# 第 36 期（2004 年 4 月）




基石

X-Programmer 非程序员

软 件 以 用 为 本

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

http://www.umlchina.com/

# 了解 UMLChina

软件以用为本。不能为客户提供所需价值的软件是可悲的。以前，开发人员以“专家”自居，客户心存敬畏，认为自己“不懂”，不敢提需求，不敢评判。最终的结果是产生了许多“鸡肋”软件——软件做出来了，也能运行，却不提供想要的价值！现在，客户的胆子大了！软件只不过是帮助我完成业务、提高效率、赚取利润的工具而已！我要的软件要能够为我的业务提供价值，运行架构还要符合我的实际情况，还要有足够的可靠性、灵活性、可用性…。软件组织的“好日子”不再有，不得不关心这些问题：如何捕获有价值的用户需求，如何组织需求，如何获得良好的软件结构，如何获得稳定的性能，如何提高软件的可用性…。

软件（技术）以用为本。不能为软件组织实际使用的技术是可悲的。为了提升软件质量，许多软件组织都在谋求技术的改进：需求的技术、分析设计的技术、增量迭代的技术、构件组装的技术……我们搜索，我们买书，我们看书，我们讨论，我们试用...但学到的技术如果不能最终用于自己的项目之中，将是极大的浪费。而这最后一段路最是艰难。

**相信【软件以用为本】。为此倾注全力，不断实践，苦心研究，采用一切可能的方式，帮助软件组织朝着这个目标进发——让软件变得好用，让技术变得好用：**

- UMLChina 训练——“聚焦最后一公里”，专注于上门为企业作 UML/UP 团队训练。训练完全和受训团队当前项目结合，【训练的过程就是指导团队用所授 UML/UP 方法实作团队当前项目的过程】。UMLChina 在团队训练方面已经有了丰富的积累。从东北到西南，已为超过 50 家企业提供服务，涉及各种类型的软件开发项目。
- 项目指导——是 UMLChina 训练的延伸。UMLChina 专家带领团队用 UML/UP 方法做项目，使团队顺利行走在 UML/UP 的路上。
- 专家讲座——请来国外国内的顶级专家，传播他们的知识和经验。著名专家有：Gerald Weinberg、James J. Odell、John Vlissides、Kent Beck、Alistair Cockburn、Martin Fowler、Scott W. Ambler、Bruce Powel Douglass、Alan Cooper、Roser S. Pressman、Kendall Scott...等。
- 新闻——全球如发生和 UML 相关的事件，都尽力为您及时报送。
- 《非程序员》——和《程序员》同年（2001）诞生，却一直保持自己的独特风格，为众多高阶软件人员所青睐。每月发行一期，杂志将长久保持免费下载的电子形式。
- 书籍——继 2002 年《人月神话》创下记录之后，2003 年《最后期限》、《人件》、《人月神话》分别排在 china-pub 总销售榜的 1、3、4 名。随后 UMLChina 把精力集中到 UML/UP 相关图书：《有效用例模式》、《UML 风格》、《探索需求》、《PEAA》、《敏捷数据》…
- 讨论组——2000 年下半年开始设立，目前注册用户已经超过 40000 人。

# I-Logix Rhapsody 2004 生成 C 代码

[2004/4/15]

3 月 30 日，I-Logix 公司发布了 Rhapsody 2004，这是其旗帜产品，基于 UML 的嵌入式开发工具，的升级版本，该版本整合了对 UML 的图形的支持。在过去生成面向对象代码的基础上，该工具增加了结构化 C 代码的自动生成功能。


在解释到增加非 UML 图形的原因时，Rhapsody 的产品市场主管 Scott Niemann 说到，“有很多人仅仅就想画图，这些人占据了很大的市场份额，有时候，软件工具必须也对那些和 UML 无关的东西提供支持。现在他们用 PowerPoint 和 Visio 设计一些高层的视图，用 UML 来支持实现”，Niemann 认为这导致了需要用两个分离的应用软件来处理一些相关的文档的情况。“但有些设计是必须和模型保持同步的，应用 Rhapsody 2004，用户在表达设计思路的时候，不需要脱离其设计环境。用户可以画自己想画的东西，并且这些东西会成为模型的一部分。”

但 Niemann 也提醒，Rhapsody 不会为这些非 UML 的图形生成代码，“那是下一步的事情”。

另外，Rhapsody 2004 将第一次生成结构化的 C 代码，Niemann 认为这是产品面向更多开发人员的举动。“许多 C 开发人员不愿意写面向对象的代码，因此我们失去了这个市场的相当大一部分。”过去的 Rhapsody 版本只支持面向对象的 C 代码。Niemann 说，“过去，我们努力让传统的 C 开发人员写面向代码的代码。但是在研究了市场、用户的交谈记录以及（技术的）前途后我们发现，大多数的嵌入式（C 语言）开发用结构化代码就可以完成了。”

Rhapsody 同样包括了一个扩展的基于模型的框架，Niemann 介绍，该框架将简化符合 DO-178B 的任务－关键应用系统的开发工作，DO-178B 是航空电子工业和航空宇宙控制系统所必须通过的一个认证标准。

Rhapsody 2004 现在是 for Windows 的版本，起价是每个开发人员 1250 美元，目标是要覆盖主要的 RTOS 操作系统，现有 Rhapsody license 执有者可以免费升级该软件。

（自 sdtimes，UMLChina 袁峰 摘译，不得转载用于商业用途）

# UML2 意味着“模型驱动”的架构

[2004/4/15]

软件架构师 Shaun Forgie 认为，UML 的 2.0 版本是模型驱动开发时代到来的号角。

他说，抽象层次的增加将会导致编写代码，不管是 Java 还是 C++，变得不再紧要，就像今天的汇编语言一样。

在和软件架构师协会的一个本地听众交流时，Forgie 认为，UML 2.0 增强了该建模语言的可扩展性（scalability）。在 2.0 版本中，各种图的描述功能更强，例如，形式化表示算法行为的“状态机”。在新版本中，过程和数据的各种形式化表示之间的关系也被定义得更加清晰。

Forgie 预言，将会有一个架构被定义来“执行”模型，而不需要创建一个可运行的程序。所有这些都会使得大多数开发工作可以更轻易地在模型层次上完成，然后自动生成代码。也就是说，手工书写代码将不再需要，甚至代码的检查也将几乎不再需要。 在基于组件开发（component-based development）方面，先前的 UML1.x 版本稍有落后。尤其是在面对实时系统时，导致了使用 UML 的困难，而且生成的代码也很凌乱。

组件，J2EE 和 .Net 这些流行的开发平台上的元素，在被提供时通常同时还提供有对每个组件行为的描述、提供的接口（port）以及用来和其他组件交流的协议。这些接口用来将这些服务组合在一起，以提供给组织中不同的组调用。这个概念加强了软件的可重用。

对模型而言，也需要被验证和执行，在手工编码和自动生成代码之前都需要检验模型的逻辑。模型同样也有归档和维护等操作。

现在，开发人员可以看到一个真正的模型驱动架构(MDA)，过程以及数据库元素都可以从模型中生成得到。这一切和过去不再一样，过去，软件开发中存在着面向对象的应用设计和另外设计的关系数据库间的不吻合。

Forgie 认为，UML 现在已经更加优美地融入到了 OMG 的标准和框架之中了。UML 现在已经被纳入最高层为元对象设施 MOF 的层次之中了（译者注：这里指的是 OMG 的四层元模型架构）。诸如 UML、Corba 以及 OMG 的组件仓库元模型都在这个系统中联系起来，它们都是 MOF 的实例。正如一个特定的 UML 模型是 UML 的一个实例，而一个现实世界的对象是这个模型的一个实例一样。

Forgie 说，模型驱动架构已经“使我节省了 50%的开发工作量，并有希望在未来五年内节省掉这剩下来的 50%”。

模型驱动架构的结构仍在发展之中，将 UML 应用到不同领域的“profile”正在逐年增加，这些领域包括系统工程、应用集成及测试。

IBM（现在包括了 Rational）奠基的 Eclipse 将成为全面实现这个复杂结构的第一个开发环境。作为吸引眼球的招数，各种图形标识的管理方法将会首先推出，然后将会是完成各种实际工作的底层结构支持，全面的实现还要再等几年。

（自 computerworld，UMLChina 袁峰 摘译，不得转载用于商业用途）

征稿

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

# IBM 将让所有 Rational 工具基于 Eclipse

[2004/3/29]

IBM Rational 正在继续同步 Eclipse 框架和整个 Rational 开发工具的产品线。星期一，公司执行官说，他们正在进行一项策略，争取在未来的 18 个月里把所有的 Rational 工具全部构建到 Eclipse 里面。


现在 Eclipse 已经集成了主要的 Rational 工具，IBM Rational 的市场经理 Jeff Hammond 在一次电话会议上说。IBM 的一些工具，例如 WebSphere Studio，已经基于 Eclipse，而且 IBM 会继续这种趋势。

“如果你看看 Rational 现在的产品，80％都和 Eclipse 集成了”， Hammond 说，“未来一年半中总的策略方向是从 Eclipse 集成到基于 Eclipse”。

IBM Rational 的总经理 Mike Devlin 说，同时，IBM Rational 在 Eclipse 工作以及它自己的工具中将继续支持开源和标准技术。

“我们为 Eclipse.org 贡献了相当多的技术”，Devlin 说，“希望能够继续贡献。我们打算把新技术先作为产品的一部分，然后，随着时间，将把它们移植到 Eclipse。”

IBM 下个对 Eclipse 开源组织的贡献之一将是围绕 OMG 组织的 UML2.0 规范方面的工作，他说，IBM 和 BEA 系统已经组立团队来开发一种标准的方式，把数据和 EJB，SDO（Service Data Objects）进行绑定。这项工作通过 Eclipse 在开源团体中找到自己的位置，Devlin 说。

有时 IBM 先让技术出现在 Eclipse 里，然后再把它构建到自己的开发工具当中，Devlin 说。例如，IBM 正在把 UML 元模型放到 Eclipse 中，将来建造一个新的、从根本上简化 J2EE 开发的工具，Hammond 说。IBM 将在这个夏天德克萨斯的 Rational 软件开发用户大会上介绍这个新的软件，他说。

另外，IBM 计划将 Hyades 技术－－一项平台交叉测试，日志和远程调试的开源技术－－加入到它自己的 Rational 测试工具中，Hammond 说。Hyades 技术现在已经在 Eclipse 平台有一个实现了。

（自 CRN，Windy 摘译，不得转载用于商业用途）

# 代码生成的未来

[2004/3/22]

模型驱动架构的先行者面临着文化的壁垒，但回报将是事件和资金的节省，以及更高的代码质量。

Wisconsin 正在使用一个 Web 系统替换原来的基于 client/server 的失业保险金应用系统。但这一次，团队通过画图来描述业务过程，而不是写厚厚的规约。

他们聘请了一位讲师，教授 Cobol 开发人员如何使用统一建模语言（Unified Modeling Language），团队在计算机上为新的应用画出了各种 UML 图。

部分软件开发组织正在转向名为 MDA（Model Driven Architecture）的系列标准，Wisconsin 只是其中一员。MDA 由总部位于 Needham 的国际对象管理集团 OMG 所制定。他们使用 UML 以及代码生成工具来构建应用，如果一切顺利的话，在完成业务逻辑的过程中，编程语言将会用得很少。OMG 的 CEO，Richard Soley，说，他已经知道有两家公司通过 MDA 生成 100% 的代码。

但是，自动代码生成只是 MDA 带来的利益中的一种。他们认为，MDA 还将带来开发时间和成本的降低、代码质量的提高、代码复用的提高，以及与应用需求更好的吻合。

Lee Carter，Wisconsin 开发部分的项目主管，认为这是有帮助的，业务用户和 IT 开发人员现在可以使用相同的语言来描述需求。“这使得我们可以关注于业务需求而不是过早地考虑底层的技术细节”。

为了表示需求，开发人员和业务分析员使用图形来表示各种用例场景，诸如一个不完整的应用是如何处理，或者如何处理被拒绝的应用请求。并表示这些系统是如何联系起来的。Carter 说，一旦一系列活动图、顺序图、协作图以及其他图设计之后，就可以通过工具转换这些模型，得到应用的代码。

他认为这些文档化应用的图可以在任何时间使用，就算历经多年。因为他们是由 IBM Rational 的软件所支持的。“图形比文字的表现能力更强，写满的一页页文字，通常没有人会再看第二遍。而且，我们可以从业务需求跟踪到代码，或者从代码回溯到业务需求。”

为了简便到 MDA 的转化，这个项目团队从 OMG 的 MDA FastStart 计划中购买了位于 St. Paul, Minn 的 Adaptive Team Collaboration 公司的服务。ATC 公司的 CTO，Chris Armstrong，担任过讲师、教练、顾问、心理专家等各种角色。Carter 说，他最近的角色是过程审计人员。

另外一个重要的决定是选用了 Curam 公司基于 J2EE 的框架为政府的社会服务部分开发应用。Wisconsin 团队使用 UML 来描述业务需求，并通过一个 PIM（platform-independent model，平台无关模型）定义，有效地集成到 Curam 软件中。系统集成 Tier 技术公司在这项工作中亦有贡献。

Carter 认为，到目前位置，项目已经超出了预期的效果。团队提前 60 天完成了项目。第一个产品版本 11 月将发布，其他部分将在两年内完成。和使用传统方法开发的类似项目相比，开发时间得到了减少。

Wilmington 的 PFPC 公司为投资管理行业提供 IT 外包服务和应用，他们说，采用 MDA，过去需要 6 个月完成的客户的项目，现在 4 个月就可以完成。

该公司的 CIO，Michael Harte，认为，MDA 已经逐渐渗透到他的团队中，带来了更大的精确性。因为这个过程强制了开发中需要做更多的前置工作，和过去相比，严重的设计缺陷可以被更早地发现。

PFPC 的高级架构师 Ian Maung 指出，MDA 帮助位于不同地方的开发人员可以更好的交流，因为（通过 UML）这个统一的接口，他们可以理解对方的意思。

Maung 认为，将来 MDA 将带简化运行平台之间的切换，因为业务和应用逻辑的定义都是平台无关的，而且只需要简单地修改就可以生成新的平台相关模型。他希望 UML2.0 中的行为语义（action semantics）可以实现对一些像 J2EE 这种基于标准的平台的 100%的代码生成，从而使得 MDA 最终可以实现自定义的代码生成。

根据几名分析人员的估计，尽管 UML/MDA 越来越多地被应用架构师所采用，也只有不到 15%的开发人员在使用 UML。批评者认为其复杂性将会拒人于门外，并且想要将已有的 IT 文化转变为代码生成的方式也是有困难的。

Forrester Research 公司的分析人员 Analyst Carl 认为，很少有开发团队有耐心深入到一个项目中很久还没有看到一行代码。“对传统的 IT 企业而言，尽早看见代码的愿望是普遍的，我认为，对于任何形式的建模而言，这都是唯一的最大的障碍”。

Meta Group 公司的 Thomas Murphy 认为，许多开发人员都有过使用 4GL 工具产生的难以维护的代码的不愉快的经验，这使得他们很犹疑。他预测，模型驱动的开发方式将会前进，不一定是使用 UML/MDA，但会面临很多挑战。

先行者对这些困难中的很多都有意识到。例如，Maung 认为宣称完全支持这些标准的工具往往都不是真的完全支持。PFPC 的首席架构师 Per Gyllstrom 指出，也有一些开发人员担心自动代码生成工具将抢掉他们的饭碗。他认为，重要的是要先在一些小型项目中实施，以消除人们对 MDA 的怀疑。

尽管 MDA 的优点之一是代码复用，但开发人员往往很难信任别人的代码，或者是机器产生的代码。

位于加州 Novato 的 Fireman's Fund Insurance 公司在进行大项目时遇到了这样的阻碍。该项目 2000 年开始，由一个精于面向对象开发的 7 人团队使用在 IBM 的保险应用架构（Insurance Application Architecture，IAA）中开发的 UML 类图的部分来生成一个基于企业 JavaBean 框架的代码，这个框架包含了持久性、消息、事件以及对象分布方面的支持。

Rational Rose 的脚本产生了超过 80%的架构代码，Fireman's Fund 后来选用了 Codagen 公司的一个代码生成工具。几个星期之内，Codagen 的规则引擎和可重用模板就使得使用 Rose 脚本来产生模型到代码转换的需求消失得无影无踪。

但是，但优先级改变时，问题出现了，核心架构团队和需要手工编写业务逻辑代码的开发人员之间出现了问题，Fireman's Fund 的一位企业架构师 Bill Nadal 说。

企业架构师团队在 2002 年重新拾起了火炬，这一次，Peter Herzum ，面向组件软件制造的一位专家，他的目标是设计可重用的组件。团队的目标是面向开发的“软件工厂”，其中组件可以被重用和拼装，以构建各种应用。

Nadal 说，Herzum 的软件 LLC 以及 Codagen 面向 IBM 的 MQSeries 的金融版本使用了 IAA 的类模型以及 Rose 脚本来生成服务以及基于 XML 的消息。在他们演示中，应用 MDA 可以自动为持久化的类、服务接口以及 XML 信息生成代码。这为 MDA 方法提供了一个论据。

Fireman's Fund 的企业架构师现在正在用他们的组件蓝图（blueprints）为安全、内容管理以及元数据管理构建核心的基础模型。将来，他们计划要关注于业务层次的组件。

“大家都认为要在一个企业全面推广是耗时且成本昂贵的”，Nadal 认为，“这通常意味着你需在要在一些单独地项目中定义一些高质量的组件，以后可以多次复用。MDA 是实现这个目标的重要一步。”

（自 COMPUTERWORLD，袁峰 摘译，不得转载用于商业用途）

# 微软的模型观点与众不同

[2004/3/22]

Prashant Sridharan，微软工具组的一个主管产品经理说，公司虽然在一般的模型驱动开发理念上完全落后，但微软相信，有一些对模型的需求和操作系统紧紧相关。

Sridharan 说，UML/MDA 阵营推动了“一次建模，多平台编码”的观念，但这个观念在实践中不太有效，因此，微软正在开发一个模型引擎和模型框架，让用户可以在 Windows 为中心的环境里对面向服务的应用进行描述。他说他希望合作伙伴们可以在其上提供 UML 工具。


白马(Whitehorse)是这个面向服务设计工具的代号，预计在 2005 年的上半年推出，和 Microsoft 的 Visual Studio 2005 开发工具一起。

Whitehorse 的其中一个要点是，它是一种基于 XML 的领域细化语言，来描述服务或网络基础设施的各个部分以及它们如何连接。微软计划为想要建立 DSL 的大公司们发布一个软件开发工具包。

只有在当你想看到大规模生产力的时候，它才有价值，因为那样做需要付出大量的努力，Sridharan 警告说。

微软打算在 Whitehorse 中提供的三个设计器将帮助用户创建基于 DSL 的图。一个逻辑基础设施设计器允许用户用方框和线条可视化地描述网络中每个硬件部分能做什么。一个面向服务的应用设计器可以用来描述网络服务和连接它们的协议，一个类设计器用来描述类或接口，以及它们之间的继承关系。

