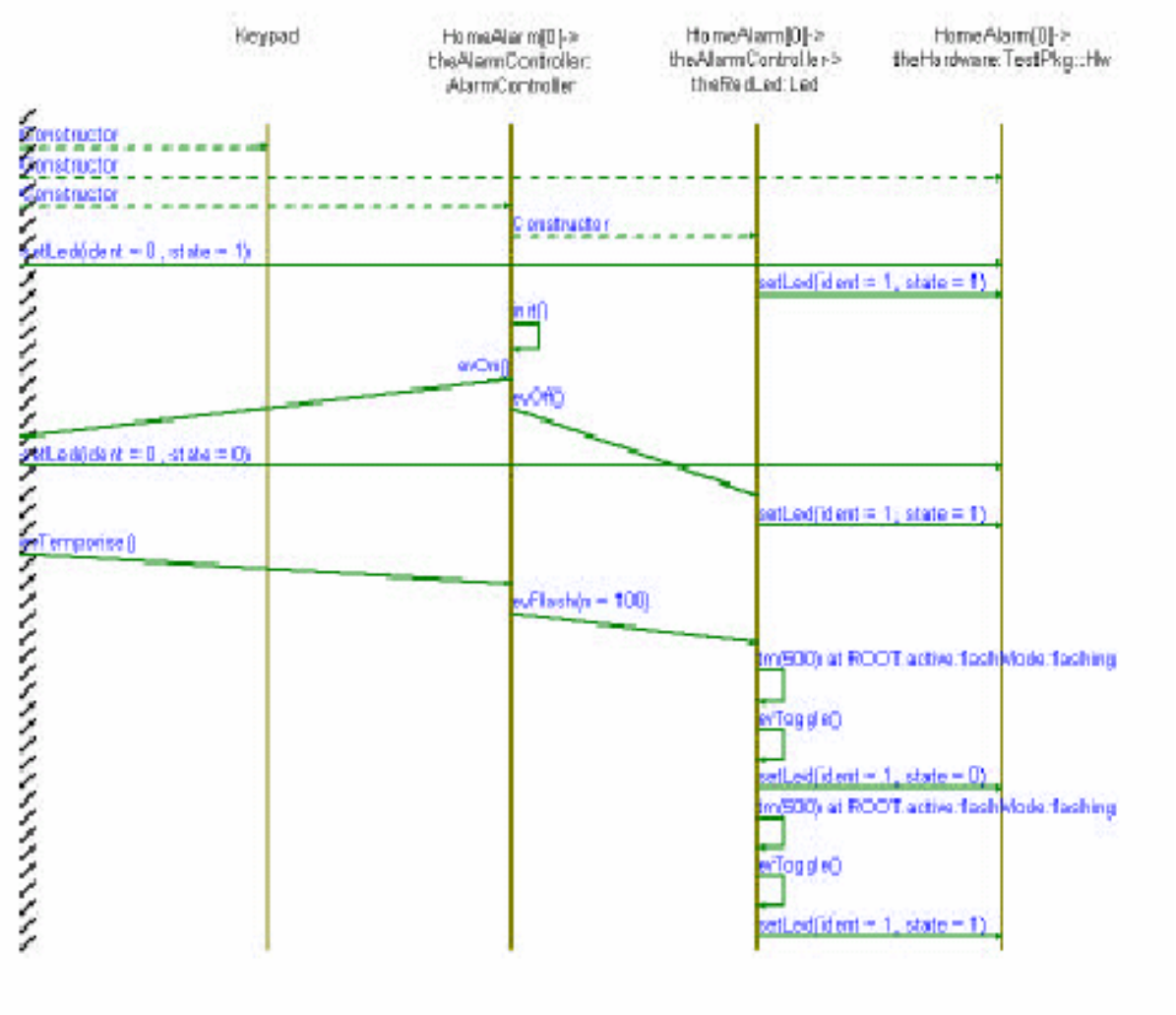
UML 模型验证

Rhapsody 会在生成的代码中通过一些手段使其在运行时向工具返回一些信息，使模型活动起来。这样就允许程序员可以在设计级就可以进行调试。可以在开发的任何阶段进行，甚至是第一天程序员没有写一行代码的时候。设计级调试可以使问题在设计周期的早期就可以被发现并马上改正，所以可将产品更快地推向市场。

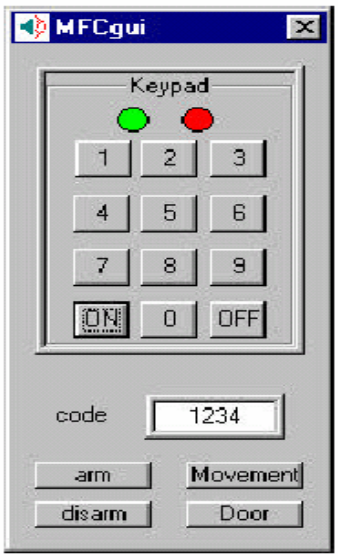


在上图中我们可以看到一个活动的状态图显示了 Keypad 类的实例在空闲（idle）状态和密码输入(code entry)状态。激活的状态通过高亮显示。

事件可以手工插入进来，象下面这样的活动的顺序图可以用来自动捕获事件发生时实例间的交互情况。



这个活动的顺序图可以检查或者甚至在分析阶段可与手工画的顺序图相比较以保证模型的正确。这种比较当然可以让 Rhapsody 自动进行。



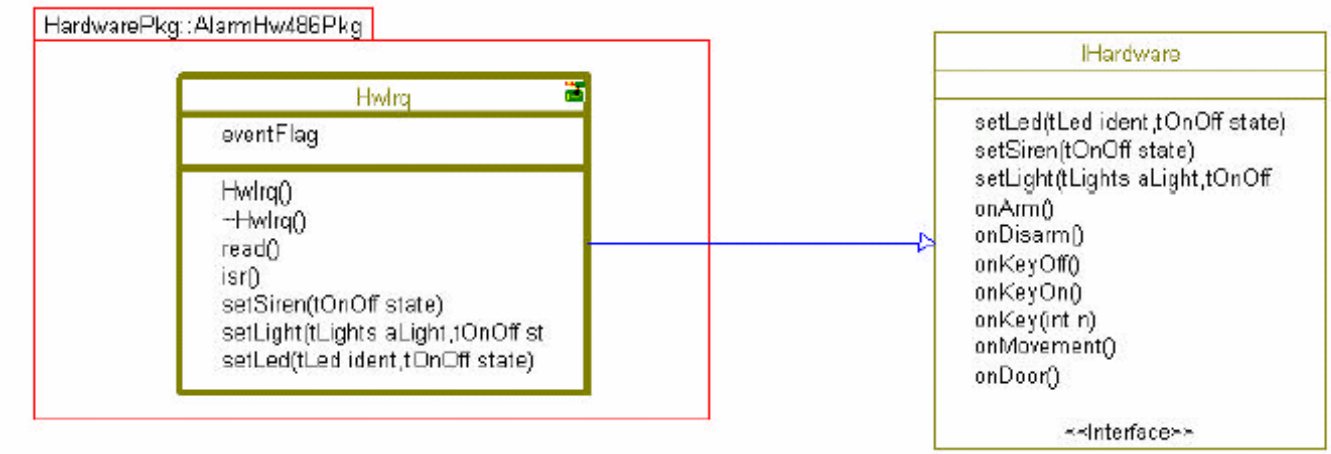
用简单 GUI（图形界面）测试 UML 模型

警报器领域一经过手工插入事件并观察看有什么事发生的测试后，可以建立一个简单的 GUI（图形界面），例如用微软的 Visual C++。用于建立这个 GUI 的类可以从 IHardware 类中继承，特别是“set”操作和当一个按键按下时调用“on”操作。这个 GUI 可以用于驱动模型又可以进行设计级的调试。

使用这个 GUI 也使客户和市场方面验证产品是否如其期望成为可能。模型任何必要的修改可以很容易而且简单地按一下按钮就可以生成新的可执行程序。

第二部分：目标操作系统（VxWorks）

一旦家用报警器通过设计和验证，就可以设定目标为 VxWorks 操作系统，比如说运行在一块通过 I/O 板连有小键盘、两个 LED、一个警笛、一个移动检测器和门开关的 ns486 主板上。小键盘、移动检测器和门开关这些硬件都产生中断。

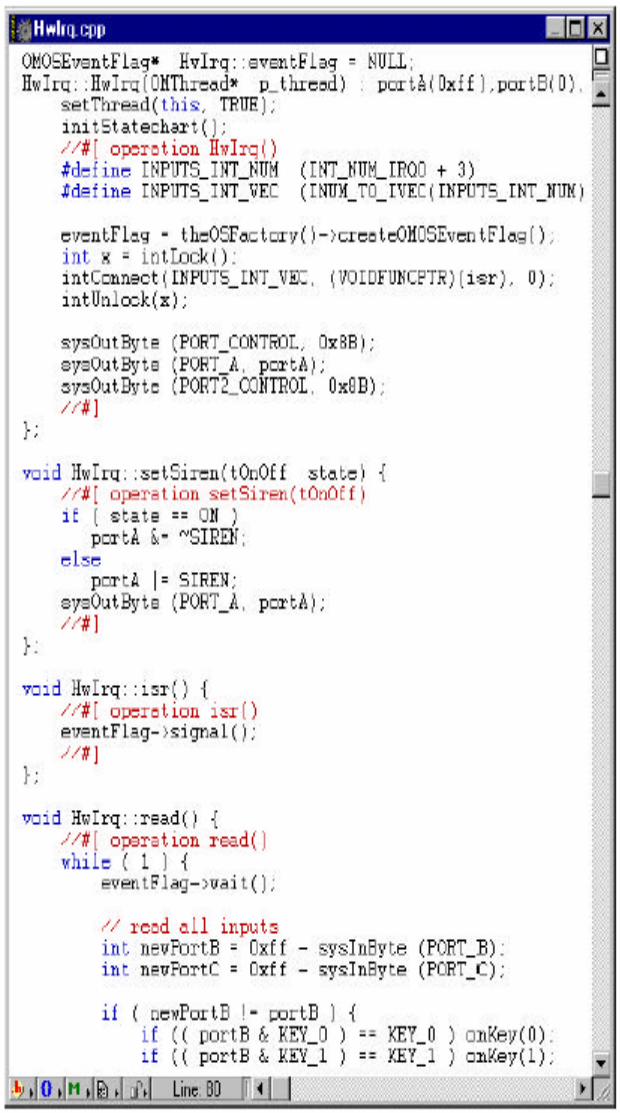


一个新的继承自接口类 IHardware 并有“set”操作的 HwIrq 类被建立。这些操作代码体可以在 Rhapsody 中写。要向 I/O 板进行写操作，用到了 VxWorks 的 sysOutByte（）操作。

构造子建立一个输出端口和两个输入端口。也用到了 VxWorks 的 intConnect（）操作。所以当第三中断（IRQ3）发生时调用 isr（）例程。最后建立一个事件标识给 isr 例程用。isr（）例程是个静态操作，只是简单地在读操作等待时设置事件标识。读操作用到了 VxWorks 的另一个 sysInByte（）操作来读入输入信号。

在 Rhapsody 中我们需要建立一个新的组件和配置来告诉 Rhapsody 哪些类要生成代码和运行在什么环境中，在这里，我们用 VxWorks486 的环境（VxWorks OS for 486 和 gnu 编译器）。然后代码可以在 Rhapsody 中生成并被编译成可执行文件可以下载到 Tornado 调试器执行。

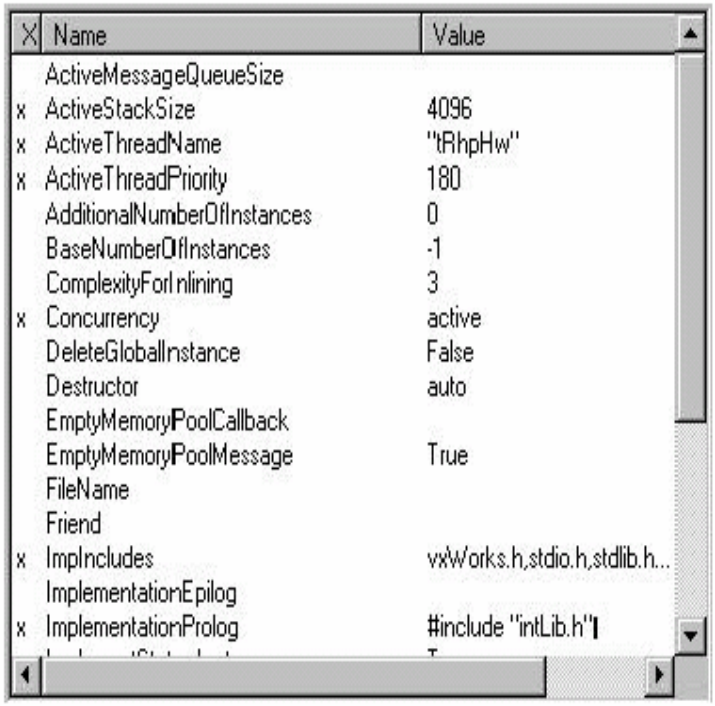
如果代码生成时被设置过，Rhapsody 就可以像前面描述的一样，允许对模型进行设计级调试。Tornado 可用于源代码级调试，同时 Rhapsody 可以用于设计级调试。



HwIrq 类被设置为活动类，所以有自己的线程，这个线程的参数可以在 Rhapsody 中简单地配置，如图中设置线程名为“tRhpHw”，堆栈大小为 4096bytes 和 180 优先级。

最后，可以从模型生成不含任何其他设置的代码，就可以用经典的 Tornado 方式进行调试，可能在发布前还要用到 WindView 来检查系统性能。

文档也可以通过按一下按钮从模型中产生，项目做完后谁会喜欢花时间（或者确实有时间）做文档啊。



结论

本文说明了如下信息：

1. Rhapsody 如何用来进行 UML 建模
2. 如何使模型能够进行设计级调试
3. 如何使模型可以通过外部的 GUI 进行调试
4. 如何使模型用于运行 VxWorks 的特定硬件平台

参考资料

	Author	*Reference*
1	*Bruce Powel Douglass*	*"Doing Hard Time" ISBN 0-201-49837-5*
2	*Bruce Powel Douglass*	*"Real-Time UML" ISBN 0-201-32579-9*
3	*I-LOGIX Inc*	*http://www.ilogix.com*

UMLChina

非程序员

软件工程杂志，已有数万用户

UMLChina讨论组

已有数万名成员

UMLChina.com

1999年11月成立，，专注UML技术的应用

专家交流

提供与世界级专家交流的窗口

使用模式和 XP 构造复杂的面向对象系统

Eduardo B. Fernandez 著，曲俊生 译

查看评论

摘要

复杂是很多重要系统的特性。这些系统都具有大量交互的实体，复杂的约束，同时需要满足非功能性需求。在语义分析模式（SAP）中，每个模式都对应一系列用例。我们利用 SAP 在增量式开发中建立一个整体的概念模型。这个整体模型为 XP 提供了一个能够考虑分布式、安全以及测试性等因素的结构。SAP 也可以应用在 XP 的每个增量开发阶段，从而保证良好软件开发原则的应用。

关键字

分析模式、复杂系统、概念模型、轻量式开发过程、面向对象分析、模式、用例、极限编程（XP）。

1 介绍

XP 大量地被应用于构造中小复杂程度的系统。由于 XP 使用的增量式开发模式——每次实现一个或者几个类[1,14]，一些整体因素，例如分布式、授权和并发等因素很难被考虑到。但是，很多重要的系统，例如制造、导航以及业务计划等，都是非常复杂的。它们之所以复杂，是因为这些系统中充斥着大量交互的实体、各单元之间的关系、各种变量值的复杂约束，同时，它们还要满足很多非功能性的需求。

XP 强调即时实现，没有明确的分析和设计阶段，因此缺乏整体观。由于 XP 没有明确地去除分析与设计阶段，因此就很少有建模的激励。在 XP 中，非常典型的操作是利用了部分模型，而后就被抛弃了。这种做法，阻止了在整体概念模型中发现共性和进行优化。代码重构并不能纠正某些概念上的问题或者考虑非功能性等方面。

模式使得我们利用前人的知识，从正确的方向构造模型。如果使用的模式是经过仔细挑选的话，它们就体现了良好的设计原则，同时，使得设计者能够透明地应用这些原则。所有这些，都保证了产生软件的高质量。

我们提倡首先建立系统概念模型，为整个（或者尽可能的接近整个）系统撰写测试代码。同时，推迟系统的实现直至模型接近完成。这样就使得设计者有时间考虑整体概念和构造更加复杂的应用。类仍然被增量地开发和测试，只是现在我们有一个整体模型的指导。我们前面已经提到了 SAP－小步实施，每次实施一系列基本的用例。SAP 可以用来增量构建整体模型。它也可以应用在 XP 的每个阶段，从而保证良好软件开发原则的运用。

2 SAP（语义分析模型）

分析模式是一系列的类和它们之间的关联，它们在一个应用中具有某些含义。也就是说，分析模式是部分应用的概念模型。但是，同样的结构对于其他的应用也许有效的，因此从重用和组合的角度考虑，分析模式就显得很有价值。分析模式可以是原子模式或者是组合模式[15]。分析模式由 Hay 的工作发起[11]，同时在参考文献[2]和[9]中，也有这方面的研究。需要指出的是，在这些参考文献中讨论的分析模式都是原子模式－它们由少数类组成；而我们的兴趣在于大型模式。从重用性方面考虑，大型模式更加有效。它们可以用来构造概念模型。我们已经提出了一种称之为 SAP 的组合分析模式。SAP 是对应少数基本用例的最小应用，它定义了一个可以与其他 SAP 联合使用构造复杂系统的语法单元。其中，用例的选择能够保证模式被应用到大量的应用系统中。我们已经提出了几个 SAP，其中一个是用于库存管理，一个用于可复用实体的预订和使用。

3 SAP 开发

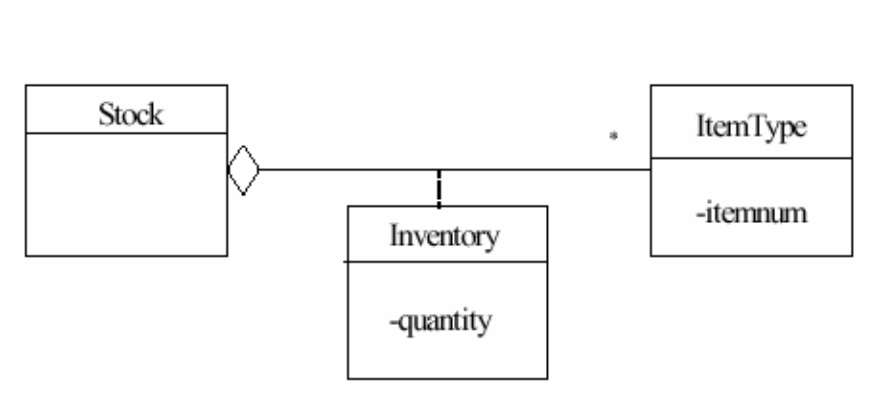
SAP 基于众所周知的原则，例如抽象、聚合、低耦合和 Regularity。它们也使用了例如授权、精确和测试性等其它原则[6]，尽管在大多数方法中没有使用这些原则。我们以 SAP 在库存管理中的应用为例，来诠释某些原则在开发中的应用。

抽象

抽象是面向对象方法的最基本原则。它提示在模型中只是包含核心的方面，剔除无关或者不需要的细节。我们利用一个库存系统的基本模型来解释这个概念。库存系统的基本功能可以概括为：

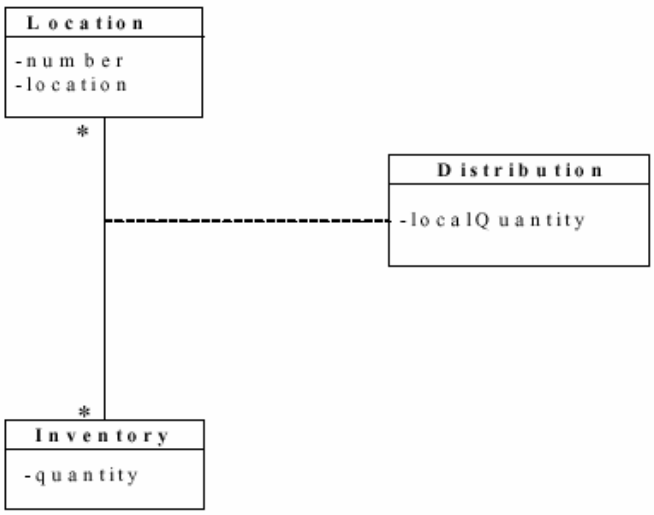
1) 记录各种库存。记录每个单元的库存数量，其中需要计算几种数量，例如库存实际数量和可供应数量。

2) 记录库存单元的位置。库存目录应该记录特定位置的库存单元分布情况。库存位置的划分应该便于快速定位库存单元。不同存储位置间的库存物资流动应该在库存目录中有所体现。



图一、 基本库存系统

最基本的库存目录只是记录某种库存单元的库存数量(图一)。其中，库存单元可以是成品、生产中使用的部件以及机械等；换句话说，库存单元是任何我们希望了解其存在和数量的物件。每个库存单元都属于唯一的一种类型。这就是对应于需求 1)的基本抽象。本模型应该作为任何库存系统模型的一个组件。类似地，我们可以对于其它的需求进行抽象。为了实现需求 2)，我们需要将库存数量分解为库存位置数量（通常意义上讲，一种库存单元可以存储在不同的位置）。这个模型有图二表示。



图二、库存分布

我们可以将图一和图二作为原子分析模型。它们所描述的抽象对应下列两个基本的用例：

- 记录具体库存单元的数量；
- 记录这些库存单元的存储位置。

分解

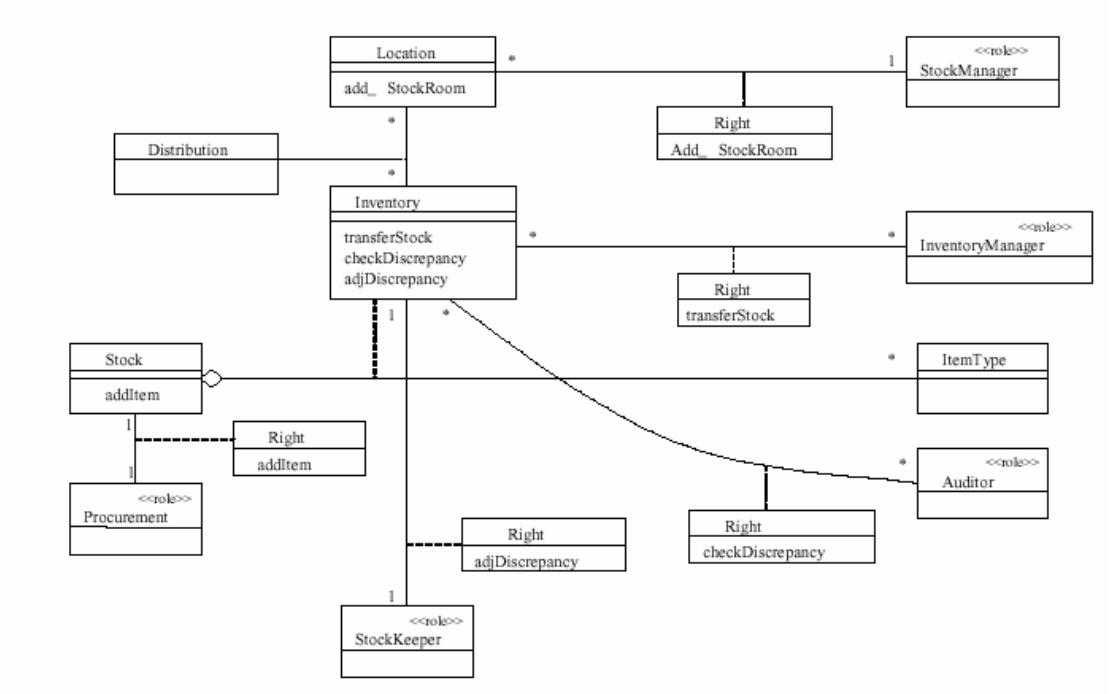
为了给一个复杂的系统建模，我们需要运用分解原则。例如，一个生产制造系统是非常复杂的，难以用一个单元来进行描述和处理。这个系统可以分解为一系列的 UML 包，例如库存控制、店铺订单、顾客订单、运输以及财务等。各个包之间的交互对于定义每个包中类的方法是非常重要的。分解对于 XP 是非常重要，在 XP 的每个阶段，对于需求的处理，都有意或者无意地运用了分解的原则。

聚合

各种模式可以组合起来构造大型的模型[6]。例如，我们可以将图一和图二中的模式组合起来，形成一个新的模式来记录库存单元的数量和它们的分发情况。图三是这个复合模式的示例。非常明显的是，这个原则在 XP 中是基本的，因为，我们需要将每个阶段的成果组织在一起。

投影

投影是将系统中不同的视图组合起来，形成一个更加完整的图形。为了描述一个机械部件，我们需要将不同的视图组装起来；一张单独的视图对部件的三维特性只能给出歪曲或者不完整的描述。XP 显然缺少这条原则，这就需要以某种方式将其引入。



图三、加入授权机制的库存系统

低耦合

这条原则是在设计模式中经常用到的。它的主要的目的是提高灵活性。这条原则在 Fowler 的分析模式中也被大量使用，它将一些方面分离出来，使之独立演进。

在上面提到的例子中，如果我们对于库存单元有一个详细的描述，我们可以利用 Composite 模式来递归地描述这个结构。这样就将一个组件或者产品的结构特性与其它特性解耦。我们也能将库存单元描述的其它方面解耦。

精确

图一所示的模型对于表述需求的某些约束来说，不是很精确。例如，为了说明在库存位置之间传递的库存总数量必须是一致的，我们需要添加一个 UML 约束－即，在 Inventory 类中添加 {库存的和必须一致}。

但是，在很多情况下，这并不准确。如果我们有更复杂的约束，这种描述就很模糊。例如，在那些对于安全要求非常严格的应用来说，这种二义性就不能容忍。Z[3]和 OCL[16]都用来为 UML 约束增加准确性。同时，在[13]中给出了作为约束语言的 Z 和 OCL 之间的区别。

授权

当我们使用原子模式或者 SAP 来构造系统时，我们了解这些单元对应的用例的角色。角色需要特定的权限来

执行用例中定义的操作。根据基于权限的访问控制原则(RBAC)，我们可以定义授权原则。例如，如图三所示，我们在库存系统中加入了授权机制——StockManager 等暗示着某些权限(以<<role>>表示)。在这个例子中，InventoryManager 被授权在各个存储位置之间转移库存。

测试性

用例与系统中的交互相对应，同时定义了各种动作的时序。而这种时序，可以用来测试系统[12] 。因此，一个按照用例建立的系统在很大程度上可以被测试。例如，对于库存系统，我们可以建立测试用例来执行这个应用的一系列动作，同时检查在店铺订单更改状态后，库存的数量是否也得到了更新。

4 SAP 和 XP

有意识地运用良好的软件开发原则需要大量的实践经验，同时，也比较容易犯错误。对于大部分的用户来说，更好地运用这些原则的方法是隐性使用 SAP 来构建系统的概念模型。正如[7]中所描述的那样，下面我们要介绍建立系统概念模型的步骤。我们假定我们已经拥有一个原子模式和 SAP 的目录。我们仔细研究了用例和(或者)其它需求，同时：

- 查找 SAP。我们首先查找那些与需求最接近或者十分接近的模式。接着，我们再尝试寻找可能运用的模式。
- 寻找原子模式。

上述步骤最终形成了一个框架，其中，模型的某些部分是相对完整的，而其它内容被部分包含甚至完全未被包含。尽管我们仍然需要将模型的剩余部分包含进来，但是我们已经有了一个其实模型。非常自然地是，我们在设计阶段可以继续添加模式。

这里要阐述的是，如果模式是在仔细应用这些原则的实践中产生的话，那么一个运用 SAP 的用户在运用原则时并未意识到他正在使用它们。这样就导致了一个高质量模型的产生。通过将每个新类从整体的角度构建，这个模型可以用来指导 XP 的设计。单独的 SAP 可以用在 XP 的每个阶段。一个良好的概念模型是易于修改的，同时也与变化的需求相适应的。一些非功能性的需求，例如上面提到的安全，应该在整体概念模型中定义。分布式、并发、实时期限以及容错性等也应该在这个层次上定义。当然，底层的结构层次必须实现和执行概念模型中定义的非功能性约束。

5 结论

我们已经建立了几个原子模式和 SAP，其中，运用了很多良好的设计原则；同时，我们也建立了一个分析模式目录，利用这个目录，即使是没有太多经验的设计人员，也可以设计出高质量的概念模型。所有这些都可以作为 XP 开发的基础，在 XP 开发中，通过将部分的实现与整个系统模型联系起来，SAP 可以在各个阶段起到引导的作用。利用上述原则建立的整体模型可以用来定义诸如分布式、安全以及其它非功能性需求。这些想法已经在两个大学的学生中经过验证，但是业界的验证是必须的。

参考文献：

1. Beck, K., Extreme Programming explained:Embrace change, Addison-Wesley 2000.

2. Coad,P.,"Object models: Strategies, patterns, and applications" (2 nd Edition), Yourdon Press, 1997.

3. Cook, S. and Daniels, J., “Let’s get formal”, JOOP, July-August 1994, 24 and 64-66.

4. Fernandez, E.B., and Hawkins, J.C., "Determining role rights from use cases", Procs. of the 2nd ACM Workshop on Role-Based Access Control, November 1997, 121-125.