使用 Whitehorse，一个设计面向服务应用的架构师将能验证应用在所描绘的网络拓扑中是否可以工作，Sridharan 说。

拖拽一个服务到面向服务设计器中的行为不光是创建一个模型，它还产生代码，因为面向服务的设计器和类设计器及它底层的代码同步，Sridharan 说，由于用户仍然需要编写商业逻辑，不应该期望生成 100%的代码，但随着时间的过去，用户们将看到日益增多的代码生成。

（自 COMPUTERWORLD，Windy 摘译，不得转载用于商业用途）

# Telelogic 升级 DOORS 套件以改进定义软件需求的过程

[2004/3/15]

在 Standish 集团对软件项目失败原因的年度调查中，去年的结果是：仅有 54%的功能被实现，而实现的功能中近半数没有被使用。根据 Standish 的报告，这种情况的原因通常源于定义不佳的需求。

Telelogic AB 公司的 DOORS（全名是 Dynamic Object Oriented Requirements System，面向对象的动态需求系统） 软件的目标是捕获细致定义的软件需求并将其转为代码。周一 Telelogic 在其套件中增加了一个新的产品，DOORS/Analyst 1.2，以捕获需求并将他们转交给其他 Telelogic 工具以生成代码。这些工具过去只能生成 C 代码，按照 Telelogic AB 公司的声明，现在它们已经可以生成 Java 和 C++的代码了。


系统可以使用标准的行业建模语言 UML 2.0 创建需求模型，UML 2.0 中增加了使用活动图、交互总览图（interaction-overview diagrams）及组件图来表达业务的能力。

模型也可以导入到 Telelogic's Doors/Architect 2.3 中，其中可以进行满足需求的系统架构的初始设计。

全球市场部的副总裁 Michael Donner 介绍，最后一步是将模型导入到 Doors/Developer 2.3 中，其中可以基于需求及架构设计系统模型。Developer 的最终输出是自动生成的 C++或 Java 代码。

Andy Gurd，Telelogic 工具集的高级项目主管，介绍说，通过使用以上三种 Doors 工具，需求分析员或者软件设计人员可以从定义好的需求开始工作，并跟踪到最终实现来满足该需求的软件。如果某些地方和最终用户的期待有出入的话，“可跟踪性就起作用了”，它能够帮助你尽快地做出补救。

Doors 可以和主流的 Java 开发工具一起使用，例如 Sun 的 Java Studio 或者 IBM 赞助的开源的 Eclipse。Doors/Analyst 1.2、Architect 2.3 和 Developer 2.3 在 4 月 30 日都将问世。这三个工具都可以工作于 Windows 2000 和 Windows XP 平台的。架构师和开发人员同样可以在 Sun 的 Solaris 8 和 Red Hat 企业版 Linux 上使用这些工具。

Telelogic 和 IBM Rational 的软件竞争，在需求分析和系统建模领域，这两个产品目前在市场份额上差不多是平分秋色，各占大约 30%。

对于高质量的软件开发而言，需求管理的重要性益发突出。Rational 试图定义五个需求管理级别并分别提供度量集。Meta Group 的 Thomas Murphy 在一份研究报告中提到，Rational 的 Rational 统一过程（Rational Unified Process）“掌握了需求管理的很多需求。”

对于高质量的软件开发而言，需求管理的重要性益发突出。Rational 试图定义五个需求管理级别并分别提供度量集。Meta Group 的 Thomas Murphy 在一份研究报告中提到，Rational 的 Rational 统一过程（Rational Unified Process）“掌握了需求管理的很多需求。”

他指出，“大多数组织都掉进了这样一个陷阱：不停地为项目增加一些不成文的插件（add-on）”，这些增加的部分或者没有进行很好的文档化，或者使用和过去的需求所不同的结构在描述。

Murphy 在报告中指出，不论是使用 Doors 还是 Rational 的方法，都需要能够理解系统的业务需求的有经验的系统分析师 。

Standish 的报告中总结到，“没有基本的系统需求的处理，一个项目注定失败”。

（自 informationweek，袁峰 摘译，不得转载用于商业用途）

征稿

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

# Telelogic 进军普通应用开发

[2004/3/15]

今天 Telelogic AB 希望宣布迈出系列行动中的第一步，在实时操作系统（RTOS）上推出建模和开发专家工具。

该公司的工具用于建造各种各样的通信，太空和国防应用，它将发布 TAU/Developer C++ 和 TAU/Developer Java，可以从基于 UML 2.0 的模型自动生成 C++和 Java 的代码。


同时，在年内，TeleLogic 也将为 C#语言推出一个 TAU/Developer 版本，设计支持微软公司的.NET 框架内部基于对象开发的 XML web service。

Telelogic 打算在现有的客户基础上进行扩展－包括汽车制造，金融和 IT，它们已经在用 RTOS 上的 TAU/Developer AgileC 和 TAU/Developer C++。

这些公司现在可以用 Telelogic 的 DOORS/Analyst 和 TAU/Architect，他们可以在更普遍的 C++和 Java 应用编程环境里为 RTOS 提供 UML 图，进行建模和代码生成。

不过这样的公司可能正在使用或打算使用 Borland 公司和 IBM 公司的 C++和 Java 竞争开发工具，它们同样在工具里使用基于标准的模型促使 UML 产生应用代码。

Telelogic 为开发者们提供了灵活性，因为用 DOORS/Analyst 进行代码生成与 UML 模型同步。同时，DOORS/Analyst 支持 UML2.0，为组件，发布和交互纵览等支持更多的图。

（自 cbronline，Windy 摘译，不得转载用于商业用途）

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

# 探索需求

## 设计前的质量

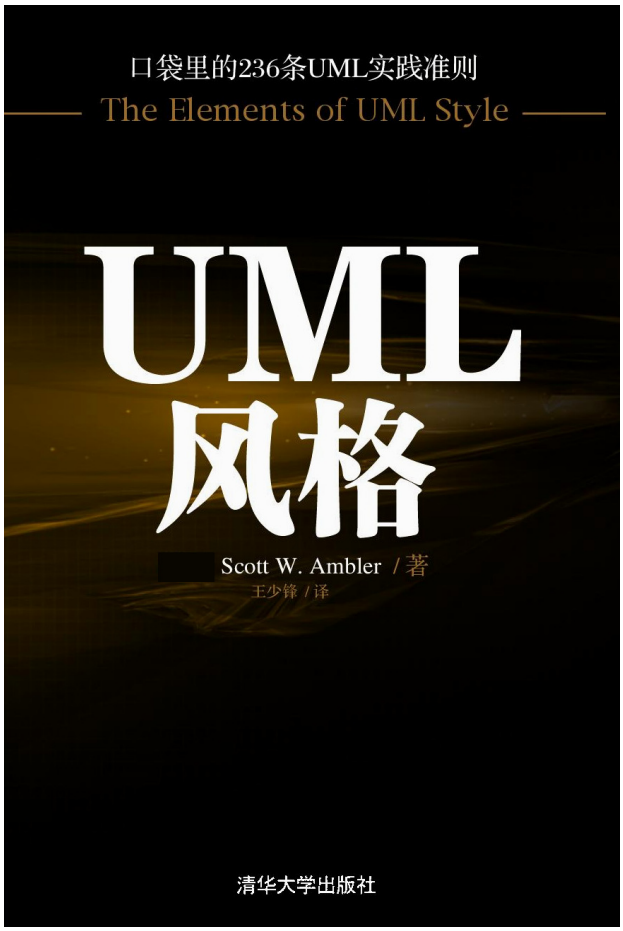
需求之于开发，就像婚姻之于人生

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

# Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社


口袋里的236条UML实践准则
The Elements of UML Style
UML
风格
Scott W. Ambler / 著
王少锋 / 译
清华大学出版社


mobile

The Addison-Wesley Signature Series

# Patterns of Enterprise Application Architecture

中译本即将上市

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

UMLChina 训练教材

# 温伯格“探索需求”交流摘要

潘加宇、熊妍妍 整理翻译

查看评论

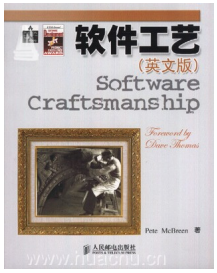
2004 年 3 月 19 日，温伯格先生应邀在 UMLChina 做了一次有关“探索需求”的交流。以下是部分精华摘要，全文将发表在 2004 年 5 月份的《程序员》杂志。

## UML

……如果你希望你的涉众懂得 UML，你就误解了 UML 这个工具和如何使用 UML…我们看到过很多事例：工程师们认为涉众应该懂得 UML，但实际不是，因此他们就责备涉众不懂 UML。这当然是错误的。涉众是客户，你应该为客户提供服务，而不是让他们来服务你。你的工作就是为客户服务……

## 您对“软件工艺”如何看待？

……我的观点是，对于开发，特别是开发软件，绝对是一种工艺……我建议软件人员尝试更多的工匠术语，如工具、家具之类的。在软件业中道理是同样的。你注意到他们的工具了吗？他们是否为每项工作选择了正确的工具？他们是否会适当地使用他们的工具？一名好的工匠不会责备他的工具，因为选择正确的工具，并保持它的锋利和干净，是工匠的职责…

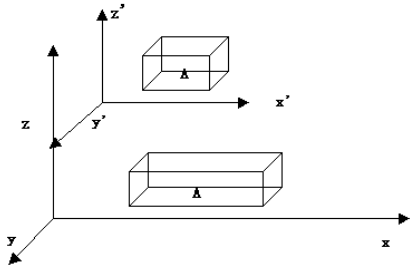


## 如何说服客户放弃不切实际的需求？

……基本的思想是：告诉客户，我们能满足他们的需求，但是这需要成本。如果你没有告诉他们所需的成本，他们当然不会知道那些需求是不切实际的。但是，判断需求是否不切实际并非工程师的责任，因为工程师不知道花多少钱才值得他们去做，他们只知道满足需求需要花多少钱……另一种方法就是告诉你的客户，因为需求的变化，一些重要的功能性或特性会因此而改变……或者你可以这么说，“我们能做，但如果我们做了这个，就不能做别的事情了，你看到底怎么办？”。他们会选择放弃其中一些要求，使需求变得更加切合实际。但要记住这重要的一点：工程师不能判断什么是切合实际的，你只能告诉客户需求的成本，他们会比较看看成本是否值得，然后做出决定。

## 如何判断客户有没有告诉我们他们真正的需求？

……他们确实不知道你能帮他们什么。这是一种情况。另一种情况是他们确实知道，但他们并没有告诉你。他们在对你说谎或他们隐藏了某些东西。……如果你的客户并不了解他们的需求是什么，那么你就会常常看到他们反复无常。……最常见的例子就是我们发现某个人说，“我们需要一个快速的系统。”看，我们不知道“快”意味着什么。这个“快”和他们头脑中已有的某些速度有关。对某些人来说快的系统，对另外一些人却是慢的。所以，你必须找出更清楚的事实。



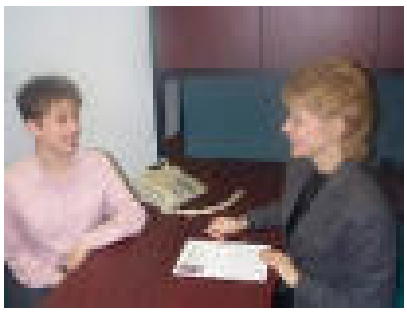
## 原型……

……最常见的错误是原型过于精细了。然后在客户得到真实体验之前已经付出太多，太迟了……对原型投入太多非常不正常，但这里也有个很好的检验方法。你们每时每刻都在开发原型吗？……他们说：“噢，不，那根本不是我们想要的。”你能负担得起把它扔开从头来过吗？但那不是真正的原型。你负担不起，因为它不是原型。

## 怎样成为一个好的需求工程师？需要哪些能力？

我想说，最重要的能力是，你需要有耐心以及理解其他人的能力……我见过很多工程师，不会回到客户那里再次询问和澄清问题，因为他们害怕再问一次。如果你怕生，就不能成为一个好的需求工程师。……问题，我回来找你可以吗？”如果你这样问，人们一般都会说好，然后你就可以很大方地回来问问题了。

另一个你将学会要问的问题是：“那么，我已经问了你好多问题了，还有其他什么你要问我的吗？”你们看，不是所有的需求工作都是问问题，然后听答案。你要打破这种思维，说需求过程就是你跟客户坐在一起，你问问题，他们回答。需求过程是获取信息的过程，一些信息在你这里，一些在客户那里，一些客户没有，还有一些客户有着错误的信息……即使我在这一行干了有 50 个年头了，我仍然怯于提问……



## 《探索需求》

……我一直努力尝试着创作一些能够留存久远的东西。我的第一本书是有关如何为一种特定的机器编写一种特定的语言的，这本书并没有被翻译成中文出版。当然，如果你写一本编程语言的书，一两年后这本书很容易就过时了。因此，我想，如果花很长时间去写一些只能留存一年的东西，的确是有些不值。……即使是在今天，30多年后，人们还是会找到我，跟我说：“你一定在我们公司工作过吧。你书中提到的那个人就在我们公司工作。他做和你书中提到的那个人一样的工作，我知道他是谁。那么，你书中提到的那个人有多大岁数？”我说：“哦，25岁。”“哦，他那年在我们公司做的时候我刚出生。”[笑了笑]所以说，现在的人仍然以同样的方式、同样的心理，做着我们在过去做的事情……. 而我们在《探索需求》一书中建议大家使用的工具，则是那些我们期望能够留存长久的工具。


录音下载：http://www.umlchina.com/Chat/weinberg.htm

全文请看：2004 年 5 月份的《程序员》杂志

# 关于温伯格

张亚勤

提到软件开发，人们就会想起微软。常有朋友问我，微软成功的秘密是什么，怎样才能让软件走入千家万户。其实，这类问题早在三十年前就有人完整地阐述和解答过；而且，即使是经历了这么长时间的技术革新，这些论述依然是非常具有借鉴价值和启发性。解答问题的正是这一系列丛书的作者--尊敬的温伯格先生。

温伯格先生是从个体心理、组织行为和企业文化角度研究软件管理和软件工程的权威和代表人物，他有着程序员、系统设计师、咨询师、专业作家的多重身份。温伯格认为：软件的任务是为了解决某一个特定的问题，而软件开发者的任务却需要解决一系列的问题。他自称为"思考着的人"（thinker，而非人们为他定义的"思想家"），同时将他思考的结论和方法通过文字传递给百万计的读者。

温伯格还是一个实干家，他所创建的学校、培训基地，主持的大学、研讨会，给一代又一代软件工作者提供了"清新的空气"。温伯格最喜欢的一句话是中国传统的一句谚语：智者千虑，必有一失；愚者千虑，必有一得。思考是自作聪明者最大的弱项，也是成功者最大的财富。温伯格说，我们不能要求每个人都聪明异常，能够解决所有难题；但是我们必须持续思考，因为只有如此，我们才能明白自己在做什么。

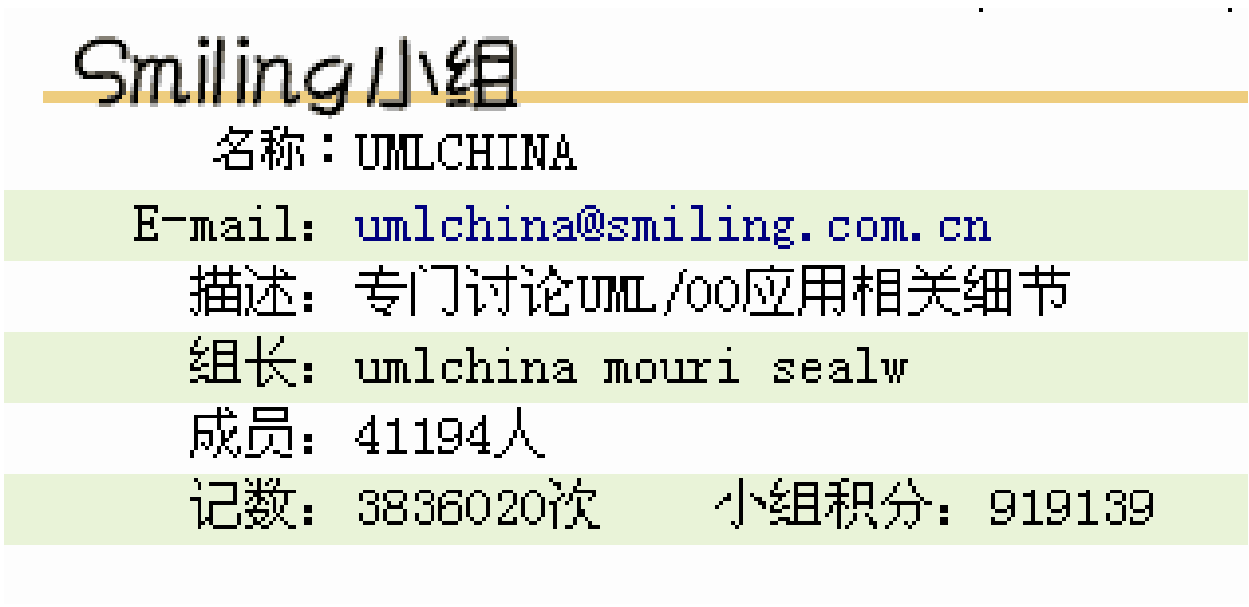
"明白自己在做什么"，是走向成功的必要条件。那些能够很早地领会或感悟到自然发展、社会发展、人类发展、行业发展、软件发展在很长一段时间内的可能趋势的先知先觉者，虽然在这个世界上不到万分之一，但是他们是时代的智者，只要他们愿意去做，他们能够很快地获得成功。他们具有非常敏感的嗅觉和洞察力，能够很好地把握未来几年的软件需求，从而进行应用解决方案的设计、前卫体验理念的构建。或者说，他们能够在行业内把握方向，技术上突破，特别的是在一些尚未发掘的领域异军突起。他们属于时代或行业的领导者，其成功一半是天才，一半是勤奋。

还有一些人，他们对趋势的领会并不十分敏锐，但是他们最大的优点在于能够在经验的基础上踏实前进。他们的成功百分之九十九来自于学习和勤奋的实践。他们是时代和行业的中坚，是事实上的社会的缔造者，当然也是行业上建设者。他们能够很清楚地知道自身的优势和劣势，根据时代和行业的现状，以及自身的经验和积累，进行主流软件开发、生产和实施。他们不一定掌握最新技术，但是他们一般来说资本和经验都非常充足，使他们保持中流砥柱位置的根本在于其能够正确认识到自身和外界的差距或互补，从而调整策略，后来居上或反败为胜。

"明白自己在做什么"，这种态度确保在进行软件开发和研究时保持理性和慎密的思考。经过了十多年的实践，温伯格先生称："技术是毫无价值的"，我的理解是，如果我们都不知道自己所作所为能给社会或自己带来什么，是根本无法找到那些有价值的技术。而他所说的无用的技术指的恰恰就是那些异想天开、不切合实际的无效劳动罢了。通过和温伯格先生的交谈和我自己在微软工作的经验，我可以负责地说，任何成功者都是其领域内的思考者的人，这种思考，使他们在不知不觉中逐渐向正确的方向转变；而温伯格的这一系列努力，正是让我们进行更深一层次思考的提醒。