5. Fernandez, E.B., and Yuan, X., “An analysis pattern for reservation and use of entities, Procs. Of PLoP99 , http://jerry.cs.uiuc.edu/~plop/plop99

6. Fernandez, E.B., "Principles for Building Complex Object-Oriented Conceptual Models", Tech. Report TR-CSE-00-24, Dept. of CSE, Florida Atlantic University, August 2000.

7. Fernandez, E.B., and Yuan, X., “Semantic analysis patterns”, Procs. of 19 th Int. Conf. on Conceptual Modeling, ER2000, 183-195.

8. Fernandez, E.B., “Stock Manager: An analysis pattern for inventories”, Procs. of PLoP 2000, http://jerry.cs.uiuc.edu/~plop/plop2k

9. Fowler, M., Analysis patterns -- Reusable object models, Addison- Wesley, 1997.

10. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., "Design patterns –Elements of reusable object-oriented software", Addison-Wesley, 1995.

11. Hay, D.C., Data model patterns -- Conventions of thought, Dorset House Publishing, New York, 1996.

12. Jacobson, I., Booch, G., and Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1998.

13. Jiang, Z., Fernandez, E.B., and Wu, J., “Comparing OCL and Z as constraint language for UML”, Tech. Report. TR-CSE-99-28, Dept. of CSE, Florida Atlantic University, May 1999.

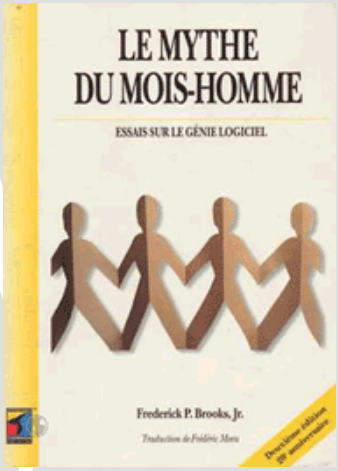
14. Martin, R.C. and Koss, R.C., “The Bowling Game. An example of test-first pair programming, “, http://www.objectmentor.com/publications/articlesBy Date.html

15. Riehle, D., “Composite design patterns”, Procs. Of OOPSLA’97, 218-228.

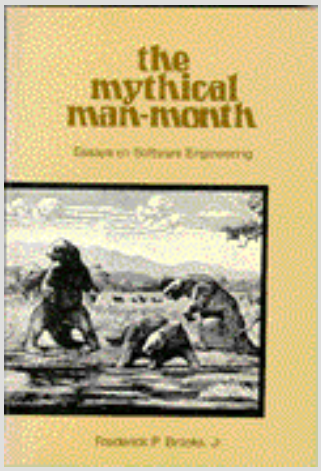
16. Wanner , J. and Kloppe, A., The Object Constraint Language: Precise modeling with UML, Addison-Wesley 1998.

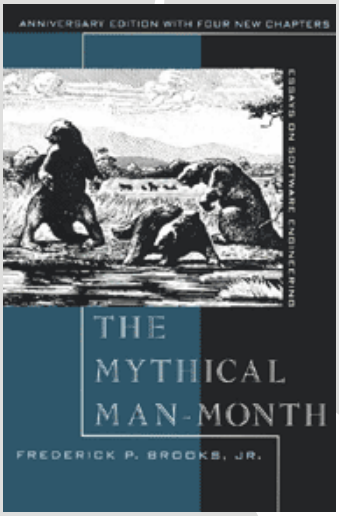
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)





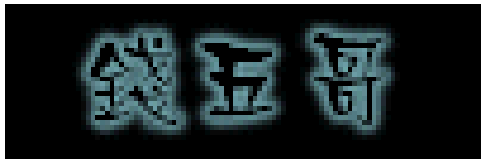
(第二版)

(第二版日文译本)

钱五哥答疑录（一）

整理自钱五哥在 UMLChina 的答疑板，网址：http://www.umlchina.com/expert/qlw5.htm，截止至 2002 年 11 月。钱五哥，计算机科学博士，专业方向：软件工程。现为 Bell Lab Research China 高级研究员。研究方向是软件工程，注重在软件过程模型、软件过程改进、软件项目调度等方面的研究。钱五哥的主页：http://qlw.126.com

查看评论



小岩：我想知道在软件产品工程的过程中，输出很多设计开发的文档和代码，如何考查这些产品，对不同项目的的文档和代码，是否可以制定出相同的检查表呢？请五哥指点

qlw：Checklist 一般是复用以往的开发经验,因此对于不同项目的同样产品,完全可以使用相同的 Checklist. 但是效果的好坏决定于这两个项目的特征,如开发类型, 产品结构, 组织模型和采用的过程. 一般来说, 同一个组织的项目还是有共性的.

小岩：谢谢五哥，我在写详细设计和编码的 checklist，好难写:(刚写完编码的，有一个问题，如果详细设计在软件过程定义的时候被裁剪掉了，那代码的 checklist 是否要包括详细设计中一些检查项呢？还有我现在写的编码的 checklist 好像只写了风格方面的事情，这样是不是不全啊（即使做了详细设计）。

qlw：不知道你说的和详细设计相关的 checklist 是什么,一般来说,没有这个必要, 它们考虑不同的问题. 除了风格以外, 完全可以考虑: 可移植性; 代码重复; 结构复杂性等等. 这些 Checklist 是经验的总结, 没有经验,也就没有 Checklist 了。

小岩：不瞒五哥，我也不知道详细设计 checklist 该写些什么。主要是我以前写的详细设计都是边写代码边写详细设计的。惭愧惭愧，现在正对着模板发呆...代码的 checklist 我写了，我最关心的代码的可维护性。我经验少，希望五哥指点。

qlw：到 www.rspa.com 上面看看,里面很多的 Checklist,以及到其他 Checklist 的 Link, 可以参考的。

==========

kinsing：There are many so-called exporters in the document review .but they can't give me more useful advices .what a pity. and the developers don't want to check other's documents, because they don't think it is their duties .can you tell me what can I do to make my system more stable? Thanks a lot. And can I email you?

qlw：If the reason is because of PEOPLE mostly, I can hardly offer any help. The people are almost unstable in SW development. I get to know there is an award-winning book called "Peopleware", hope it can give some ideas.

==========

John Zhu：五哥对形式化语言的前途如何看

qlw：形式化语言一直是人们想要而有没有得到的东东. 据我所知,B,Z 语言应该在使用中某些工具也开发出来支持它们. 不过它们的应用范围相当局限. 随着应用系统逻辑复杂性的提高, 对形式化语言的需求有涨. 不过还是应该各行其道。

==========

liz：请教钱五哥您对软件测试熟悉吗，能否指点一下有哪些资源或者相近的资源可以寻找？（我只找到 www.cetus-links.org，对照着它的 oo_testing link 找别的没找到）

qlw：关于 OO 测试的资料.，这些 Link 可能有用：

(http://www.mtsu.edu/~storm/literature.html)(http://lglwww.epfl.ch/Research/OO_Research/toos/)。

==========

qw.cnwinds：我原来一直是以面向过程的方式编程，一般的程序也能独立完成。但现在学习 ooad 的感觉是分析出来的东西不知道如何写出最终的软件，（主要是接口的定义，类之间的抽象程度的控制）所以也不知道分析的对不对。可否介绍一些有完整例子的书或文章给我学习呢？非常感谢！

qlw：OOAD 本身无法检验分析或者设计的正确性,我想各种 OOAD 方法主要是为了提供分析和设计阶段的一致性, 并试图解决分析设计和编码的一致性. 而对于正确性检验是另外的研究课题, 也许形式化语言可以解决这个问题。

==========

Zjb：请问有关 CVS 的使用要点，比如说如何进行用 CVS 进行管理，你能否介绍一下哪里有比较好的文章。谢谢

qlw：CVS 是通用的 CM 工具, 有人在几十万行代码的项目中使用过 CVS, 可见是有一定的管理能力的. 我想主要的管理行为集中在(1)权限管理(2)版本管理(3)Build 管理. 由于 CVS 本身只是一个 CM 工具, 需要结合很多具体的 policy 来辅助 CVS 的使用。

==========

ellen：钱五哥：你好！ 请问一个项目的报价原则主要从哪方面考虑？各方面占百分比大概是多少？

qlw：我想无论是承包商，还是提供商， 都会从一下几个方面考虑经费问题：(1)硬件费用(2)软件开发费用(3)软件维护费用(4)市场情况(5)风险. 其中(2)会比较复杂. 与(1)项目类型(2)第三方软件使用(3)性能要求(4)质量要求. 都有很大的关系。

==========

Jack Xiao：I never think about project manage and other things before, Now I try to enter this area. I found it too difficult to operate if we try to execute a develop plan according the rules defined in Rational Unified Process. There is too much document to create and too much procedure to complete. I have no experience on completing a plan in a formal way, so, how do you complete a development plan step by step. Any advice on reading material or others.Thanks!X.M.Ch.

qlw：I think if you are following the plan, whether the plan is RUP style or other style, you will have to follow the procedure, which may require a couple of documents. That is PM. However PM activities are based on the type of projects. If the project is focus on speed and changing requirement, you may use XP or other kinds of lifecycle, which is light-weighted, if the project is focus on Product Quality, you will need RUP or CMM compatible process. In my past projects, XP-like procedure is used. Currently CMM compatible Procedure is used.

==========

邓红波：Testability 体现在哪些方面？如何去度量？

qlw：Testability 是一个很大的问题,就我的体会说两个方面:(1) 需求的可测试性在于需求描述的是否清晰,accept criteria (2) 设计的可测试性在于设计是否合理. 比如模块化程度。

==========

ysy：自己目前正在做的事情是如何建模一个软件过程,使之体现 CMM 的思想,也即:怎样在过程中体现 CMM 的一些关键过程域,比如配置管理,需求管理,软件项目计划等.因为 CMM 并没有指出怎样实现这些东西,所以我们要想做一个基于 CMM 的平台,帮助企业实现过程改进,就必须要解决上述问题.想问一下你是如何看待这个问题的.谢谢!

qlw：我想你的意思是设计一个方案/工具,使之 100%或者已知比例的满足 CMM 的要求,这个事情有一些困难, 主要原因是 SEI 并不提供对工具的评估.是否达到 CMM 的要求是 Lead Assessor 针对项目一个人决定的. 当然辅助项目管理, 使之靠近 CMM 的要求是可行的, 目前很多工具都说自己如何好, 如何满足各种标准, 其中包括 CMM.

==========

westwf：我在程序员中听到了很多对软件工程师的误解，他们认为系统分析员是客户需求的二传手而项目经理是客户的代言人和监工。他们都是吸程序员血汗的寄生虫，您能否帮我想个方法说服他们呢？

qlw：[代言人和监工]的问题可以说是一个开发模式的问题,现实中,多数瀑布模型就是这个样子,程序员没有太多自己发挥创造性的机会,很多企业认为这是很正常的事情,即员工应该各尽其责.但这是对程序员的一种伤害,即 PM 和分析人员在达成协议时,并没有真正征求开发人员的意见. SW-CMM v1.1 中对此有明确的解释. 目前国外和国内的某些企业在实施 XP 模型,其中程序员的作用得到很大提高.

==========

邓红波：我的意思是当软件交付到测试阶段时，如何评估它的可测试性，以及如何根据它的可测试性来设计测试以最大降低人员和资金的投入？

qlw：[可测试性]的好坏是在测试阶段之前便确定的, 因此这个评估也不应该在测试时才进行. 抛开时间问题不说, 如果一定要评估可测试性,那么可以依据你的需求描述和设计定义,以定量或者定性的方式来评估. 依据可测试性来设计用例似乎并不准确, 可测试性是能否测试的问题, 设计和实施测试用例是一个测试目标的问题. 不同阶段的测试,UT/IT/ST 都应该有自己的测试目标.

==========

Jack Xiao：I don't know what’ s the meaning for PM,XP。Now I'm reading a book ,I think it's very good.In this book, the author provided some docs needed built. I don't know whether you have read it,If we follow his(their) advices,the docs is not too much. But I always doubt whether it is enough.When you use CASE like Rose, the actually created docs is few: use case, class diagram, interactive diagram...;of cource,you should attached many docs into approriated diagram.I don't believe that there is a project developed completely based on a formal method.Time to time,I doubt what we should created,and why.I have seen too much docs created just for project: no use,no actual target.If possible,would you pls send me aclassical one you have developed(just docs)?Thanks! The book name not printed:System design and snalysis in a changing world(English Edition)

qlw：I think whether the document is enough or not is determined by your development policy, i.e. you managers. The control of document is more important than the document numbers. I have not read the Book you have just recommended, from the name, it seems a practical book. Rose will not generate much document, however the control software work product, i.e. the Diagrams is more important. I think the purpose of document is for project management and inter-group communication. Sorry to tell you that I am currently not focus on project development, old project are more hacker style and less document can be found

Jack：Thanks.I think I don't have enough practice on project development in OO,so don't be sure where to go and how much is enough.Could you pls introduce some docs or books on project process control,maybe you have write some one:-)?To my comprehension,the documents is the products of project development process.As you said,the document is for project management and inter-group communication.For chinese software engineers,most of them can provide excellent solution,but in mind,not on documents, so am I :-). If we can't create needed document, the project is half failed. How to control to build right documents on time in each step(analysis,design,implement,deploy...)?

qlw：Whether the document is adequate for your project is dependant in your project manger and senior management, and the necessity of document in their minds are probably the project management and communication

==========

babyblue：我想问一下有关工作量估计的问题，首先工作量的估计什么时候做？怎么做？工作量的修改如何反映到项目计划和其他方面？谢谢！

qlw：工作量的估计发生在项目计划时,主要通过历史数据,PM 经验和当前项目规模的估计. 工作量发生改变时应该及时调整项目的进度。

==========

Weiyunliang：小弟对 JAVA 的发展前景不是很清楚，请五哥指教。

qlw：虽然 Java 有一些天生的缺陷, 但 java 的好处可能比缺点更显著.应该是一个很好的方向, 但决不代表 Java 可以 Cover C/C++, 也许将来可能会取代部分 RAD 工具. Java 在嵌入式开发中的前景我不十分了解,不过似乎和嵌入式的系统对性能的要求有矛盾。

==========

Jingle：请问五哥采用组件技术(比如 COM)进行系统分析和设计的步骤是什么，和传统的面向对象设计方法(如 OMT)的区别是什么，ROSE 对组件技术提供了那些支持？

qlw：我想从分析和设计过程上面看基于组件和基于对象的并没有很大区别. 对组件和对象在系统中的地位也是类似的, 但是基于组件的开发需要依据系统需求来选择和评估组件, 或者反过来依据现有组件的功能来划分系统功能. Rational 就有一些关于这方面的 guide。

==========

gsyn77：近来正在从事 web 开发工作，对于 web engineering 有没有什么需要特别注重的方面？如果不是采用 oo 的方式，比如说 j2ee，.net 等，是否也可以运用 oo 的分析和设计方法?。

qlw：Web 上面的脚本构造确实和一般的结构化开发有所不同，代码本身很容易变得重复冗余，可以参照 refactoring 的思想来优化. 关于分析设计,OO 当然没有问题. jsp/applet 可能会比较好，至于说到使用 COM 技术，我个人不十分看好。这个东西不属于开放系统。

==========

王永锋：五哥，能否告之小弟一些自动测试工具的网址，小弟特别需要一些测试工具测试软件！多谢五哥！

qlw：自动测试的工具非常多, GUI 方面的比较好的就是 rational 的 SQA，也许现在改名了，可以到 www.rational.com 上面看，还有 visual test. 另外还有很多协议测试工具，这个具体的情况就比较复杂了。

==========

xftang：能不能描述一下 'Software architecture'和 Framework 和 Design pattern 的关系？Framework 本来应该就是面向领域的，为什么又有 Application Framework, support Framework,etc.JFC,MFC,WFC 严格的来讲是属于 Toolkit 还是 Application Framework?BTW:infrastructure 是不是 software architecture 的一个别名.怎么理解软件复用的目标：面向领域，以体系结构为中心，过程驱动

qlw：SA, Framework, DP 的关系应该从系统分析设计方面看，Archi 是设计时首先要考虑的问题，C/S, B/S, Corba 都时 Archi 的问题，DP 相对比较具体，就是针对具体问题。Framework 的概念应该在两者之间，可能只存在于大型项目。xFC 都是类库，应该更接近于 Tool Kit，但是它们确实也定义了某些基本的 Framework,这些 Framework 并针对具体的某个应用，而是适用于本平台的所有应用。infrastructure 和 SA 是两个问题，也许在某些论文中会混用。

==========

Roy：Dear Sir, I would like to know that whether there is any standard method to estimate the progress of development. I can estimate my progress according to my experience, but what can I do for others. Please give some equations to me if there is.

qlw：Schedule is generally derived from current resource availability, effort estimate and historical data. I think if you mean some theoretical method, there are really much of them. In General, you need to setup process model using Gantt/PERT/CPM and using some MAX-FLOW... algothrim to generate a schedule map, they are commonly used in machine scheduling

Roy：I don't think developer's personalize character is good for development, however, it's a genius; and how can I avoid this situation in ordinary life of development, as you know the collaboration ship is also very important to complete the project.

qlw：Take advantage of each developer's personality is the ability of the PM, but there is a maturity problem. Generally, in the beginning, it is a chaotics and then rule-governed and adaptive in the end

Roy：Would you present some professional&effective methods for testing example designing? Must there is differences between academic and practice and, I don't think that the theory directs practice for always, well, this is your subject, any suggestions or contributions of you to introduce?

qlw：Testing strategy for different software and in different stage of development is much different. However the basic idea is same: you must cover as many states as possible, that is the fundamental theory of Computer Science: FSM.

==========

罗子豪：钱五哥，你好。我是一个项目经理，以前从事面向结构的软件开发过程；现在公司让我准备一个大型项目，从面向对象的方面进行需求、设计和开发。我对于此方面只是略知一点（也就是面向对象的编程），请钱五哥给一些建议，我该做什么准备，当然越快越好。（我对我当前从事项目的业务方面很熟悉）

qlw：[OOAD]目前比较可用的系统化方法就是 UML 了,也许从 ROSE 入手会更快.不过也不要期望过高,工具是你让它干什么,它才干什么的. 至于材料.你可以从 UMLChina 上面找。

==========

Roy：I was only a pupil, as I think I am a civilization-blind in the modern age. Dramatically, I am a worker in a software company, Hi-Tech so-called by the people who seems to know all of the world, need I admire them? Doctor, please reply me. Yes, I like the untruthful world, it gives opportunities to me, and same to you. At the end there is a question for you: which is the first step when a project starting, time schedule or resource plan(face to investor)? Sincerely.

qlw：To some extent schedule is derived from resource availability as I have replied you in previous questions. However things are different for different projects: fixed-resource vs dynamic-resource; end-date-driven-schedule vs flexible-schedule. So which goes first depends on your situration. Ganerally there must be something predefined rather than dynamic all the time. If Dynamic all the time, you will have to use some new method, such as Adaptive Software Development, the book I am reading

==========

jordan-gxj：钱五哥，我现在面一个难题,想请你帮忙指点： 我已是一个 30 岁左右的人了，我目前已参与或主持了一定数量的软件系统项目，各方面都有些涉猎，如 MRPII，GIS，DATAWAREHOUSE，BI 及电信网管及电信 MIS 系统，应该说知识面挺广，但涉足都不是很深。我现在的状况是现实情况不允许我深入项目技术细节（因为我是一个 MASTER，到那里都是当项目经理），再加上处于这样的年龄我也不想涉猎。技术细节及编程，另外现在好

象 PMP 很热门，我想做个“职业经理人”，但我的疑问是获取 PMP 就能脱离技术细节吗，能成为成功的 PM 吗？请你指正我的想法。期盼回音

qlw：我的年纪也许不比你小,不过我从来不认为新技术是"年轻人"的事情,现在我正在深入研究各种网络技术, 程序设计更如同写散文. Anyway, (不知道你说的 PMP 是不是 Project Management Process.) 我认为当好 SW PM 最重要的一点是理解程序设计的复杂性,真正理解项目要求和掌握人的心理. 对于技术细节的掌握到理解即可,不必事必躬亲. 我想对你来说不算难事. Good Luck

==========

haljf：钱五哥，有没有软件可靠性工程方面的资料，我很想深入研究一下软件可靠性的理论、方法和管理以及与软件工程过程的集成。也许你也听说过关于软件质量的这样一个说法：同样一个软件，我一个人几天搞定，IBM 要用几个月；我的软件几天就出问题，IBM 的就是不出问题；我的软件几个月后就没有人用，IBM 的几年以后还有人用，而且还收服务费。我现在很关心的就是我们软件可靠性，希望使我们软件的质量上一个档次。谢谢。

qlw：你说[可靠性]是指 Availability,Robust 还是 Quality? Anyway, 程序对于 Exception, Fault 和 Requirement 都应该给予满足. 相关材料可以参见 Fault Management 和 Software Testing, Formal Review, Code Inspection 和 Software Quality Assurance

==========

polokent：五哥,我是情话的博士生,多次有机会参与了一些大型 web 系统系统的设计和一些 consultant 一起工作,构造了过一些系统,我们的做法回头再看,有些和 rose 象,先进行商业流程分析,然后重构流程,建立 use case,然后就根据 use case 建立所有的人机界面也就是所有的网页,然后再由程序员做实现,当时做的还算成功,没有什么大问题发生,感觉就是 PM 管的事情太多,太累,最近我接了个项目,由于资金问题,人手更少,程序员没有经验,前端的商业模式分析人员有流程分析经验,又缺乏使用 rose 的经验和能力那我就想利用 rose 的能力,来补足我的资源不足的缺陷,其实我对 rup 也不太懂,我该怎么办,

qlw：看你的情况，我感觉 ROSE 未必能解决什么问题。成功的 Project 多数是稳定的、基于成功经验的。让一些新人，开发新型项目，项目本身资源不足，还要采用他们不熟悉的技术，那么项目的风险更大了。

==========

Xmblade：想问五哥：如何证明一个模型的合理性和有效性？

qlw：证明一个模型的合理性和有效性是一个很好的研究课题。你所说的这两个模型的性质最后可能会归结为模型的完备性和模型的效率。即便描述方式恰当，前者至少也应该是一个 NP 问题。而对于后者，恐怕不存在证明，它做需要的是和其他模型的比较。

巍巍：钱博士，你好．我有一些问题要请教. 1．基于组件的软件工程现在在国内的公司里应用得多吗？一般是在什么行业的开发应用的比较多？从卖组件库的公司买来组件来组装是不是真的适合软件开发呢？2．我有些感觉是决定行业应用项目成功的主要因素不是它到底是面向对象还是基于组件，而是真正理解行业的需求，一些很成功的行业应用项目至今还是用传统的结构化系统分析方法．我以前呆的一个公司开发部领头的也是个清华博士毕业，整天开会就是研究 J2EE，CORBA，MVC，EJB，JSP 什么的，可最后项目出来，很多客户都说，这不就是些网页嘛，你们的技术优势到底体现在哪里呢？你们说的那些 J2EE 对我们有什么好处呢？最后客户转向了一个使用微软技术的公司．我的问题是软件工程到底最适合哪些行业应用呢?3．以您的经验来看，IT 系统集成业务的项目管理与软件开发业务的项目管理有什么不同？前者最需要哪些知识和经验，有哪些困难和风险面临着系统集成项目经理？4．您对 Software Of Unknown Pedigree(SOUP)看法如何？

qlw： (1)从我个人的经验看,如果选择得当,组件对开发的促进还是相当大的．至于国内目前的情况,我没有做过调研,也就没有发言权了;(2) 我同意你关于项目成功主要"在于对行业需求的了解".问题是如果你重复开发同类产品,你是否想建立复用机制呢？早期的做法是函数库,然后是类库,现在部件占优．(3) 用户永远不会关心你的复用机制．你需要向客户展示更强的功能;更高的质量和更低的成本，这个才是技术．(4)软件工程适用于任何的软件开发,理由是无论什么软件的开发,都需要适当的技术和管理.(5)IT 系统集成业务的项目管理与软件开发业务的项目管理应该有很多区别，由于客户,需求变更情况,技术要求,质量要求,售后服务等方面的不同,需要采取不同的措施．Lucent 在集成项目(或者说是面向客户项目)方面有一些很好的实践.(6) 关于 SOUP，我并没有什么了解,用看的一句话回答你"Alan said a related issue was raised by Commercial Off the Shelf Systems (COTS). These were being increasingly used in systems. But they did not have a SIL rating, nor a known pedigree in many cases. They were SOUP: Software Of Unknown Pedigree. "

巍巍：关于重用性，大家都是做开发的，以前写了无数的代码，我想问的是究竟重用的有几个 percent?对于应用层的开发，能重用的很少。再说了，JAVA 本来就是 OO 的，要重用可以直接写 JAVA 代码，何需 EJB？对于 servlet 和 JSP，我可以用 JAVA 把逻辑做好，然后在 JSP 里面调用，当然 JSP 只负责与用户交互的处理。这样系统也很清晰，并没有搞混。这同以前我们做应用程序的开发是一样的，只不过一个是用 swing，一个是用 HTML 做客户端的交互。这同用 servlet 没什么区别，但开发调试都很方便。对于 EJB，其实也只是系统的一个框架，我想核心仍然该是我们的 JAVA 代码，也就是所谓的商业逻辑，EJB 只不过是提供一个分布式应用开发的环境。其实所谓的 OO，那也只是一种软件工程的理论，我觉得过程调用非常能解决问题。我看大家很多情况下也只是做一个 class 然后不断的使用过程调用。还是以前的老方法。OO 并不能解决软件开发的问题，就象软件业历史上所提出那些理论一样，也许过几十年回头来看，现在大家所推崇的 OO 又会被别的理论所代替。就象一句笑话说的那样：计

算机软件工程就象是在漆黑的屋子里寻找黑猫。OO 并没有让我们找到那只猫。 ----------摘自 javaunion.org，请问钱博士如何看待这种观点?

qlw：说的很有意思。原文并不虚言，复用（reuse）确实一直被大家提到，但是真正使用的确实不多。原因很多，我想可能的原因是：（1）技术走的太快了，人们更不上理解（2）硬件发展太快了，软件性能不必考虑（3）市场变化太快了，采用新技术和快速交付间相互矛盾。（4）其他主客观原因。我不很理解 EJB，应该是组件技术。对于大型应用（也许以后更多）开发应该应该很有帮助，但也许并不仅仅在于解决复用问题。等有空看仔细了在议:-)

==========

城墙：你好！我有多年的ＥＲＰ软件咨询、销售和实施经验，对客户的需求了解较为透澈，但目前的国内软件在实施时对客户的适应性非常差，所谓的ＢＰＲ工程和ＥＲＰ软件实施实质上是两张皮。于是我想自己对在总结多年经验的基础上自行设计开发，但我没有大型的软件工程设计经验，只有这样的想象性描述，不知我能否利用ＵＭＬ来建模，设计。我是９２年毕业的计算机专业。

qlw：用 UML 来分析设计是一个好主意,便于和你的开发人员交流. 也易于保证分析设计的连续性. 你可以参考 UML 的书籍来做. 另外,我很感兴趣你所说的两张皮问题,能否给一些解释? 你负责的产品有哪些呢?