我相信不论您是否从事软件开发、研究或管理工作，都能从温伯格先生谆谆的话语中收到启发。

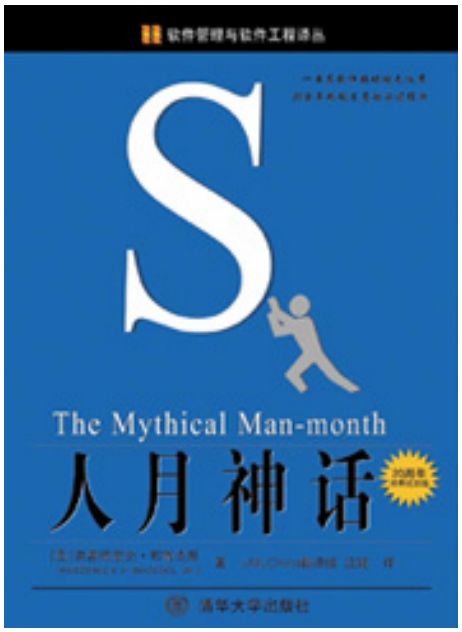
**本文写于 2003 年 8 月。本文作者为微软亚洲研究院院长兼首席科学家。**

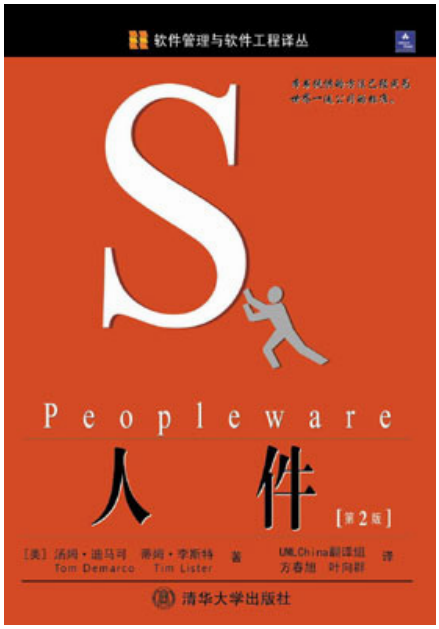


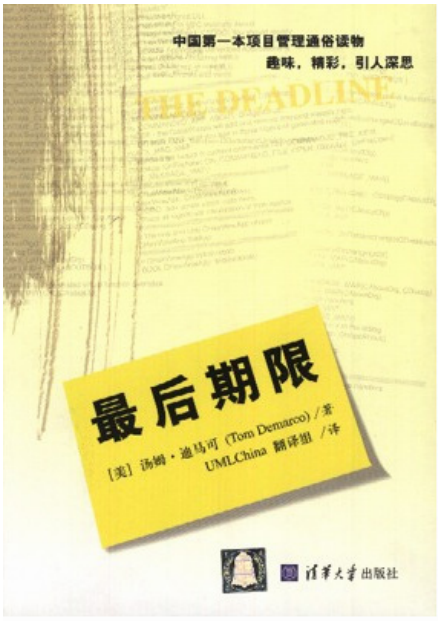

斐力庇第斯从马拉松跑回雅典报信，虽然已是满身血迹、精疲力尽，但他知道：没有出现在雅典人民面前，前面的路程都是白费。

学到的知识如果不能最终【用】于您自己的项目之中，也同样是极大的浪费。而这最后一段路最是艰难。

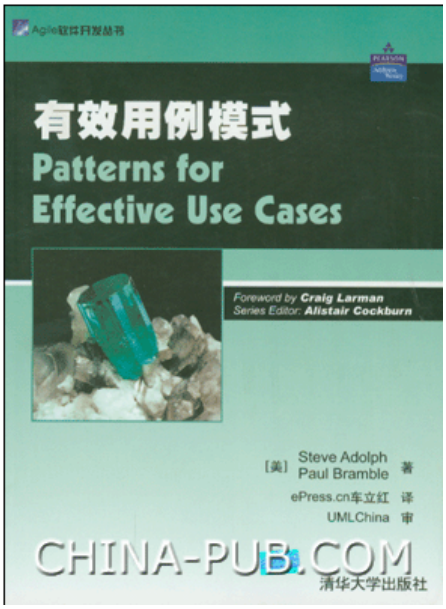
UMLChina 聚焦最后一公里，所提供服务全部与您自己的项目密切结合，帮您走完最艰难的一段路。

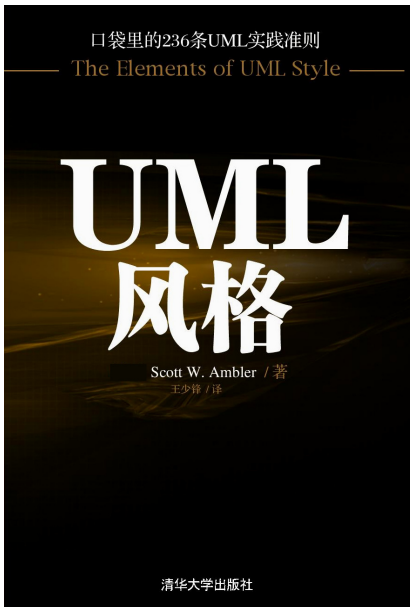


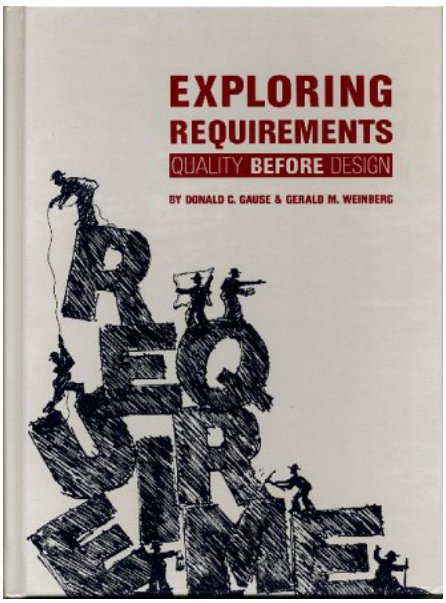



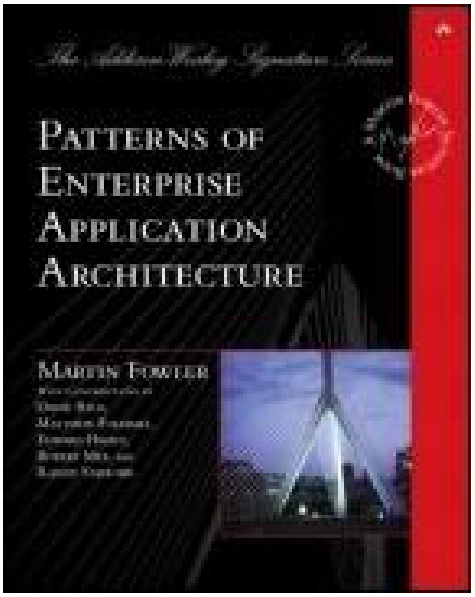



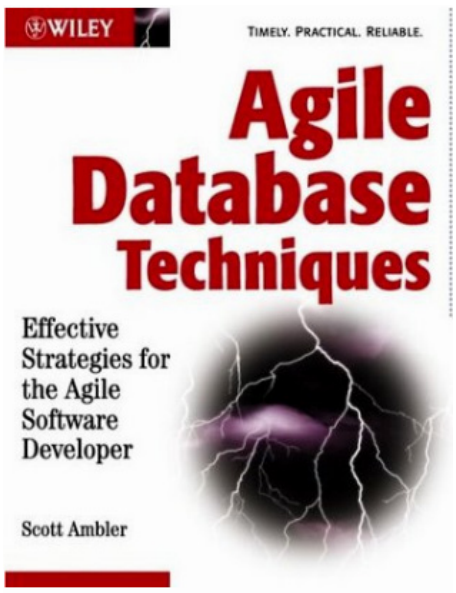

- 2002 年《人月神话》创下销售记录
- 2003 年《最后期限》、《人件》、《人月神话》分别排在 china-pub 年度总销售榜的 1、3、4 名



# 论银弹的存在（v0.3）

张恂 著

查看评论

“颠覆软件工程[4][8]？软件工程是软件工业的立业之本，深深扎根于数百年来工业、产业和信息革命造就的肥沃土壤。某些人出于各种目的，提出颠覆软件工程，就像要颠覆建筑工程、土木工程、机械工程、装饰装修工程……一样荒唐幼稚，无异于痴人说梦！除非你是在自己家里摆弄几个软件自娱自乐，顺便向别人炫耀一下自己的智力，那可能是不需要软件工程的。”

—— IT之源 张恂，2004年3月25日

**注：**是的，您发现本文尚未最终完成。这是我们首度尝试网络开放式写作，热忱欢迎感兴趣的读者与我们一道来创作，提供正反面的论点、论据和数据。相信在此文的形成过程中，也一定会存在不少疏漏和错误，望大家及时批评指正。

我会及时综合大家的意见，把其中我认为合适的内容陆续添加到文章中，并记下贡献者的名字。来信请寄：zhangxun2001@hotmail.com

## 变更历史

0.3 2004-4-15 添加了银弹的证据，补充了参考文献，修缮了通篇的文字叙述。

0.2 2004-3-28 具备主要观点，初次发布在 www.iturls.com

## 致谢

首先感谢清华和电力出版社先后出版了《人月神话》的中文版、影印版，为本文提供了宝贵的研究素材。

（……）

## 目 录

## 引言

有人看到这个题目，可能会想：什么人胆敢说银弹是存在的？No Silver Bullet（NSB），这个世界软件工程界最著名的论断之一，出自美国 1999 年图灵奖得主、世界软件工程界的先驱、权威和元老——费雷德里克•布鲁克斯博士、教授，怎么可能出错呢？！多少年来，国内国外，上上下下，学术界、工业界，大家都众口一辞地认为没有银弹，他怎么敢冒天下之大不韪，说“有银弹”呢？这个人怎么了，如此不知天高地厚，是不是有什么毛病，难道不会死得很惨？

的确，这两年来随着《人月神话》中文版、影印版[2][3]陆续在国内的引进、翻译、出版、造势和鼓噪，如今“没有银弹”这个时髦的舶来品已经成为满天飞的口水词，被软件界的一些人天天挂在嘴上，“这个不是银弹，那个不是银弹，OOAD 不是银弹，UML 不是银弹，用例不是银弹，RUP 不是银弹，MDA 不是银弹，软件工程也不是银弹，”……凡此种种，真是举不胜举。布氏银弹（NSB）在人们眼里就好像是可随身携带的万精油、万能膏，只有能找到地方，贴上一剂准保管用。

可事实的真相果真如此简单吗？我要告诉大家，今天我是明知山有虎，偏向虎山行。布氏银弹到底是什么，是不是像大家所说的那样没有银弹，为什么说有银弹，为什么又说布氏银弹没有意义，老布的真实意图又是什么？请看下文分解。

## 写作动机

最近，在一次闲谈中，我向一位圈内朋友询问他对国内软件工程现状和未来发展的看法。他是这方面的活跃分子，本以为会听到他一番踌躇满志的表达，却没想到他摇摇头，我问他怎么了，他说“软件工程在国内已经**被搞臭了**”，所以现在他在外面搞活动，也不敢声张，不敢跟软件工程沾边了。他的这一番话完全出乎我的意料，着实令我吃了一惊！

“软件工程无用论”和“软件工程虚无论”，在中国可谓由来已久。在我们这些天天和软件打交道的人看来，软件工程就像家常便饭，完完全全是一种日常工作和生活的方式，真的难以想象没有软件工程的软件开发会是什么样子。可是，经过一段时期的观察，我逐渐发现了这样一个严酷的现实，想不到现在国内还有这么多人（至少在某些专业的网站、媒体和论坛上）稀里糊涂地怀疑软件工程的重要性和必要性，对“软件工程到底是什么”的理解也是随心所欲、五花八门。

我反复思考了导致这一现象的方方面面的原因。不幸的是，看来很多人在看了《人月神话》之后（或许根本没有仔细看），就认为布鲁克斯所持的是一种悲观论调，进而从没有银弹论推导出“软件工程不是银弹”，认为软件工程基本上是无用的神话，宣扬软件工程是在追求空无缥缈、并不存在的银弹，30 年来软件工程毫无建树……等等等等。

可是他们错了，完全把《人月神话》的意思搞反了！

又经过一番认真仔细的调查研究，我发现了一个令我惊讶的事实：布鲁克斯 的“没有银弹” 说并没有错，可是我们大部分跟着喊“没有银弹”的人很有可能是错的，我们没有真正理解老布！

## 论点一：银弹不是万灵药的同义词

首先，我们需要明确一点：**银弹不是万灵药的同义词**，也就是说我们不能随随便便地在句子当中把“银弹”这个词当作“万灵药”来用。这一点证明起来并不困难。

**证明：**

采用反证法。

假设“银弹”等于“万灵药”，那么“OOAD 不是银弹，UML 不是银弹，用例不是银弹，RUP 不是银弹，MDA 不是银弹，软件工程也不是银弹”当然都是对的，因为世上原本就没有万灵药。

可是，转念一想，“世界上不存在万灵药”这个命题应该是个公理，本来就不需要证明（喜欢哲学、形式逻辑的朋友可能对此感兴趣），难道布鲁克斯教授前后间隔 9 年时间写了两篇重要的文章，花了这么大的精力，就是为了求证“软件工程当中没有万灵药”？这是不是很荒唐？显然这里存在矛盾，我们前面的假设不能成立，说明布式银弹并不是万灵药的意思，它另有含义。

## 到底银弹为何物？

那么，到底什么是“银弹”呢。下面让我们还是回到大名鼎鼎的**布氏预言**进行一番仔细的探究。在有没有银弹这个问题上，如果一定要较真，离开原文到处空喊“没有银弹”，恐怕不是科学的做法。

布氏银弹的原始定义：

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

我的译文是：

“没有哪一项单一的进展，无论是在工程技术还是管理技术上，能够独立地许诺在一个 10 年内，在软件的生产率、可靠性或简洁性方面，哪怕是取得一个数量级的提高。”

《人月神话》中文版[3]的翻译是：

“没有任何技术或管理上的进展，能够独立地许诺十年内使生产率、可靠性或简洁性获得数量级上的进步。”

请大家注意前后两种翻译的对比。我认为汪颖翻译的意思还是不错的，可惜不够准确，存在一个小小的缺陷：没有把“一个数量级”的“one”翻出来，结果就变成了含含糊糊的“获得数量级上的进步”（原书中好像好几处都有这个毛病，我看的是老版，请看过最新版的读者告诉我这个毛病是否还存在，如果改过了，我就把这段话删掉）。当然从整体意义上来看，这可能不是个大问题，但是老布原来说的到底是一个数量级，还是两个、三个数量级，严格地讲没有译出是错的，应该明确指出。

我们顺便计算一下银弹技术所要求的年平均增长率：

根据定义，有 $(1 + x)^{10} = 10$，故 $x \approx 0.26$

如果以上我没有搞错的话，布氏银弹实际上要求我们的软件开发生产率**每年**要保持平均 26%左右的增长率（或在 10 年内增长 10 倍）。在我们讨论有没有银弹的时候，当然不能离开 NSB 的这个基本判断标准。

## 论点二：布氏银弹之无聊面

当然，实现真正的布氏预言要求我们只能利用一项单一的技术就可以连续 10 年保持 26%的平均增长率，这对于软件工程实践来说可能确实有些困难（限制条件太多了）。1986 年后的第一个 10 年这个目标没有达到，现在离第二个 10 年也只剩下 3 年了，这次布氏银弹有望现身吗？假使依然没有如愿，我们是否应该为此感到悲观和沮丧呢？不！

让我们换一个角度来考虑更加现实一点的情况。其实，单纯追求布氏银弹在现实的软件开发环境中并没有实际意义。为什么这么说？至少有以下三点理由：

- 对于一个具体的软件产品或工程项目来说，不顾其他一味地提高生产率有什么意义？片面地提高生产率（或可靠性、简洁性）并不是我们真正的目的，企业只有在保证质量、真正满足客户需求并且能获得利润的情况下，追求高生产率、高可靠性或高简洁性才是有意义的。所以，如果一项技术或一件工具不是银弹，不能保证 10 倍的改进，我们并不能因此而随意地轻视它、拒绝它、抛弃它。

- NSB 为什么偏要规定只能采用单一技术？对一个具体的软件产品或工程项目来说，在实践中只要我们的组织还存在缺陷，只要是预计改进效果优于以往，能够保证总体成功、实现质量和业务目标的各种工程技术和管理方法，我们都应该勇敢果断地进行尝试，综合灵活地加以运用。

- NSB 要求 10 年提升 10 倍，可是在评估一项技术的时候，我们通常考虑的是它对我今年的项目、产品能否有切实的帮助，像 Grady Booch 那样世界顶级软件企业的首席科学家对于战略性技术也顶多考虑 3-5 年，可见谈论 10 年 10 倍是没有实际工程意义的。再者，某个工程技术、管理方法虽然有效，但是如果它给项目带来的改善没有达到 26%，比如只有区区的 20%、10%，难道就不好了，就应该抛弃，应该置之不理？

不难发现，老布在下“没有银弹”这个论断的时候确实采取了一些窍门，以保证至少 10 年 NSB 不会失效。可根据上面的分析，NSB 本身对我们的实际工作又有什么意义呢？至少我们**不能想当然地把“是不是银弹”作为指导我们开展实践，挑选、采用某项技术或工具的标准**。邱嘉文在写给我的信中认为：老布所谓的银弹，也许就是指的是“一项了不起的软件领域的革命性技术”。具体的数量上的判定标准，只是一个“虚指”，就好比我们常说的“一日千里”中的“千”。

那么，布老先生为什么要提出没有银弹之说，NSB 有什么用？当然这不是空穴来风。据我的了解和分析，原因可能是这样的：1986 年的时候，由于作为学术教授的他不满于**当时的** CASE 工具开发商所作的过多不实的宣传，于是借 NSB 一说揭示他们的工具存在着很大的局限性——不能解决软件工程的根本问题，而同时他又为我们指出了一条明路（“There is no royal road, but there is a road.”[2]），他心目中的银弹其实就是指那些未来能够解决根本问题的技术。所以，**布氏银弹原本不过是用来嘲弄国外某些 CASE 工具开发商过份宣传的一句玩笑话，结果却被我们国内一群无聊的发烧友没头没脑地拿来攻击软件工程！**

## 论点三：揭开真相—有银弹

作为革命性的核心关键技术，各行各业的银弹广泛存在，比如某些计算机硬件技术就是 N 倍银弹。为了保险起见，老布对软件工程 NSB 只前瞻了 10 年。可是现在没有，不等于将来没有。那么，软件工程的银弹是否真的永远不存在？

上文提到，NSB 是针对整个行业水平提出来的指标，对于某个具体的企业、项目来说并没有实际意义。我们要求的是获得综合性的整体增长，单一的银弹没有价值。此外，银弹要求增长一个数量级，也就是原来这个项目我要花 10 天，现在我只用 1 天就完成了，是原来的 1/10。这是否真的很难办到呢？

我想，对于国内大多数的软件开发机构来说，一个项目的效率或者质量提高 26%，甚至放宽到 30%、50%以上都不是一件很难办到的事，因为我们的工作中还存有大量低效、甚至错误的部分。我不相信无论在管理还是在技术上，我们的软件工程水平都高到了与世界先进发达国家平起平坐的那个程度，连 30%的再增长空间也没有了。（参见文末的证据）

（补充：增长率曲线）

实际上，我说老布心里是有银弹的，或者说他始终期盼着银弹或准银弹的出现，他在自己的文章中也多次重点提到并花了大量篇幅来分析银弹存在的可能性。具体来说，就是重视业务分析、需求分析和软件设计，越接近认识问题、解决问题的本质越好。而且，事实上他也已经点题了——《No Silver Bullet: Essence and Accident in Software Engineering》，告诉我们要区分解决根本问题和次要问题的技术，到了 20 世纪末 IT 界更要关注前者的研发和大力改进，我想这才是他写作这两篇文章的本意。可惜我们国内很多人出于各种原因，莫名其妙地把后半句忘掉了，或者装着没有看到，甚而把 NSB 错误地拿来否定软件工程。

根据布鲁克斯的分析，在 1986 年的时候有一些技术、工具和方法的确不可能是 10 倍银弹，比如高级编程语言、分时技术、统一的编程环境，因为它们解决的仅仅是次要问题，但有一些则是可能的、潜在的、发展中的银弹，老布已经给了我们许多提示，他提到了软件重用、需求精化和快速原型、递增式开发、优秀的设计，并认为其中一些技术相当有希望。如果我们按照这个思路，仔细观察分析一下国外软件工程这 20 年来的发展道路，可以发现类似的银弹（恐怕不能说是 100%的银弹，但追求 100%有意义吗？）已经出现或即将出现（甚至已经存在一段时间了，参见文末证据）。所以，如果我们说过去的某某编程语言不是银弹应该是保险的，但如果说“软件重用不是银弹，OOAD*UML 不是银弹，MDA 不是银弹”就要小心了，很有可能犯过于草率的错误。

和老布、许多人一样，我也是一个乐观的程序员，所以从心底里一直怀疑没有银弹论的真实性。幸好我不是孤立的。十年间已有几位大师（如 Brad Cox、David Harel）对 NSB 进行了有力的反驳，为此布鲁克斯在他的第 2 篇文章《“No Silver Bullet” Refired》[2]中进行了很好的诠释。

其实，我们真的没有必要去追捧所谓的布氏银弹说，那只是一篇举世名著的引子，一剂能激发你进行深入思考、认真观察现实的清醒药，当然，妄图用 NSB 来颠覆软件工程[4][6][8]则更是南辕北辙了。

反过来，我们应该找眼于把眼前的一个一个项目做好，哪怕能够让一个项目的质量或效益提高 10%、5%都算是一种成功，更何况让所有项目都 100%地按时按质成功完成都也已经是一种奢望了！做到这些，别无他法，软件工程是最重要的保障。我们确实应该听从老布的真言，更加认真和热情地关注业务建模、领域工程、软件的需求分析和架构设计、软件重用、软件项目管理等等当代国际先进的软件工程管理和技术领域，这些都是能给我们的现在和未来带来实际效益的“银弹”！

说实话，我们大抵是没有什么资格去和世界软件产业的领导者谈论什么银弹的。在 30 多年来已经发展成为一个庞大学科的软件工程的方方面面（我想到几个重点的，比如建模语言、架构、框架、设计模式、分析模式、软件过程、软件重用、领域工程、软件测试、配置变更管理等等），我们有多少原创性的东西，有几样拿的出手，并赢得世界学术界和产业界一致掌声的成果？**至今中国好像还没有自己的 OO 大师吧**。现在看来，跨入 21 世纪，我们仍然需要继续在更广泛的范围内做艰难的软件工程的启蒙工作。所以，**千万不要被误人的没有银弹论所蒙蔽！**如果我们不管 3721，把凡是自己不熟悉、不掌握的“新”技术、“新”方法都贬低为假弹、哑弹，把孩子和洗澡水一起抛掉，岂不是冒很大的风险？NSB 可能适合北美，但没有银弹论在中国泛滥的结果只能使我们更加落后。

邱嘉文写道：有没有银弹不重要，重要的是我们必须向前进，如果现在的某些软件工程方法不能带领我们前进了，我们就寻求改进就是了。如果银弹是“不断寻求改进”，那么，这样的银弹才是我们应该追寻的目标，并且无处不在。

## 论点四：重新定义银弹 — 一颗榴弹

根据前面的讨论，一方面至少我们不能排除布氏银弹存在的可能性（见文末证据），另一方面争论布氏银弹的存在与否可能从一开始就没有什么实际意义，而单单提出没有银弹说恐怕也不是布鲁克斯教授真实的本意。

所以，现在我们有理由修正银弹的定义。说白了其实谁都知道，**核心技术就是银弹，谁掌握了软件开发的核心技术，谁就掌握了银弹。**另一方面，关系到项目成败的核心问题也是银弹，不管它带来的改进是 10%，20%，还是 5 倍，10 倍。

在社会经济学上有黑猫白猫之说，在软件工程领域，我本人和很多朋友一样，更加看好**花猫和中西合璧的复方药**（您听到最近康复的凤凰台刘主播的新闻了吗？）**软件工程的本质就是一个不断取得各方面因素动态平衡的过程。**而且，在我心里，软件工程上能够解决实际问题、有效的银弹必然是一颗榴弹（子母弹）。

质量改进三元（架构、过程、组织管理）。

## 论点五：你的银弹不是我的银弹

……

## “大家都错了？”

既然银弹不是万灵药的代名词（见上文证明），那么把银弹当成了万灵药的人就可能犯了错误：第一种可能是在说废话，第二种可能是在不经意中忽略、排斥了那些真正对你有帮助的技术和方法。因此，我估计**那些说“没有银弹”的人中的大部分人，并没有真正搞懂银弹的含义，没有领会老布的精神，因而在某种程度上也就没有看懂《人月神话》这本书**。很可惜，我平日里敬佩的一些国外大师、专家们在引用 NSB 的时候也犯了同样的错误。

这里先引用几段误用 NSB（把银弹、万灵药混为一谈）的例子：

“多年以来，技术界执着地寻找解决开发过程一切问题的“银弹”，这些努力无疑是值得尊敬的，不过结果却令人叹息：尽管事实上这种银弹并不存在。从上个世纪六十年代末、七十年代初开始出现的软件工程（Software Engineering），曾被当作包治百病的万应良药”。[6]

（评语：可见作者是听说过有银弹这么一说的，但他显然并没有搞懂到底什么是银弹，因而这段话也就成为了无聊的废话，构成了后面论证的错误前提和一个假象，因为如前所述，别人——真正的软件工程专家从来不会把软件工程当作万灵药，世界上从来也没有什么万灵药。）

（其他著名的例子待补充……）

如果我的判断是对的话，为什么会有这么多人道听途说，以讹传讹，跟着犯这个明显的低级错误，形成圈内一个热热闹闹的“没有银弹现象”？我想，原因就是大家一直耳熟能详的、媒体为我们总结的“浮躁”和“低水平重复”，根本原因就是没有静下心来，好好学习，认真读书。

## 没有银弹论者的心态分析

……

## 论点六：走下神坛的布鲁克斯

在我眼里，布鲁克斯教授好像并非真正意义上的软件工程“大家”。

《人月神话》也并非正规的学术论文集，而是汇集了软件工程发展的早、中期各种深邃思想和精辟观点的札记（或杂记），而且还有可能存在一些小的语义矛盾和问题（有待进一步研究），当然这是世界软件史上一部非常难得和宝贵的作品。

另一方面，我们也要为时下某些 IT 出版社、所谓的最专业媒体的市场策划以及各种形形色色的枪手、托们进一言：**天底下哪来那么多的圣经和必读经典？**据我所知，自创世纪以来世界上圣经好像只有两个正式的发布版本：《新约》和《旧约》（请信教的朋友指正）。按我的理解，圣经必然是要我们每个礼拜时不时地捧起来虔诚地诵读的。软件界存在圣经吗？我们不要人为地制造新的神话，以免将来被人笑话。

离老布首次提出 NSB 竟然已经过去 17 年了！有意思的是，RUP、Use-Case、UML、OOAD、MDA、CBD、设计模式、架构、框架、分析模式、重用、领域工程、业务工程……等等等等，**几乎 17 年来软件工程所有主要的努力、取得的重大进步都是朝着解决布老提出的“根本问题”这个方向去的**。想到这里，令人不禁油生敬佩之情，惊讶于他老人家的眼光。可惜，国内业界似乎很少有人看到这点，只有一大片浮躁、浅薄甚至错误的没有银弹的声音。我想布氏银弹的真正意义和价值正在于此，他有先见之明，并用他自己独特的方式为业界指明了方向，大家已经而且正在朝着这条道路前进，这就是他的伟大之处。

## 当代淘金记

最后，我想给大家讲一个有趣的故事（纯属虚构，切勿对号入座）：

世界杯淘金赛如火如荼，黑马中国队遭遇尴尬

（2004-3-26 IT 之源资深记者张恂于索夫特威尔山脉全球独家报道）

据传，举世闻名的索夫特威尔山脉的群山之中蕴藏着储量巨大的金矿。为此，不久之前，世界上各个国家、地区纷纷派出自己的能工巧匠组成梦之队，雄心勃勃地加入到一场激动人心、史无前例的世界杯索夫特威尔淘金大赛中。现在，这场比赛还未进行到第 5 周，就已经出现了戏剧性的热闹场面。这里本记者将向大家重点汇报美国队、西欧队、日本队、印度队和中国队这 5 支热门队伍的进展情况。

首先分析一下各队的实力。美国队，大家很熟悉了，历届夺宝大赛的冠军得主。他们队员的胆子大、脑筋活、情报灵，很善于总结自己和他人的经验教训。说实话，在本记者眼里，美国队个人的实力并不是最强的，可能还不如中国队，但奇怪的是，美国队好像有种魔力，总能吸引到世界各地的各路高手加盟，所以他们的整体实力总是保持第一。

西欧队实力排名第二，但他们的现场分析能力和组织纪律性在所有各队中是公认最强的。

日本队一开始缺乏自信，工作方式老是模仿美国队，亦步亦趋，但后来经一位叫戴明的美国队退役老向导的指点，很快赶了上来，不但挖到了不少金子，还顺便挖到了很多名贵的依莱克钻石。目前他们的实力位居第三，有时甚至还超过美欧。这三个队咬得最紧，有得一拼。

印度队和中国队一样，同属第二梯队，本属新秀，但同样是有贵人相助，得到了亨弗利、吉尔伯两位高人老向导的指点，水平提升很快，后来大部分队员干脆就给美国队打起零工来，帮着打炮眼、运矿石、淘金泥，结果也赚得个盆满钵满。

不过，中国队挖到的金子其实比印度队多，而且他们是各个队伍中自尊心最强的一个队。由于历史上他们曾经挖到过 4 大名石，加上这回又招募了大量武功盖世的挖宝高手，还没有开挖就被各大传媒评选为最聪明、最耀眼、最靓的黑马，因而队员们个个豪情壮志、势在必夺。然而，不知为什么，赛了两周中国队目前在总体上还是有些落后，进步不够明显。不过也不用灰心，可能是因为他们晚到的缘故吧。

各队首先在进山入口处的大矿——普罗古兰敏山区挖掘。美国队最先到达那里，并且掘到了第一桶金，淘光了那里大部分已探明储量的矿山。从第 2 周开始，在各队还在迷恋普罗古兰敏山的秀丽风光和看似巨大的储量的时候，美国队就已经开始向深山进发寻找更大的金矿，并陆续有了新的斩获。

可是，美国队不知是过于大方还是粗心大意，一路上遗留下了很多找矿的路线图、矿井分布图和各种挖矿的工具设备。可能是由于美国队太成功了，逐渐有些美国队队员开始离队，开始为其他各队担任起向导、顾问和后勤来了，他们卖图纸、卖筛子、卖各种机械，甚至还卖可乐，也忙得个不亦乐乎。他们还帮助本队物色、雇佣、吸纳了不少出色的国际外援，现在美国队里差不多有 1/3 的队员原先来自印度队，1/4 原先来自中国队。

目前按照收获程度，领先的依次顺序是：美国队（几乎占据了所有已知的大矿，并几乎主导了各大新矿的发掘），西欧队和日本队差不多，中国队、印度队挖到的金子最少，虽然也时不时地挖到一些特大金块。

此外，在大约进行到第 2.5 周的时候，曾经有一位叫布鲁克斯的资深老向导告诉传媒说，他看不下去了，他向各队预言道："你们这是在瞎忙乎，这里已经没有什么真金白银可挖了，新的富矿在比泽尼斯山、瑞哈门特山、迪赞恩山……如果你们相信，就朝那里走……"。想不到天生幸运的美国队居然听信了这番话，派出大队人马沿着布鲁克斯指点的道路纷纷向新矿区进发，而且仅仅用了不到 1.5 周的时间，果然发现了比普罗古兰敏山储量大好几倍的举世大矿藏。

可是布鲁克斯的这句话一传百传，传到了中国队耳朵里不知怎么却只剩了半句话，变成了"你们这是在瞎忙乎，哪里还有什么真金白银可挖呢？"。这下，许多一直在普罗古兰敏山挖矿的中国队员感觉一下子失去了信心，开始整日为"继续驻留在原地深挖，还是重新找新矿，还是干脆放弃、退出比赛"而争论不休，结果眼睁睁看着手边可挖的金子越来越少。

黑马中国队遇到了尴尬。您说，中国队应该怎么办呢？……

## 临别感言

我发现，软件史上至少有两个著名的软件工程经典论断（一个是软件开发的瀑布模型，另一个就是没有银弹论），在不经意之中从侧面帮了美国人大忙，帮助他们在软件工业上确立并保持了长期的世界领导地位。真正聪明的美国软件人，根据自己内心真实的判断，并没有听命于这些所谓的经典判断，所以他们尽管经历了种种坷坷绊绊，但仍然取得了举世瞩目的辉煌成就。正如上面所讲的这个当代淘金故事，没有人确切地知道他们到底挖到了多少软件工程（当然也包含软件工艺）的"金子"，我想数量一定是惊人的（有谁愿意露富呢？）。可是，在如何看待这件事情上，好像还有很多中国的软件人没有清醒，他们固执地以为天下已经没有金子可挖了。

我们当然不能怪文斯顿•罗伊斯和布鲁克斯老先生，人家写著作是出于好心肠，是就事论事，并没有说错，说的也的确是真话，只能怪我们自己，谁叫我们没有认认真真地当好小学生，没有学会独立思考、真正领会他们的精神，凡是外国著名专家、著名大师、著名机构提供的经验、开的方子一概都乐意照单全收呢？

我的建议是：如果你通过仔细的评估，预计到有什么开发技术或管理方法能使你的项目或产品获得短期和/或长期的成功，那么就勇敢地去尝试，理性地去分析和总结，即使偶尔失败也不要气馁。这些事情并不难做到，因为你肯定不是世界上第一个这么做的人，所以千万不要再没头没脑地说这个不存在，那个没用。请大家务必关注自己身边时不时浮现的银弹、铜弹或铁弹，抓住每一个改进的机会，奇迹就在脚下！

## 银弹（准银弹）存在的候选证据

软件重用方面：

1）Several organizations have obtained reuse levels around 90% in certain projects or areas: … Motorola: 85% reuse and a 10:1 productivity savings ratio in compiler and compiler-tool test suites … [7, p6]

2）Our experience and our knowledge of what many organizations have achieved through reuse persuades us that management may expect substantial gains: … Time to market: reduction of 2 to 5 times; Defect density: reductions of 5 to 10 times; Maintenance cost: reductions of 5 to 10 times … [7, p6-7]

3）浪潮通软早先推出一个新产品需要两年，用了新模式只需 8 个月左右（注：3 倍准银弹，不错）。西部世纪软件用上了“柔性组装”之后，人工成本降至传统开发模式的 1/4（注：4 倍准银弹，不错）。总之“变革非常大”…… 选用标准化和成熟化的模块组装应是最有效的方法，后者可以把缺陷密度降至原来的 1/5 到 1/10。[5，A24-A26]

人才方面：

4）超高产开发人员的生产效率一般是普通开发人员的 4-10 倍……超高产开发人员一般只占整个开发群体的 1%或 2%。[1, p169-170]

（更多数据待补充……）

## 参考文献

[1] Booch G.著，邢春丽等译，《面向对象项目的解决方案》(*Object Solutions: Managing the Object-Oriented Project*)，机械工业出版社，2003年8月第1版。

[2] Brooks F. P., Jr.著，《人月神话》（影印版），中国电力出版社，2003年3月第1版。

[3] Brooks F. P., Jr.著，汪颖译，《人月神话》，清华大学出版社，2002年11月第1版。

[4] 第二书店，《软件工艺》广告“颠覆软件工程，开发回归人本.....”，http://www.dearbook.com.cn/subject/softwarecraft/index.xml

[5] 高丽华，《软件进入“PC时代”》，计算机世界报（第10期，总第984期），2004年3月22日。

[6] 韩磊，《开发回归人本：软件工程遭遇危机？》，第二书店，http://www.dearbook.com.cn/subject/softwarecraft/09.xml

[7] Jacobson I., Griss M. and Jonsson P., *Software Reuse: Architecture, Process and Organization for Business Success*（《软件重用》影印版），世界图书出版公司北京公司，1998年3月第1版。

[8] 熊节，《软件工艺》译者评论，第二书店http://www.dearbook.com.cn/subject/softwarecraft/07.xml

征稿

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

Agile软件开发丛书

PEARSON Addison Wesley

# 有效用例模式

# Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

清华大学出版社

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

# 探索需求

## 设计前的质量

需求之于开发，就像婚姻之于人生

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

# Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社

# UML 2.0 新特性

# 第一部分（活动图，类图，通信图，用例图）

Granville Miller 著，郝黎凯 译

查看评论

## 引言

统一建模语言（UML）是以可视化方式描述软件系统的结构和行为的标准语言。UML的2.0版 期望可以在2004年4月30号发布。本白皮书描述了新版UML的外部特征。

UML2.0在可视化建模方面有了许多的革新和增强。它可以描述现今软件系统中存在的许多技术比如模型驱动架构（MDA）和面向服务的架构（SOA）。本白皮书将会描述这些革新和增强的部分。

本文并不是UML教程，它的重点在于新版本的增强特性上，读者需要对UML的早期版本的有一定的了解[1]。它特别适合那些已经在使用UML1.1来进行软件系统设计的读者。本文的目的是提供一个对UML2.0的一个比较精炼的介绍，使得读者可以很快的了解到UML2.0的重要特性。

本白皮书的内容依据被对象管理组织（OMG）采纳的UML 上层结构规范（Superstructure Specification）（ptc/03-08-02）。虽然在该规范最终版本确定以前，UML规范的内容还有可能发生变动，但是本文中描述的UML元素发生变化的可能性不大，另外如果规范有变动，我们将会发布一个补充文档。

对UML比较陌生的读者可以访问http://bdn.borland.com/together。参考《Practical UML™: A Hands-On Introduction for Developers》

## 活动图

我们首先来讨论活动图。活动图是UML所有图中变化最大的一种。活动图的目的发生了相当大的变化，它本来用于描述工作流程(以下简称“流”)，现在它也包含一些必须的新特性使得它可以支持工作流程的自动化(automation)。