==========

Flybird：我也是一个新手，搞软件也有一段时间，觉得必须要用有效的工具和正确的理论来指导软件开发工作。我现在用的是 playcase。你觉得这个工具如何。我这里有的大公司出的 case 工具就只有 visio2002。但不知道行不行。能不能给我介绍一下那里有 playcase 的详细资料。 对于 UML 的学习，你觉得是先结合工具来学习，还是先学习理论在进行实践好。我的工作很忙，我想一边结合工具的使用，一边学习 UML 知识。你觉得这样好吗，请多提建议。

qlw：我对 Playcase 不甚了解，似乎 Playcase 的作者高展对它很有信心。我想虽然 Rose 也不过是实现 UML 的一个工具，但是其功能可能比 Playcase 更强。Playcase 的主页是 www.mrcase.net.另外我认为先有一些理论知识会更好一些。

==========

xi rao：Sometimes, I confuse about some design procedure, could you comments my steps? 1. proposal: talking about why did this project 2. techology overview and checklist: write down the step how to use the new package or library. Add some usecase diagram 3. Write the specification with use case analysis 4. Analyze: refine the noun, verb build the conceptual。l model 5. Design: draw collaboration and class diagram with every method with some pattern design method. 6. Build a complete checklist for all procedure 7. Coding 8. Testing: I want to do "pressure testing", but don't know how. Problem

with the project: 1. The class diagram and collaboration diagram is too complicated and can't print on one page. I feel hard to read them. 2. About 3000 line code small project get too much diagram 3. The step2 could go wrong because of new technology, so waste a lot of design, re-do a lot of work. Could you gave me some suggestion or comment my steps? Thanks

qlw：(1)Pressure Test need tools which simulate many users, you may develop them by yourself or just find one on the internet(2)Whether the Diagrams is too much for you or not depends only on your objective, i.e. what do you want your project be.(3)You are right, things are changing quickly, you can try XP, although I think some practices in XP are not so good for some kind of project, such as the "Simple Design". And there is also a policy called "Do the simplest thing that could possibly work.(DTSTTCPW)"

==========

simple best：钱博士您好，我正在读在职硕士，对 SE 感兴趣，做这方面的论文可以吗？Design Pattern 可以当作论文写吗？

Qlw：Design Pattern 的研究当然可以当作论文来写,问题是你有没有新想法.博士论文需要一定的理论基础.不过如果你的想法非常有创意,而且针对具体应用,应该也是好论文.我浏览过一篇 USC 的博士论文,其中主要内容是通过对 Effort 的研究,考察 SW-CMM 的有效性,理论的东西不多,但是有很多实际数据。

==========

Csut：钱博，请教您个问题（但愿不会让您觉得可笑）我们在用 ROSE 设计一个项目，开发语言选定 VC6，不只怎么组织各小组开发模块的集成工作？为了便于集成，设计时应注意什么问题？请您指教或推荐些学习资料。谢谢！

qlw：好像这是你重发的问题,我上次没有答的原因是不清楚你的项目情况,这次还是不十分明白. 因此你问应该考虑什么问题. 我想从大面上看,首先考虑接口问题. 如果采用 OOAD 中,我认为类结构可能是极为重要的部分. 可以到这个 Cklist 上看看: http://www.construx.com/chk.htm

==========

sun_ping：钱五哥:您好！ 能否提供一个有关结构化分析方法（SA 、SD）的实际例子或相关站点。谢谢！

qlw：[结构化方法 SAD]可能需要看看 Idef 系列, 具体 site 我并不清楚, 你可以自己查找. 另外那个 PlayCase 也许可能用来当辅助工具. Software Architecture 也是一本很好的关于分析设计的书. 关于数据库设计, 看看 ER 方法就好, 工具有 ERWin 和 ERStudio 等等, 记得 umlchina.com 论坛上面有相应材料和工具。

==========

ken：我现在在一家系统集成公司做项目管理部的负责人，也是才进来的，感觉工作千头万绪的，什么进度监控、版本管理、知识管理等等，都是一片空白，我准备从项目评价开始做起，切实督促到每个项目组，将问题统揽在一起，你有何高见！

qlw：[项目管理]的问题相当大，很难一句话讲明白。推荐你看看 CMM 的资料，如果项目比较小，可以参见 XP 等资料。如果工具具备，不妨学学 RUP. 但无论如何，看看 CMM，尤其是 Level2+3，都是有很大好处的。具体的一些实践，不妨看看 MS 的一些项目管理书籍。此外我还推荐一本书: Adaptive Software Development, 也许很快会出版。

==========

kaitty：钱老师，您好。我是 UML 的初学者，有一些问题不明白。RUP 提供了六种视图，是不是在项目开发过程中每一种都要用啊？还有，我开发的服务器程序没有用户，怎么画 Use Case 视图啊？请多指教！

qlw：UML 设计中没有必要用到所有的视图。

==========

wuzheng：CMM 是卡内基梅隆大学的教授提出来的软件工程标准。我想知道的是 CMM 评估师的职责是什么？到底怎样才能成为一个评估师？没办法本来想问问公司里的 QA 但据他说中国现在还没有哪个个人或是公司有这样的评估资格。我将来很想往这个方向上发展。请问我该怎样做才能达到我的目的呢？请问美国是否有 CMM 的评估师资格考试？

qlw：CMM 的职责在 www.cmu.sei.edu 的主页上面有几个文档，专门介绍与 Audit 相关的内容，tr007.96 中介绍了对 LA 的要求。

==========

hobby：请问如何在公司中学习和实施 psp？

qlw：学习和实施 PSP 是两个截然不同的问题。北航关于 PSP 有过实验，记得曾见过他们的实验数据。实施 PSP 可能会比较困难，它要求每个人对自我的严格管理，不如试一试 XP.

==========

吴华胜：我是一名大四的学生，现在正在做毕业设计，老师要求我们用 java 作为平台开发一个网络数据库在线查询系统。而且要用 uml 作为系统分析阶段的工具。但是以前从来没有学过，不知道从哪里下手。我现在正在看关于面向对象系统分析的书籍，不知道我的学习方向是否正确？您能否指点一下，在我应该怎样分阶段地学习 uml 呢？

qlw：[关于用 UML 设计 Java 查询项目]你的方向应该不错，但是需要注意把焦点放在 UML 上面，OOAD 的方法还是比较多的

==========

wingrun：钱博，你好！我是一个软件工程的新手，现在才刚刚接触软件工程。我现在感到有些迷茫，因为对它不了解，所以不知道自己该看些什么，做些什么。能不能把软件工程的现状讲解一下？给我指明一个方向：怎么样来学习软件工程？谢谢！

qlw：【关于现状】，国外对软件工程的研究都是很实际的，所有你在开发中遇到的软件工程问题，都有人研究，管理方面：CMM，RUP, XP, Measurement; 技术方面：各种 OO 分析和设计。此外对逆向工程的研究似乎一直比较多。

==========

Xu Lei：钱老师： 你好！我想知道，你为什么叫钱五哥呢？

qlw：钱五哥是我网上的绰号而已，就象有些人可能用“主席”，“老大”之类的诨名一样，不必在意:-)

==========

Charles：錢小弟 ： 你連 Rational ROSE 都不懂 ，還敢在這裡丟人現眼 !!

qlw：Rational Rose 又不是什么高深的东西，我很早以前调研过这个东西，也许还是 4.x。个人认为 Rational Rose 最大的好处是帮助大家用一种共同语言描述一件事情，是描述工具（UML），本身不是什么理论性的东西。不是说学用 ROSE 就会分析、会设计，关键是对问题的理解和把握。难道不会用 MS WORD 就不会写文章了么？记住：完备性、形式化才是真正的好东西。能保证/证明你的程序没有错误的东西，才是最重要的。

==========

张小二：请教您关于软件工程理论的体系问题。我认为，1、软件工程是一个理论基础；2、软件工程过程（如 RUP）是软件产品生产的过程；3、建模工具（如 ROSE、ERWIN、PowerDesigner 等以 UML 为标准的 CASE 工具）是在软件工程过程中运用的实施工具（包括开发工具）；4、软件工程管理如 CMM、PSP、TSP、项目管理等是对软件工程过程的管理和改进。您看这个认识是否妥当？您觉得软件工程的整个理论体系应该如何认识？

qlw：【关于软件工程理论的体系问题】1、软件工程是一个理论基础【是实践经验的总结，上升到理论】；2、软件工程过程（如 RUP）是软件产品生产的过程；【yes】3、建模工具（如 ROSE、ERWIN、PowerDesigner 等以 UML 为标准的 CASE 工具）是在软件工程过程中运用的实施工具（包括开发工具）；【yes】4、软件工程管理如 CMM、PSP、TSP、项目管理等是对软件工程过程的管理和改进。【软件过程的范围相当大，简单地说，和人、物和活动都有关系。】。

bone_liu：五哥，本人认为设计模式这东西并不实际。设计模式带来的好处是柔性与易维护性，但它使一个简单的东西变得复杂。所以，本人认为只要文档写得的齐整，就不用去套什么设计模式了。

qlw：【设计模式】是很实际的东西，它将复杂多变地东西程序化或者部分形式化，使之可重复，而这是软件工程的目标之一。文档是一种形式，关键在于里面的内容

==========

lucy：钱博士：你好，问你两个问题 我正在做关于远程教育系统建模的论文，我们实际上已经完成了用 PHP 编写的远程教育网站，主要负责教学环节，如课程课件点播、答疑、作业、信息查询等等，而后台的数据库管理使用 POWERBUILDER 开发的，是 C/S 模式，主要用于教务管理，如人员管理、学籍管理、成绩管理等等。 现在做学位论文，要用 UML 建筑这个系统的模型，比如用例图，我的设想是有教师、学生、管理员三个角色，我们在 C/S 端设立了拥有不同数据操作权限的几级管理员，比如系统管理员、各系教学秘书、系主任、校教务处管理员等。在建立用例的时候，我设计的用例非常多，有大概 50-60 个，包括学生的选课、资料下载、作业、考试、查看学分、查看成绩、教学评价；教师方的课程管理、课件管理、作业管理、题库管理、学生成绩管理、学生评价；教师和学生共同的留言板、实时讨论、个人信息维护、信息查询、查看消息、问卷调查；管理员方数据库管理、人员管理、教学管理、教务管理和系统管理。这样画出的 use case 图很庞杂，而且也没有体现出面向对象的继承、泛化等联系，因为我的用例之间没有什么联系。由于开发的时候是以网页为模块独立开发的，现在要找出他们之间的联系有点困难。 我也曾经考虑过，先画成使用包，学生，教师，管理员,最后还是画了个总用例图，因为觉得分开画别人不容易看明白。倒不是我觉得用例太多，而是觉得我总结出的用例没有体现出 UML 的精髓。用例之间没有什么联系。 我画的类图基本上是以数据库设计时的 ER 图为蓝本，再加上一些接口类，比如注册表单、课件管理表单等等之类的，不知道是否合适。 我知道这样非常不对。我也是因为要写学位论文了才开始学 UML，刚开始没有花太多工夫设计。但现在要应付答辩，总不能太不像样子吧。能不能给一点意见呢？？？

qlw：【PHP 编写的远程教育网站】这种先编码后设计再分析的做法应该 Stop 了。Use Case 属于分析阶段，建议你假定没有目前并没有你的网站，也不知道要用 PHP,还是 JSP 还是 CGI,重新考虑需求。这样可能就不会被 web page 干扰了。

==========

盈盈：请教钱五哥，我是做界面设计的，我想请问你没开发人员做界面的时候都有什么好的界面规范？还有哪里能找到完整的界面规范呢？

qlw：【界面规范】有很多，go to www.rspa.com, use it as the entry point

==========

钱小弟：五哥你好:我是做软件过程改进的,起步不久,以前的工作主要是规范了软件开发的流程（包括软件开发过程、评审、变更控制等）。现在想建立一个集成化的 CASE 环境，将项目管理工具和其他支撑性工具集成进去，最终目的是实现项目管理的办公自动化，近期目的是实现项目计划管理、进度管理、版本控制、配置管理、文档管理。目标有了，但具体应该怎么做，我还比较迷糊，请五哥指教，我下一步应该这样做，有没有比较好的工具可以利用。

qlw：【集成化的 CASE 环境】是一个很好的想法，我想其中主要需要考虑的问题是处理好灵活性和严格性的关系。至于实现，需要评估各种平台，以便减少开发工作量和提高可用性。基于 web 或者 Groupware 都是很好的选择。

==========

Yanite：我是一个软件工程初学者。老板让我做一个软件的测试 只给了我一套软件（没有源码）我想知道如何在集成测试和系统测试中最大限度的找出 bug （我现在只找了几个 bug 而那个写程序的说不是软件的问题）能否帮帮我？这里先谢啦！

qlw：【集成测试和系统测试】确实没有必要看到源码，但是必须有相应的文档，比如需求文档，概要设计文档等等，依据系统的功能等要求来设计测试用例。如果你测出的 bug 是环境配置等问题，可能说明需求文档中关于系统环境的描述不完整。

==========

老六：老五，我喜欢用例子学东西，来的快！听说好的软件工程文档做完后，找些初中生来写程序就行了。你有这样的文档例子吗？

qlw：我还真没有听说过敢用【初中生】来按文档写程序的项目经理。从某种意义上看，程序构造是一门学问，可以同程序设计相较。目前很多项目建立可能会省略程序详细设计，而直接有程序构造人员实现。

==========

Ywli：您对在中小型 IT 企业中实现过程改进有什么建议与看法？

qlw：【中小型 IT 企业中实现过程改进】是一个很实际的问题。目前有一下一些可行的方法：(1)SEI PSP/TSP(3)RUP(2)XP/FDD(3)Adaptive Software Development/Agile. 对于（1）来说，凡是实践过 CMM 的人可能都会觉得这是一个负担很大的事情，换言之，杀鸡未必用牛刀，（2）的问题是赖于工具，成本高。（3）太简单，很多地方没有涉及到。（4）是最近比较热的方法，我本人觉得确实比较适合中小企业，它从人的协作入手来解决管理问题，详细信息可以等我和朋友们合作翻译的一本书《自适应软件开发》出版。

==========

boyu：你好，请教一个问题：我是测试人员，从编码出身，但编码时间不长。我接触软件工程不包括学校内的时间有 3 年了，也一直在关注着国内软件工程的成长，现公司准备进行实施软件开发规范，虽然了解一些规范与不规范的厉害关系，但没有深刻的理解，无法去说服程序员按规范进行，请教一下，我现在该如何去做，是重新回去开发，去体会，还是在继续学习，漫漫会理解的？

qlw：如果【无法去说服程序员按规范进行】，那么可以花钱请专家给培训，这样可以促进规范的实施，人的因素可能是最大的问题了。

==========

陈渊：软件开发费用应该怎么算?

qlw：【软件开发费用】并不那么容易计算，单纯从技术方面看，可以采用如下策略：程序规模 -> 工作量（人•日）—> 软件成本，其中要考虑生产率问题。此外还有很多其他成本，包括设备等等。

==========

xiajia：我在一家软件公司研发部做负责人，公司很新，研发部都是很年轻的人，我们合作开发了一个管理软件。我们用的平台是 Microsoft.Net,我有个搭档，他是我们的技术核心，我主要负责项目管理。在此期间，我和我的搭档一直试图将 UML 的模型概念贯穿到整个开发过程中，我们做了这样一些事：需求分割成众多用例包，再配以泳道图表示业务流程，活动图表示系统流程，配以单元测试用例。然后用 powerdesigner 作为数据库的建模工具，用状态图表示存储过程，用 clearQuest 作为测试的管理工具。 项目从 4 月 5 日（不包括需求）开始。现在项目开发已经接近尾声。我的感觉是： 1 在尚未成熟的团队中有一个技术核心对风险的降低起了很大作用，但是过分依赖灵魂人物存在很大隐患。需要标准化的东西。 2 在团队中推广 UML 尽管会遇到很大阻力（比如人员素质、时间压力、成本压力等），但一个团队要想不断降低开发成本，提高质量，这是很必要的选择。 3 我们的需求分析员直接来自项目所适用的领域，这是个让我又觉得成功又觉得冒险的地方，成功在于产品会很贴近用户，冒险在于花了比较长的时间让他转换思维，培训使用 visio 等工具。 一直忙碌，现在我停下来总结心得，我觉得我非常希望组织一支高质量的团队，但我的搭档认为人员素质的限制将是我最大的瓶颈，这个问题上我没有很好的答案，因为我觉得这是一个很难定性的问题。我希望通过培训和不断的实践能弥补这些，我希望得到您的帮助。等候您的回音，谢谢。

qlw：一个新项目和成熟项目的管理完全不同，风险所在也不尽相同。我同意你观点中的（1）和（2），这些都是将新项目转变伟成熟项目的关键（主要是技术因素）。（3）则不相同，这件事情的解决方法可能需要从软件过程方面入手，建立一套合适有效的需求分析过程以及配套的评审过程等就可能使这个问题得以缓解。人的素质确实是最大的问题，这个问题相当复杂，比如说某人可能技术很好，但是心不踏实等等。在人员的领悟能力可接受时，

一种可行的方法使让开发人员感觉项目有前途，始终有新技术可学，这样他们就不会动摇。

Xiajia：多谢五哥回答我的问题，你的回答给了我很多提示，我想问：在这样的情况下想建一支好的团队，我想从 2 个方面入手，一是让需求过程尽可能地贴近用户，也就是说，检验需求地标准是客户听懂了，看懂了，在项目组内部把它转换成 uml 语言，使沟通一致。二是让开发人员在理解模型的基础上积极开展组件/类库的试验和积累，激发他们的创造力。你觉得可行吗？ 另外我还有个难题：如何让一个团队中的不同角色接收 uml 培训？培训的质量怎么控制？ 关键技术采用什么语言或在什么平台下实现是不是一开始就应该稳定？

qlw：你说【2 个方面】，我觉得没有什么大问题，不过还是应该以客户为中心，应该说开发人员对客户的需求理解了，听懂了，而不是反过来。团队中的成员应该按照角色进行 UML 培训，多数情况下，SE 和 Designer 应该非常熟悉 UML 的各个细节，Developer 应该熟悉 OO。培训的质量怎么控制的控制确实是个难题，也许可以通过收集反馈来进行评估。关键技术采用什么语言或在什么平台下实现应该在项目早期确定，以便降低风险。

==========

sword_hero：我在概要设计是这样做的，概据需求分析先想到一个框架，其中某些问题就直接想出来就行了，对于某些问题的细节我就通过编程来试试看实现，先编个大致的框架，在编程中我就可以大致的知道很多需要注意的地方以及如何分配功能，但由于工期比较紧，往往在我编程中想概要设计应该如何设计的阶段 boss 就要来催了，如果我说概要设计还没好，他就会说，我都看你在编程了，怎么回事？那我只有把编的代码略加修改就投入使用了，而概要设计书一般写的比较晚，我这样做行不行？有什么后果，请您指点一二，谢谢

qlw：【概要设计】、原型开发和关键技术实验之间地关系取决于你采用地开发模型。一般来说原型有两种，抛弃型和改进型。如果因为某些原因将一个抛弃型原型投入使用，一般来说都会造成不同程度地可扩展性问题，这对系统地进一步发展极为不利。

==========

Ares Chen：我看过很多描述如何写好“需求分析”的书籍、文章，每一篇都在说需要注意“准确、可行性...”等，对于有具体客户的独立项目，我觉得这些都好理解，也可以去努力实现，但对于企业产品的研发，我们没有具体的客户，也没有具体的需求，我们该怎么来确定我们的需求分析的准确和可行呢？我希望听听您的意见。

qlw：我想你说的【企业产品】可能是指 Commercial software，这种产品的需求源自市场调查，可以参见 MS 的开发方法，如微软的秘密一书。。

==========

学习者：我想请教一个关于在软件开发过程中，各个阶段的大体费用或者各个阶段占用总体费用的比例（不用考虑硬件设备，只要相关的需求分析，软件设计和开发）。盼复，谢谢。

qlw：关于【各个阶段的大体费用】与不同软件类型、开发团队构成、开发模型、以及开发工具的关系十分密切。如果仅考虑人员成本，你可以根据各个阶段所用时间和人员个数来估算。可以参见 FPA，或者 COCOMO 方法。

==========

Goober：请教钱博士，如何对软件的变更进行分类、控制、管理，另外对于 SCCB 来说如何正确组织其成员、进行合理分工、如何确定那种类别的成员处理相应级别的变更？

qlw：【软件的变更】的具体实施和你们公司的具体情况有关，我以前参见的 SW－CMM 项目中，几个项目组对 SCCB 的实施都不尽相同

==========

study：做程序设计时用程序流程图，但 windows 下使用定时器 timer，在程序流程图中使用什么表示，循环吗?但定时器控制下有许多并行的任务，怎么表示?

qlw：【流程图】不适合描述 timer 这样的并发模块，同时也包括多线程。试一试 UML 里面的图

==========

zjuxxl：钱博士：我选择了“基于 UML 自动测试系统建模方法的研究"作为博士课题，最初想法是通过对自动测试系统的共性与特征分析，提出一个通用的、可复用的对象框架，使用扩展的 UML 对框架进行描述，并提出一种形式语言来描述扩展的 UM，最后基于该形式化语义研制 CASE，随着课题的深入，感觉题目太大，钱博士能否提一些宝贵意见。

qlw：【基于 UML 自动测试系统建模方法的研究】题目比较散，有几重含义（1）使用 UML 方法（注意，不是基于）（2）自动测试系统（3）形式化方法。也许我没有理解你的用意，我认为有意义的工作不是在于描述这种应用/测试系统的行为，而是自动测试本身。自动测试的形式化远比自动测试系统的形式化有意义的多，而且也是两个概念。

==========

Sabrina：请问，如果选择了项目用螺旋模型的方法进行软件开发，那么应该按照怎样的规范（或流程）做？采用这种方法后，是否要重新定义软件生命周期？具体如何操作？

qlw:【Spiral Model】的关键在于迭代周期的里程碑确定，目前的趋势是短周期，并且加入周期评审。此外 Sprial Model 就是一种生命周期模型。

==========

little gp：钱博，我在具体做项目时遇到了如下的选择：1）将项目开发队伍开赴客户那里 2）将开发队伍留在公司，

只要求项目经理或业务骨干数人留下。第一种方法，能够很及时的满足客户随时提出的要求，但成本太高；第二种方法，成本低但对客户的要求反映慢一些。目前我们采用前者。 您觉得那种方式更好些，或者您觉得还有那些应该考虑的方面？

qlw：【人员安排】问题的分析挺好的，但是需要和你的具体要求像结合，此外，还可以考虑请客户代表来你们公司常驻。

==========

叶子：用 uml 分析，什么时期是用来确定开发费用的，是在开始分析之前，还是用 USE CASE 做了需求分析之后决定？

qlw：【开发费用】的确定必须看你用什么样的测量方法，如果采用 FPA,那么早期，既在需求确定以后就可以，如果采用 SLOC，则很难准确的在早期估算成本。

==========

Vindey：我很喜欢软件过程，但是学习了很久，感觉软件过程实施和改进都是很不容易的事，尤其是在中小企业里。我不知道您对于 XP 的具体项目实施有没有经验，可否指点一二。

qlw：其实不管是 CMM，还是【XP 的实施】都会对企业有不小的影响。可能只要有以下几个应该注意的问题：（1）管理层的支持 （2）Commitment Among All team leaders and members (3) Plan and schedule is reasonable (4) effective evaluation (5) team motivation and self motivation, Hope help.