第一个值得注意的变化是活动图的结点不再叫做活动（Activities），而是改称动作（Action）。活动成了一个更高层次的概念，它包含一个动作序列。因此一个活动图展现了一系列的动作，这些动作一起组成了活动。

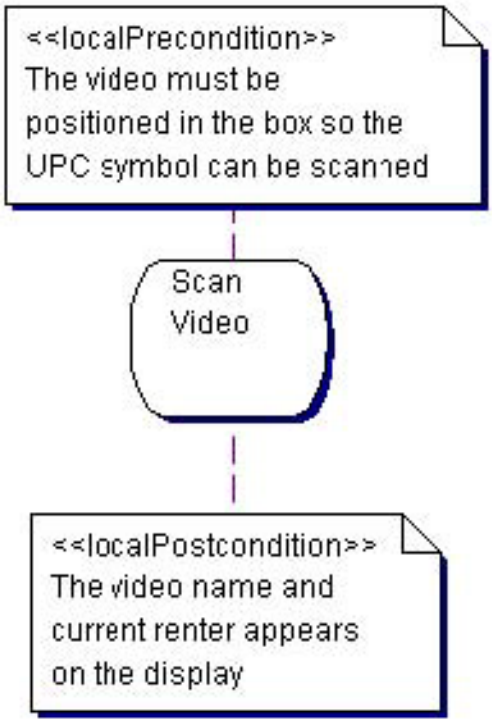



图1 动作，前置条件和后置条件

动作拥有前置条件和后置条件，这些约束表现为附着在动作上具有相应构造型的注释。如图1所示：动作要触发必须满足前置条件。动作完成后必须满足后置条件。

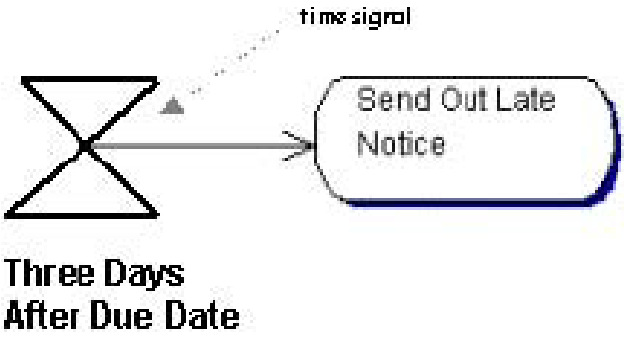



图2 触发活动的定时信号

动作可以有多个输入流，在UML2.0中，当所有的输入流都触发时，动作才会触发。信号是另外一种触发动作的方式，UML2.0引入了一种新的信号－定时信号，预先指定的时间段结束后，定时信号就被触发。

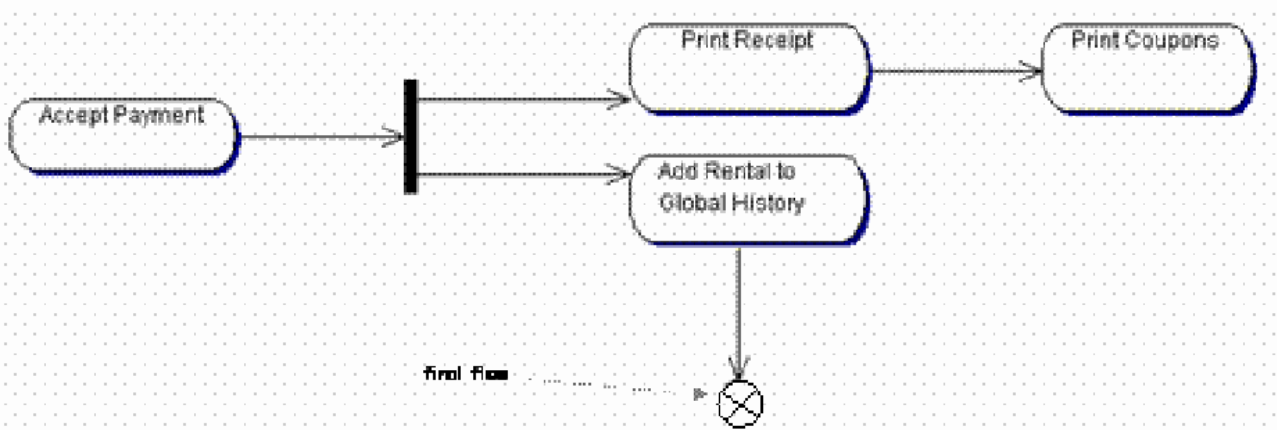



图3 流终结点

流终结点（Flow Final）是一种新的控制流符号。它类似于活动终结点(Activity Final)。不同的是流终结点仅表明活动图中的一个流结束了，活动图中的其他流仍然可以进行。而活动终结点意味着活动已经完成了。这两类终结点都不能有出边。

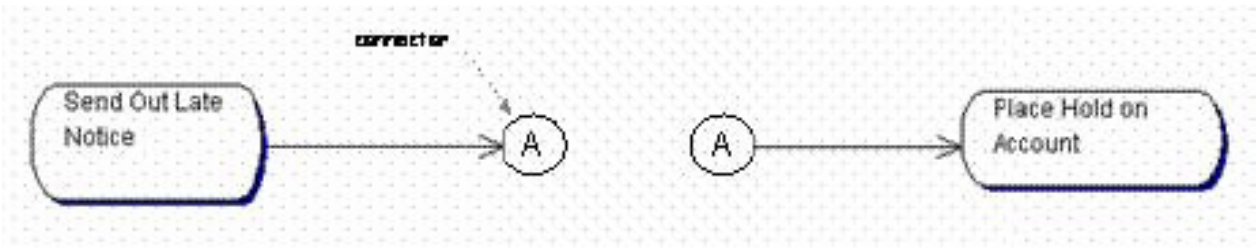



图4 连接器（Connector）可以使流可以跨越多个活动图

最后一个扩展是连接器。连接器描述了流跨越活动图，从一个活动图延续到另一个活动图的情形。如图4所示：在两个活动图中有两个名字都是A的连接器，这意味着左图中的A和右图中的A在逻辑上是同一个结点，动作“Send Out Late Notice”的下一个动作就是“Place Hold On Account”。

**组合**

UML2.0包含3种新方式的对动作进行组合或者分解：子活动；扩展区域；扩展泳道线。例如：一个动作可以被分解为一系列其他动作的组合，这些动作的组合就是子活动。如图5所示：“Return Video”动作就实现为一个子活动。

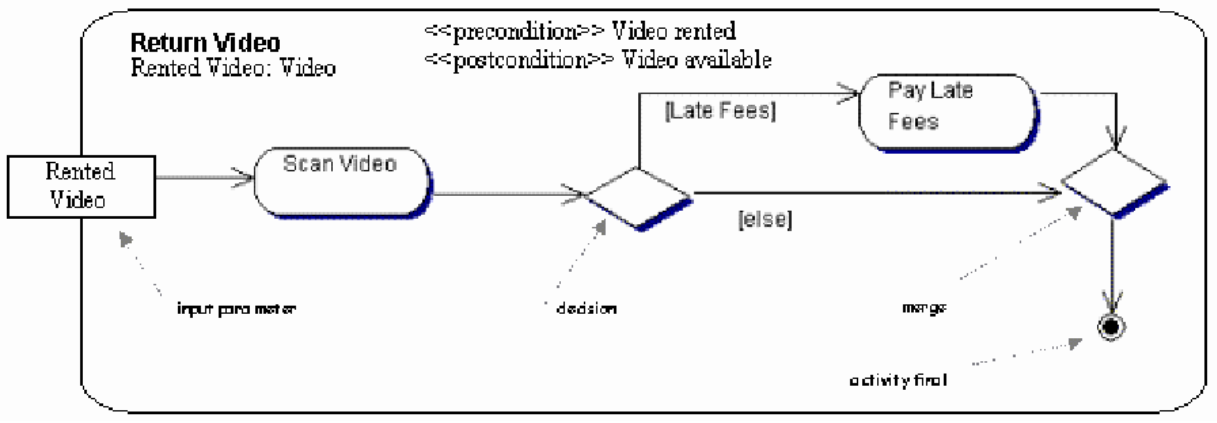



图5 子活动图

活动除了可以通过活动图来描述外，还可以通过活动结点来显式声明。活动名字在活动结点的左上角，名字下面是参数和类型。活动的逻辑（一系列动作）包含在活动结点中，前置条件和后置条件可以放在活动结点的中心位置。

活动通常有输入输出。例如，“下订单”活动可以接受一个定购清单并发送要求的货物（或者简单地发送一个装运清单）。活动结点图的边沿可以放置输入输出对象来表示输入输出参数。输入参数在左边，输出参数在右边。

另外一种用来组合动作的方式是扩展区域（Expansion Region）。扩展区域是一个活动的嵌套区域。它的输入是一个集合。对于集合中的每一个元素扩展区域中的活动都被执行一次。其实，扩展区域是一种迭代，它从语义上类似于输入集合上的“for”循环。

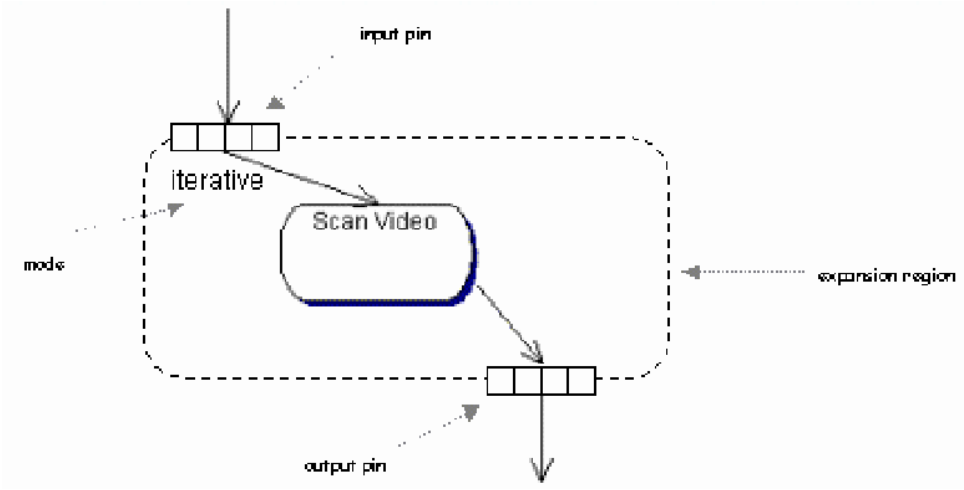



图6 返回多个video的扩展区域

扩展区域的输出集合必须和输入集合类型一致。实际上，不仅集合的类型要一致，集合中值的类型也要相同。当扩展区域执行的时候，输入集合的某一个位置上元素对应的输出会被插入到输出集合的相同位置上。输入输出集合大小相同。

扩展区域有三种执行方式：并行模式，迭代模式，流模式。并行模式就是扩展区域的各个执行对象可以同时执行。迭代模式则要求这些执行对象顺序执行，一个执行完了才能开始另外一个。流模式中，所有的输入同时进入扩展区域，扩展区域可以在整个输入集合自由操作。扩展区域的左上角是执行方式的标记。

扩展区域中的单个动作由一个速记符号来指明。动作符号不能出现在活动区域中，他们只能和输入输出脚相连。

新版的UML中还对泳道线进行了扩展。通常我们会碰到画多维泳道线的问题。如下图中所示：Raleigh 的架构师负责一个动作，而California的商务分析师负责另外一个动作的情况如何用单个泳道线来建模呢？

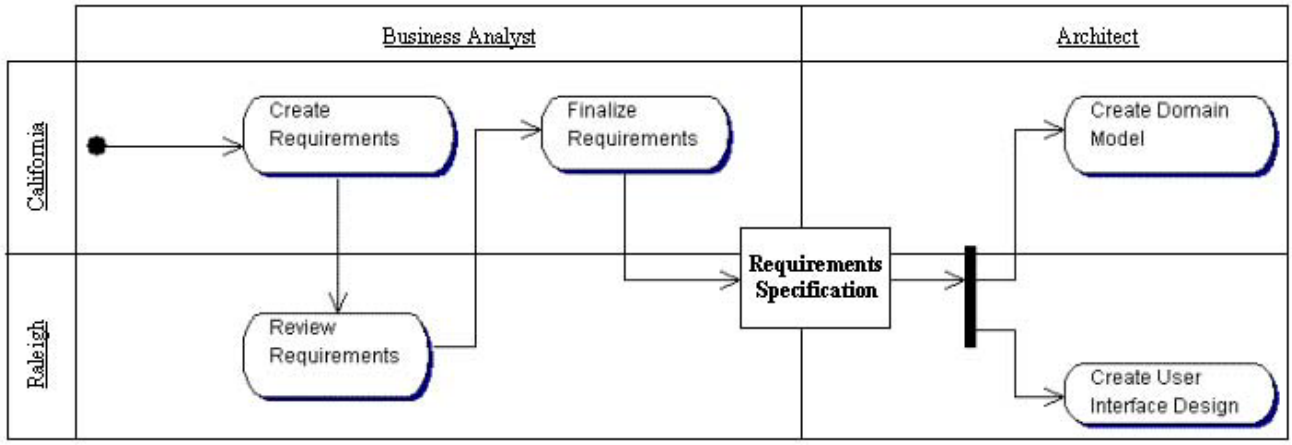



图7 使用活动分区来表示动作的角色和位置

为了解决问题，UML 2.0 引入了活动分区（Activity Partition）。活动分区根据活动的一组特征（比如角色，地域，组织等）来组织动作。分区可以嵌套使得维度可以多于两维。如图7所示的活动既有地域属性，又有部门属性。

**异常**

异常处理由两个阶段组成。首先，我们在代码中的某处抛出异常（一个异常可以被多次抛出）。通常，可能抛出异常的代码被放在一个代码块中，在Java语言中，该代码块作为“try”语句的一部分被放在一对大括号中。

在异常处理的第二阶段，异常被捕获并被放在“catch”块中的代码进行处理，“catch”块是可以嵌套的，其中的处理应该能够处理异常反应的问题。

有时候，异常反映的问题可以解决，那么程序可以继续执行。当问题不能解决的时候或者需要用户介入处理，或者程序终止运行。

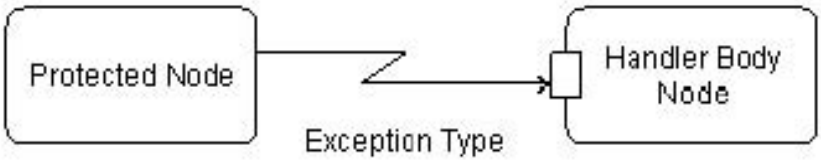



图8 UML2.0 异常处理图符

在UML2.0规范中，“try”块在活动图中被称作“保护结点”(Protected Node)，而“catch”块则被称作“异常处理实体结点”（Handler Body Node）。异常发生的时候，需要在一组异常处理实体结点中查找可以处理这种异常的结点。如果有匹配的，那么对应的处理实体结点就会执行。如果没有匹配的，异常就会被向当前“try”块的外部传播（如果“try”块外还有“try”块，那么异常就会被外部的“try”块捕获，如果一直没有捕获，最终会被系统捕获）。

以一个简单的浏览器为例，它的功能是提示用户输入地址（URL），并把地址可以定位的资源在显示器上显示。该活动的逻辑是：

1. 打开新的地址

2. 打开显示窗口

3. 拷贝内容，

4. 关闭和地址关联的流以及显示器

如果一切顺利，资源的内容会被正常显示。

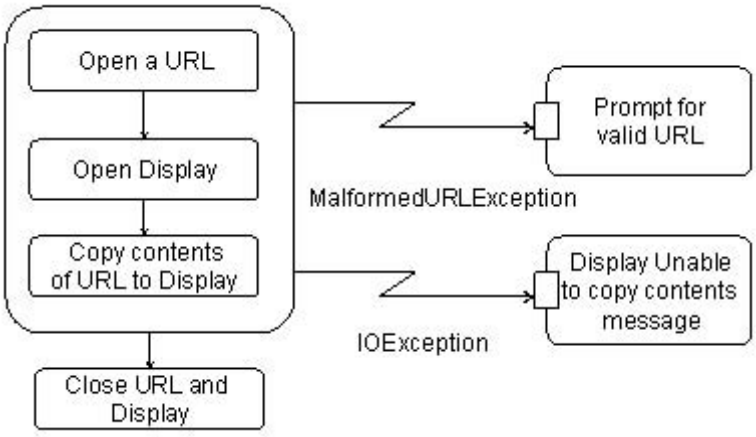



图9 一个简单浏览器的异常处理

在上述的过程中，有两个地方可能会出现异常。一个是URL格式可能不正确而导致出现非法地址格式异常(MalformedURLException)，它会在打开地址的时候被检测到。另外一个异常是输入输出异常(IOException)，它可以在很多地方发生，例如地址可能是有效的，但是地址所在的资源不可读或者显示不可用等。

在如图9所示的例子中一共有4个活动，三个放在保护结点中（UML2.0中，活动可以嵌套）。另外有两个“异常处理实体结点”代表我们上述的两种异常。异常处理实体结点和保护结点的后继处理是相同的，因而我们可以把关闭地址和显示窗口活动作为我们的最终(finally)结点（译者注：类似于Java语言中的“finally”块）。

## 类图

类图是UML中最熟悉和常用的图了，它的变化不是很大。仍然是由名字，操作，属性组成，并且类的名字，每个操作和属性分别放在一个单独的格子中。但是对于属性的表示方式有所增强。将属性建模为字符串还是关联关系一直是个问题。把属性表示为方格中的字符串的好处是可以表示属性的可见性(visibility)。将属性建模为类的关联关系的好处是可以清楚地表现出属性的多重性。新版的UML使得这两种建模方式的可以等价。如下图两种表示方式的属性都既可以表现可见性又可以表现多重性。

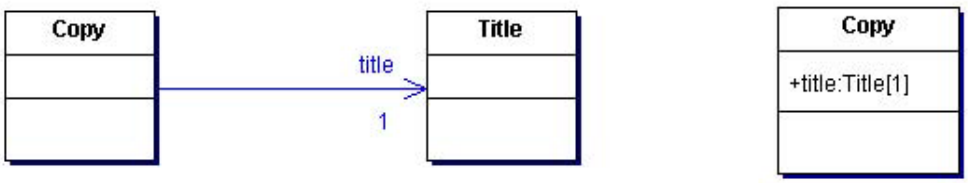



图10 类属性的两种等价表示

为了支持这种等价的表示方式，属性的定义方式得到了增强。大多数的变化发生在UML1.5版中。下面我们来分析一下属性的特征(Property)来展现属性定义语法上的改变。属性用如下的语法来定义：

[visibility] [/] name [: type] [multiplicity] [=default] [{property-string}]

可见性[Visibility]特征用单个字符来表示。有四种可见性，分别用四个符号表示：公开的(+),私有的(-)，保护的(#),包级别可见(~)。；

“/”代表可导出(derived)特征，如果一个属性可以通过其它属性来构造，那么该属性就是可导出的。例如：线段的长度属性可以由该线段的两个端点属性推导出来。可导出特征是UML2.0在类图的属性定义中引入的新特征；