==========

吕锴：书写用例的时候，有的 use case 性质不同，但过程相似，是否应合并。比如：买入股票和卖出股票，过程几乎相同。

qlw：我想不应该合并。

==========

shi zhongjie：我们刚开软件工程的课。关于数据耦合、控制耦合、公共环境耦合、内容耦合，以及功能内聚、顺序内聚、通信内聚、过程内聚、时间内聚、逻辑内聚、偶然内聚，我想知道关于这些的一些具体的例子，麻烦啦

qlw：关于和【耦合】和【内聚】，它们描述的都是系统内部模块间的依赖和独立关系，属于一种定性的描述。我想需要了解它们的具体定义。请给出具体定义和英文描述。单从字面上看，数据、控制、环境耦合分别描述了模块间在数据信息、控制信息和共同环境依赖性方面的依赖关系。

shi zhongjie：我们发的教材是：《软件工程导论》张海藩 著（第三版）清华大学出版社。原文如下：数据耦合—

“如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。”；控制耦合——“如果传递的信息中有控制信息（尽管有时这种控制信息以数据的形式出现），则这种耦合称为控制耦合。”；公共环境耦合——“当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。公共环境可以是全程变量、共享的通信区……”；内容耦合——“如果出现下了情况之一，两个模块间就发生了内容耦合：- 一个模块访问另一个模块的内部数据； - 一个模块不通过正常入口而转到另一个模块的内部；- 两个模块有一部分程序代码重叠（只可能出现在汇编程序中）；- 一个模块有多个入口（这意味着一个模块有几种功能）。”好多阿，先就这些吧。的确说的不清楚，尤其是“控制耦合”和“内容耦合”，水平有限，希望钱博士能帮帮我。谢谢啦！

qlw：根据定义，【内容耦合】最好的例子就是 longjmp,这种跳转会进入其他处理模块的内部，另外 robot 程序也是内容耦合的例子。【控制耦合】的例子实在太多了，传递改变模块内部流程的参数都引起控制耦合。

==========

张凤川：关于代码生成的问题： ？、应用 Rose Realtime 能生成系统的所有代码，而不仅仅只是框架？ ？、它能使开发复杂实时系统的工作量大大简化？ ？、如果是这样的话，为什么国内做 Realtime 相关（如通讯）的公司中没有大规模推广和应用呢？ ？、UML2.0 中引入了很多新的特征，而且新闻中称：“其它公司，如 Telelogic 也在致力于利用 UML2.0 从图形化的用户模型中自动生成代码。”，是否意味着基于 UML2.0 的 CASE 工具也将象 Rose Realtime 一样，能生成系统的所有代码？ ？、如果 CASE 工具做到了这一点，那么软件工程中的分工模式是否会演变为类似建筑业的分工模式：建筑设计师、工程管理者、建筑工人？ 问题也许幼稚，请见谅！

qlw：在软件工程历史上，【代码生成】问题已经讨论了很久很久:-) 由于各种原因，自动生成代码的质量并不能满足要求，特别是在对性能要求很高的行业中；自动代码生成的灵活性也不高；一般代码生成工具都会产生框架代码；各种代码复用机制和工具也对代码自动生成构成挑战。

==========

Acidrain：我想在学校里如果实际工程经验很少，看教材上那些高深的理论实在是味同嚼蜡，至少我软件读了 7 年出来工作了 2 年才发现自己所知实在甚少，回想当年的课本才发现原来里面还是有不少可以用到的好东西，当年没有根基无法理解而已，现在来看真是感同身受了， 钱兄赞同否？ 你对目前学校的软件工程教育（我们当年好象本科都没开过这种课，硕士阶段学的也就是泛泛而谈）有何高见？ 没有实际工程经验的学生应该如何理解软件工程？ 学校里好像还是学术论文派大大压倒工程派，2 年的实际工作我现在算是知道软件工程的重要性了。

qlw：关于软件工程教育，我个人觉得应该在保证理论基础的前提下，增强实践性。很多软件工程方法和。计算机技术，如早期的 IDEF, Jackson，FPA，现在的 UML，FFP 以及比较稳定的数据结构、编译原理和操作系统，都需

要按照理论学习的方法来讲授，但是必须将这些技术方法和最新的实践结合在一起，如编译原理，特别是优化在开发工具的应用。FPA 在 Web 应用中的使用等等。

==========

yeahy：钱五哥，你好。我和 9 个同事在维护着公司自己开发的 MIS 系统，系统的业务逻辑很复杂，我们每天都忙着为用户添加新功能，感觉挺被动的，软件的架构有点老，想重写又觉得工程浩大。想请教一下钱五哥，有没有什么办法改变一下？我觉得软件工程技术能够帮助我们，但是不知道怎么着手。期待您的回复。:)

qlw：维护【系统的业务逻辑】复杂的系统确实是一个很困难的事情。一般来说，如果原始程序的模块化程度比较高、原始程序的数据定义可扩展性好、对需求变更（MR）的管理比较完善、测试（回归）比较完善。维护起来还是会比较顺利的。

==========

大志：我有个朋友问过多次这个问题，他 30 多岁，才想开始学软件，每天的学习时间也就 2 个小时，有没有可能在 短时间里比如 2 至 3 年做出成绩，就是改行吃这碗饭， 但他又担心随着 IT 行业的变化，到头人财两空 有没有高人给指点一下，感谢……

qlw：软件开发不是一个随便玩玩就可以的事情，一些非专业人员的程序更本无法投入使用。能够发布使用的产品即需要在软件工程的管理下进行，同时也很需要开发人员的编程水平，完成同样的功能，高水平程序员和业余程序员写出来的程序，在性能、可靠性、可维护性等方面都相距很远。最近看有报道说高水平技工在国内看好，我想在软件开发领域也是一样的。国外，很多程序员会一辈子搞程序开发，因为他们把这当作一种事业和一种追求。

UMLChina

非程序员

软件工程杂志，已有数万用户

UMLChina讨论组

已有数万名成员

UMLChina.com

1999年11月成立，，专注UML技术的应用

专家交流

提供与世界级专家交流的窗口

Frederick Brooks 传

20 世纪最后一年也就是 1999 年的图灵奖，授予了年已 69 岁的资深计算机科学家布鲁克斯(Frederick Phillips Brooks, Jr.)。布鲁克斯这个名字在中国知之者不多，但在美国却是大名鼎鼎。因为他在 60 年代初只有 29 岁时就主持与领导了被称为人类从原子能时代进入信息时代标志的 IBM/360 系列计算机的开发工作，取得辉煌成功，从而名噪一时。以后他作为硬件和软件的双重专家和出色的教育家始终活跃在计算机舞台上，在计算机技术的诸多领域中都做出了巨大的贡献。从某种意义上说，对于布鲁克斯而言，图灵奖是一个“迟到的荣誉”。

布鲁克斯 1931 年 4 月 19 日生于北卡罗来纳州的杜哈姆。1953 年从杜克大学毕业，取得学士学位以后，进入哈佛大学深造，1955 年取得硕士学位，1956 年取得博士学位。值得指出的是，布鲁克斯取得的是计算机科学的博士学位，是一位“正宗”的计算机博士，是世界上第一批获得计算机科学博士学位的少数学者之一。他的博士论文课题工作是在哈佛著名的计算实验室(Computation Laboratory)进行的。大家知道，40 年代被称为 MARKI 的世界上第一台程序控制的机电式计算机 ASCC(Automatic Sequence Controlled Calculator)就是由艾肯(Howard Hathaway Aiken,1900～1973)在这里设计，并获得 IBM 的支持而开发成功的。请大家注意，叫 MARK 的计算机有两种。除哈佛艾肯设计的 ASCC 被叫成 MARK 外，英国曼彻斯特大学由威廉斯管的发明人 F.C.Williams 和 T.M.Kilburn 等人研制的 MADM 计算机(1848 年)也被叫成 MARKI，这是一台用威廉斯管作存储器并可存储程序的计算机，有时也叫做“婴儿”机(Baby)。通常提到的 MARKI 指哈佛的那一台。布鲁克斯最终完成的博士论文题目为“自动数据处理系统的分析设计”(“The Analytic Design of Automatic Data Processing System”)。从博士论文开始，布鲁克斯的一生就与计算机结下了不解之缘。

在哈佛取得博士学位以后，布鲁克斯进入 IBM 公司设立在纽约波凯普茜的实验室当工程师。这个实验室从 50 年代到 80 年代一直是 IBM 开发计算机的中心。布鲁克斯在这里参加了 Harvest 和 Stretch 计算机的开发，任体系结构设计师。这两个型号的计算机都引入了一些新技术，在 50 年代后期至 60 年代初期有很大影响，尤其是 Stretch 计算机，首创先行控制方式，最多可重叠执行 6 条连续的指令，后来被发展成流水线方式，因而被认为是世界上第一台流水线计算机。布鲁克斯在其中的创造性贡献是解决了程序中断系统的设计，在数据格式中出现不均匀的字符分布时如何设计其二进制代码等问题，并从而在 1957 年取得了他的第一个美国专利“程序中断系统”(Program Interrupt System,专利号 3048332，与 D.W. Sweenly 共有)，发表了他最初的两篇学术论文。

1959 年，布鲁克斯曾被调至 IBM 在约克郡高地的研究中心工作，但第二年又重新被调回波凯普茜的实验室，因为当时 IBM 内部在计算机的研发方向上产生了重大的分歧。1960 年时，IBM 的计算机生产线上的产品是 8000 系列，但遭到一些人的反对，其领头人是伊万斯。伊万斯虽然只是衣阿华州立大学电气工程系的一个本科毕业生，但 1951 年就加盟 IBM，曾参与或主持过 IBM701、7070、1410、7070 等多种型号计算机的开发，已经积累相当丰富的知识和经验。他经过认真分析，认为主要继承 IBM 原有技术的 8000 计算机，即使研制成功并上市，过不了几年，即到 1964 年就会丧失生命力，缺乏市场竞争能力。因此他主张 8000 机下马，采用新的技术开发新的计算机，尤其是要开发新的操作系统。伊万斯的意见使 IBM 分裂成为两派，一派支持，一派反对，而反对派的领头人正是布鲁克斯！两派的争论和对立非常尖锐，又势均力敌，因为伊万斯的学历没有布鲁克斯高，但资历却比他老，双方的支持者人数也差不多。以小沃森(Thomas John Watson, Jr.)为首的 IBM 决策层于 1961 年 5 月担着极大的危险最后采纳了伊万斯的意见，是年秋宣布成立一个名为 SPREAD(系统程序设计、研究、工程和开发，Systems Programming, Research, Engineeringand Development)的委员会作为“taskforce”(类似于我国过去经常采用的所谓“攻关领导小组”)，由 13 人组成，主席为汉斯特拉(John w. haanstra)，副主席为伊万斯，布鲁克斯是成员之一。作为争论中胜方的伊万斯冷静地分析了形势以后，做出了一个令人大感意外的决定，他亲自找到布鲁克斯，请布鲁克斯主持日后被称为 IBM/360 的这个新项目。伊万斯这一举动主要基于以下两点考虑，一是如果由他自己来主持 360，那么原来反对他意见的另一派人很难团结在他的周围；二是涉及这样重大改革与创新的项目，应该让年轻人来挑头。他自己虽然当时也只有 34 岁，但布鲁克斯比他小 5 岁，更加年轻。难能可贵的是，布鲁克斯作为争论的负方，慨然接受了伊万斯的邀请，同意负责这个他曾经反对过的项目！这个故事很像我国京剧舞台上的“将相和”(虽然伊万斯并未“负荆请罪”)。伊万斯和布鲁克斯双方在这件事上所表现出来的明智、大度和勇气都十分令人钦佩和赞叹。其结果和效果就是整个 IBM 公司的职工果然团结起来，实现了痛苦而艰难，然而却是历史性的转变和飞跃。IBM/360 的开发总投资 5 亿美元，达到美国研究原子弹的曼哈顿计划投资 20 亿美元的 1/4。在研制期间，布鲁克斯率领着 2000 名程序员夜以继日地工作，单单 360 操作系统的开发就用了 5000 个人年。因此，当 1964 年 4 月 7 日，在 IBM 公司纪念其成立 50 周年的庆祝大会上宣布 360 系列计算机的时候，小沃森完全有理由声称“这是公司历史上宣布的最重要的产品”。确实，IBM/360 以其通用化、系列化和标准化的特点，对全世界计算机产业的发展产生了如此深远的影响，以致被认为是划时代的杰作。而 360 的推出，也使 IBM 在短短两年时间内，即到 1966 年，其资本就增加到 45 亿美元，职工总数净增 6 万，达到 19 万，成为名副其实的“蓝色巨人”。到 60 年代末，360 系列机的市场占有率达到 15%，到 70 年代中期，超过了 50%。各计算机生产厂商纷纷以 360 为榜样，推出各自的系列机，有的则直接采用 360 的操作系统，比如著名的 Amdahl 公司的所谓“插接兼容式”计算机(Plug Compatible Computer)就是这样。为此，伊万斯和布鲁克斯两人常常被并称为“IBM/360 之父”。

当然，IBM/360 到今天早已是“昨日黄花”了。IBM 公司在 70 年代就推出了 370 系列替代 360，以继续保持

其技术上的优势。我们之所以用了一定篇幅介绍 360 的故事，是因为其中不乏让我们的企业家、科学家和工程技术人员深思的一些问题。IBM/360 的特点我们只简要介绍如下：它是集成电路的计算机，体系结构既适于事务处理，又便于科学计算；系列中各机型(规模由小到大，功能从弱到强，包括 20、30、40、50、65、75 等 6 个型号，后来扩充了 25、85、91、195 等型号)具有兼容性；有标准的输入输出接口和通用的输入输出设备，它们与中央处理器相对独立；软件既有兼容性又有可扩充性，从而可最大限度地保护用户的软件投资。这些特征大多都成为以后计算机设计与开发所遵循的基本原则。

360 成功以后，布鲁克斯离开 IBM 回到其故乡，为北卡大学(UNC)创建了计算机科学系，担任该系系主任长达 20 年(1964～1984 年)。卸任以后仍在该系任教至今，因此他培养的学生很多，可谓“桃李满天下”。除了教学以外，他还致力于发展美国的计算机技术和计算机在国防等方面的应用，有许多社会兼职。1966～1970 年，他是 ACM 全国委员会的委员；1973～1975 年出任 ACM 体系结构委员会(所谓 SIGCA)的主席；1977～1980 年布鲁克斯在美国国家研究院计算机科学技术部任职；1983～1984 年他是美国国防科学委员会人工智能攻关领导小组的成员，1986～1987 年是上述委员会另一个攻关领导小组“计算机模拟和训练”的成员；1985～1987 年他担任军用软件攻关小组组长。他的研究领域除了计算机体系结构、机器语言设计、软件工程、大型项目管理以外，还包括动态体系结构的可视化(如“走查”Walkthrough)、人机接口、交互计算机图形学等等，十分广泛。例如关于虚拟现实，涉及布鲁克斯曾参与领导攻关的计算机模拟和训练，他十分重视，1992 年 3 月由美国国家自然科学基金会 NSF 主持的虚拟现实研讨会，就是由布鲁克斯等人倡议并在北卡大学召开的，这次会议对 VR 进行了定义并就其研究方向提出了详细建议，奠定了 VR 作为独立研究方向的地位。布鲁克斯在筹备及组织此次会议上做出了重要贡献。1987 年布鲁克斯当选为美国工程院院士，他同时也是英国皇家学会和荷兰皇家科学与艺术院的外籍院士。

布鲁克斯的著作不多，但影响都很大。1963 年他和依费逊(Kenneth Iverson, APL 发明人，1979 年图灵奖获得者)合著了《自动数据处理》(Automatic Data Processing，Wiley)一书，这是该领域中最早的专著之一。1969 年此书再版时有 2 个版本，其中一个是专门论述在 IBM/360 上的数据处理(书名为 Automatic Data Processing,System/360Edition)。1975 年，他把他历年来所写的有关软件工程和项目管理方面的文章汇集成书，书名为《人月神话》(The Mythical Man-Month: Essay on Software Engineering，Addison Wesley)。由于本书是他领导 IBM/360 软件开发经验的结晶，内容丰富而生动，成为软件工程方面的经典之作，出版 20 年之后,1995 年又再版了一次。最近的一本专著是他与荷兰特文德理工大学(Twente Technical University,位于荷兰与德国接壤处的恩斯赫法)的勃芬夫教授(G. A. Blaauw)合著的《计算机体系结构：概念与发展》(Computer Architecture: Concept and Evolution，Addison Wesley, 1997)。勃芬夫是布鲁克斯在哈佛时的同学，后来又在 IBM 共事多年，曾一起开发过 3 个型号的计算机，这本书实际上是对计算机体系结构半个多世纪来发展变化的一个全面的回顾和总结。作者在书

中风趣地把整个计算机家族叫做“计算机动物园”(Computer Zoo)，对其中的主要成员逐一作了剖析。除了上述学术性著作外，1995 年，他与苏泽兰特(I.E. Sutherland，“计算机图形学之父”，1988 年图灵奖获得者)等还合编了一本书，书名是《Evolving the High Performance Computing and Communications Initiative to Support the National Information Infrastructure》，由 National Academy Pr.出版，论述了有关高性能计算机计划及信息基础设施(也就是所谓“信息高速公路”)建设的一系列问题。

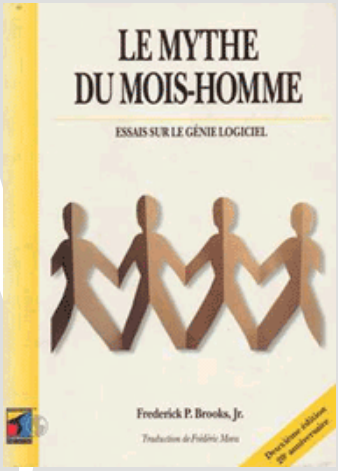
在授予图灵奖之前，ACM 在 1987 年曾授予布鲁克斯“杰出服务奖”(Distinguished Service Award)，1995 年曾授予他以纽维尔(A. Newell, 1975 年图灵奖获得者，1992 年去世)命名的 Newell 奖。加上这次的图灵奖，布鲁克斯成为继克努特(D.E. Knuth, 1974 年图灵奖获得者)之后的第二位同时拥有 ACM 三个奖项的计算机科学家。IEEE 也先后向布鲁克斯颁发三个奖项，即 McDowell 奖(1970 年)，计算机先驱奖(1982 年)、冯·诺伊曼奖(1993 年)。AFIPS 在 1989 年授予布鲁克斯 Harry Goode 奖。数据处理管理协会 DPMA1970 年授予他“计算机科学”奖，并命名他为该年度的风云人物。1985 年布鲁克斯因在开发 IBM/360 上的杰出贡献而荣获全国技术奖章(National Medal of Technology)，同时获此殊荣的还有伊万斯和 IBM 的另一位功臣布洛克(Erich Bloch)。物理学界的富兰克林学会(Franklin Institute)也曾授予布鲁克斯 Bower 奖。

购买

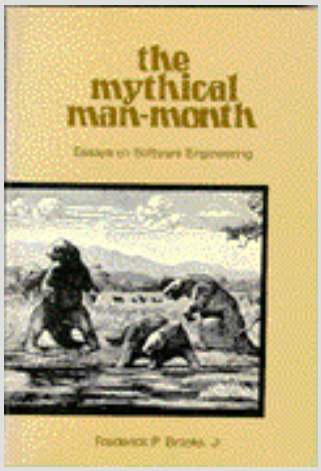
原书名	The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)
原出版社	Addison Wesley
作者	（美）Frederick P. Brooks, Jr.
译者	UMLChina 翻译组汪颖
书号	7-302-05932-2
页码	370
开本	32 开
定价	￥29.80
折扣价	￥23.84
出版日期	2002-11-1

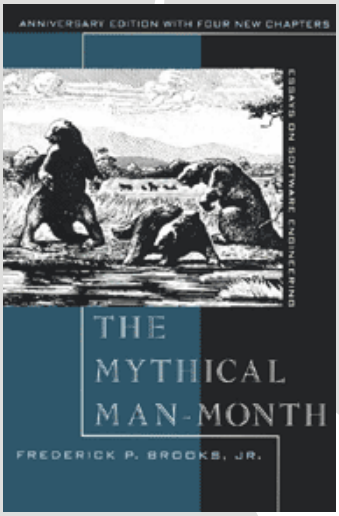
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)



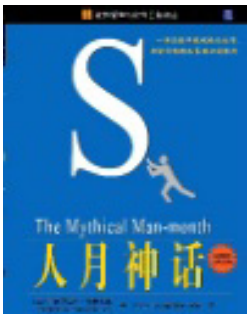


(第二版)

(第二版日文译本)

《人月神话》各章精选

第 1 章 焦油坑



史前史中，没有别的场景比巨兽在焦油坑中垂死挣扎的场面更令人震撼。上帝见证着恐龙、猛犸象、剑齿虎在焦油中挣扎。它们挣扎得越是猛烈，焦油纠缠得越紧，没有任何猛兽足够强壮或具有足够的技巧，能够挣脱束缚，它们最后都沉到了坑底。

过去几十年的大型系统开发就犹如这样一个焦油坑，很多大型和强壮的动物在其中剧烈地挣扎。他们中大多数开发出了可运行的系统——不过，其中只有非常少数的项目满足了目标、时间进度和预算的要求。各种团队，大型的和小型的，庞杂的和精干的，一个接一个淹没在了焦油坑中。表面上看起来好像没有任何一个单独的问题会导致困难，每个都能被解决，但是当它们相互纠缠和累积在一起的时候，团队的行动就会变得越来越慢。对问题的麻烦程度，每个人似乎都会感到惊讶，并且很难看清问题的本质。不过，如果我们想解决问题，就必须试图先去理解它。

第 2 章 人月神话

Brooks 法则：

向进度落后的项目中增加人手，只会使进度更加落后。

第 3 章 外科手术队伍

在计算机领域的会议中，常常听到年轻的软件经理声称他们喜欢由头等人才组成的小型、精干的队伍，而不是那些几百人的大型团队，这里的“人”当然暗指平庸的程序员。其实我们也经常有相同的看法。

但这种幼稚的观点回避了一个很困难的问题——如何在有意义的时间进度内创建*大型*的系统？那么就让我们现在来仔细讨论一下这个问题的每一个方面。

第 4 章 贵族专制、民主政治和系统设计

法国城市兰斯（Reims）在建筑风格上的一致性和上面所说的大教堂形成了鲜明的对比。设计的一致性和那些

独到之处一样，同样让人们赞叹和喜悦。如同旅游指南所述，风格的一致和完整性来自 8 代拥有自我约束和牺牲精神的建筑师们，他们每一个人牺牲了自己的一些创意，以获得纯粹的设计。同样，这不仅显示了上帝的荣耀，同时也体现了他拯救那些沉醉在自我骄傲中的人们的力量。

第 5 章 画蛇添足

在开发第一个系统时，结构师倾向于精炼和简洁。他知道自己对正在进行的任务不够了解，所以他会谨慎仔细地工作。

在设计第一个项目时，他会面对不断产生的装饰和润色功能。这些功能都被搁置在一边，作为“下一个”项目的内容。第一个项目迟早会结束，而此时的结构师，对这类系统充满了十足的信心，熟练掌握了相应的知识，并且时刻准备开发第二个系统。

第二个系统是设计师们所设计的最危险的系统。

第 6 章 贯彻执行

假设一个项目经理已经拥有行事规范的结构师和许多编程实现人员，那么他如何确保每个人听从、理解并实现结构师的决策？对于一个由 1000 人开发的系统，一个 10 个结构师的小组如何保持系统概念上的完整性？在 System/360 硬件设计工作中，我们摸索出来一套实现上述目标的方法，它们对于软件项目同样适用。

第 7 章 为什么巴比伦塔会失败？

那么，既然他们具备了所有的这些条件，为什么项目还会失败呢？他们还缺乏些什么？两个方面——*交流*，以及交流的结果——*组织*。他们无法相互交谈，从而无法合作。当合作无法进行时，工作陷入了停顿。通过史书的字里行间，我们推测交流的缺乏导致了争辩、沮丧和群体猜忌。很快，部落开始分裂——大家选择了孤立，而不是互相争吵。

第 8 章 胸有成竹

系统编程需要花费多长的时间？需要多少的工作量？如何进行估计？

第 9 章 削足适履

创造出自精湛的技艺，精炼、充分和快速的程序也是如此。技艺改进的结果往往是战略上的突破，而不仅仅是技巧上的提高。这种战略上突破有时是一种新的算法，如快速傅立叶变换，或者是将比较算法的复杂度从 n^2 降低到 n log n。

更普遍的是，战略上突破常来自数据或表的重新表达——这是程序的核心所在。

第 10 章 提纲挈领

我记得曾经有一个项目，在三年的开发周期中，机器指令计数器的设计每六个月变化一次。在某个阶段，需要好一点的性能时，指令计数器采用触发器来实现；下一个阶段，成本降低是主要的焦点，指令计数器采用内存来实现。在另一个项目中，我所见过的最好的一个项目经理常常充当一个大型调速轮的角色，他的惯性降低了来自市场和管理人员的起伏波动。

第 11 章 未雨绸缪

因此，管理上的问题不再是“*是否*构建一个试验性的系统，然后抛弃它？”你*必须*这样做。现在的问题是“是否预先计划抛弃原型的开发，或者是否将该原型发布给用户？”从这个角度看待问题，答案更加清晰。将原型发布给用户，可以获得时间，但是它的代价高昂——对于用户，使用极度痛苦；对于重新开发的人员，分散了精力；对于产品，影响了声誉，即使最好的再设计也难以挽回名声。

因此，*为舍弃而计划，无论如何，你一定要这样做。*

第 12 章 干将莫邪

就工具而言，即使是现在，很多软件项目仍然像一家五金店。每个骨干人员都仔细地保管自己工作生涯中搜集的一套工具集，这些工具成为个人技能的直观证明。正是如此，每个编程人员也保留着编辑器、排序、内存信息转储、磁盘实用程序等工具。

这种方法对软件项目来说是愚蠢的。

第 13 章 整体部分

和古老的神话里一样，现代神话里也总有一些爱吹嘘的人：“我可以编写控制航空货运、拦截弹道导弹、管理银行账户、控制生产线的系统。”对这些人，回答很简单，“我也可以，任何人都可以，但是其他人成功了吗？”

第 14 章 祸起萧墙

当一线经理发现自己的队伍出现了计划偏离时，他肯定不会马上赶到老板那里去汇报这个令人沮丧的消息。团队可以弥补进度偏差，他可以想出应对方法或者重新安排进度以解决问题，为什么要去麻烦老板呢？从这个角度来看，好像还不错。解决这类问题的确是一线经理的职责。老板已经有很多需要处理的真正的烦心事了，他不想被更多的问题打搅。因此，所有的污垢都被隐藏在地毯之下。

有两种掀开毯子把污垢展现在老板面前的方法，它们必须都被采用。一种是减少角色冲突和鼓励状态共享，另一种是猛地拉开地毯。

第 15 章 另外一面

面对那些文档“简约”的程序，我们中的大多数人都不免曾经暗骂那些远在他方的匿名作者。因此，一些人试图向新人慢慢地灌输文档的重要性：旨在延长软件的生命期、克服惰性和进度的压力。但是，很多次尝试都失败了，我想很可能是由于我们使用了错误的方法。

第 16 章 没有银弹

在所有恐怖民间传说的妖怪中，最可怕的是人狼，因为它们可以完全出乎意料地从熟悉的面孔变成可怕的怪物。为了对付人狼，我们在寻找可以消灭它们的银弹。

大家熟悉的软件项目具有一些人狼的特性（至少在非技术经理看来），常常看似简单明了的东西，却有可能变成一个落后进度、超出预算、存在大量缺陷的怪物。因此，我们听到了近乎绝望的寻求银弹的呼唤，寻求一种可以使软件成本像计算机硬件成本一样降低的尚方宝剑。

但是，我们看看近十年来的情况，没有银弹的踪迹。没有任何技术或管理上的进展，能够独立地许诺在生产率、可靠性或简洁性上取得数量级的提高。本章中，我们试图通过分析软件问题的本质和很多候选银弹的特征，

来探索其原因。

第 17 章 再论《没有银弹》

《没有银弹》中声称和断定，在近十年内，没有任何单独的软件工程进展可以使软件生产率有数量级的提高（引自 1986 年的版本）。现在已经是第九个年头，因此也该看看是否这些预言得到了应验。

*《人月神话》*一文被大量地引用，很少存在异议；相比之下，《没有银弹》却引发了众多的辩论，编辑收到了很多文章和信件，至今还在延续[3]。他们中的大多数攻击其核心论点和我的观点——没有神话般的解决方案，以及将来也不会有。他们大都同意《没有银弹》一文中的多数观点，但接着断定实际存在着杀死软件怪兽的银弹——由他们所发明的银弹。今天，当我重新阅读一些早期的反馈，我不禁发现在 1986 年～1987 年期间，曾被强烈推崇的秘方并没有出现所声称的戏剧性效果。

第 18 章 《人月神话》的观点：是或非？

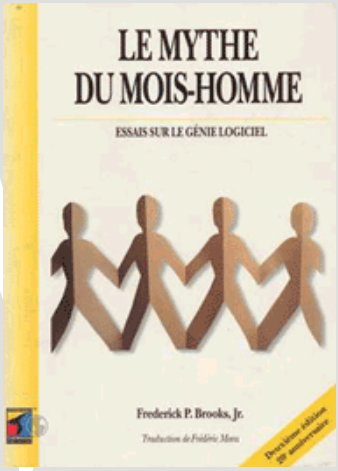
现在我们对软件工程的了解比 1975 年要多得多。那么在 1975 年版本的人月神话中，哪些观点得到了数据和经验的支持？哪些观点被证明是不正确的？又有哪些观点随着世界的变化，显得陈旧过时呢？为了帮助判断，我将 1975 年书籍中的论断毫无更改地抽取出来，以摘要的形式列举在下面——它们是当年我认为将会是正确的：客观事实和经验中推广的法则。（你也许会问，“如果这些就是那本书讲的所有东西，为什么要花 177 页的篇幅来论述？”）方括号中的评论是新增内容。

第 19 章 20 年后的人月神话

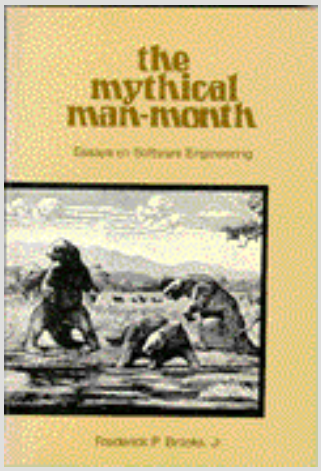
为什么《*人月神话*》得以持续？为什么看上去它仍然和现在的软件实践相关？为什么它还拥有软件工程领域以外的读者群，律师、医生、社会学家、心理学家，和软件人员一样，不断地对这本书提出评论意见，引用它，并和我保持通信？20 年前的一本关于 30 年前软件开发经验的书，如何能够依然和现实情况相关？更不用说有所帮助了。

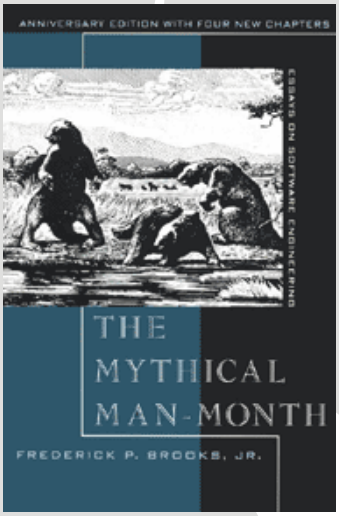
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)





(第二版)

(第二版日文译本)

《人月神话》软玉生香

刘天北 著 查看评论

从尺寸说起

豆豆先生（Mr. Bean）（编注：著名喜剧片的主人公）赞许惠斯勒的《画家母亲的肖像》，首先谈的一点就是："这幅画比较大（quite big），所以了不起（excellent）"。这种批评套路，中国古已有之：品茶时赞美“热得好”就是。今天我们谈谈 Brooks 的《The Mythical Man-Month》（中译为"人月神话"，下简称 MMM），也效法英国来的专家，从尺寸说起：这本书的一个突出优点，在于它比较小。

小有两点，一是开本，一是页数。尼采说，我的野心是用十句话说完别人一本书才能说清的事（他还加了一句--甚或是，一本书说不清的事）。可是对于出版工业，这样的野心家不要也罢。多数书长，还是因为所涉及的领域本身头绪众多，不过总可以用得上法国人的俏皮话：他没有时间往短里写。把书写短不是人人都可以办到的，这需要良好的分寸感和结构感--都是很难在匆忙的 IT 从业者中找到的素质。是以一般技术书籍，多为人高马大，卷帙繁浩（当然还有国内影印版的惯常做法，把原来的大开本缩印成小兄弟，但考虑到增加的阅读难度，结果并没有想象的那么经济）。“大”，对于书--尤其是技术书--来说，不见得就“了不起”。以我个人的阅读习惯，多于五百页、开本又超过“大 32 开”的技术专著，那就只能当字典查，又好比皇宫里的大龄宫女，偶一调情也可，若要三千宠爱集一身，不啻为非分之想。本来一卷在手，有如软玉生香在抱，总以“轻”、“薄”为妙，若是庞然巨物，体态狼伧，未免让人产生心理障碍（借此一角向通读了《C++ Primer》的各位致意：你们都可以去做《大内密探零零发》里艳福齐天的皇上）。眼下说的这本 MMM，正是技术书中的赵飞燕，可作字面意义上的掌上舞：大 32 的开本，算上注释、索引共 332 页，要紧的是，每章前的大量篇幅留给题图和题记，19 章下来，实在要读的内容不到 200 页。

如果决定读这本书，你也许可以参照我个人的经验做出时间预算：并非出于固定的读书计划，而是为了谋杀早晨巴士上的无聊时间，我选中了这本相貌可人、插图众多的 MMM（它的前任似乎是袖珍本的斯各特船长传）。在前一周的周日下午我兴味盎然地覆盖了书末的三章：后记（“The MMM after 20 years”）、作者著名的单篇论文《没有银弹（No Silver Bullet）》和《再论没有银弹（No Silver Bullet Refired）》；这之后，第 1 章以下的若干章大约以每次 1 章半到 2 章的速度平均分布在 5 次车中颠簸的早高峰时间（每次约 35 至 40 分钟，并有一次坐过站）；最后，一个睡眼惺忪的周六上午让我完成了其余部分--我没法相信，这种"休闲"的阅读形式适用于任何一本其他的技术“名著”（可能的例外是《Peopleware》（《人件》，即将由清华大学出版社出版，译者为 UMLChina 翻译组方春旭、叶向群），我们一会儿还会谈到它）--或许你明早会带上本《Planning XP》试试。

美味的分析

那么，除了小，这书还有什么好处？对，再就是：它比较好读。我看书也纯粹是"以貌取人"，外表即是一切（法国人说：皮肤是人最深刻的部分）。就像饭菜， "好吃"就好，营养等等在我辈老饕看来，还是第二义，而出于“精神食粮”这话的本意，“好读”也该是对书的最高赞许。

好读有两层意思，读起来容易，和读起来可口。小书不都易读（听说过《逻辑哲学论》？），易读的书也不都可口（有些易读则易读矣，但就好像小儿止咳糖浆，来多了就是对你智力的侮辱--还用提余秋雨老师吗？）但对于一本技术书而言，把自己的主题讲清楚了，而且还易读，也是了不起的成就。好些人写书，一味在“内在美”上下功夫，殊不知，“可接近性”才是首要的一环：我们不也说“可口”、“可人儿”吗？“可”（也是 accessibility 里的 ability）最要紧。

回到正题，谈谈全书的结构：我手上这本 MMM 的二十年纪念版（Anniversary Edition），共 19 章（另有简短的前言--初版和再版各一篇--和尾声）。其中，第 1 到 15 章是初版内容，最长的一章 17 页，最短的第五章刨去题图和题记，不过薄薄两张半纸而已。作者称这些章节为"随笔（essays）"，每章各自处理相对独立的主题（“人月”问题、软件项目的人员构成、开发中的交流、文档、排错等等），因而你基本可以不顾及章节间的联系，单独从你感兴趣的任何段落入手。书末的四章里，第 18 章"Proposition of MMM：True or False?"是对之前各章所有观点的一个列表--一个非常便于翻检和温习的设置；其他的三章（对于“银弹”的两篇讨论和后记）篇幅在二十到四十页，值得专门对待。对于一本初版于多年前的技术书籍，作者并没有逐条修订原有的内容，而选择了重印这些章节，同时在后记中根据二十年来的技术上和认识上的变化检讨得失。

现在，你大概能模糊地看到这本书的样子：很薄，随手翻上去每章很短，并包含大量整页的插图（从侧面看，书页的颜色黑白参差）。换言之，可读性首先来自于外表，你拿起它，马上就感到美味当前。

然后你会注意到书的风格。关于原版技术书的语言风格，我曾经在别处做过一个区分：可读的/可翻译的（对另一对为人熟知的法国概念的蹩脚模仿）。“可读的”风格主要出现在用母语写作的作家笔下。他们能够自如地使用语言中的元素，几乎总能找到对特定概念最合适的表达。这些表达，往往植根于该语言自身，因此很难转换为另一种语言。比如本书作者给讨论“系统空间限制”的一章的标题是“ten pounds in five-pound sack”。对任何具备高考英语水平的读者，这里的意思都非常直观，但是谁能轻易地给出它的中文等价物？"削足适履" ？（我碰巧知道中译本就是这么翻译的。）“五磅的麻袋装十磅”？另一方面，“可翻译的”文本则多为移民作家的产物。对于一个概念，他们缺乏“自然的”表达方式。他们在用别人的语言，一串串的词像临时借用的家具一样摆放那里。你

可以简单地逐字翻译这样的文本，甚至可以采用机器翻译而不会犯大错误（从 Jacobson et al 的 OOSE 摘引随便一句："When strong semantics are provided at the behavioral description level for all of these fundamental issues, they radically affect the selection of an appropriate resource structure."），但是无论译或不译，都很难马上明白他说的是什么。当然，这只能是对实际情况的最简化的 0 阶模拟（一些喜欢在行文中加入俚语的本土作家几乎不能归入任何一边）。但是，当我说出"可读的"时，我首先想到的就是 Brooks 和另一位大师 Martin Fowler。二者的风格具有不少共同之处：亲切、不卖弄，能用最普通的英语说明（有时是令我们的汉语绝望的）复杂问题。

本书的作者 Brooks（也许应该说"小"布鲁克斯才对，因为他的全名是"Fredrick P. Brooks, Jr."），曾担任 IBM 的大型机 System/360 及其操作系统 OS/360 的开发项目经理，被称为“S/360 之父”，之后转入大学，执教多年，因而难能一见地在实际项目管理和教学上都具备世界级水平。在离开 IBM 近十年之后（1975 年），他根据大量切身实践和教学经验写出了本书，在风格上，也兼具教师的醇厚耐心和工程师的技术素质。我个人最受益于书中随处可见的精彩的比喻。仅仅举几个例子：“焦油坑”、“大教堂”、“外科手术队伍”、“银弹”，就我记忆而言，几乎每一个都是对所指对象最贴切的形容。而这些隐喻的光彩，又通过众多题图、题记（有时不失幽默）的反射，获得了熠熠生辉的效果。

在很多情况下，过分追求“幽默感”也会产生止咳糖浆式的效果。典型的例子是 Peopleware。与 MMM 相比，那本书总是给我一种推销员教材的印象。二书作者技术背景（我不认为 DeMarco 们曾参加过严肃的软件项目）、宗教背景的不同（Brooks 是位虔诚的基督徒）也许可以解释卖弄和朴实、空洞和厚重之间的差距。

来自法式餐馆、手术台和狼人的启示

让我们回顾豆豆本人的批评实践。在我看来，它具备难以错认的形式主义特征：主要考虑“外在”的、形式的（比如画的大小）要素，而忽略作品实际再现的内容--我们记得豆豆本人唯一一次提及画中的那位老太太时的措辞：停在仙人掌上的、丑恶的老蝙蝠（"a hideous old bat who looks like she's got a cactus lodged up her backside"）。我也要采用这样的策略吗？读者们（我设想，看到这篇书评的各位多数是技术专家）恐怕会讥之为买椟还珠，另外，MMM 的内容也远比“老蝙蝠”更有可谈之处。但即便如此，逐一考察书中的观点也是不现实的：首先，没有人比作者已经做过的更好，如果你确实需要，不如去看原书第 18 章；其次，全书并没有一个一以贯之的中心思想可以提取，对于比如说沟通、文档、排错、工具等领域，每一个都值得专文仔细讨论。经过考虑，我决定谈谈那些无需我重读原书就可以回想起来的内容--这也就是经过我的巴士阅读实践（基本上是一个高通滤波过程）幸存下来的精华成分。

人月

首先是“人月（man-month）”。熟悉软件项目管理的各位肯定清楚，人们常常根据人月来估计工作量（并相应收费），比如一个项目五人两月完成，那么总工作量就是 10 人月。本书以此命名，套用唱片工业的术语，可以说"人月神话"是本"专辑"中主打篇目，而除了以之为标题的第二章之外，并没有太多内容与此有关。

称之为“神话”，作者的用意也并非完全否定作为计量方法的人月，而是要理清这个概念中隐含的种种错觉。根据我勤勉而不可靠的记忆，这里的主要论断包括：1. 人/月之间不能换算，换言之，两人做五个月完成，不等于说五人做两个月就能完成；2.（也是最有争议的一点）在项目后期增加人手，只能使工期进一步推迟；3.项目越大，单位工作需要的人月越多。或者说，归根结蒂，作者要粉碎的是“人月”概念可以线性把握的神话：无论是开发人员的人数上，还是工作量本身上的变化，都可能导致最终完成时间的非线性变化。作者的戏剧性表述是：Adding manpower to a late software project makes it even later. 这个观点事实上并非像它看上去那么有挑衅性，而作者在初版以及后记中，都表示这包含了合理的夸张，后记里甚至援引进一步的研究表示，增加人手不一定会使工期进一步推迟，不过肯定会使工程效率进一步降低--而如果一定要增加人手，越早越好。

得出这样的结论，主要的考虑是增加人手带来的隐性花费（人员培训、多人工作时的通信成本等等）以及项目进程中存在的无法逾越的关键路径（crucial path）。对于项目管理人员来说，这应该已经是一个不言而喻的真理。但由于最终决策者的压力，实际项目中却往往会发生与此违背的情况。作者把一份法式餐厅菜单塞进了书中作为题图，我也建议大家学会菜单上的那个句子：Faire de la bonne cuisine demande un certain temps. Si on vous fait attendre, c'est pour mieux vous servir, et vous plaire（做好菜需要一定的时间。如果人家让您多等会儿，那是为了让您最后吃得高兴。） 对于说法语的客户/老板，这会是一个好的搪塞。

概念完整性

如果一定要找到贯穿全书的概念的话，"概念完整性（conceptual integrity）"可能算是一个。宁可少添加一些七七八八的功能（anomalous features）,也应该保证，整个系统体现的是完整的一套设计理念 ：这是引入概念完整性的含义。概念完整性的受益者包括最终用户、系统开发者、培训员和服务人员。概念完整性保持得好的系统更易用，需要较短的培训时间和学习时间，同时也更易于开发。但是在软件生产的实践中，各级决策者都很容易产生强调“功能”而忽视概念完整性的倾向。因此我们看到了太多包含大量“功能”，而就整体而言不知所云的系统，也出现过太多次由于一两个附加功能的实现难度而导致整个系统开发推迟、甚至难产的事例。

所以作者提出，概念完整性是系统设计中最重要的考虑。在保证概念完整性的各方面中，我认为作者提出的两点最为有趣。第一，要克服系统的“第二版效应（the second system effect）”。一个系统设计师在设计第一版系

统时，往往出于较弱的自信心以及量力而行的考虑，会尽量剪裁要实现的功能数量。而当第一版系统成功发布，开始第二版的设计时，随着自信心的增强，大量以前被压制的提议都会重现，设计师在塞入新的功能时也会不再那么保守。这很可能导致一个臃肿而缺乏概念完整性的第二版系统（作者的 1995 年举的例子是 Windows 31 之后的 Windows NT）。第二，整个项目团队的有效组织有助于保证概念完整性。首先，要区分产品人员和开发人员，architect 和 implementer。这里所说的 architect 相当于一般而言的产品经理。只有他以及少数的几个人能确定整个的产品需求，而不应把即使是最细微的需求留给开发人员确定。在需求确定的过程中，多次反复，包括来自开发人员的反馈都是必要的，但为确保概念完整性，做出决定的必须是少数，甚至一个人。另外，在开发中可以考虑“外科手术式的”开发团队，即，由唯一的“手术师”（技术大拿）以及多个助手、测试员、联络人员、文档/工具/配置管理员、秘书等组成的队伍。显然无论是产品/开发人员的区分，还是外科手术式队伍的引入，核心原则都是 1）职能细分；2）将高要求的工作集中在少数人手中。事实上，作者在其他的章节里说过，提高软件项目的质量的最好办法之一，就是找到合适的人。“伟大”的设计师和普通设计师之间的差别，比采用任何先进工具/方法论/管理模式前后的差别都要大。

银弹和其他

另外，增补部分中的"银弹"是作者独创的又一个概念。在古代的狼人传说中，只有用银质子弹才能制服这些举止无常的怪兽。因此作者也用“银弹”一词，命名人们渴望找到的制服软件项目这头难缠的怪兽的法宝。“没有银弹”意味着，没有任何一种方法（无论技术上的或是管理上的），单单采取它就能将现有的软件开发生产率（可靠度/简洁度）提高一个数量级。

作者的论证是简洁而（在我看来）有效的：软件开发的困难来自两个方面：本质的和偶然的（类似于经院哲学中的本质/偶性的对立）。本质的困难是软件开发本身所固有的，无法用任何方式取消的，而偶然的困难是其中的非本质因素，可以通过引入新工具、方法论或管理模式来消除。关键在于，只要本质的困难在软件开发中消耗百分之十以上的工作量，则即使全部消除偶然困难也不可能使生产率提高 10 倍。

就像作者其他很多论证一样，读完关于银弹的部分我们或者会说“本来就是这样，这还用说”，或者会说“这么简单的事，我以前怎么没有想到”。正是这种介乎漠视和震惊之间的摇摆体现了这些论断的价值：它们也许在理论上已经是被过分强调、被超越、被唾弃的了，但即便如此，在实践中我们往往连这些基本的原则都会完全不顾。对我们而言，MMM 或许首先是一本关于态度的书。它指认的问题首先是，对待软件开发工作时应采取怎样的态度。正是因为某种态度，我们才会热衷于寻找解决一切问题的银弹（RUP 也好，CMM 也好，XP/Agile 也好）而从未诚心诚意地贯彻其中任何一个原则；我们才会急于在系统中塞入一个又一个的功能，无视用户使用时的实际

效果；我们才会无法相信自己能够组成“外科手术式”的团队，无法找到不做产品设计的开发人员；我们才会一次次接受形形色色的人月调整，最后把一个个项目带进了“焦油坑”。

据我看来，软件工程首先是一种缓慢的艺术。在这中间，筹划、交流、冥想以及迭代占有很大比重，同样（如果不是更重要的），错位、反复甚至离题也应据有一席之地。即便强调工程的可控性，也难以排除这些必要的环节（记得法式餐馆的例子？）。我们所能做的，是给它们足够的时间和容忍（你知道我不是指纵容延期）。

Brooks 在书中提出的两组时间比例也许值得留意：一个“能转（ready to run）”的程序（program）要变成程序产品（programming product）需要 3 倍于编程的时间；同样，能转的程序变成程序系统（programming system）也需要 3 倍的时间。准此，则要获得程序系统产品（programming systems product--有点拗口，但仔细想想，还可以？），需要 9 倍于编写“能转”的程序的时间。另外，他在提到工期规划时说，整个工期的三分之一给计划（planning），六分之一编码（coding），四分之一组件测试和早期系统测试（component test and early system test），最后四分之一系统测试。

类似的划分，远比我们常见的各种方法论中的“里程碑”粗糙得多，但是，有多少人在实践中相信编码应只占整个工期的六分之一甚至九分之一？是否仍是“态度”的原因使我们失去了缓慢工作的能力？我们能把这种态度命名为“浮躁”吗？当所有竞争者都许诺只用那六分之一就能完成整个项目的时候，你能坚持那个“一”吗？我们能像提高气功境界那样，从一种态度跃迁到另一种吗？软件开发是一种气功吗？外科手术、银弹和法国菜都是气功吗？狼人能用气功治好吗？气功也能用人月估计吗？豆豆和老蝙蝠，谁更像狼人？狼人也会浮躁吗？浮躁的狼人究竟更需要气功还是银弹？还是书评？一篇豆豆式的书评？

尾声

写到这里，我愿意介绍我手头这本 MMM 的由来。打开书皮，我能看到封底处“Low Price Edition (LPE) authorized for sales only in India, Bangladesh, Pakistan, Nepal, Sri Lanka and Maldives”的字样。原有的书店标签（我印象中是"Graham，250rs"）不知什么时候脱落了。但即使缺乏这个标签，仅仅“LPE”还是能让我回想起在班加罗尔 Graham 书店度过的那个匆忙的黄昏：第三或四层是技术专区--我开始只找到了 Jacobson 们的 OOSE（有点困惑于复杂的书店布局）--看到另外一人手中的 Software Project Survival Guide--我求助于一个店员，他立刻拿给我那本书，和一本 MMM--另一个店员看到它们，又递来一本 Peopleware.

本文的完成，主要与我一次偶然的阅读有关，与一些论坛中朋友们对 MMM 的热衷和猜想有关，而与该书中文版本的推出可能只有间接的关系。我无意以此给这个译本作任何广告（我在别处谈过我对中文翻译的态度），当

然更无意作反广告。不过，如果现在手头没有这本书的话（一个不幸的假设），我会到书店翻翻那个译本，看看插图复制的质量（尤其是那些铜版画--LPE 能让我数清狼人脚爪上的毛），再看看书后还有没有注释和索引。仍然是那些外在的、肤浅的因素会决定我是否购买，就像在任何一个豆豆式书评的作者那里一样。

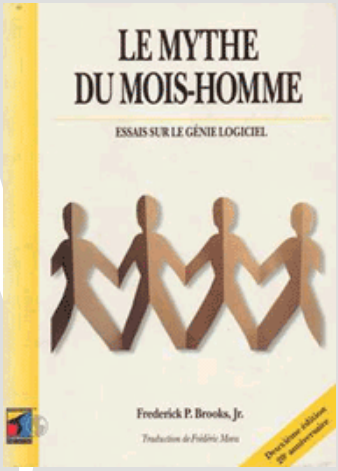
[参考资料]

- The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition) by Frederick P. Brooks, 1995 Addison-Wesley, 322 pp
- Peopleware : Productive Projects and Teams, 2nd Ed., by Tom Demarco, Timothy R. Lister, 1999 Dorset House, 264 pp
- Object-Oriented Software Engineering: A Use Case Driven Approach by Ivar Jacobson et al, 1992 Addison-Wesley, 524 pp
- Bean: The Movie, by Mel Smith (Director), 1997, 90mins

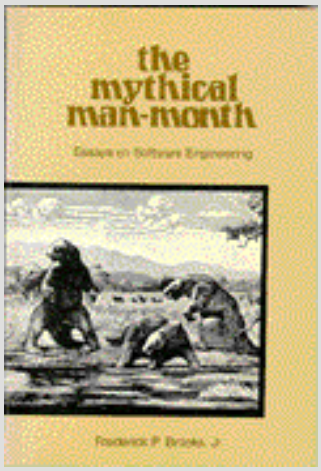
本文最初出自 IBM DeveloperWorks 中国网站（ibm.com/developerWorks/cn）

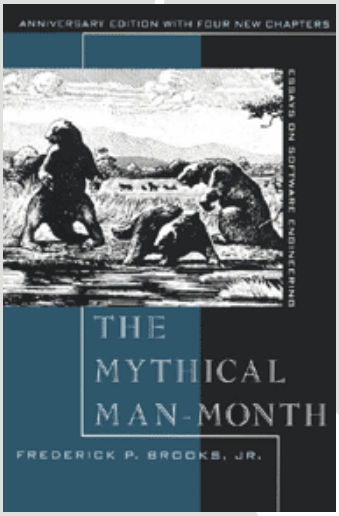
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)





(第二版)

(第二版日文译本)

《人月神话》与实践

Adams Wang 著 查看评论

关于《人月神话》一书，已经有了许多评论和讨论。可能，作者本身的经历——**“被认为是‘IBM 360 系统之父’，他担任了 360 系统的项目经理，以及 360 操作系统项目设计阶段的经理。”**——已经是最好的评论。

许多朋友认为现在的软件工程数据比较理论化，可操作性不高，往往只能了解一些理念。在面对具体项目的时候，还是有些迷茫。而在整个翻译的过程中，Brooks 的观点以及治学态度经常令人叹为观止。这里，就自己的一些体会和实践同大家探讨。

“焦油坑（The Tar Pit）”一章中，对编程系统产品的观点**“*编程系统产品*（Programming Systems Product）。和以上的所有的情况都不同的是，它的成本高达九倍。然而，只有它才是真正有用的产品，是大多数系统开发的目标。”**很有意义，至少在面对现在许多老板们对管理的观点**“现实世界中的管理就是在更大程度上以人员的生命为代价，让他们更努力、更长时间地工作。经理们总是不停地吹嘘他们的人员的加班时数和能从这些人身上榨取更多时间的小把戏。”**（《人件》，即将由清华大学出版社出版，译者为 UMLChina 翻译组方春旭、叶向群）面前聪明的工作。这种情况下，开发产品的质量一定会下降，甚至惨不忍睹，因为开发人员唯一能控制的是质量，当他们不得不牺牲质量，痛苦面对自己的工作，践踏工作的乐趣时，可以想象项目成本会大量地增加，并且项目的发布往往伴随着一大批程序员的倒下。

团队组建

“人月神话（The Mythical Man-Month）”提出了这样的论断，（盲目地）**“向进度落后的项目中增加人手，只会使进度更加落后。”**这中间还涉及到如何组建你的开发团队，或者面向一个软件开发任务时，如何规划开发计划、划分任务项、分配资源。紧接着，在“外科手术队伍（*The Surgical Team*）”中，Brooks 提出了用外科医生＋副手来组织团队，保证设计思路的完整性。其中，还提到了采用“语言专家”来帮助疑难问题的解决；安排工具维护人员，也就是现在意义上的系统管理员来保证系统开发、管理环境的有效运行；而其他人来解决一些文件管理等工作。

在一个小型的 C++项目操作中，对上述方法的实践中

- 由 PM 和结构师一用一备来分析需求、进行框架设计，确保整个项目的概念完整性。分析设计的产出物达到框架示意代码的级别，这部分框架代码主要是帮助团队对项目开发的理解，不存在于正式的代码中；

- 安排开发人员负责配置管理。这里的配置管理不仅仅局限于文档、软件产物的管理，而是在使用框架驱动的迭代开发时，需要对框架和各个组件不断地进行编译、整合、检查记录 Bug。这位开发人员往往是团队中的主程序员（Chief Programmer），对各种开发方法、方法学有着一定的经验；

- 安排人员对工具进行预研，如了解 STL 类库等。该角色具体的人员在不同的阶段会进行调整。因为，他/她需要对语言、类库、开发技巧进行学习研究，往往会占用大量的工作时间。在实际情况中，前期是有主程序员承担；后期，由 PM 承担；

这样的安排的确能解决产品思路的一致性，在很大程度上提高生产力和产品质量。不过，它要求：

1. PM 和结构师有**非常**良好的沟通，包括分析方法的风格、对开发的理解等，他们之间的不一致直接会导致项目的开发方向；

2. 对分析的结果，进行良好的贯彻。软件行业具有年青、有朝气的特点，同时也有些浮躁。有的开发人员好高骛远，对代码质量重视不够，影响框架的稳定性。

这样安排的风险在于“外科医生”的工作负荷可能过大，尤其在国内的公司中，他们往往同时兼任 PM 和系统分析的工作。同样，相应的绩效考核机制也不容易具备。

另外一个风险在于“一用一备”的安排，在“为什么巴比伦塔会失败？（*Why Did the Tower of Babel Fail?*）”的“大型编程项目的组织架构”中详尽地论述。不过，在国内的开发环境下，似乎很难实行。比较折中的方案，是对某种类型的项目，包括业务类型、开发方法、语言类型，建立开发和管理指南，从而保证概念完整性。在后续所经历的一系列 Domino 类型的项目开发中，进行了尝试，相应在实践中遇到的问题在于如何贯彻实施。

上述方法潜在的一个声音是“软件重用性”。提到重用性，可能大家脑海里马上想到的是对象、类库等。不错，面向对象的方法、商业组件（类）库的确极大地提高了软件重用性。但对于中小型企业，甚至于一个成熟的开发团队，积累自己大粒度的框架是提高软件生产力的一个重要措施。设计模式中一些模式，如 Visitor、Observe 本身就可以作为软件框架来使用。以设计模式为基础，根据开发类型积累特定业务领域的框架是完全可行的最佳实践。而 Brooks 一再强调“概念完整性（Concept Integrity）”，以及在“外科手术队伍”中推荐由外科医生来负责系统的开发，保证了产生的系统是少数人思维的结果，它们往往简捷、纯粹，没有众多人的影响，经过了项目的洗礼之

后，往往能成为重用的框架。相反，在分析设计阶段，安排许多开发人员来共同开发，尽管同样完成了项目，但会相应导致“画蛇添足（The Second-System Effect）”特征——“***它极富有创造性，极端复杂，非常高效。但不知为什么，同时也感觉到粗糙、浪费、不优雅，以及让人觉得必定存在某种更好的方法***”，而重用的框架往往是捕捉到了问题的根本，用简单优美的方案，解决 80%的问题。

贯彻实施

Brooks 在“贵族专制、民主政治和系统设计（*Aristocracy, Democracy, and System Design*）”一章中，一针见血地指出了成功软件具有高度的概念一致性。“**结构师难道不是新贵？……至于贵族专制统治的问题，必须回答“是”或者“否”。就必须只能存在少数的结构师而言，答案是肯定的……**”，《人月》中的体系结构更加类似于现在的需求概念，而非现在所意义的体系结构。不过，在具体的实践中，需求和体系结构依然应该由少数的人来承担。这样的观点可能会引起争议，但就个人观点而言，软件行业实际上是“精英行业”。高素质、具有丰富经验的需求分析人员、结构师往往是软件企业的核心骨干人员，相应的一般编程人员相对的流动性会大一些。