多重性[multiplicity]在没有明确指定时为1；

缺省值[=default]定义沿袭自UML1.x；

特征串[{property-string}]是由若干字符串来表述的属性特征，有如下特征字符串：

{readOnly} ：只读属性；

{union} ：表明可导出属性是由其他属性联合的结果；

{subsets <property-name>}：表明属性是一个继承得到的属性的子集；

{redefines <property-name>}：改变一个继承得到的属性的名字；

{ordered}：某类型的一个有序集合,集合中的元素不能重复；

{bag}：集合中的元素可以重复；

{seq} 或 {sequence}：集合元素可以重复，并且有序；

{composite}：组合属性；

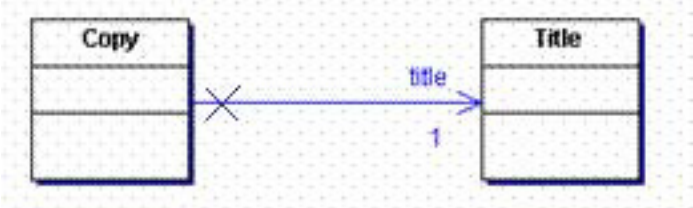



图11 导航关系

类图中泛化的用法和原来相似，但是类的导航（Navigate）关系有了一个新的图符来描述。如上图所示，在两个类的导航关系中，如果导航关系的一端有箭头，则箭头代表了导航的方向。然而，如果有一个”x”出现在导航箭头的起始点上则表明这种导航关系可以不发生。如果既没有箭头，也没有“x”，则表明导航是双向的。

## 通信图(Communication diagrams)

通信图就是过去的协作图(collaboration diagrams)。和序列图中的术语一样，通信图的结点被称作生命线(lifeline)。这些结点通过消息连接在一起，这些消息代表了交互作用发生的顺序。

消息可以被并行发送，它的表示方法是在序列号后边放一个字母。比如：在录像带出租事务的结尾的两个消息：“打印收据”（Print Receipt），“添加出租记录到全局历史记录”（AddRentalToGlobalHistroy）可以被并行的发送来减少处理时间时。从“10a:printReceipt” 10b:addRentalToGlobalHistory”可以看到，他们的序号相同，但序列号后边分别有字母a,b代表这两个是并行的消息。

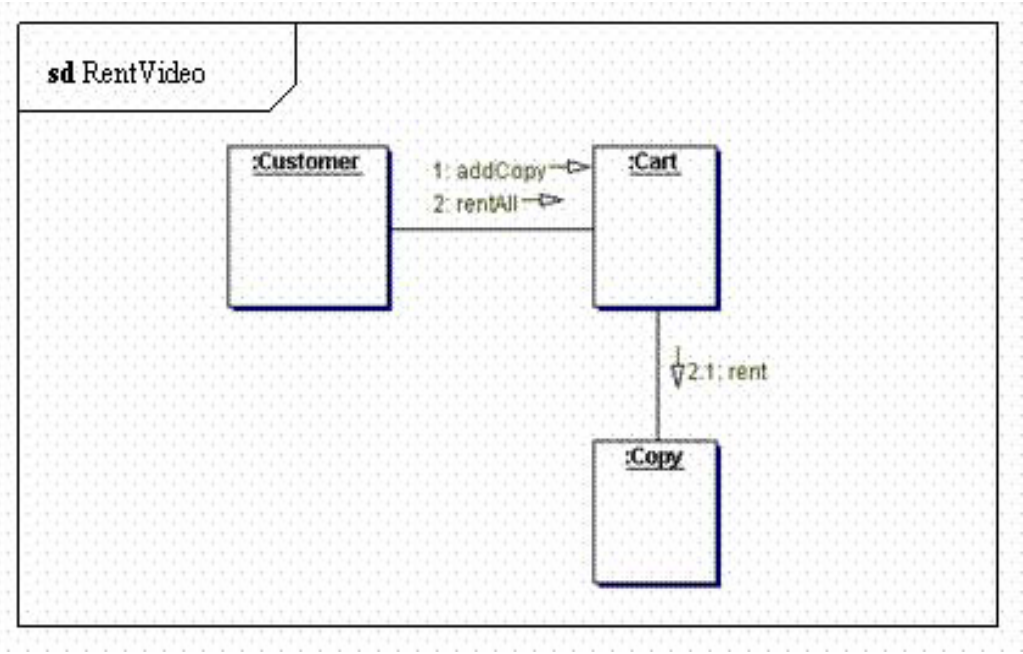



图12 出租录像带的通信图

最后，通信图可以被放在一个框架(frame)中，框架是一个方形的容器，容器左上角是交互的名字，容器内部是生命线和消息。

## 用例图

UML1.3中加入了扩展点来表现用例是如何扩展的。在UML2.0中，扩展点功能进一步加强了。它展现了一个用例扩展另外一个用例的逻辑必然性。另外也清楚地指出了扩展的确切位置。

我们来看看电话会议用例如何扩展打电话用例的。首先开始普通电话用例，然后在电话会议一方的电话机上，轻轻按下挂起键(flash)，并且拨下特征激活号码，听见拨号音的时候，就可以电话会议另外一方了。这里，基础用例清楚的被扩展了。

为了展示用例扩展的条件，一个注释被附加在扩展关系上。这个注释表明了打电话用例扩展的条件和扩展点。这样就将扩展点表达得更加确切。

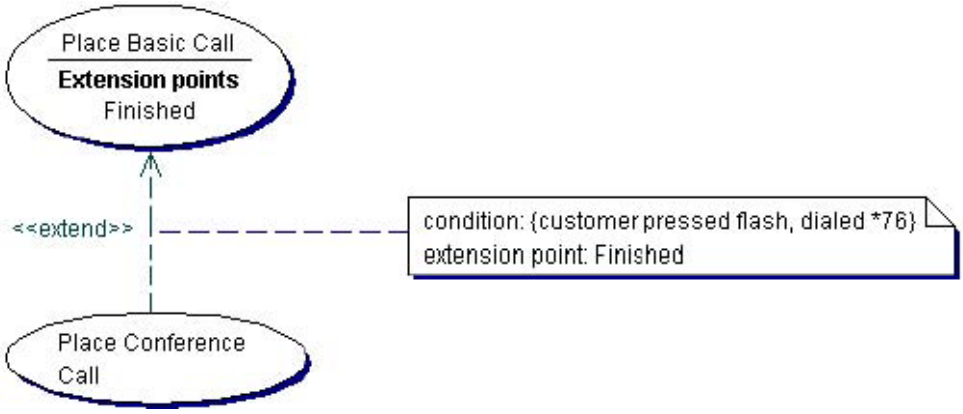



图13 用例扩展的条件

## UML 一致性

UML的早期版本没有说明怎么才叫对标准的支持。结果是UML工具厂商们可以比较容易的声称自己支持的UML特性是符合标准的。一方面这是有利的，因为厂商们可以更好的发展UML；另外一方面则是不利的，它导致不同厂商之间的工具产生的模型之间非常难以交互，而从一种工具产生的模型转换到另一种工具产生的模型则会丢失很多信息。

UML新版本定义了38个兼容点。兼容点是一个UML领域（例如“用例”）。所有的UML2.0实现都必须实现一个兼容点(Kernel)。其它的37个（包括用例在内）都是可选的。这样，用户就可以知道某一个UML建模工具支持哪些模型元素，以及对这些模型元素支持的程度。

对每一个兼容点，有四个兼容（兼容等级），它们刻画了兼容的程度：

•不兼容：实现和兼容点的语法，规则，图符不兼容

•部分兼容：实现和兼容点的语法，规则，图符部分兼容

•完全兼容：实现和兼容点的语法，规则，图符完全兼容

•互操作兼容：实现和兼容点的语法，规则，图符完全兼容，和兼容点的XMI模式也完全兼容

对38个兼容点的支持程度构成了三个兼容级别。基础（一级），中等（二级），完全（三级）。基础兼容需要实现头10个兼容点。中级兼容还要实现接下来的15个兼容点。完全兼容则要实现所有38个兼容点。换句话说：基础兼容需要兼容类图，活动图，交互图，用例图；中级兼容需要兼容状态图，轮廓图(profile)，组件图和部署图。如果要达到完全兼容，则还要兼容行为图(Action)和其它一些高级特征。

## 结束语

UML正在迅速的成为可视化建模环境来满足当今软件开发中对于技术（例如：异常）和沟通（例如：活动分割）的需要。UML覆盖的领域也在扩展中，它不再仅仅用来描述软件系统。随着SOA和MDA的兴起，UML不仅需要成为一个平台独立的系统描述语言，而且必须能来描述和自动完成商业流程。

最终，UML是一个工具框架，用来帮助我们更好的开发软件。在最新的版本中，这个框架变得相当的大了。这些工具是必须的，用来解决不断变化的挑战。从这个框架中选择正确的工具要依靠教导和经验。

本白皮书用来开始一个教导过程。这一部分描述了13种图中的4种。余下的两部分内容中描述了其它图的信息。

## 参考文献

Armour, Frank and Granville Miller. Advanced Use Case Modeling: Software Systems, Addison-Wesley, 2000.

Fowler, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Addison-Wesley, 1999.

Miller, Granville. “The Latest Status of Version 2 of the UML,” The Coad Letter, Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30374,00.html

Miller, Granville. “What's New in UML 2? Model Exceptions,” Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30169,00.html

Miller, Granville. “What's New in UML 2? A Question of Profiles,” Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30168,00.html

Miller, Granville. “What's New in UML 2? The Use Case Diagram,” Borland Developer Network, 2003, http://bdn.borland.com/article/0,1410,30166,00.html

OMG, UML Superstructure Specification (ptc/03-08-02), http://www.omg.org/

**本文原文为Borlabd公司白皮书，获得Borland公司的许可刊登。**

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

# Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

Scott Ambler

The Addison-Wesley Signature Series

# Patterns of Enterprise Application Architecture

中译本即将上市

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

UMLChina 训练教材

# ChattaBox：使用 UML 和 SDL 设计并发通信软件系统

P S Kritzinger 等 著，车皓阳 译

查看评论

## 摘要

本文所描述的案例研究的是并发通信系统（CCSs）的软件工程。我们使用最佳实践软件工程方法学来说明和设计 VoIP 系统，而后实现。方法包括统一建模语言（UML）以及规范和描述语言（SDL），对于特定项目也混合使用两者。

这里实现的 VoIP 系统，称作 ChattaBox，允许用户通过语音以及其它一些功能进行通信。系统需求和静态设计是用 UML 图来进行的。动态设计最初也用 UML 来做，随后就转向 SDL，使用 Telelogic 提供的工具来做。SDL 设计的结果使用这个工具来验证。

最终系统要验证正确性、性能和可用性。它要满足工程过程最初阶段定下的所有要求，同时还要保持稳定，兼容协议。在工程过程自身完成评估之后，我们得出结论，软件工程标准对于 CCS 工程领域来说是非常重要的。更进一步，UML 对于高层设计功能很有帮助，但不能充分提供协议验证。在软件工程过程中提出的最大的缺点就是不充分和易于出错的转换过程，那需要手工除错和人为干预，转换 SDL 图弥补了这一点。

本项目由 CapeTown 大学 DNA 实验室承担。DNA 实验室与南非法国大使馆资助的 INT（巴黎近郊）有一项正式的研究合作协议。这个项目还由国家研究基金（NRF）以及与南非西门子通信公司的合作项目 THRIP 资助，Telkom 作为工业成员也参与了这个项目。

## I. 引言

在今天这个复杂技术构成的世界里，有效的软件工程过程对于开发软件制品来说十分重要。然而，软件工程领域仍然没有提出一种方法，适合开发所有类型的系统。本文关注于 CCSs 系统的工程设计。使用移动通信和网络协议的应用程序是这些系统最基本的一些例子。我们研究了一个案例，用以调查并发通信软件系统的工程过程。案例研究的主要目标是分析使用最佳工程实践和 CASE 工具创建这样的系统时，可能的软件工程路线。为了这个目的，我们设计并实现一套称为 ChattaBox 的 VoIP 系统。

起初，创建这个软件的基本需求是要支持 VoIP 通信[1]。但是，为了使它变得足够复杂，可以作为案例来研究，ChattaBox 演化成了一个全面的工业强度的分布式应用程序。加入了象语音邮件和动态地址这样的服务。而且，我们还决定在 ChattaBox 系统服务器端应该支持负载均衡、安全和数据管理。

在这次案例研究的过程中，我们有效地组合了现有软件工程方法，用以构建复杂的软件系统。现有的设计方法有 UML[2]和 SDL[3]。而且，在这个领域中面向对象设计范型的有效性也同时得以检查。

而后提出了正向工程方法[4]。与这种方法相一致，规范和最初的设计用 UML 完成。之后，UML 设计图转化成 SDL 规范，进行校验和验证。开发过程如图 1 所示。

为了实现正向工程过程，需要有一种方法把 UML 设计转换成等价的 SDL 规范。为了这个目的，我们布署了 UML suite 4.5 以及由 Telelogic 提供的 SDL 和 TTCN suite 4.3[5]。

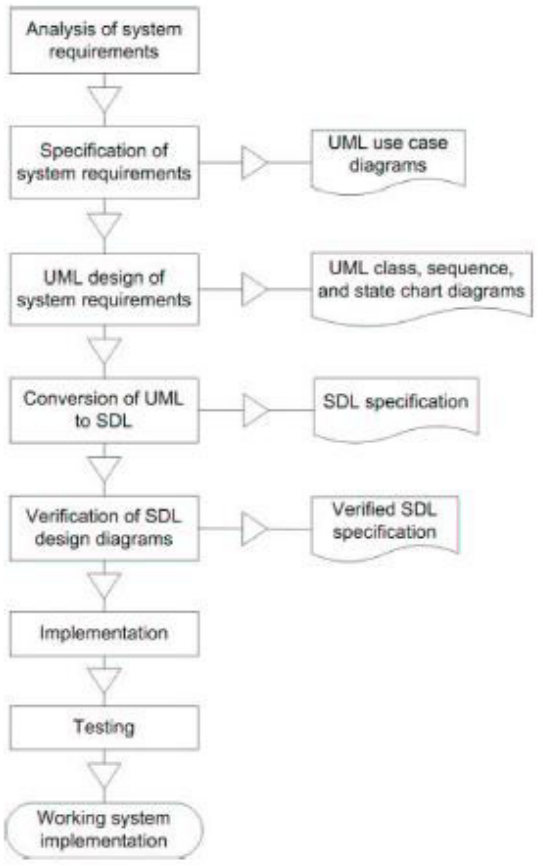



图 1 使用 UML 和 SDL 进行正向工程

## II 背景

在实际开发系统之前，要进行 UML、SDL 和 VoIP 中所涉及协议的背景研究。

### A．统一建模语言

根据 Booch et al.[2]，UML 是一种用于软件工程的图形化建模技术。最近这些年，由于面向对象方法学的研究兴趣逐渐增温，它的用途也变得日益广泛起来。自 1997 年 11 月 11 日起，UML 已经由 OMG 组织标准化。尽管 UML 主要是设计用来辅助软件开发，但还是可以用它来对不同类型的系统建模。它的主要目标是方便程序员、软件工程师和客户之间的交流。为了完成这个功能，它提供了措施可以对软件开发生命周期的每个阶段，从需求分析到实现进行建模。

使用 UML 的一个好处是它提供了许多不同的图，这样对特定的系统就会持有许多不同的视图。UML 的另一项正面作用是有广泛的 CASE 工具支持。

另一方面，正如以前提到的那样，UML 主要的不利之处在于它没有形式化的语义。这使得仿真 UML 模型测试正确性变得很困难，而这也恰恰是 CCSs 工程主要关注的地方。

### B．规范和描述语言

如 Belina et al.[3]所述，SDL 是一种形式化语言，可用于确认和描述软件系统的功能行为。它由 ITU（国际电信联盟）标准化，SDL 有图形格式（SDL/GR）和文本格式（SDL/PR）。

SDL 最主要的长处在于其仿真和元级执行模型的能力上。使用这种语言生成的系统可以校验和验证。特别是，在创建并发通信系统时，可以在实现之前检测到错误。这是节省成本而且高效的。而且，从高层系统图到低级过程图，SDL 图形成了细节层次。

SDL 的一个缺点是它目前只有很少的 CASE 工具支持。进一步的缺点是，SDL 图类型很少，因此系统就不能确定出足够多的视图。最后，SDL 并没有提供需求规范。

正是因为这个原因，我们仅在 ChattaBox 系统设计的后半部分选用了 SDL。下面会给出 VoIP 的详细讨论。

### C．VoIP

VoIP由协议栈组成，设计它们是要在包交换网络上传送语音数据。声音信号是使用codec编解码器数字化编码的。而且数据被封包，在终端之间通过IP网络（LAN、Internet，等等）进行传输。

用于案例研究目的的VoIP系统涉及的实现有：实时协议（RTP）、实时控制协议（RTCP）、会话描述协议（SDP）和会话启动协议（SIP）。这些内容在[6]和[1]有详细描述。在VoIP系统的顶层，SIP是标准协议，用于启动涉及如语音[7]等多媒体元素的交互式用户会话。SIP与SDP联合使用，用于给多媒体会话过程[8]的参与者传递媒体流信息。

实际的语音包由RTP传输，这个协议通过单播或多播网络服务[9]为程序管理实时多媒体传输。最后，RTP由RTCP实现，这使得它可以监控数据传输。监控过程使得接收者可以检测是否有包丢失或补偿延时抖动的现象。VoIP中涉及的协议足够复杂，可以在CCS工程中全面测试UML和SDL的使用情况。

## III 相关工作

组合应用 UML 和 SDL 引起了许多研究者的兴趣。Holz[10]觉得软件工程过程应该同时使用这两种语言。他认为 UML 适合系统的分析和早期设计工作，而 SDL 则更适合详细设计和充分高层实现语言。更进一步，他建议扩展 SDL，包含 UML 的概念。与他同出一辙，Bjorkander [11]和 Dimitrov et al.[4]也宣称应该组合 UML 的表达能力和 SDL 的一致性与语义。

而且，许多研究人员也曾经研究过把一种语言映射为另一种语言。例如，Selic et al[12]曾经借助一定的扩展把 SDL 映射为 UML。与此相似，ITU 发布了推荐标准 Z.109，详细解释了如何将 UML 映射为 SDL。

最后，两种语言的标准化组织扩充语言，包括了 ITU 最新版的 SDL、SDL 2000 的概念，并试图匹配 SDL 和 UML。同样，OMG 目前正在推出 UML 2.0，它将包含验证 UML 规范的概念。

很明显，这两种语言都有各自的支持者，研究它们的组合方法将会有更广阔的空间。

## IV 软件工程过程

为了进行案例研究开发的ChattaBox系统使用下面列出的过程完成工程任务。

### A．需求分析和规范

软件工程过程的第一个阶段是需求分析和规范。一开始错误地理解系统需求可能会产生有缺限的软件。ChattaBox的需求使用UML用例图表示，用例图专门用于系统需求工程。这些图有三个组成部分：参与者（绘制成棒状小人）、用户与之交互的系统（绘制为盒/块）和用例（显示成系统模块中的气泡）。参与者可以是人或者是系统以及它们的组件。

在ChattaBox系统中，要确定两种主要的需求分组：终端用户需求和内部系统需求。

终端用户需求，正如上面所说的那样，指的是由ChattaBox提供给公众用户的功能。建立这些需求需要确定软件的所有可能需要的功能。ChattaBox提供的主要功能是使用户可以建立语音会话链接。另外，还确定了大量其它的功能需求。在图2中，用例图组成了ChattaBox终端用户的部分需求规范。

在这个用例图中，从用户的观点来看很容易看出ChattaBox需要什么。ChattaBox需要为每个用户保留一个帐户，提供登录设施。帐户允许用户保存个性化设置和地址簿。另外，软件要允许用户调用基本话务功能，例如选择或回复呼叫。还要提供状态服务，反映用户是否可以接收呼叫。

除了上面提到的基本功能以外，还要为ChattaBox建立高级的需求，包括语音邮件服务等。

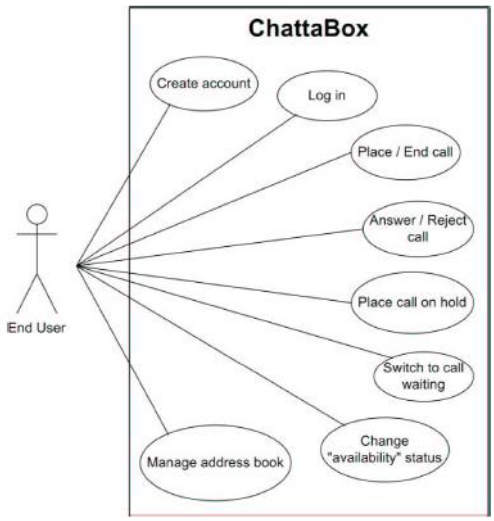



图2 ChattaBox的终端用户需求

内部系统需求用来描述ChattaBox分布式结构中对不同组件的需求。每个组件必须给其它组件提供服务。这些需求也使用用例图映射出来。

### B．设计

设计阶段开始于对ChattaBox系统架构的概括，这要使用UML组件图描述。这些图是显示高层通信以及组件和对象分布的好地方。

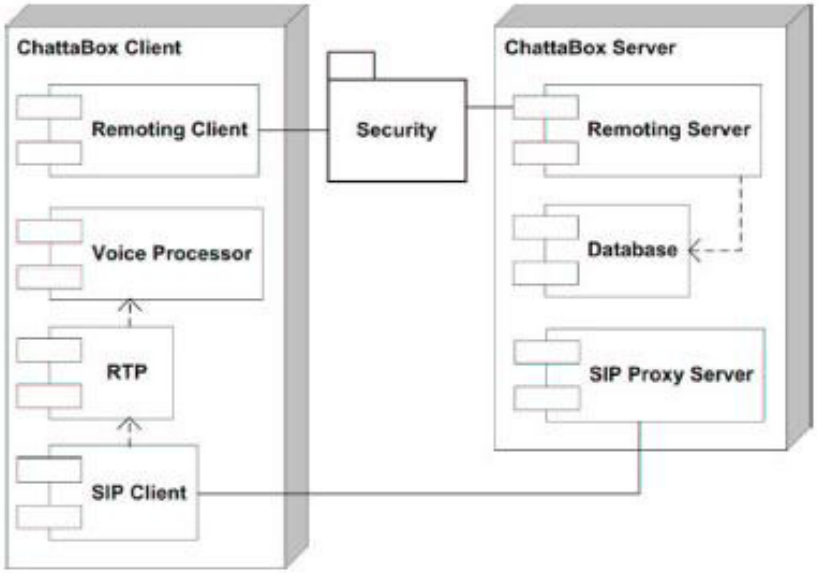



图3 概括ChattaBox系统

图3概括了ChattaBox系统的架构。大体上，这个系统由两个实体组成：客户和服务器。

ChattaBox服务器有三个主要的组成部分：

1） 远程服务器 – 这部分依靠微软的.NET远程基础架构完成抽象网络调用，使得用户状态变化和语音邮件这样的设施成为可能。注意为了验证的目的这个组件需要安全包。

2） 数据库 – 数据库组件管理数据的持久化。

3） SIP代理服务器 – SIP代理服务器负责转发SIP消息，它在会话建立阶段使用。

ChattaBox客户由下面四个部分组成：

1） 远程客户 – 远程客户与远程服务器通信，同时使用安全包验证用户。

2） SIP客户 – SIP客户发送和接收SIP消息，许可建立会话。

3） RTP – RTP组件处理实时语音数据，在两个ChattaBox客户端之间进行传输。

4） 语音处理器 – 语音处理器从通过RTP发送的声音输入流中捕捉语音数据。

客户和服务器验证使用对称密钥和基于X.509[13]的证书机制。

1） 静态设计：下一阶段关心设计结构和静态层面。这个阶段布署了UML类图。类图描述了系统中对象的类型和它们之间存在的不同类型的静态关系。它们也表明类的属性和方法，以及对象之间相关的关系约束。图4中给出了类图的一个示例。

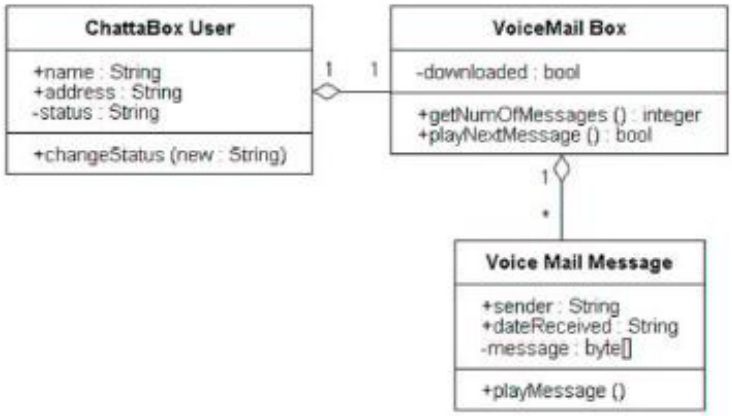



图4 ChattaBox用户，VoiceMail Box和Voice Mail Message类

2） 动态设计：设计过程最后阶段要处理系统的动态行为。绘制UML交互和状态图是用来捕捉设计的这层因素。交互图是描述对象组如何以某种行为进行交互的模型。一种特殊类型的交互图、时序图都被用来捕捉单个用例的行为。时序图描述对象之间传递的消息，它使得设计者可以理解和把握整体的控制流程。对于并行过程建模来说，它们特别有意义。图5给出了时序图的一个例子。

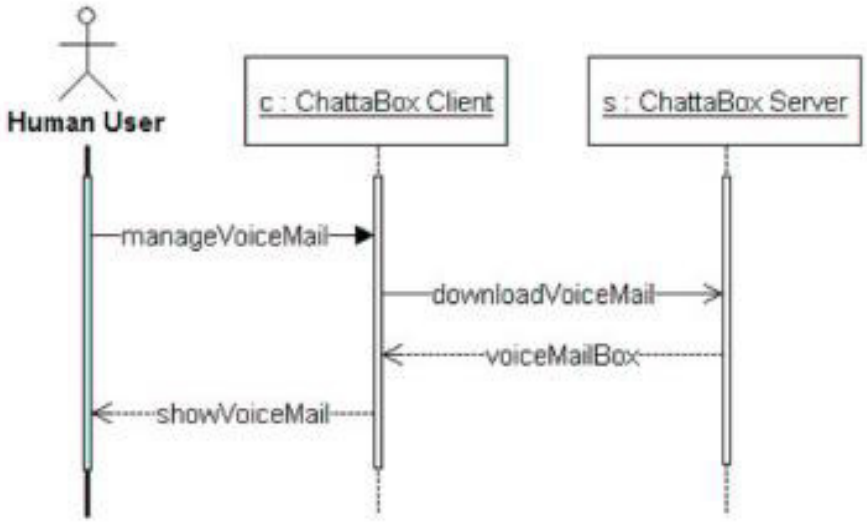



图5 语音下载时序图

状态图描述特定对象可能会处于的所有状态，以及作为达到这个对象的事件结果状态会如何发生变化。我们发现状态图在描述跨越几个用例的对象行为中非常有用。图6给出了状态图的一个示例。描述大量交互对象的行为来说，它们尤为有用。但是，状态图及时序图的组合完整概括了设计的动态因素。

这个案例研究的主要目的是验证SIP协议实现的正确性，这样在设计阶段就关注于系统这方面的内容。

### C．把 UML 转换为 SDL

按照客户机/服务器模型划分，ChattaBox的SIP组件中完整的设计结果由两套类图和状态图组成。每套图都单独转化形成一个独立的SDL系统，它与环境进行交互。

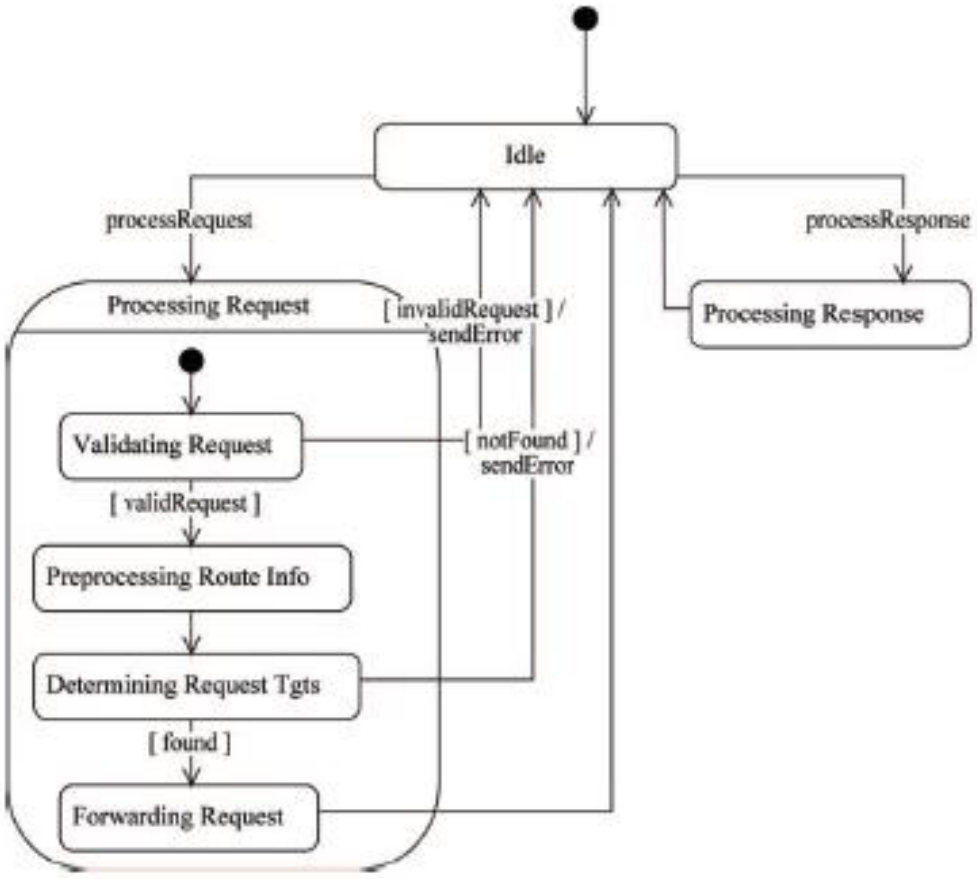



图6 处理SIP消息的状态图

这个转换并不是个简单的过程，为了这一步，在UML中需要遵循特殊的语法。最初的类图和状态图，在它们适合完成转换之前，不得不修改数百次。例如，为了将UML类正确转换为SDL制品，要使用原型。

系统设计的客户机和服务器部分单独进行转换。一经转换，就会发现部分原始UML设计并没有被转化。因此SDL图必须扩展以保留它们的含义，同时在SDL中要保持语法和语义上的正确性。结果就需要详细了解SDL，对SDL图求精的过程是极度耗时的。

### D．设计验证

在使用SDL描述ChattaBox中的SIP之后，它就被认为是完整的，并且认为与UML设计是等价的，验证就被执行。这些内容可以由Telelogic SDL和TTCN suite 4.3等仿真器和校验工具来完成。

仿真器允许分组给系统发送选择信号，并监控结果行为。对于环境发送的每个信号，都要监测这两个系统的反应和初始化行为。这意味着缺少设计，应当改进。除此之外，校验工具还用来对每一套SDL图都进行状态检测。结果表明系统没有死锁现象。

一旦SDL设计被正确验证，就开始实现ChattaBox。UML和SDL都被当作参考模型，以确保可以准确反映设计过程的结果。实现细节超出了本文讨论的范围。

### E．测试

为了确定实现的并发通信和分布式系统是否正确，测试是必不可少的。要确定两个主要的正确性度量。第一个是要测试是否所有的系统组件都完成了相应的功能。第二是要测试终端用户是否对ChattaBox产品表示满意。

1） 组件测试：在实现阶段，单个组件要单独测试。测试包括黑盒和白盒测试。下面是测试中得到的一些结果：

- SIP正确性：SIP呼叫能够成功跨越多个领域和代理服务器。SIP呼叫序列遵循SIP RFC标准[7]。SIP呼叫要能正确对付错误，也可以优雅地中断（称作teardowns）。
- RTP正确性：要正确处理RTP连接，会话要干净地关闭。语音数据要可听，要能正确地传输才行。
- 分布式架构完整性：分布式架构要能有效地均衡负载。除非发现合法的证书以及档案和语音邮件要能正确地管理，否则服务器将无法运行。

2） 用户测试：系统由十人组中进行测试，他们的年龄分布从18岁到49岁。两人分为一组，进行测试，这两个人都使用ChattaBox进行通信。用户要用系统来执行任务。之后他们要通过调查问卷判断系统，问卷是事先做好的。分析完用户测试的采样数据之后，我们得出了下面关于ChattaBox系统的结论：

- 尽管有迟滞现象，但声音质量很好。
- 系统可以响应启动呼叫、档案改动和语音邮件管理等行为。
- 用户界面很有效，相当容易使用。
- 系统并非不可预测，可以按照用户的想法进行工作。

总的来说，ChattaBox成功地满足了要求。组件和用户测试表明它是一个稳定的，正确的系统。

## V 结论

在案例研究的最后，我们可以得出下面这些结论。

### A．UML 和 SDL 的软件工程

把UML用于需求分析和最初的设计，SDL用于验证和测试设计，这样的选择把软件工程视为一个整体。案例研究中获得的经验也证实在完成复杂系统工程时，有结构的软件开发过程是很重要的。

而且，我们也评估了使用UML和SDL实施CCS工程的价值。UML有很大帮助，它可以从高层快速开发系统。如果这条路可行且有效，仍然需要改进转换过程。

使用软件工程方法验证系统正确性的做法已经受到注意。尤其在把独立组件整合进最终系统的时候，开发大型通信和分布式系统被证明是特别困难的。

完成了耗时的测试之后，还要与系统进行会话以确保它可以顺利运行。很明显，任何减少SDLC的作法都是有用的。进而，了解系统设计逻辑是否正确可以在实现阶段节省大量的时间。如果设计正确，实现中遇到的错误就不大可能会是逻辑错误，逻辑错误需要花费大量时间来解决。

### B．面向对象设计范型的适宜性

UML设计完全是面向对象的。我们发现在最终系统中用到的许多概念并不适合使用面向对象范型来建模。这些概念包括事件、远程对象和线程。大多数分布系统的实现要用到这些概念。因此我们建议采用UML扩展以更好地对这些概念进行建模。

而且，如果对正在使用的概念没有深刻地掌握，完整而详细的设计就不总是可以实现的。例如，最终实现中用到的许多制品在设计阶段并不为人所知。再者，在不具备这些知识的时候就开始建模意味着设计只能停留在相对较高的层次。

### C．CASE 工具的使用

作者对在软件工程过程中CASE工具的位置理解得尚且不好。图形对于团队成员之间的概念交流来说是很必要的，可以用来澄清不能很好理解的概念。使用Telelogic和Microsoft的CASE工具将会大大完善软件开发生命周期（SDLC）。

从ChattaBox案例研究初始阶段直到系统实际实现是一段很长的时期。但是，我们现在对使用软件工程构造复杂的系统和团队协作有了更为深刻的见解。

## VI 未来的工作

下面提出了一些对在未来工作和案例研究中选用不同的软件工程道路的建议。

把UML设计转换为SDL的过程应该更加高效。在转换之后，如果没有对SDL图进行扩充，效率会提高。

应该进一步调查，开发工具允许在UML和SDL之间进行逆向工程。有了这种工具，开发人员就得以保证在任何时候都能操控系统的表示。相应地，要用SDL验证设计的正确性，同时也要用更为友好的UML进行归档。

另一种方法是采用UML 2.0或SDL 2000进行案例研究。这样的案例研究会帮助弄清楚这些方法学的扩展是否允许在软件开发周期内仅使用一种技术，从需要和设计到验证，最后到实现，测试和归档阶段。假如说这些语言可以成功地完成CCSs工程，那么它们对于以后的软件工程师也会是一种有价值的工具。

## 参考文献

[1] H. Schulzrinne and J. Rosenberg, “The IETF internet telephony architecture and protocols,” *IEEE Network*, vol. 13, pp. 18 – 23, May/June 1999.

[2] I. J. Grady Booch, James Rumbaugh, *The Unified Modelling Language User Guide*. Addison-Wesley, 1999.

[3] D. H. Ferenc Belina, “The CCITT-Specification and Description Language SDL,” *Computer Networks and ISDN Systems*, vol. 16, pp. 311–341, 1989.

[4] R. D. Evgeni Dimitrov, Andreas Schmietendorf, “UML-Based Performance Engineering Possibilities and Techniques,” *IEEE Software*, vol. 19, pp. 74–83, January/February 2002.

[5] Telelogic, “Telelogic Web Site,” http://www.telelogic.com.

[6] H. Schulzrinne and J. Rosenberg, “The Session Initiation Protocol (SIP): Internet-centric signaling,” *IEEE Communications Magazine*, vol. 38, pp. 134 – 141, October 2000.

[7] M. H. H. S. E. S. J. Rosenberg, “RFC2543: Session Initiation Protocol,” http://www.faqs.org/rfcs/- rfc2543.html.

[8] M. Handley and V. Jacobson, “RFC2327: Session Description Protocol,” http://www.faqs.org/rfcs/- rfc2327.html.

[9] S. C. H. S. R. F. V. Jacobson, “RFC1889: Real-time Transport Protocol,” http://www.faqs.org/rfcs/- rfc1889.html.

[10] E. Holz, “Application of UML in the SDL Design Process,” http://citeseer.nj.nec.com/- 268645.html, June 1998.