从个人发展的角度，安安静静地作为一个螺丝钉仿佛不是这个时代精神。这个问题同企业与人之间的关系一样，很难一言以蔽之。就具体的项目操作，理想情况是程序员较少地参加前期分析工作，由有经验的同仁来负责，给出具有框架代码程度的需求和框架产物。

在较早的项目中，有的一般编程人员不安于编码实现，过早接触于一些 PM 性质工作，以及项目压力等其他因素，造成了代码质量不高，使得框架的重用程度降低。而在另外公司的一个项目中，由于企业的文化，同事们严格按照分工进行开发，提高了质量。后者看似少了机会，但从长远的角度看，大家对设计的了解更加透彻，对流程的理解更加深刻，为以后的发展打下了基础。

同样在这一章节中，Brooks 提出了“在等待时，实现人员应该做什么？”的问题。他认为“**首先，必须设定良好定义的时间和空间目标，……同时，在物理实现的级别，也有很多可以着手的工作。**”实际项目中，早期的投入往往人员较少，一般 15 人月左右的项目，初期就 2～3 人，这其中包括了需求分析、设计人员，以及主程序员。主程序员会对系统有初步的了解，对系统开发采用的技术、工具、开发技巧进行研究，同时同 PM 一同负责搭建软件工程环境和软件开发的环境。这样的安排，出现的一个问题是编程人员与分析设计人员的沟通。它需要大家通畅、自由的交流，经可能对开发达成共识，减少信息的丢失。

项目交流的方法，Brooks 就 OS360 开发经验，在“贯彻执行（*Passing the Word*）”一章中，进行了具体的讲解。如，会议与大会、多重实现、电话日志等。就小型项目而言，如果不考虑异地开发的情况，较正式的组内会

议可以安排在一周一次，即周例会的形式。另外，在里程碑之处安排评审会议，以对开发的进展、对需求的实现以及项目计划进行调整和修正。而对于异地开发，或者用户不在本地的情形，则建议采用电话会议等形式，效果虽然不如本地开发好，不过有聊胜于无。

“为什么巴比伦塔会失败？（*Why Did the Tower of Babel Fail?*）”（编注：巴别塔）又对软件开发中的交流问题进行了强调，指出了文档是沟通的一种重要方式。后续的 “提纲挈领（*The Documentary Hypothesis*）”则用类比的方式阐述了如何定义软件项目的文档集合，“另外一面（*The other face*）”讨论了程序文档的一些形式。这些观点对实际工作也具有指导意义，相应的 Rational Suite 中 Requisite Pro 工具使用参考中，推荐了软件项目的参考文档集合。在 C++和 Domino 若干项目的实践中，发现最小的文档集合包括需求文档（Software Requirement Specification）、项目计划（Software Development Plan）、框架设计、设计元素说明。其中，项目计划不仅仅是进度，而是从软件的配置管理、生命周期、资源分配、风险分析、培训等方面进行描述，这部分的内容往往不局限于单个项目，而是在同一种类型的项目中具有共性。因此，可以在不同项目中共享。

总而言之，Brooks 对文档的建议是**“项目经理聪明的做法都是：立刻正式生成若干文档作为自己的数据基础，哪怕这些迷你文档非常简单。接着，……”**以及**“如果一开始就认识到它们的普遍性和重要性，那么就可以将文档作为工具友好地利用起来，而不会让它成为令人厌烦的繁重任务。通过遵循文档开展工作，项目经理能更清晰和快速地设定自己的方向。”**

面向变更的开发

“稳定状态的生产思想特别不适合项目工作。我们倾向于忘记这一点：项目的全部目的就是让自己死亡。项目生命中的唯一稳定状态是死后僵硬……”（《人件》）；

“软件开发是减少混乱度（减少熵）的过程，所以它本身是处于亚稳态的。软件维护是提高混乱度（增加熵）的过程，即使是最熟练的软件维护工作，也只是放缓了系统退化到非稳态的进程。”——“未雨绸缪（*Plan to Throw One Away*）”

软件开发本身就是一个具有无序趋势的活动。当人们四处游走，寻找一个类似于计算机硬件那样流水线化的方式方法时，可能应该静静思考一下，这本身就是与开发内在发展规律相悖。并且，软件开发是人类的思维创造活动，同诗歌、乐曲一样，好像人类历史上还没有什么为上述创造进行流水线化的尝试。

从软件开发产物的角度而言，CMM 规范 2 级中的配置管理一般包括四个活动：配置标识、版本控制、变更控制和度量。配置标识和版本控制都是为了给变更提供一个稳定的承载基础。而 Brooks 对 360 机器开发的进行了

描述“**在 System/360 工程模型中，在一大堆常规的黄颜色电线中，常常可以不经意地看到紫色的电线束。……这些更改过的接线使用紫色电线，看上去就像伸着一个受了伤的大拇指。**”体现了配置管理的雏形，进一步提出了在软件项目中“**……软件开发也需要用到“紫色线束”的手法。对于最后成为产品的程序代码，它更迫切地需要进行严密控制和深层次的关注。……而且需要文档化。**”——“整体部分（*The Whole and the Parts*）”

在 CMM 试点项目以及后来的项目实践中，为项目的文档产出物进行标识是比较容易实现的。Internet 上有大量的资料和模板可供参考。版本控制可以使用 VSS 或者 Open Source 的 CVS，而 CMM 规范中反复出现的“基线（baseline）”概念，在配置管理实践的初期或者小规模的项目中，并不是强制性的。当配置标识和版本控制实践到一定的程度之后，再推广基线和变更控制，就相对容易得多。

即对于某种类型的项目，可以采用制定配置标识规范、识别文档类型、确定该类项目的目录结构、建立版本控制机制来进行初期的实践，具有一定的成熟度之后，再实施变更管理。变更使用“**阶段（量子）化、定期变更……量子（阶段）化变更方法非常优美地容纳了紫色线束技术：直到下一次系统构件的定期发布之前，都一直使用快速补丁；而在当前的发布中，把已经通过测试并进行了文档化的修补措施整合到系统平台。**”这种方法能在一定程度上缓解项目压力、变更和质量之间的矛盾。

迭代开发

Brooks 在“20 年后的人月神话（The Mythical Man-Month *after* 20 Years）”明确提出了迭代开发的概念“**增量开发模型更佳——渐进地精化**”。这种开发方法对项目产生的激励作用是不可估量的，作者在北卡罗来纳大学时，“**我常常会被屏幕上第一幅图案、第一个可运行的系统对团队士气产生的鼓舞效果而感到震惊。**”而在 C++的现实项目中，在极大的压力下，当第一个可执行的原型出现在开发人员面前时，长期的疲惫、沮丧一扫而空。从而，为继续开发提供了动力。这里，长时间高负荷工作并不是被提倡的，但增量开发是软件开发动力学的一种重要手段。

迭代开发可以用“外科手术队伍”、OO 及设计模式的实现来实践。面向对象框架的分析设计思想，并不一定非要用面向对象的语言实现。它的一个重要特点，是固化用户要求或者是待开发系统中相对稳定的部分，以牺牲模型某个维度的代价来得到较稳定的模型。然后，在该框架的基础上不断地迭代。“外科手术队伍”则由具有丰富经验的“外科医生”操刀，来主持需求分析和建立迭代框架。少数的人能确保框架不带有过多不必要的非结构性的功能特色，从而保证框架的简捷和良好的扩充性。

迭代开发本身是对变更这个“怪物”的一剂良药，不同迭代周期能较好地容纳阶段（量子）化的变更。在变

更的同时，始终有可运行的系统供调试、测试，从而确保项目的质量——质量决定成本，**“远远超过最终用户需求的质量是一种取得更高生产力的手段。”**（《人件》，即将由清华大学出版社出版，译者为 UMLChina 翻译组方春旭、叶向群）。

同样，Brooks 在“未雨绸缪（*Plan to Throw One Away*）”中提出了抛弃型原型的概念，并在“《人月神话》的观点：是或非？（*Propositions of* the Mythical Man-Month: *True or False*？）”中强调**“因此，*为舍弃而计划，无论如何，你一定要这样做。*”**迭代开发是容纳原型，包括抛弃型和非抛弃型。而且，它是一个很好说服管理层和用户的途径——因为，不同的迭代周期和任务本来就在计划之中。在 C++的项目中，对于没有坚持在框架设计完成之后，抛弃框架代码原型，至今还有些遗憾，相应所付出的代价就是产品质量和开发人员对该项目信心的丧失。

另外一个提高项目产品质量的方法，Brooks 在“整体部分（*The Whole and the Parts*）”的“系统集成调试”中给出了专家意见，**“使用经过调试的构件单元”、“搭建充分的测试平台”、“一次添加一个构件”**和**“阶段（量子）化、定期变更”**。这在目前的实践中依然有指导意义。现在的软件测试水平相信业界人士都非常清楚：很少有规范化的测试；白盒测试基本不做；项目压力过大，不断压缩测试时间……。因此，仔细的系统集成测试非常有必要，能够做到《人月神话》中，OS360 的系统测试一半已经非常不错了。

项目计划

项目计划的重要性相信每个人都了然于胸。Brooks 在“祸起萧墙（*Hatching a Catastrophe*）”一文中提及了项目计划/跟踪。另外，在其他的许多章节中，阐述了计划文档化的重要性。这里，项目计划不仅仅指的是项目进度，采用 Microsoft Project 所画出的仅仅是项目的进度。

在具体的实践中，项目计划可以从以下若干方面考虑：

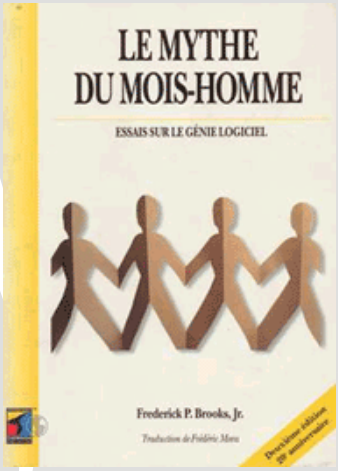
- 项目描述：包括项目定义、名称、背景、目标与范围、交付物和验收标准等；
- 项目组织结构：定义项目人员组成。注：这部分内容会根据项目的实施不断调整；
- 软件生命周期：依照项目特点决定合适的周期。并不是所有的项目都适用与迭代周期，也不是瀑布模型就一无是处，甚至有的项目需要瀑布模型作为主干，在某个阶段加入迭代特性。
- 项目管理：包括客户管理、进度管理、成本管理、风险管理、培训管理等；
- 配置管理：定义项目目录结构、产物标识方法、版本控制等；

项目计划必须根据用户、项目特点仔细地考虑，它可能是寥寥数语，但是它的作用不在需求规格说明之下。项目计划和需求规格说明是项目文档集合中最基本的强制性文档。

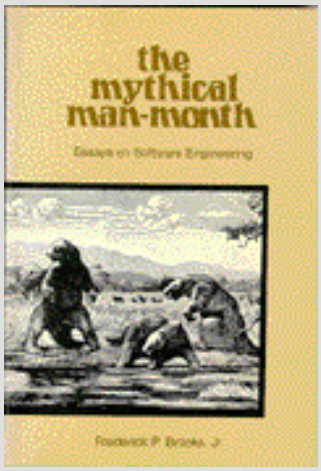
软件开发中，理论与实践是个亘古不变的话题。我们很有幸诞生在一个信息共享的时代，能够接触到前人睿智的思想。**“这个神奇的时代远远没有结束，它依然在飞速发展。更多的乐趣，尽在将来。”**——Brooks。

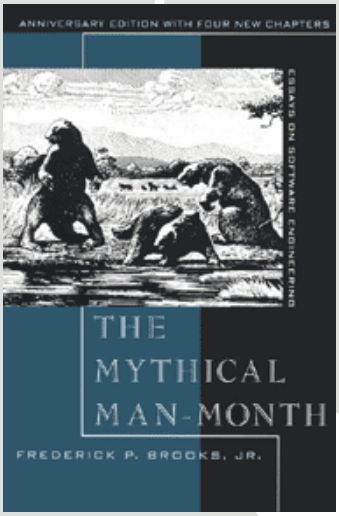
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)



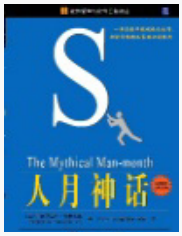


(第二版)

(第二版日文译本)

《人月神话》20 周年纪念版评论集

陈懋成 编译　　查看评论



购买

原书名	The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)
原出版社	Addison Wesley
作者	（美）Frederick P. Brooks, Jr.
译者	UMLChina 翻译组汪颖
书号	7-302-05932-2
页码	370
开本	32 开
定价	￥29.80
折扣价	￥23.84
出版日期	2002-11-1

Brian Kernighan

我唯一一本读过一遍以上的书，是 Fred Brooks 的《人月神话》，实际上我每过一两年都重读一遍。部分原因是这本书文笔很好，部分原因是书中的忠告很有价值，即使是 25 年以后。当然，现在很多细节上的地方，和我们做事情的方法，都有不同。我们的工作更自动化，计算机的“马力”更强劲，但书中依然有许多好的忠告，我非常推崇这本书。这是我唯一能想起来的你能从中体会到乐趣和思想的计算机科学书籍。

Ed Yourdon

有些书对于读者和作者象是年金，它们年复一年的分红。《人月神话》就是这么一本书。我直到在 1994 年接到 Brooks 教授的电话，才真正完全赏识它。

原因来自他的电话，电话中他说，他的出版商请他修订他在 1975 年第一次出版的这本书。我立刻表示了我的一点妒忌、羡慕；因为我的出版商从未叫我修订一本出版接近 20 周年的书。确实，我甚至表示了这样的意见： 当代的软件工程师不会考虑阅读这本如此古老的书，因此，可能无法卖出一本。“哦，不”，Brooks 教授回答道，“《人月神话》到目前为止，每年稳定的卖出 10000 本。”

嫉妒、羡慕和突然的现实：什么是我们拥有的象年金一样的东西。我高兴地说，自此书出版至今，我已读过 1975 版至少四遍；在接到 Brooks 电话后，我再次将它从书架上拿下来，重读了一遍。但这次，是有特殊目的的：实际的原因来自于他的电话，Brooks 教授告诉我，目的是发现自这本书 1975 年出版以来，计算机领域是否有有意义的事件发生。

我必须申明，这样的问题是相当难以回答的。Brooks 继续告诉我，他现在已基本离开软件工程社团，将他的大部分专业精力投身于虚拟现实领域的研究。所以，在准备再版这本书时，他想知道：什么已经改变了，什么还没有？他的原书中哪些是正确的，哪些是错误的，哪些是不切题的。

当然，我不是向他提供同样信息的唯一的人；我的一些同僚、大量的业界领袖、作者、顾问和摇旗呐喊者都被邀请回答这个问题…。我们所有人很高兴地做了。并且，象你所期望的一样，我们的输入被 Brooks 教授处理、分析、过滤、综合进那个非凡的新版本——它是真正的国际财富。

原书的内容仍在那儿，现在新增了四章。它们是 Brooks 教授对他的思想的反思和对我们的反馈的反应。新增的第一章由恰到好处地浓缩的原书主题组成；包括可以被称为书的中心论点的东西：大型编程项目忍受管理问题，这些问题因为劳力的分配而与小项目有本质上的差别；软件产品的概念完整性是如此关键；概念完整性是困难的但可达到的。第二章小结了二十年后 Brooks 对这些主题的观点。第三章是他 1986 年首发于 IEEE Software 的经典报告：《没有银弹》的重印。最后一章是对 Brooks1986 年的断言 “在未来十年，依然没有银弹”的思考。

年轻的软件工程师、吝啬的研究生、懒惰的软件老手常请我标示出当前为止最好的软件图书。“如果我带着仅有的一本计算机书在沙漠荒岛上，”他们问，“应该是哪本书？”这是个荒谬的问题，但人们坚持要个答案。假如你真的被放逐到这样的小岛（或者你决定躲藏到这样的地方去避免 2000 年软件崩溃的恐惧！），《人月神话》应该紧随着你。

Jason Bennett

背景

在我提交我的关于《人月神话》的书评前，我意识到没有介绍此书的第一版是我的疏忽。那是 1975 年。程序员辛苦的在最早的哑终端上工作，这些终端连着来自 IBM 的笨重主机。也许，如果足够幸运，程序员可以工作在一台小型计算机，它来自于业界的新星公司 DEC，和它们的 PDP 系列计算机上。FORTRAN 和 COBOL 是主要语言，夹带着一些 PL/1。C 是地平线上的亮点，刚刚开始离开 Bell 实验室走它自己的路。在整个工业范围中，汇编语言仍在广泛的应用。进入 Frederick Brooks 的书，工业界的原作之一。Brooks 在反思他在 IBM 的时间，特别是他在 1960 年代中期工作于 System/360 和它的操作系统，OS/360（独特的名字，不是吗？）的一段。Brooks 试图指出哪些做对了，哪些没有，特别是如何构造和管理全部程序。他因此开始书写软件项目管理的第一个条约，一个软件工程不可缺少的部分。二十五年后，我们仍然读这本书，学习它。在业界，大部分书六个月后就无用了，这本书则是空前的。记住，虽然，最终会有书能达到它的水平。

哪些是好的？

你可能很想知道，为什么这本书能持续如此长的时间。答案是简单的：书中的技术对大众的教训来说是次要的。简单说，Brooks 指出下列几点：

1. 编程必须进入到软件工程，以便持续改进技艺的状态；
2. 任何好的工程产品，必须是概念和体系结构的完整结合；
3. 软件开发中的焦油坑（the tar pit）可以通过尽责、专业的过程得以避免；

书中有如此之多的经典语录，以致这篇简单的书评不能全部包括。当然，最著名的是 Brooks 定律：加人将延迟工程（隐义：这样做没有帮助）。在 MMM（《人月神话》）中外科开发团队、第二系统效应、文档的重要性等均被覆盖，有些还是第一次出现。通过这本书，Brooks 覆盖了所有因素，这些因素是成功完成一个主要软件项目必须做的；同时，在书的各部分中，他给出了一致的软件工程与项目管理的坚实的基础。事实上，这是第一本主要的正确汇集工程师需要的关于大型软件项目开发知识的书，书中相关知识来自于对已完成项目的看法。除了原书，Brooks 1986 年的随笔《没有银弹》也被包括进去，同时带着 Brooks 在这本书出版二十五年后的思想。这些随笔每篇都值得一读，因为它们给出了当前工业所处位置的精确评价。可以这么说，容易的部分已在我们身后，让我们从这儿开始实际的工作。

哪些是有害的？

关于这本书哪些是有害的，答案是：人们没有读它。没有特别的好理由，他们仅仅是太忙于读最新的关于今天时尚的书了。当然，具有讽刺意味的是，今天的时尚可能是明年的笑话，而 MMM（《人月神话》）将从现在开始下一个十年。如果，学校发给你这本书，再次读它（你可能不是第一次:-)）。如果你从没听说过它，明天开始读。

书中哪些是对我们的有用的？

为什么我们每一人都应该读这本书？对于一串半组织化的小组哪些项目管理是要紧的？在我们为世界大同努力奋斗时，我们是否做的不好？嘿，是或不是。

首先，我们没有商业软件的压力。如果 Apache 明天发布而不是今天，没问题。每个人可能是坏的，所以我们不会做的更糟。因为如此，大部分开放源码项目，依赖一个或几个领导者，他们追踪着每件事和版本。我们没有许多拥有大量不同文档的项目。

其次，我们也依赖于低的期望值。没有人对我们有成功的预期，所以，对于这个运动，任何一个成功都是个胜利。现在，所有的眼睛都盯着我们。如果一个发布版延迟一个月发布，每个人都会说我们在保持时间表上做得不如微软。如果我们因为我们的可怜的设计而重写了一个应用，我们将不能提供有效的替代品。我们将不得不做

得比其他人更好以证明我们自己，并且我们不能做面向特定平台环境的产品。如果开放源码社区比工业界学习 MMM 学得更好，那么，我们将获胜。

再次，作为一个工业，我们必须提升我们软件的期望。我们要为精确的时间表努力奋斗。我们需要避免第二系统的影响。我们需要知道为什么你不能加人到已经延误的项目去加速开发。长话短说，我们需要仔细工作以及时产出合格的软件，而不是仅仅将原料放在那儿。

Francis Glassborow

我第一次接触这本书在近二十年前。作为一个学校老师，我毫不怀疑这本书应是我的天才计算机专家的小伙子们需要读的。也许我的成功不能表示它的价值，也许人们不知道它。

计算机图书常常在用旧前很长时间就已过时了，但这部书是一个值得注意的例外。作者描述了管理的失败导致延误了在恰当时候交付的问题，在今天——《人月神话》首次出版以来二十周年后，依旧在发生。加倍的工作力量在今天和他那时一样不能解决时间（表）的流逝。

如果，你已经拥有了老版本，你仍然会为带有额外的四章的新版感到高兴；如果你以前从未读过它，把其他事放在一边，立刻补上这重要的一课。象老版本对暴露关于延迟六、七十次的 IBM 感兴趣一样，这个版本增加了一些关于微软的注释（尽管如此，每晚重建开发项目的策略不能解释为何 Windows95 的 M8 beta 版的版本是 950 版）。

Chris Larson

自 Frederick P. Brooks, Jr. 出版经典著作《人月神话：软件工程随笔》已经 20 年了。在这部注重实效、明晰的书中，Brooks 剖析了许多工程管理的神话，这些神话来自他在年轻的软件工业中有意义的实践。典型的，他抨击了在项目中增加人手可以促进项目的完成的幻想。带着实例、幽默、严密的逻辑，Brooks 展示了这些神话实际上如何给软件项目带来灾难并导致延迟。

你不能用人力换取时间

他的思想恰到好处地应用于文档项目的管理。有多少次查看文档项目的经理思考通过增加人手缩短时间表？这是个简单的导致陷入的谬误，但 Brooks 清晰地表达了这样的思想—“人力可以与时间互换”的实际效果。

Brooks 说，“导致大量软件项目进入失控的原因是时间表的缺漏，这比所有其它因素的组合还多。”他给出了几个原因。首先，Brooks 责备可怜的估计技术，它错误的“搞乱了前进中的努力。”因为估计的不确定性，经理们缺少“礼貌的倔强”去支持那些看上去比其它需要更长的时间线的步骤。一旦时间（表）流逝，倾向于加入更多的人力，而这就像“火上浇油”，会导致一个新的灾难生成的循环。当然，一个基本的错误就是假定人力可以严格的和时间交换。Brooks 指出，这个公式仅在项目成员不需要和其他人交流的项目中成立，比如：拾棉花。

然而，一旦项目中包含连续任务和依赖关系时，可以交换的思想立刻就土崩瓦解了。“交流的努力，”Brooks 说，“必定导致加入大量要做的工作……。三个工作者要求的成对交互交流三倍于两个；四个六倍于两个。”这个交流的时间（不包括培训时间）吸收了许多增加人力分担任务带来的好处。

那么，答案是什么？

Brooks 认为由一个组织，其结构类似于外科团队的，由一些专家设计并完成全部项目的核心工作而由另外的人力在特定的途径上支持他们的努力，来执行产品开发可以治愈上述病症。

Brooks 说到，这条仅有的路是在一个项目早期完成“概念完整性”的通途。大部分关于这些完整性的重要的表述通过这个规范发生，“那是一本计算机手册加上性能规范……第一批文档中的一篇产生计划中的一个新产品，最后的文档完成它。”

文档是关键

Brooks 继续雄辩的支持文档，列出用户需要的每一项，以免只见树木不见森林：程序的目的、环境、选项、逻辑、“运行时间、”等等。并且认为：“大部分文档需要在程序编写前起草，因为它把基本的计划决策具体化了。”

作为技术交流者，我们需要听到来自工程方面的更多的各种各样的支持。尽管 Brooks 的环境——运行于大机上的大型编程项目——已经转换到我们今天的分布式计算环境，Brooks 的逻辑和清晰的思想依旧吹走了仍然威胁卷入项目管理中的迷雾。

Frank Chance

介绍

出版于 1975 年的《人月神话》是软件开发方面的经典作品。1995 年版包括了令人感兴趣的新的几章，但原来的随笔依然是这本书的心脏与灵魂。在这本书中，Brooks 解决了如何组织和管理大规模编程项目的问题。这些项目要求成百上千的程序员，产生几百万行代码（想想 SAP、Oracle 数据库引擎、Windows2000）。这部书由一系

列简明的随笔组成。在这篇评论中我将讨论开篇随笔——我的最爱之一。

焦油坑

Brooks 将大系统编程作比喻作史前的焦油坑来开始他的第一篇随笔："记忆中，我们看到恐龙、猛犸象、剑齿虎正在挣脱沥青的魔爪。挣扎得越剧烈，陷入的越深，没有哪只野兽足够强壮或熟练，它们最终都沉没了。大系统编程在过去的十年间就像焦油坑，许多大而强有力的野兽在其中已经惨烈地失败了。大部分已实现并在运行的系统，很少有达到目标、时间表和预算的。大和小、厚重和细实，一个接一个的团队卷入了沥青（陷阱）。没有什么事情似乎会导致这个困难——任何特殊的手掌都能被拉出来。但同时并相互作用的因数的相互聚集导致运动越来越慢。每个人似乎都惊讶于问题的难缠，难于面对它的本质。"

记住，这些话写于 1975 年。今天它们仍然可用吗？考虑一下 WindowsNT5.0。第一次计划于 1997 年发布，随后延迟到 1998 年早期，1998 年末，然后是 1999 年（为此它被重新命名为 Windows2000）。这儿是一些公开的估计：

- 5,000 程序员。
- 35,000,000 行代码。

显然，NT5.0 是个大系统编程项目。同样显而易见，Brooks 的焦油坑在今天同 1975 年一样普遍！

让我们继续 NT5.0 的例子。假设最糟糕的情况，全部 35,000,000 行代码都是新编的。有理由假设开发工作大致在 1994 年开始。所以我们有：

- 5,000 程序员 X 5 年 = 25,000 程序员年
- 35,000,000 行代码 / 25,000 程序员年 = 1,400 行 / 程序员年。

如果你是个程序员，或者你只接受过编程课程的教育，这个数字（1,400 行每年）似乎令人惊异的低。我们当中的大部分人都能在一两天内堆积出接近一千行的代码。什么使得 Microsoft 的程序员一整年才产出 1,400 行代码？

两种可能性跃入我们的脑海：

- Microsoft 雇用了 5,000 名不合格的程序员去开发 NT 5.0。

或者

- 写一个大规模的程序系统产品远难于堆砌出单一的程序。

Brooks 将讨论认为后一个答案是正确的。他由定义术语开始：

（1）程序

一个独立的程序是我们两天编程狂欢的结果。它是准备自己运行于我们编程的那台机器上的。如果我们加上文档、通用化代码、编写测试用例、使得代码可以由其他无关的编程人员来维护，我们就有了：

（2）程序产品

另外，如果我们接受我们的程序，并且完整地定义了它的接口使得它达到预定义的规范，并且测试了它和大量的其它组件的交互作用，我们就有：

（3）程序系统组件

并且如果我们都做了（加上文档、通用化代码、编写测试用例、使得代码可维护、定义了接口、测试了交互作用），我们就有：

（4）程序系统产品组件

Brooks 用手边的三倍规则说明在上述每个步骤中的工作要求：

（2）＝3 倍（1）的人力

（3）＝3 倍（1）的人力

（4）＝9 倍（1）的人力

或者，换句话说，开发一个独立的程序仅仅要求开发一个程序系统组件的 1 / 9 的人力。

回到 Microsoft 的例子，如果我们将这个 9 倍的因子乘以 1,400 行每程序员年的生产力测量，我们得到 12,600 行每程序员年（举例来说，假设我们掌握每一程序员，并且使得他们独立工作，堆砌在单一的程序上）。在一篇独立的随笔中，Brooks 引用一个发现这点的经理的话说，平均他的每个程序员仅能将他的一半时间用于开发——其它时间由文书工作、会议和各种其它任务所占据。把这些因素考虑到 Microsoft 的例子中，我们达到了 25,200 行每程序员年。那么，Microsoft 的程序员开始看来非常可敬。另一个测量自 1975 年来有了很小的改变，Brooks 引用的估计是 1,000 行每程序员年。如果上面引用的 1,400 行每程序员年是精确的，那么，它表现了在 1975 年到 1995 年 20 年间，生产力仅仅提升了 1.75%每年。这个结果证实了 Brooks 的另一个假定——程序员的生产力相对是个常量，它不受开发所用的语言的影响。因此，实际的生产力收获来自于迁移到高级语言编程，这些语言每行表达了更多的实际工作。尽管目标是大系统项目，Brooks 的解释常常被广泛的应用。例如，这个第一篇随笔用标有“手艺的快乐”和“手艺的悲哀”的小节来结束。在悲哀中，他讨论了荒废的问题：

“…这个人们已经工作了很长时间的产品，显然在完成前将被废弃。同事和竞争者已经在热烈地用新的和更好的主意反击。人们的孩童般想法的取代已经不仅仅在构思，而且付诸时间表。这一切总是似乎比它的实际更糟

糕。新的和更好的想法通常在完成之前不被应用；它仅仅被谈论。真老虎永远不能和纸老虎相比。”

小结

Brooks 的随笔涉及到了大系统编程所固有的多种挑战，但对任何投身于软件开发的人来说读这本书都是有用的。题名的随笔（《人月神话》）讨论了许多编程任务的不可分割性，和为什么增加人力到软件项目中无法产生效用。我的另一篇最爱是“贵族、民主和系统设计”（概念完整性的讨论）和“计划和投放之路”（在付运前多次交付的明确计划的益处）。一些问题已经因为技术的进步而废弃，例如关于如何在一个大型团队中分发写好的文档。

然而，你可能惊讶 Brooks 面对的许多问题今天如何阻止我们。另外的益处是 Brooks 简洁、清晰的作品读起来令人愉快。如果你是个程序员，如果你和程序员一起工作，如果你管理程序员，你应该阅读这本书。

TAL COHEN

人们说在计算机世界中每件事都在迅速的变化，然而，在二十多年间，一些重要的事情依旧没有改变。

如果某个工作可以由十个人在一个月内完成，人们说这个工作要求十个人月（man-months，也叫“person-months”）。简单算术表明，如果你分配二十个人去作同样的工作，它应在五个月内被完成。（译注：应该是半个月）我想这个逻辑在许多项目中大概都是正确的，否则，经济学家将不会那么令人喜爱。

然而，在软件设计的世界，这个承诺是个彻底的谬误。没有那条途径可以产生它。一个能由一个程序员在两个月内完成的程序，也许会花去两个程序员三个月的时间。早在 1975 年，当软件工程还是非常年轻的专业时，Frederick Brooks 敏锐地观察到人月概念仅仅是个神话。

在 70 年代，软件工程项目管理的问题背景是：大部分经理是受的经济领域的教育而不是计算机，并且他们熟悉的理论不能简单的用于软件项目。事实上，没有用于软件设计项目的相同的理论。因为直到今天，大部分软件项目仍不能按照时间表发布，所以我们有个大胆的暗示：这个问题象许多 Brooks 在书中谈到的那些问题一样，仍未解决。

在 20 周年纪念版的序言中，Brooks 写到：

> 让我高兴和惊讶的是，《人月神话》在 20 年后依旧流行。

实际上，这真令人羞愧。它指出了在二十年间，软件工业没有学到一些严重的教训，它仍在付着学费。软件工程依然被认为和其它工程专业相比是个很陌生的艺术。真的，我是第一次接纳软件编写中存在艺术的观点。它是一种美丽的、精巧的艺术，仅仅能被那些掌握同样艺术的人所欣赏。我相信，当一个真正的代码艺术家完工时，它比建筑更加迷人、比油画更加引人入胜、比音乐更加令人思潮澎湃。然而，一个建筑师不会让房屋设计的艺术

外观将他从房屋可以经受住最起码的地震的保证中转移出来。这个事实正渐渐的被大部分软件专家——那是些认为自己是艺术家而拒绝承认自己是工程师的人，所理解。（一个有趣的解决方案，是由我的好朋友提出的，他认为他自己是个“软件架构师”。）

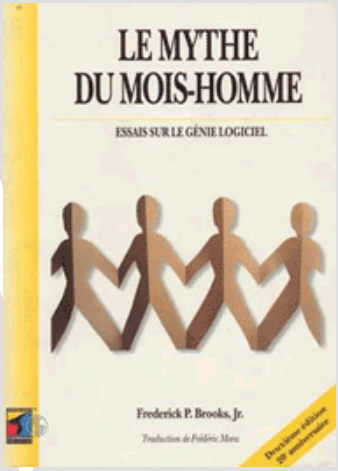
Brooks 的工作是简单的而且对于那些被认为在软件业务领域是个专家的人是必需的，尤其对于在这个领域从事管理工作的人。这部书不仅仅指出了问题，而且也提出了些有趣的解决方案（例如：“外科团队”）。它指出了这个领域的每个专业人士都应该知道的一些非常重要的缺陷（例如“第二系统”效应）。最后，它提供了许多重要的见解和案例研究。

书中大部分材料和今天依然相关，就象它在原书写出来的时候。尽管，众所周知，部分材料已经过期时。在 20 周年纪念版中，相对于原文——叫做“经典”——的修订，Brooks 聪明地决定加入修改的章节。那些用来讨论暴露于 90 年代灯光下的出现的问题。我个人认为，在读完前面的每一章后应读第 18 章中有关的部分。第 18 章的标题是“人月神话的主题：是或非？”，它包括了从 1 到 17 章的修改信息。

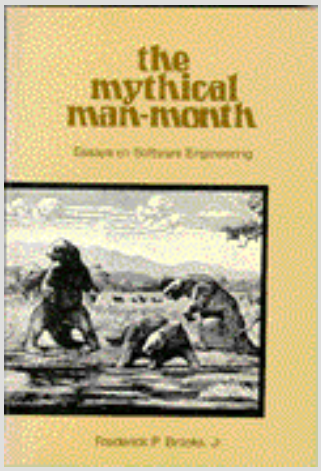
最后，这个新版本包括重印的 Brooks 的著名随笔，“没有银弹”，那是在都柏林 IFIP86 会议上的特邀论文，后来在 Computer 杂志上发表。在这篇文章中，Brooks 推测在他的文章出版（1986 年）后的十年，不会发现有技术可以大大增强软件开发过程。九年过去了，Brooks 悲哀地回顾：他还是正确的。

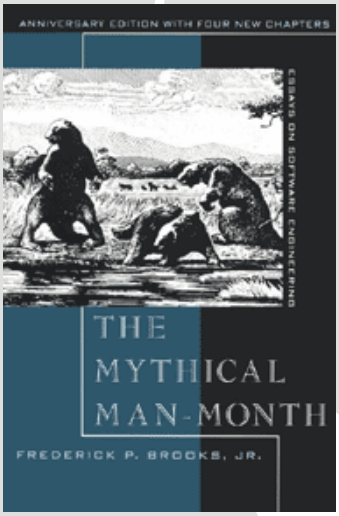
各种译本封面欣赏

(第二版中文译本)



(第二版法文译本)





(第二版)

(第二版日文译本)

软体尚方宝剑(Silver Bullet)何在?

—— Fred Brooks 和 Brad Cox 的不同观点

欧阳进(台湾) 查看评论

前言

二十年来，人们一直寻找解决软体危机的方法，包括结构化、人工智慧、物件导向等方法；但软体大师 Fred Brooks 在 1986 年发表的文章里预言在 10 年内找不到解决软体危机的尚方宝剑，历经了十年，果然不幸被他言中。1995 年里，Brooks 与 Cox 两位大师分别再深思软体尚方宝剑，Brooks 仍持怀疑态度，而 Cox 则相当乐观。

本文以时间的前后，依序交叉介绍两位大师的见解，期能引起读者对软体未来的些微兴趣。由于笔者的学养和功力远不如两位大师，只能介绍他们文章里的精华，并未添加笔者的阐释或见解以免您受到笔者有限能力的误导或局限您的思想和视野。希望您有空细读两位大师的原文著作，培养您自己的思维和创意。

介绍

二十年前(1975)，IBM 大型电脑之父——Fred Brooks 出版一本书："**The Mythical Man-Month**"。

收集了他在 1960 年代领导 1000 多人共同发展 OS/360 大型软体系统的心得和经验。从实际经验中，他体会到开发大型软体过程中，难以汇集参与人员的设计理念然后提供给使用者一致的设计概念(conceptual integrity)，因而导致软体的高度复杂性，使得大型软体系统往往会进度落后、成本暴涨及错误百出，就是所谓的软体危机(software crisis)。

经过了 10 年(1986)，Brooks 发表了一篇著名的论文——

"No Silver Bullet: Essence and Accidents of Software Engineering"

他断言：

「在 10 年内无法找到解决软体危机的尚方宝剑（银弹）」 (There will be no silver bullet within ten years)。

这文章激起许多软体专家的讨论与争辩，而"No Silver Bullet"也成为脍炙人口的名词。Brooks 认为软体专家所找到的各种方法皆舍本逐末，解决不了软体的根本困难——即概念性结构(conceptual structure)的复杂，无法达到概念的一致性，软体自然不亲切不好用！

到了 1990 年，曾首先提出"Software IC"名词的 OO 大师——Brad Cox 针对 Brooks 的观点而发表了一篇重要

文章——

"There Is a Silver Bullet"

说明他找到了尚方宝剑——即有些经济上的有利诱因会促使人类社会中的文化改变(culture change)，人们会乐于去制造类似硬体晶片(IC)般的软体组件(software component)，将组件内的复杂结构包装得完美，使得组件简单易用，由这些组件整合而成的大型软体，自然简单易用；软体危机于焉化解了。

在 1995 年初，Brooks 的上述名著第二版出炉了，书中含有一篇关于尚方宝剑的新文章—— "No Silver Bullet Refired"。文章里，Brooks 赞扬 Cox 的文章，但他认为 Cox 误解了他的本意。

尽管历经了十年，软体开发方法也有所进展，但 Brooks 仍然认为尚方宝剑仍未出现；大型软体系统的开发工作仍然困难重重，只能渐进地改善，不要奢望短期内会出现尚方宝剑，一举解决软体的困境。

在 1995 年底，Brad Cox 发表了新文章——"No Silver Bullet Reconsidered"。这文章里，Cox 更深刻探讨文化和思维变迁(paradigm shift)的话因与效果，从人类文化的观点阐释软体危机的原因，而得到乐观的结论——软体的尚方宝剑是可能的。

过去 10 年，Brooks 的预测是千真万确的，果真软体的困境不但未解决，反而更加严重。那么下个十年又会如何呢？有趣的是，大家来欣赏一下两位大师的深思灼见，然后再看看您个人的见解，也许会激励出许多新创见也说不定，不是吗？

1975

Brooks 在"The Mythical Man-Month"书中之看法

该书是论文集，其中有一篇文章叫**"The Mythical Man-Month"**，他就以此作为书名。在 1956~1965 之间，Brooks 实际领导 IBM 360 大型电脑的开发计划，包括硬体结构及庞大的 OS/360 作业系统在内，因之他具有「IBM 大型电脑之父」之尊称。由于 OS/360 是多达 1000 位程式师共同合作的大型软体开发工作，让他深刻了解到大型软体开发的技术和管理上所面临的种种困难和挑战。于是，他就将其领导开发 OS/360 软体系统的经验心得收集在这本书里。

人们常拿 Man-Month （多少人，做多少个月）来计算软体的工作量，但是 Brooks 发现软体的开发工作是需要人与人之间密切沟通的，使得设计工作不易分割，所以 Man-Month 为单位的计算方法是有问题的(mythical)。也就得出著名的 Brooks 法则——

「对于进度已落后的软体开发计划而言，若再增加人力，只会让其更加落后。」(Adding manpower to a late software project makes it later)

这是该书名称的涵义。

Brooks 从经验中深深感觉到参与人员之间的概念一致性(conceptual integrity)是软体设计中最重要的一环(conceptual integrity is the most important consideration in system design) 。一旦软体师们有一致的设计概念，其所共同创造出来的软体才有可能达到简单(simplicity)及好用(straightforward) 之效果。

为了达到这效果，Brooks 主张应注重软体的主架构(architecture)，并将它与实施细节(implementation)分开来。所谓主架构，就是：对使用者介面之完整而详细之叙述(the complete and detailed specification of the user interface) 。主架构汇集各软体师的不同思绪，融合成为一致的概念，然后呈现给使用者，让使用者感觉软体是源自于单一的设计理念(single philosophy) ，而不是基于庞杂无序的众多思绪。将主架构与施行细节分开来，是让大型软体计划获得一致性概念的有力途径(separation of architectureal effort from implementation is a very powerful way of getting conceptual integration on very large projects)。

1987

Brooks 在"No Silver Bullet"文章中的看法

自从"The Mythical Man-Month"一书上市以来，历经了十年，尽管软体专家们努力研究新方法来解决软体的困难；但是 Brooks 认为这些方法，包括高阶语言、OOP 、AI 等皆舍本逐末，只解决了一些概念的表达(representation)技巧而已，并无法解决根本性的概念结构(conceptual structure)问题。在"**No Silver Bullet**"文中，他藉亚里斯多德(Aristotle)的用词而将软体之困难分为两种：

●Essence ——此为概念上(conceptual)的根本困难。

●Accidents ——此为将概念转换为电脑表示法时，在表达(representation)上之困难。

例如，OOP 上的抽象资料型态（即类别）观念，只是改进了设计的表示方式而已，所以抽象时只去除掉表达性(accidents)的复杂和困难，但并未去除掉任何根本性的复杂。

由于没办法解决这种根本性的困难，使得原本单纯可爱的软体，逐渐演变为进度落后、成本暴涨、错误丛生等，像恶梦中的狼群般蜂踊而至，于是哀号而希望有种**「银弹」（silver bullet 又译为尚方宝剑）**能即刻平息牠们。然而 Brooks 认为：

「不仅眼前找不到尚方宝剑，由于软体的本质使然，未来也不太可能找得到。」 (Not only are there no silver bullets now in view, the very nature of software makes it unlikely that there will be any)

Brooks 列出的根本性困难包括：

- **复杂性**(complexity)——「复杂」是软体的根本特性，可能来自于程式师之间的沟通不良，而产生结构错误或时间延误；也可能因为人们无法完全掌握程式的各种可能状态；也可能来自新增功能时而引发的副作用等等。
- **一致性**(conformity)——大型软体开发中，各小系统之介面常会不一致，而且易于因时间和环境的演变而更加不一致。
- **易变性**(changability) ——软体的所处环境常是由人群、法律、硬体设备及应用领域等各因素融合而成的文化环境，这些因素皆会快速变化。
- **不可见性**(invisibility)——软体是看不见的，既使利用图示方法，也无法充分表现其结构，使得人们心智上的沟通面临极大的困难。

这些是软体的本性，Brooks 认为没有捷径可解决之(there is no royal road)。但可以渐进式地改善之。他认为可行的方案有：

* **买来装配**(buy and build) ——软体的建造尽量使用现有零组件，不要一切都从头做起。

* **快速雏型**(rapid prototyping) ——使用重复式的开发方法(iterative development) 来渐进地改善软体的雏型，裨逐步厘清使用者需求。

* **有机成长**(growing organically) ——有生命的东西皆是由小慢慢长大的，建造大型软体的方法应该是逐渐成长，而不是一次建造完成。

* **优秀设计者**(great designer) ——人是软体设计的核心，良好的方法可改善人的创造过程，但无法激励人的原本创造力，须藉重有创意的人。

1990

Cox 在"There Is a Silver Bullet"文章中的看法

> And in "No Silver Bullet: Essence and Accidents of Software Engineering," he(Brooks) argues that the difficulties are inevitable, arising from software's inescapable essence--not from accident, but from some deficiency in how programmers build software today.
> (在"NSB" 文中，Brooks 声称这些软体的困难是无法避免的，是源自软体上无法逃避的根本质——亦即不是来自一些旁技细节，而是今日程式师设计软体时缺点)

接着，Cox 针对 Brooks 所提的软体困境所在，Cox 从新的观点来阐释之，而得到乐观的结论。

Cox 认为软体危机是必然会面临的(irresistible)的困境，但并非是无法克服(immovable)的。当全球经济进入资讯时代，市场对资讯殷切需求而产生的经济诱因(economic incentive)，将促成人们社会中的文化改变(cultural change) 。加上目前遭遇的软体困境：成本过高、品质不良、以及人们无力去改善它们。使得人们对软体的看法和思维产生巨大的变迁(paradigm shift)，从过去一行一行撰写指令的土法炼钢方法转而尝试去建立可重复使用的标准组件(standard component)以及组件交易市场(market)。在这种新环境中，软体的价值体系(value system)、权力结构(power structure) 、以及软体师与消费者关系重新定位；彻底从过去重视程式的撰写过程(process) 转而重视组件价值、拥有、及买卖系统等基本文化要素、而导致软体产业的革命(software industry revolution)。

软体组件类似于硬体的晶片(IC)一般，把组件的复杂封装起来，对使用者而言，调组件再也不复杂了。因之，建立软体晶片市场，让人们不再受困于组件内程式撰写的复杂，而专注在组件本身的使用、价值和买卖上，软体的复杂和困难就自然烟消云散了。

因之，文化和思维的变迁将是一把尚方宝剑，能克服软体的困境；而且这种变迁是即将来临的，并非海市蜃楼！

1995

Brooks 在"No Silver Bullet" Refired 文章里的看法

自从 1987 年 Brooks 发表**"No Silver Bullet"**一文之后，激发了许多人对到底有没有尚方宝剑的讨论与争辩，例如前述的 Cox 之"There Is a Silver Bullet"文章，以及 Harel 的"Biting the silver bullet"文章等等。

Brooks 在理解各方的反应，以及检验当今各种软体设计方法之后，他仍认为尚方宝剑尚未出现(the

magic solutions are not just around the corner)。

Brooks 赞扬 Cox 的"There Is a Silver Bullet"文章是一篇极佳的文章，可是他认为 Cox 误会了他的本意，误解之处有二：

① 如上节所述，Cox 在文章中写道：

"...but from some deficiency in how programmers build software today."

Cox 把根本性困难解释为软体建造方法上的缺失，事实上 Brooks 所谓的根本性困难是指软体上概念性的复杂所导致的，无论使用什么方法，无论在任何时刻，皆必然呈现的软体固有特性。

② 如上节所述，Cox 在文章中写道：

"...Brooks argues that the difficulties are inevitable..."

事实上，Brooks 认为他自己只是持怀疑之态度，并非认为尚方宝剑是完全没希望的。例如，Brooks 认为软体系统的复杂是可分层次(level) 的，若软体下层的小组件(object)组成上层的大组件，再由大组件组成更大之组件，层次分明地组织起来，则复杂度就可以减低了。还有，当软体由小而大渐进地有机成长时，虽然复杂有系统地增加，但可确保软体的正确性。导致这种困难的原因仍可藉渐进式地改善或去除，只是没有捷径罢了。

Brooks 认为 OO 是个有希望的途径，但由于牵涉到人们思维方式的转换，使得企业组织必须先投下资金(front-loaded cost) ，才能逐渐回收其利益(down-stream benefits)，因而 OO 的进展会是慢如蜗牛。既使它可解决软体危机，也不会是在眼前。

1995 年底

Cox 在"No Silver Bullet" Reconsidered 文章中的看法

Cox 认为 Brooks 所谓的根本(essential)困难——complexity、conformity、changability 及 invisibility——并非最根本，而是由一个隐藏在深处的病因(cause) 所造成的症状(symptoms)罢了。这个病因就是：

> 目前文化中缺乏有利的诱因来鼓励企业公司生产、买卖软体的组件(component)以及更少的子子孙孙组件(subcomponent)。

如果买卖各层组件者皆有利可图，自然会想尽办法将组件封装得简单易用，则 Brooks 所指的软体困难自然消失了。

Cox 觉得 Brooks 的观点是以技术为核心(technocentric) ，而他的观点则是以人为核心(human-centric) 。由于观点的差异，对软体危机的阐释自然会获得不同的结论。他的结论是：尚方宝剑是可能的，只是这把剑是软体思维的变迁(paradigm shift)，而不是技术(technology)。

想一想，造铅笔的公司卖掉了一支铅笔，就少了一支铅笔，亦即少了这铅笔的各小组件，如铅笔擦、笔心等。为了要再造另一支铅笔，就得付费去购买铅笔擦和笔心，使得制造笔心等小组件之企业公司也有利可图。因之，铅笔到铅笔擦、笔心等制造者皆会用心去把组件包装好，使其简单易用！

但是，在软体中，制造试算表软体的公司，卖出一套试算表软体，只是个拷贝动作，并不会少掉一个位元；不必再向小组件的制造者购买，使得小组件制造者无利可分，自然无法进行生产、包装和买卖了；这是目前软体业的困境。

在未来新文化之中，使用者(end user)每使用一次某物件(object)，就必须付一次费用给这物件的生产者，而这生产者则必须付一次费用给其内部各次物件(subobject) 的生产者，依此类推到最小的组件为止。在此环境中，各层次物件之复杂结构皆封装得完义无缺，使得各层物件皆简单好用，软体的复杂与困难就迎刃而解了。

结论

在前言中已提到，本文只是介绍 Brooks 与 Cox 的文章精华，笔者因能力所限而不做推论，当然不会有结论，盼您在细读过这两位大师的原文后，才做出您自己的结论。如果您愿意，也盼您有所结论时，能与笔者分享，自当感激不尽！再会了。

[参考资料]

[1] **Brooks, Frederick.** *The Mythical Man-Month.* Reading, MA:Addison-Wesley, 1975.

[2] **Brooks, Frederick.** "No Wilver Bullet:Essence and Accidents of Software Engineering," IEEE Computer(April 1987"), PP.10-19.

[3] **Brooks, Frederick.** *The Mythical Man-Month*, the 20th anniversary edition. Reading, MA:Addison-Wesley, 1995.

[4] **Cox, Brad.** *Object-Oriented Programming:An Evolutionary Approach, Reading,MA:Addison-Wesley.*

[5] **Cox, Brad.** "There Is a Silver Bullet." *Byte*(October 1990), PP. 209-218.

[6] **Cox, Brad.** Superdistribution "Objects as Property on the Electonic Frontier. Reading, MA:Addison-Wesley, 1995.

[7] **Cox, Brad.** "'No Silver Bullet'" Reconsidered," *American Programmer* (Novermber 1995), PP.2-8.

[8] **Harel, D.** "Biting the silver bullet:Toward a brighter future for system development," *IEEE Computer*(January 1992), PP.8-20.

Fred Brooks

Fred Brooks 在 1956-1965 之间领导 IBM System/360 大型电脑的开发计划，建立了 IBM 在商业电脑市场上的盟主地位。目前任教于 University of North Carolina at Chapel Hill.

1975 年他出版的**"The Mythical Man-Month"**一书，是软体设计方面的经典名著。历经 20 年，该书第二版终于在 1995 年上市，一出炉就发行 250000 册。1986 年，他发表了重要文章——"No Silver Bullet"，激起众多人士对软体危机与奇迹的好奇与争论，一直持续到今天。

Brad Cox。

Brad Cox 是著名的 OO 大师，他的重要著作——

"Object-Oriented Programming: An Evolutionary Approach"

家喻户晓的「软体 IC」一词就是来自他的著作。他是 Objective-C 语言的创造人；目前任教于 Geoge Mason 大学。他最近出版一本新书－－

"Superdistribution: Objects as Property on the Electonic Frontier"

探讨在朝向全球性资讯密集产业过程中所面临的各种问题和困难。Cox 相信化解软体危机的尚方宝剑是文化的改变，即人类思维模式的变迁(paradigm shift)，而不是技术(technology)。

本文原载《物件导向杂志》

UMLChina

非程序员

软件工程杂志，已有数万用户

UMLChina讨论组

已有数万名成员

UMLChina.com

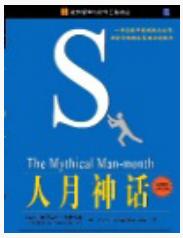
1999年11月成立，，专注UML技术的应用

专家交流

提供与世界级专家交流的窗口

《人月神话》出版后的网友评论

摘编自：chinapub 和 cnforyou



原书名	The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)
原出版社	Addison Wesley
作者	（美）Frederick P. Brooks, Jr.
译者	UMLChina 翻译组汪颖
书号	7-302-05932-2
页码	370
开本	32 开
定价	￥29.80
折扣价	￥23.84
出版日期	2002-11-1

gcc_0000：

个人看法：作者有预见性，抓住软件的本质！ 5-star

但是在当前软件工程讨论中急功近利的气氛下，不久就会受到攻击(大规模的)。

本书阐述作者的观点(当时是 1975 年)。这些观点在之后的软件工程学术发展中，不断的被众多学者引用和发扬。所以从大量的流行的软件工程书籍中，大家已经详尽的了解到大多的观点，并且有详细的具体实施方法 (恰巧是本书所未列出的)。就如同本书最后一章中讲的，一个很有趣的场面：作者在飞机旅途中，看到邻座正在阅读此书，但直到下飞机，他仍然没有做出作者期望的惊叹的表现。在作者不满足的追问他读后感想时，他说：“书里面所说的观点我都知道!”。作者便放弃了介绍自己的想法！(这可能减少某些人有意攻击本书观点的欲望，另一方面也是作者本书的意图)

中文版：编排者有意维持作者/原出版商的设计思路，但只学到其表面。(恰巧如同本书的意图的另一面)

太厚，〈〈生存手册〉〉500 页比他还薄 1/5。

装订有问题：每页空白留足，无可厚非，但相对的字体太小，过于靠近装订线处，加上书过厚，读起来甚累!! (软玉生香, 从形体上变了)

小字体推荐〈〈Software Architecture--Perspectives on an Emerging Discipline〉〉, 250 页，只有它 1/4 厚度--小巧玲珑！

zglcl008：

当你负责过几个项目的开发后在评论本书是否有用吧，别作无畏之争了。参加过大的项目的开发，无论你是程序员还是项目、计划、测试、进度、质量的管理人员，如果你对书中的有些观点还是无动于终，那你还是缺乏总结与思考的。无论对与错，适用不适用要多想想借鉴而已。

Jiseng：

书不错，就是 32 开太小，不太习惯，而且俺上班看的时候被领导 K 了一顿，说俺上班时间看小说，:'(没办法，难怪领导会搞错，哪有科技书起个这名字的..."XX 神话",书还没读完，读完后给大家汇报情况...

swachian：

不错的散文书，译的也很有可读性。

第一次就印 2 万册，可见其面向读者面之广了。

mah-jong：

平心而论，书是好书。但如果指望从这本书里找出解决开发中面临的软件工程问题似乎不太可能。这本书的意义从我的角度来看有两个，第一个是作者用通俗易懂的笔法写出了一些常识。第二个是没有银弹那篇文章的观点，就是解决开发过程中问题的万能药是不存在的。我们都知道，很多软件从业人员对自己的工作有着类似宗教的感情，有时候这种感情会导致管理者和开发者产生“人有多大胆，地有多大产”的错觉，从而忽略了常识。这对于工程来说是一件很不好的事情。这本书和其他几本类似的书（比如死亡之旅）类似于解毒药，可以让我们比较冷静的去看待软件开发。

书是好书，虽然我已经看过其中的大多数文章，但还是打算买一本中文本来看看，当作是小说读读也未尝不可。我猜作者的写作目的也不是宣传什么万能药，而是期望告诉读者，一些做法解决不了问题，而另一些做法会让事情变得更糟。

xdsmile:

昨天 china-pub 送来的，一口气读了快一半了。同我以往见过的任何软件工程与开发管理的书籍不同，这本书几乎没有任何软件工程上的规范讨论之类，所有的观点都是作者实践中总结出来的，而且也不套用软件工程的原则，而是自己起名，自己定义其含意，行文幽默；最重要的是作者并不强调应该干什么，而是提出问题，提出解决方案，提出适用范围，提出其方法的局限性，指出在他的实践中是怎么用的，产生了什么后果，并总结其教训。类似此类的观点在本书中比比皆是，有很多发人深醒的论点。20 年前的这本书称其为经典的确名副其实，看完这本书，我想就不难理解为什么美国的信息产业有如此巨大的实力了。

另外，不得不说的是必须得感谢译者，从译文的字里行间可以看出其中的辛苦，看出译者所花费的心血和努

力，尤其是每章的标题，翻译的尤其贴近中国文化的内涵。

还有，呵呵，就是其低廉的价格，在这个名字就是金钱的年代，如此的好书才不到 30 大洋，真是令大家跌破眼镜。较之某些××××就上百的书来讲，其中的性价比可想而知，感谢为这本书出版而努力的人们。

play:

hehe 有道理。

我喜欢看老外的书，因为他们组织书的内容的结构本身，以及行文过程中体现出来的是个人的思维。看的是他们的思维过程和思路，而不是得出什么什么结论。

因为，谁也无法说什么样的结论就是好的，什么样的结论就一定不正确。所以，国内很多书我一般只是拿来当作手册，而国外有些书，虽然他们写的内容和我们的环境可能有很大差异，但从他们的思维过程和发展过程本身，有很多东西值得学习，和琢磨。

望国内也多出一些值得推敲，甚至能引起大家争论的好书。共勉！

xdsmile:

看来这本书引发了大家的许多争论，其实对于书籍是仁者见仁，智者见智的；有个小故事，在英国的一列火车上，有个人正在看书，旁边一位老者看他那么专心，好奇的问他在看什么书？那人说在看霍金的《时间简史》，并说自己很久没看到这么有趣的书了，那个老者听了说：“哦，是那本书呀，我看了几遍都看不懂”，那人很奇怪，说：“不会吧？我觉的很容易懂呀，没什么难的，我是个商人，很少看书的，都能看懂”，老者笑了笑，没再说话。后来才知道，那位老者是苏联科学院的资深院士。

从这个小故事，我们其实就可以看出，一本书对于任何人，其体会是不同的。

比如，我在书店看了《计算机程序设计艺术》，觉的太枯燥了，根本不想看，可我知道那是套好书，它并不会因为我看不懂就没有其价值。

imhexi:

看了前面的几节样章，一直期待出版。哈，终于出来了，上来买先！

感觉很不错，比较实用，还有很难得的一点就是行文通俗，在软件工程的书籍里面比较罕见。

ricky:

我买了一本，请大家告诉我哪一块写得好可以吗，我怎么没有看出来

mah-jong:

那些内容写得好是见仁见智的东西。我个人比较喜欢巴比伦塔、另一面以及没有银弹这三部分。刚才出外面办事，顺便到书店里面买了一本中文本，九折，价格还不错，呵呵。回来大概翻了翻，译的还可以，不过原书中那种简洁而不乏味的语言风格没有表达出来，可能译者可能过分强调和原文符合，所以表述上有些啰嗦。比如："我们可以通过遵循优秀的而不是拙劣的实践，来得到良好的设计"（pp239）。"而不是拙劣的"这个词和"，"完全没有必要保留。这本书属于散文性质，不是技术手册，只要不翻译错，怎么让人读起来顺畅就怎么翻好了。

easyso:

我现在看到第 7 章——为什么巴比伦塔会失败？

下面我补充一些材料，帮助各位，特别是非基督徒理解《圣经・旧约・创世纪》11 章 1-8 节。有一点说明：一般现在都称 The Tower of Babel 为巴别塔而不是巴比伦塔。

Part 1of 2

CUV-和合本，Chinese Union Version

1 那时，天下人的口音，言语，都是一样。 2 他们往东边迁移的时候，在示拿地遇见一片平原，就住在那里。3 他们彼此商量说，来吧，我们要作砖，把砖烧透了。他们就拿砖当石头，又拿石漆当灰泥。 4 他们说，来吧，我们要建造一座城和一座塔，塔顶通天，为要传扬我们的名，免得我们分散在全地上。5 耶和华降临，要看看世人所建造的城和塔。 6 耶和华说，看哪，他们成为一样的人民，都是一样的言语，如今既作起这事来，以后他们所要作的事就没有不成就的了。 7 我们下去，在那里变乱他们的口音，使他们的言语彼此不通。 8 于是，耶和华使他们从那里分散在全地上。他们就停工，不造那城了。

TCV-现代中文译本，Today's Chinese Version

巴别塔

1 起初天下的人只有一种语言，使用一种话。 2 他们在东方一带流浪的时候来到巴比伦平原，在那里定居。 3

他们彼此商量：“来吧！我们来做砖头，把砖头烧硬。”于是他们用砖头来建造，又用柏油砌砖。 4 他们说：“来吧！我们来建造一座城，城里要有塔，高入云霄，好来显扬我们自己的名，免得我们被分散到世界各地。” 5 于是，上主下来，要看看这群人建造的城和塔。 6 他说：“他们是同一个民族，讲同一种话；但这只是一个开始，以后他们可以为所欲为了。 7 来吧！我们下去搅乱他们的语言，使他们彼此无法沟通。” 8 于是上主把他们分散到全世界，他们就停止造城的工程。

gigix:

终于看到了第一个有价值的批评。在我看过的圣经版本上写的是“巴别塔”，所以看到 Adams 用了“巴比伦塔”这个词的时候，我担心他会被人批评。只是没想到，这位真正的批评者来得这么迟，让另一些人跳梁了这么久。

（另：巴别通天塔建造的地方是幼发拉底河与底格里斯河交汇处的一个小村庄，名叫 Shinar。我很喜欢这个村庄的名字。）

读者当然有批评的权利，你尽可以使用这种权利。不过，到目前为止，对于人月的批评，只有 easyso 的批评是打到了点子上，其他的批评者都只是在那边自说自话地叫喊“不好”、“不好”而已。

对于这本书，有很多人说好，并且说出了它好在哪里。而 easyso 之前那些所谓的“批评者”呢？你们的观点在哪里？你们的论据在哪里？你们知道这本书不好在哪里吗？对于这样一本经典著作，我早就想看看会有怎样的批评，可是这些“批评者”们太让我失望了。

adams_wang:

关于“20 多年前的书，现在有些地方看得不是很懂，毕竟我对 20 多年前的技术及软硬件状况不了解”，我想解释一下，书中提到的东西，并不是只存在于 20 年前才有的硬件开发中。在现在单片机、通讯、嵌入式等开发中，情况基本一致。另外，根据个人经验，该书中的观点同样适用于现在的软件项目开发中。

而谈到软件行业的基本技术，如操作系统理论、数据结构、编程语言、数据库理论基本成型，相应的实践也由先驱们前扑后继，积累了大量的经验。另外，对一般的软件从业人员而言，项目中面对的是社会学的问题，而不仅仅是技术问题。真正的技术突破由那些科学家等人已经完成，甚至推向市场。我们只是应用他们的成果，从这个角度讲，大多数应用软件开发人员只是沉浸在“高科技”的幻想中，所以，软件项目管理中更多的是人的管理，而非一般的技术“突破”问题。如果这方面仍然有理解的难度，个人有些遗憾。

很荣幸能同您就书中观点进行探讨。

easyso:

翻译得不错了，如果能多了解一点圣经和基督教的指示，就可以翻译得炉火纯青了。

ququxiao:

我赔！高明的译者多了，有几个象你这样急功近利的，还要开研讨会，更想上新闻联播吧，可惜上不去！

lzj310:

我作为项目经理、部门经理或技术负责人有两年时间，一直为书中所提出的问题困扰。感谢你翻译的这本书，她解决了我部分的问题。另外我了解到该如何思考技术管理中出现的问题。

cuixian:

书太厚，字太靠中缝，阅读不方便。

mah-jong:

再谈《人月神话》的翻译

今天没事，用东北的歇后语来说就是“下雨天打孩子，闲着也是闲着”。再来谈谈《人月神话》的翻译。这几天这里有关人月神话译本的争论我都看了，和一般的论坛相比，这里有着一种难得的幽默感。正方观点说嫌翻译的不好你自己来呀，还有巴比伦塔的翻译是致命的问题，其他问题是没有的。反方的观点是翻译的不好就不该翻译名著，译者和出版商拿着这么个译本就召开讨论会有点恬不知耻。从我这种普通读者眼里来看，无论是正方还是反方都是些极有幽默感的人，实在是佩服得不得了。先来谈谈正方的观点。

正方的观点一类似于食客挑剔厨师的菜，厨师一怒之下说你来炒好了。我猜这世上绝大多数饭馆都不会做这种事情，这是砸自己的饭碗嘛。同样的，作为读者付了钱买书，觉得翻译的好就叫声好，觉得有问题就说说。当然，如果不喜欢，译者可以让读者闭嘴。读者也可以不谈，但从长远来看，这种做法是在砸出版社的招牌、汪先生的招牌和 UMLChina 的招牌。当然，如果诸位说我就干的是这一锤子买卖，以后怎么样我不管，那我也无话可说。翻译和出版社出书是为了赚钱，但对于读者来说这是件好事情，语言文化的差异、国内的外汇管制制度和收入偏低，让大多数读者不能很幸运的去买自己想看的原版书，而有人来做这种经典书籍的翻译工作，出版社和翻译者得到了收入、读者得到了想看的书，无论对那一方都是有利的事情。在可以预见的时间里，这种事情还是比较好的。利人利己的事情为什么不好好做呢？

正方的观点二是认为巴比伦塔这类翻译才是问题，其他问题不是问题。那我就想知道了，我前面举的那个句子您认为很流畅吗？这种句子在这本书里比比皆是，再举一例，又是随手翻到的（pp62）,“面对估算过高的难题，结构师有两个选择：削减设计或者建议成本更低的实现方法来挑战估算的结果”。当然，这些句子不能说是错译，还是表达出了作者的意思，但您觉得这种翻译符合中国人的语言习惯吗？容易理解吗？当然，兄弟资质愚钝，可能对这种深奥的东西理解不了。但我读原文的时候可没有这种感觉，作者的写作让我这种 E 文很烂的人都可以读得明白晓畅。还有前面说的那个“可爱”的问题，我怎么就没看出来有什么可爱的地方呢？所以，我说译者没有把作者原文的那种清晰简洁的风格表现出来。当然，如果正方认为这种译文没有什么问题，我只能怪我自己，中学怎么没有好好上语文课呢？对这么简单的句子都认为不够清晰，对这么“可爱”的事情都没看出可爱来。

现在再来谈谈反方的观点。我觉得用不着无事生非。首先出版书是要赚钱的，总得造造声势。这没什么可指责的，而且这本书确实是好书，多做做宣传也是件好事嘛。再有就是翻译资格的问题，这不是什么大问题，经典也是人写的（人是人他妈生的，妖是妖他妈生的，呵呵），而且这本书翻译的还可以，比起大量的句子都没翻译通的书还是好多了。再说了，这个译本老兄不喜欢可是等更好的译本出来再买嘛。

最后再来谈谈我自己的观点，如果用 10 分制来评价，这本书的选题还是非常好的，可以给 10 分。翻译的还可以，可以给 7 分。版面设计可以给 9 分，和原书的风格很接近，可惜封面差了点，所以扣了 1 分（里面的插图虽然不太清晰，但考虑到这个价格，也就不必扣分了）。期望译者的下一本书可以翻译的更好些（如果汪先生还做这件事情的话）。

就这么多，顺便补充一句，我现在有点怀疑正方和反方都是出版商的托，为了搞出一个热烈的气氛，让我这种傻不机机的普通读者关注这本书并掏腰包。我觉得没有必要这么做吧，想买这本书的人，肯定会买的，再说汪先生翻的还可以。

xtall:

这才是有价值的批评。

mah-jong,你不会也是托吧？让我这种傻不机机的普通读者关注这本书并掏腰包。

mah-jong:

我倒是挺想做托的，可惜还没有这个机会，呵呵。我愿意谈这本书倒不是因为我付钱了，主要是因为这本书的风格一直是我欣赏的，还有什么麦克康吉尔写的软件工程不是科学丫，Yourdon 的死亡之旅之类。这些读物对于个人的具体工作没有太强的指导作用，但从一般意义上来读非常开阔思路，还是很有帮助的。

对于译著通常我没有太高的要求，反正我的英文应付技术书还不成问题，而且也有自己找原版书的渠道。买些译著来看也不是想给人挑毛病，而是觉得母语的东西亲切些。翻译的好，我就放在手边，翻译得不好，就翻翻丢到阴山背后。

fhlix:

排版太 e 了！

明明可以印成小册子，随身携带着看，却搞得臃肿不堪，有凑页数的嫌疑。不知道什么时候有影印版。

easyso:

巴比伦和巴别的区别可是比较大的，查金山词霸关于 Babylon 和 Babel 的解释，主要有：

Babylon: 巴比伦式的城市指奢华糜烂的城市或地区，经常是罪恶和腐败的；

巴比伦, 奢华淫靡的城市, 任何大的富庶的或罪恶的城市。

巴比伦古巴比伦王国的首都，位于幼发拉底河沿岸的美索不达米亚境内，作为首都大约建于 公元前 1750 年，在它被亚述人毁灭后（ 公元前 689 年），它又被尼布甲尼撒二世重建于王室显赫之时，巴比伦是世界七大奇观之一的空中花园所在地。

总结，Babylon 的含义是关于罪恶，与 Sodom、Gomorrah 是比较类似的词；

而 Babel，巴别在《旧约全书》中希纳的一个城市（现在被认为是巴比伦），当建筑者们不能理解彼此之间的语言时，通天塔建筑被迫中断了；

【圣】(古巴比伦人建筑未成的)通天塔(上帝因他们狂妄, 责罚他们各操不同的语言, 彼此不相了解, 结果该塔无法完成。见旧约创世纪。)

[babel][喻]摩天楼; 难以实现的计划

[babel](意见等)不一致; 杂乱声

n. 混乱，嘈杂

n.空想的计划

总结：Babel 有难以实现、混乱、嘈杂之意。

mah-jong:

大哥，你可真幽默。当然，这个词翻译确实有点小问题，但这不妨碍理解作者的意思。作者和译者把圣经的上下文都放在那里了，我想，任何人只要读了，都可以理解作者的意思，而作为译者，能够表达出作者的意思也就够了。译者犯的这个错误其实无伤大雅（我倒不认为这是错误，只是编辑的疏漏罢了），再说也不是没有巴比伦塔的这种译法，老兄这可是鸡蛋里面挑骨头。如果不信，可以用 google 搜索“巴比伦塔”和“圣经”这两个词。

至于给译者挑毛病，只是希望译者能够做得更好些。这本书的翻译，我只能说没有明显的错译。看得出来，译者是下了功夫的，我随手翻到的几页至少没有不通的地方，但只限于此。译者的中文表述上需要继续加工，比如我前面举的那个例子，我再顺手举个例子，是随手翻到一页（未加选择的翻，pp77），“对软件系统的体系结构师而言，存在一种更加可爱的方法来传播和推行定义”。“可爱”？！确实挺可爱的，呵呵。

出版社把书稿交给译者，译者再转包给学校里面的学生，这是公开的秘密，也无所谓对错。可是读者总能要求出版社和接书稿的人细心一点吧。

cat_hangzhou:

很遗憾还不能看到书。看了样章...感觉不错。

其实对一个真正的程序员来说，翻译...至少我感觉不是太重要...能让我看懂就可以了。也许译者不是很专业，因此一些概念没有用术语表示...但是名词无非是一种指代，偶看这种书的时候，就当做宏处理掉了：）

就象骂名已久的 TIJ 第一版，呵呵，开始看有点累,但是明白了译者的习惯了以后，也能看下去啊。

当然了，侯老师译本更好，偶是当看小说一样看完的...呵呵，侯老师不要生气：）

其实老外的书...和国内的思路不同。对他们来说，他们很愿意将自己的想法用一种简单的方式表述...就想曼昆的经济学。

功力不到，反而在怪老师讲得不好...有一点点的过分。

译者很辛苦，因为西方有很多成语、典故，不见得所有的人都知道...就象巴比伦塔。好的翻译，很可能就会带上厚厚的注释（尤利西斯）。但是不能这样要求每个翻译。也许对他来讲，他认为这些是人所共知的。

就翻译来说，我的观点：第一要准确，不要扭曲了作者的意思；第二就是要明白，不要出现很拗口的话。呵呵，《设计模式》这本书，我打开来很多次，但是每次都看不下去...看不懂...我不敢说译者的水平如何，只是觉得

自己的火候还不到。

读书...很多时候是一种相互交流的过程...我已经在考虑这方面的问题，恰好有这样一本书，可以帮我解惑。如果我对这一领域根本没有涉足，那么我想作者译者的水平再高，与我来讲也不过是对牛弹琴，而我自己的感觉是味同嚼蜡（就象我看生物技术的书一样）

人月神话，本来就不是一本纯技术的书。也不是管理的书。这本书只是一些经验的积累。我在带项目，感觉就是很累，有很多问题不知道怎么解决。很希望在这本书里能够得到灵感。是的，我只希望得到灵感，并不指望得到答案。

关于有人提到的 20 年的技术...我想不用争论什么。如果要争论，那么首先要争论的是什么是技术！做程序时间不久，水平也不高，但是我一直认为，对一个项目（工程）来说，重要的不是采用什么技术，而是思想！如果程序员还是停留在具体一段代码一个方法的争论中...那么也许你可能成为一个很资深的程序员，但是永远成为不了可以高屋建瓴的程序员/项目经理。

很感谢译者能把这本书带进国内...很希望能够看到有新鲜的思想。呵呵，至少是对我而言。

中国的教育是填鸭式的教育。很希望大家能不做养殖场的鸭子，而是做只野鸭。

我决不指望译者能按照我的口味来翻译，我会努力尝试去适应译者的口味。毕竟，是我需要看，而不是译者。

最后，如果能通过漫画、小说让我明白其中的道理、思想，我会很感激。但是我知道这个，基本上不可能。于是，退而求其次，至少，说明白问题。:）

呵呵，没看过全书，不好评价。仅从样章上看，译者还是有一点中国文化的功底的。打星...实在不好打，因为我怕样章是全书唯一翻译的好的:）但是就凭那一个样章，我认为全书也值三颗星了:）

yusc_ww:

没看书就有这么多废话，I 服了 u

cat_hangzhou:

呵呵，很不好意思...污了您老法眼。

书...我昨晚看掉了一半...说实话，看起来很顺，但是没觉得融会贯通。

另外，废话并不是针对这本书的，而是有感于不少无谓的争论。

嘿嘿，看了一半，感觉...废话至少不是错话：）

自己的...既然是废话，那么给个鸡蛋吧

wlcun:

在校大学生注意了，去你们的图书馆找一本《论大型软件的开发》，武汉大学出版，1.10 元，如果你找到，你会发现就是《人月神话》，新版本也只是“增加了一些新的想法和建议”（作者语），我发现以前出版了不少经典好书啊，版面设计精致，略发黄的书页让人爱不惜手，如果你想收藏，就给清华面子，要不以十倍价钱赔给图书馆（我就这样干了）。最后一句：现在的书价太黑，把上 china-pub 的一半时间去泡图书馆吧。

liuty2006:

大概翻了一下。没什么价值---至少对于我。二十年前的书,对于突飞猛进的计算机技术来说,的确老点。

Thecleverestman:

正确的思想永不落后

ggggwan:

昨天去海图买了这本书，回家看后的感觉是有点失望，总的感觉是这本书不如微软那本《快速软件开发》好，不知道是翻译的原因还是原文的原因，读起来不是很流畅，没有那种让我看完后有酣畅淋漓的感觉，看过的软工书只有快速软件开发让我有这种感觉。也可能是文中有很多观点在其他的软工类书中也都有介绍了，看过后没有那种比较震撼的感觉。

另外，文中对软工中的问题缺乏实例讲解以及如何解决的论述，比较泛泛，不容易抓住要点。最近做的一个项目，集成了很多 IBM 的软件，看 IBM 的文档经常就会给我这种感觉，说了很多东西，但是我需要的或者认为重要却说的不多，而且经常写出一些让人看后费解的话，不如微软的文档傻瓜，搞的我很是头大。作者是 IBM 出身的，是不是传染病？

还有书的大小采用了 32 开，搞的页码比较多，翻起来很难受，不如小 16 开的好，留白太多（虽然序言中说明了这是 anders wesley 出版社编辑的决定，但是我觉得不爽，因为我不认为有什么必要在书边做笔记），虽然我

买书可以报销，但是留白多总是让人感觉好像花 25 块钱享受的却只是 20 块钱的服务。

gemlei:

浮躁的 UMLChina 从来就没有自己的观点，那着外人的成果标榜自己的权威。“人月神话”是一本能误导初学者的好书，你可以说它的预言是诡辩，因为没有详细的统计数据和深入的项目背景解剖。泛泛而谈，包容万物。

因为它想什么都是，所以它什么都不是。阅读“人月神话”每个章节像是在做一道道“辨错”题，在项目不同的背景，资金，人员素质，盈利模式下，有着不同的答案。感觉是在学习 CMM 后一个用于练习的案例，仅此而已

hyqzmy:

很好的《人月神话》怎么会翻译成这样？试讨论一二

鄙人以前看《人月神话》英文版，毕竟功底不够，看得慢而无趣。前不久无意之中看到中文版译出，更兼翻译者简历赫赫，急急搜购一本，意图一快，不料读了几章，一些意见却是不吐不快了。

众所周知中文贵在简洁，美在简洁，翻译起来很少会出现译文比原文长的情况，可是看看几个小例子：

第四章“贵族专制、民主政治和系统设计”，原文为“Aristocracy, Democracy, and System Design”。纵使 Brooks 老先生是隐喻大师，我在文中也没有看出一些“贵族”气来，“政治”意味也几乎没有，莫非译者喜欢用搞政治的办法来搞软件？鄙人看来，“专制、民主与系统设计”已经足矣，要更贴切地表达 Aristocracy,则可以用“精英制、民主制与系统设计”。

这个问题实际不大，到了第五章我实在就忍不住要来写帖子了。这章名为“The Second-System Effect”，被译为“画蛇添足”，具体文中遇到这个词组时，又被翻译为“开发第二次所引起的后果”或者“开发第二个系统所引起的后果”。天哪！这里显然 Effect 直接做“效应”译就完了，文中讨论的后果大部分也并非“画蛇添足”，而是在开发升级版本的产品期间常见的一些误区的讨论。我不知道译者是否熟悉软件开发的版本习惯和硬件/系统产品的系列化升级编号习惯，但是从全章描述的内容看，直译的“第二系统效应”甚至更简单一点称为“V2 效应”，恐怕都比什么“画蛇添足”、“开发第二次所引起的后果”或者“开发第二个系统所引起的后果”来的准确而没有歧义与误导。

限于时间，我不再一一举例讨论了。有心的读者朋友不妨在阅读期间自己对对书上提供的原文。看得出来汪先生在翻译中花费了很大的心力，专注与投入是毫无疑问的。汪先生简历赫赫。应该也属青年才俊之流。可是为

什么我们计算机专业的外文经典一翻译过来往往就错漏百出呢？如果这样，如何相信这批人马做出来的软件可靠呢？

据说软件业界有个最佳实践叫做“同行评审”。这本书出版卖钱之前，我看到了同行吹捧，同行评审又有没有呢？那么一大批的编委会，莫非竟是一张虎皮么？

gigix:

有这么严重么？

Aristocracy 和 Democracy 这两个词，是来自于亚里士多德的政治学说。亚里士多德认为，“良好的”政治体制有三种：君主制、贵族制和民主制。这两个词就是这三种政治体制的后两种。所以，如果你问我这个标题有没有“贵族味”和“政治味”，我就觉得它有。当然，你翻译的“精英制、民主制与系统设计”也不错。不过我觉得，最多只能算难分伯仲。如果要说谁更准确地译出了 Brooks 引用亚里士多德的味道，我倒觉得 Adams 的翻译似乎更好一些。

至于你提出的第二个问题，我更觉得无意义了。“程序员在第二次开发类似的项目时加入一些不必要的特性”，这样的行为用“画蛇添足”来描述不正是很贴切吗？至于你所提出的“V2 效应”，恐怕更多的人会想到希特勒的远程导弹，而不是“软件的第二个版本”。看来，这个翻译顶多也只能算难分伯仲了。

中文是不是“贵在简洁美在简洁”，或可商榷；是不是要“翻译起来很少会出现译文比原文长的情况”仍然或可商榷。比起刻意追求“简洁”，想来还是易读性更加重要。“公共汽车”这个词，在英文中只有 3 个字母、1 个音节，请问你要如何追求“简洁”的翻译？

hxtan:

翻译得确实不错

这本书翻译得确实不错，但仍有一些瑕疵，例如：

1、“的地得”偶有混乱。这常常是我判断一个人中文表达是否规范的重要标准。

2、有的术语译得不够到位。例如，将 "An Integrated Approach" 译成了“一条完整的路”。

yuhai:

昨天拿到了书,还没有仔细看.

书的内容暂且不说,只是感觉印刷(或装订?)不是很满意,字太靠近书脊,看得时候比较费劲,要用点力气把书摊平才行。书的行距和间距比平常的要大,大约 23*24 吧,不知道是不是为了多印几页好提高定价?

总之在不了解内容的情况下,单看印刷情况我是没有购买的兴趣的.

gbhome:

终于拿到了这本《神秘的人月》，嗯。一本多年来看到被无数的工程、实践著作引用的文献，终于乖乖地躺在了我的手心里。

第一件事就是把中国人写的那些页数，统统裁下来扔进垃圾筒，本来中文的表达力已经限制了我们对全书的理解，请不要再让那些什么 UMLChina 之类的废话把整本书的导向都弄错了吧！在《神秘的人月》诞生的时候，别说 UML，连 C++都还没有呢。什么印度，呵呵，印度注定要失败的。

不过客观地说，翻译的水平虽然没有我期望的那么高，但居然也不像我想像得那么差。总体而言，还算差强人意。排版不错，我喜欢这种小开本的书，封面也不错。如果是裘宗燕老师来翻译，一定又是一本旷世译作。

对于软件工程我当然有自己的看法，但是我不像许多中国人，在译作前大谈一番，好像比作者还牛。我们没有资格这么做，人要知廉耻。对于这样一本书，在得不到英文原著的情况下，我们要透过中文的表象，来虚心地看每一个字。已经一个星期了，我还在看《焦油坑》这一章，做了几十页的笔记。

无论如何，清华大学此番引入这本书，也算了了我多年的心愿，感恩不尽。

huangyz

书绝对经典，对软件开发很有启发，我在 15 年前武汉大学求学时，已拜读（75 年初版本），是武大的老师翻译的中文，此书我至今仍珍藏在身边。巴比伦塔、外科小组的比喻至今印象深刻。民主与专制、画蛇添足等问题仍是软件开发中的常见问题。20 周年版我一定会买，只是有点胆心翻译的质量。有谁是否能评论一下翻译的质量？

tjuwayne

原著的价值当然毋庸置疑，译者也是专业水平相当高，而且态度严谨，令人赞许（当然，我不配这么夸奖译者，而是该说“崇拜崇拜”）。美中不足的是（仅为个人习惯），尽管译作在语句含义上并没有走样，但是有的地方表达得并不符合一般人的习惯，也经不起语法上的严格推敲。

附带一提，我在本书中竟没发现任何错字，尽管单凭这点不能说明本书的价值高，但说明了译者和评审者一丝不苟的态度。