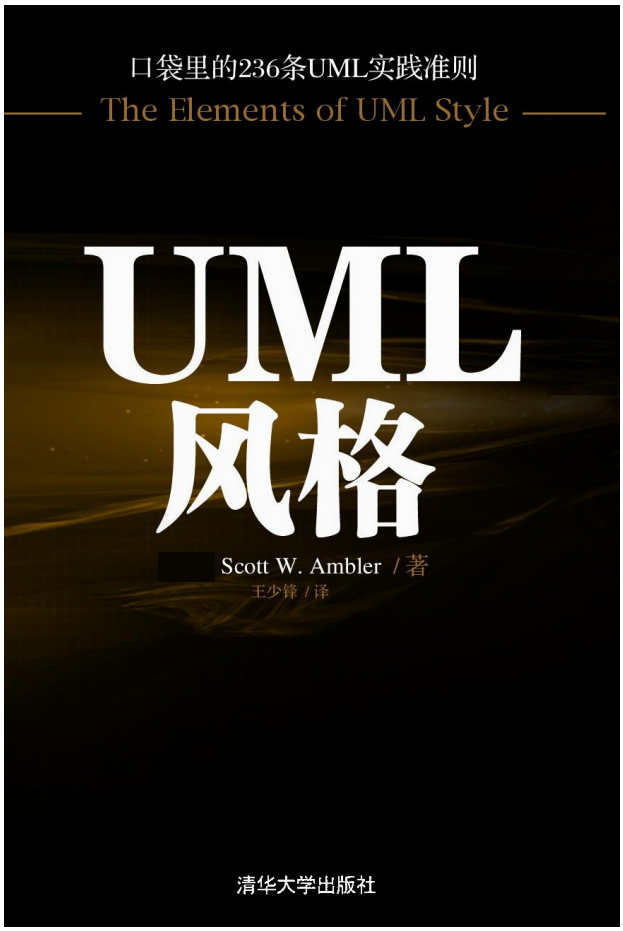
[11] M. Bjorkander, “Graphical Programming Using UML and SDL,” http://www.telelogic.com/- download/ paper/graphicalprogramming.

[12] B. S. J. Rumbaugh, “Mapping SDL to UML,” http://www.rational.com/products/rosert/prodinfo/- reading/sdl2umlv13.pdf.

[13] W. Stallings, Network Security Essentials: Applications and Standards. Prentice, 200

征稿

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


口袋里的236条UML实践准则
The Elements of UML Style
UML
风格
Scott W. Ambler / 著
王少锋 / 译
清华大学出版社


mobile

# RUP 的分析和设计工作流中的 Aspect

Eduardo Kessler Piveta、Augusto Jun Devegili 著，samying 译

查看评论

## 摘要

本文主要考虑 RUP 的分析与设计工作流，并且确定这个过程由于 AOSD（面向方面的软件开发）而作的改变。我们定义了一个新的角色，称之为“方面设计师”（Aspect Designer），并且描述了怎样、何时、哪一个结构必须被创建或被修改。我们同时提出了一些需要在面向方面设计中进一步讨论的问题。

## 1 简介

AOSD 是一种新的软件开发思路，因此它需要自己的过程、建模技术和工具。然而，在软件开发阶段，还不能很清楚的知道需要何种机制、工具和语言进行高级分解。一种建议是采用目前已经存在的过程支持 ASoC 建模，直到我们有足够的理论和实践背景去开发基于这些模型的新过程。

Rational 统一过程(RUP)是一种软件开发过程，这种过程聚焦于对开发团队分配责任和活动。它的主要目标是提供一个过程，这个过程使得软件开发具有可预测（成本和时间）的特性。

在本文，我们尽量清楚的表明 AO 的采用对于用 RUP 方法开发的软件的影响，包括 RUP 关注的：角色、制品、活动和工作流，特别是在分析和设计工作流。

## 2 RUP: Rational 统一软件过程

根据[2]“RUP 是一个软件工程过程”。在软件开发组织中，它提供了一个有规律的方法去指定任务和职责。它的目标是在确定的时间进度和预算下，用高质量的产品满足用户的需求。尽管 RUP 是一个过程框架，可是它的一个重要特性就是延展性：它可以适应并扩展以满足用户的特殊需要。

有 4 个基本的建模元素：角色、活动、制品、工作流。角色表示了开发团队中具有一定行为和职责的人员，例如：系统分析员、架构工程师、设计工程师。活动是工作角色完成的工作：分析用例和执行者、复审设计、进行性能测试。制品是一个过程产生或应用的信息，例如：软件架构文档或者一个设计模型。工作流描述了一些活动的顺序，这些活动产生了一些有价值的成果，并且展示了角色间的交互关系。分析与设计工作流，实现工作流和移交工作流都是工作流的例子。

## 3 对工作流产生怎样的影响?

表 1 描述了面向方面在什么地方以及怎样影响了 RUP 工作流

| 工作流 | 高 | 中 | 低 |
|---|---|---|---|
| 业务建模 | | | √ |
| 需求 | | √ | |
| 分析和设计 | √ | | |
| 实现 | | √ | |
| 测试 | | | √ |
| 发布 | | | √ |
| 配置管理 | | | √ |
| 项目管理 | | | √ |
| 环境 | √ | | |

表 1 RUP 工作流怎样被 AO 影响

**业务建模：**这个工作流集中关注功能需求，没有受到方面(AO)的太大影响。

**需求：**尽管非功能需求在这个工作流被识别，可是作为一个方面(AO)被建模时，它通常影响后续工作流。

**分析和设计：**这个工作流将在下一部分讨论。

**实现：**尽管在 AOSD 中实现工作流相当不同，我们在考虑实现活动时是十分近似的。用两种活动考虑 AOSD 是十分必要的：实现类和单元测试。

**测试：**单元测试的策略是每一个方面都要被定义。在集成测试时，为了作为一个整体测试系统，编制一些应用类和方面是十分必要的。

**发布：**不影响这个工作流。

**配置和变更管理：**不影响这个工作流。

**项目管理：**不影响这个工作流。

**环境：**这个工作流关心的是为软件开发进行技术环境的准备。由于面向方面需要新工具和设计、程序和工具指南，方面改变了员工执行工作的方式和生产的制品。下面的角色必须被按照方面来考虑：工具专家、架构师、测试设计师、技术设计师。

## 4 分析和设计工作流

这个工作流的主要目标是，将需求转换为系统怎样被实现的说明。它能被解释为，用几种不同的策略，实现从需求到分析和设计的模型。

这个工作流产生的模型应当涉及功能和非功能的需求。通常功能需求在建模（分析）早期被确定，非功能需求在设计建模时涉及。分析是一个探索性任务，集中建立软件是“什么”，设计则集中考虑一个软件部件“怎样”去建立。

在针对一些非功能需求(日志\跟踪\保持\安全)按照方面建模的设计阶段，我们认识到“方面”（aspect）扮演了一个重要角色。因此 RUP 中分析和设计工作流中受到的影响最大。

### 4.1 角色

在这个工作流中有几种相关的角色，某些角色在 AOSD 中受到了更大的影响。

尽管在系统分析中，我们主要关心功能需求，AO 此时并不成为主要关心的方面。然而架构工程师和设计工程师直接受到“方面”（aspects）的影响，必须考虑他们的活动。我们建议设计角色被专门分为两个子角色：域（domain）设计者和“方面”（aspect）设计者。因为他们关心不同的应用（功能和非功能）。

架构师的责任是设计模型，这些模型由一系列的模型元素构成，模型元素提供了系统的行为特征。在传统的面向对象软件过程中，模型来自分析模型针对设计而做改变。在 AOSD 中，分析模型应当被最低限度地修改。为了方面（aspects）自身的建模，我们需要一个独立的模型，设计模型应当主要集中在于域（domain）类和方面（aspects）。接口也是同样，域（domain）类和方面（aspects）类两者都要有接口的详细说明书。

在分析和设计工作流有一些相关的角色。

**数据库设计师：**因为他的职责在于建立数据模型（如果应用包含数据库），如果系统应用面向方面（aspect-orientation）的数据库[3]技术，面向方面将关系到他。应当持续考虑一个方面，面向方面设计者的职责是认识如何设计持续的方面。

**封装体设计师（Capsule designer）：**负责在系统的设计中考虑实时的问题，由他或她作为一个方面（aspect）设计实时封装体。**封装体设计师**被看做一个专业的方面设计师（aspect designer）

**设计复审者：**他们必须将方面（aspects）作考虑因素，去回顾、评价在工作流中生成的制品。

### 4.2 工作流细节和制品

AOSD 用以下描述的活动呈现：

**设计候选的架构：**在处理应用的非功能需求的时候，架构师将生成一些制品，这些制品包括设计模型、软件架构文档和考虑方面的实现用例。在基于方面的详细实现，开发模型中包括方面（aspects）是很有必要的。

**执行架构合成：**这个活动构造概念的例证（proof-of-concept），在这里架构合成至少是一个对于关键架构需求的解决方案，应当考虑横切关注点（crosscutting concerns）怎样影响架构概念的例证（proof-of-concept），于是产生一个考虑方面的架构概念的例证制品原型。

**精化候选架构：**因为架构根据方面（aspects）而变化，这个工作流的大多数活动细节是考虑方面。

**分析行为：**这个细节的主要焦点是功能需求。因此在细节中方面（aspects）不是一个问题。

**设计部件：**在这个细节我们需要一个新的活动： 方面设计（Aspect Design）。当架构细节在方面的更高抽象

层次（定义、精化）被关心时，在这个活动中，方面设计师通常关心设计层次的元素，例如：结合点，建议，继承层次和支持子系统。

**设计数据库：**在这个细节方面设计师和数据库设计师为存储和保持数据定义规约和策略。

### 4.3 工具

定义了怎样描述横切关注点（crosscutting concerns）后，在分析和设计理论模型时，非常有必要扩充存在的工具（例如：RATIONAL ROSE 和 RATIONAL ROSE REALTIME）去支持这些扩展。

## 5 相关工作

另外有研究项目探索 RUP 适用于面向方面的规则，例如 RUPIM[4]，RUP 扩展提供了三个方面的不同扩展：持久性、分类、网络。尽管大部分变化发生在执行工作流（主要依据在应用类和 DOMAIN 类关系的分离），为了合并这些从 PIM 到架构的设计元素，分析和设计工作流做了一些改变。在我们工作中，RUP 的设计模型被分成两个不同的模型。DOMAIN 模型和方面模型。显然实现工作流好象不需要大的变动，实现活动（DOMAIN 和方面）是正交的和同时被执行的。

尽管我们不能描述方面所需的符号，但是我们的确需要为 AOSD 扩展 UML。UCMS[5]是可能被用作方面相关制品的的符号。在[6]，作者常通过创建一个新的命名的元素——**方面**（aspect）来扩展 UML 元模型。方面属性和织入（weave）定义有关，操作和织入声明相联系。关系包括泛化、联合、依赖。

## 6 未来工作

本文是识别与指出面向方面（AO）何时并怎样影响 RUP，特别是分析和设计工作流中。未来的工作将集中在进一步精化的探索性论文，并扩展到其他工作流，详细设计广泛的角色、制品、工作流细节，工作流，以及概念和规律。

未来研究集中于用统一的方式去表达在设计层次中的横切（crosscutting）和用在某方面的架构。研究 AO 的设计参数，AOSD 和极限编程（Extreme Programming）的关系、重构，都是是非常有趣的方向。

## 7 总结和讨论

RUP 进行一定的修正后能被应用于实现 AO 系统。这些修正是小范围的。主要的问题特别是在设计阶段，缺乏一个已经定义的方式描述在 AO 系统中的横切关注点（crosscutting concerns）。其它可能是为了高度关注 AOSD，而创建一个新的开发过程。关于 AOD，可以指明的一些讨论问题是：

- 怎样描述 AO 软件架构?
- 怎样在设计中表达横切关注点（crosscutting concerns）？
- 当没有方面和部件之间、方面和方面之间，所有可能关系的清晰定义时，我们怎样描述方面（aspects）？

这些问题的答案能更好的帮我们描述软件工程过程关系。

## 参考文献

[1] Johan Brichau, Clarke Siobhan, Maurice Glandrup, and Lodewijk Bergmans. Advanced separation of concerns. In ECOOP Workshops, 2001.

[2] Philippe Kruchten. The Rational Unified Process: An Introduction. Ob-ject Technology Series. Addison-Wesley, Upper Saddle River, 1999.

[3] Awais Rashid and Elke Pulvermueller. From object-oriented to aspect-oriented databases. In Proceedings of the 11th International Conference on Database and Expert Systems Applications, Lecture Notes in Computer Science 1873, 2000.

[4] Paulo Borba, Augusto Sampaio, and Tiago Massoni. Progressive imple-mentation of aspects. In OOPSLA 2001 Workshop on Advanced Separa-tion of Concerns in Object-Oriented Systems, 2001.

[5] R. J. A. Buhr. A possible design notation for aspect oriented program-ming. In Proceedings of the Aspect-Oriented Programming Workshop at ECOOP'98, 1998.

[6] Junichi Suzuki and Yoshikazu Yamamoto. Extending UML with aspects: Aspect support in the design phase. In Proceedings of the Aspect-Oriented Programming Workshop at ECOOP'99, 1999.

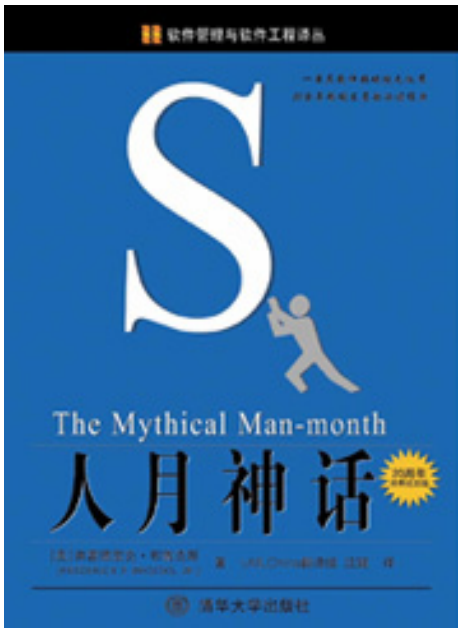
[7] Martin Fowler. Refactoring: Improving the Design of Existing Code. Object Technology Series. Addison-Wesley, Upper Saddle River, 2001.

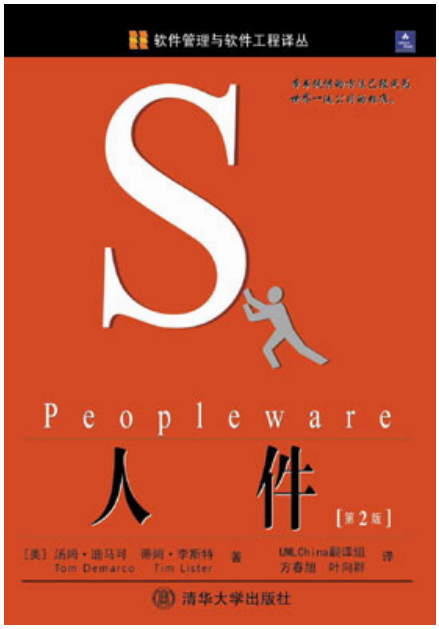
[8] Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, Upper Saddle River, 1999.

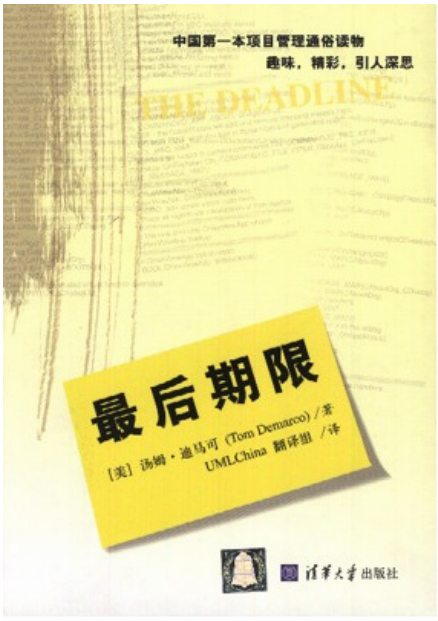

斐力庇第斯从马拉松跑回雅典报信，虽然已是满身血迹、精疲力尽，但他知道：没有出现在雅典人民面前，前面的路程都是白费。

学到的知识如果不能最终【用】于您自己的项目之中，也同样是极大的浪费。而这最后一段路最是艰难。

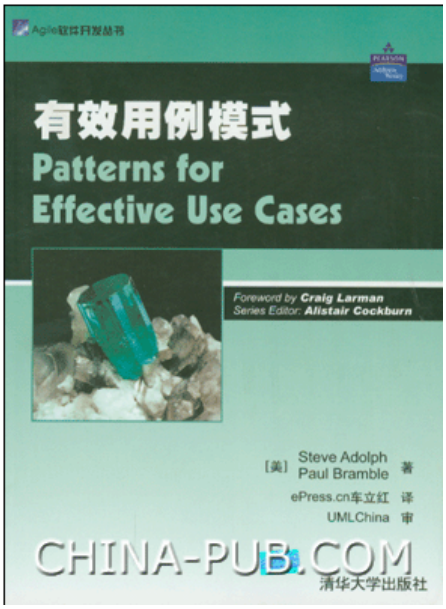
UMLChina 聚焦最后一公里，所提供服务全部与您自己的项目密切结合，帮您走完最艰难的一段路。

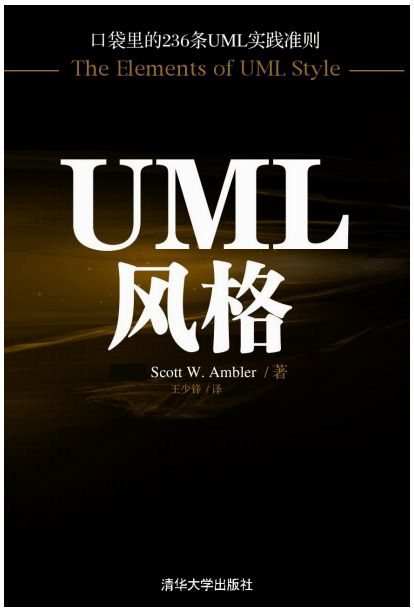


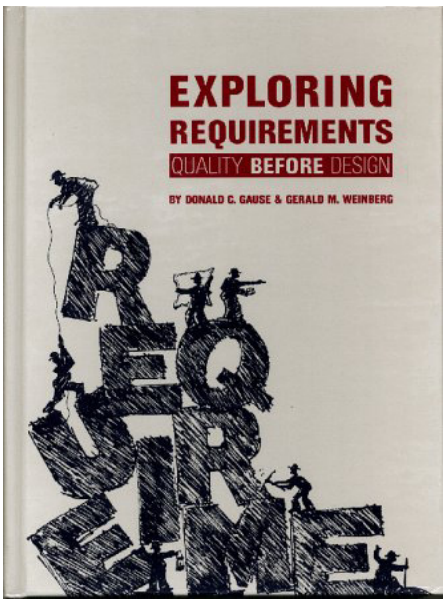



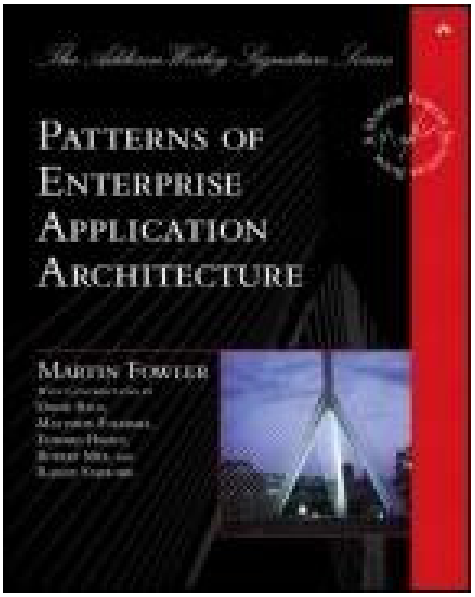



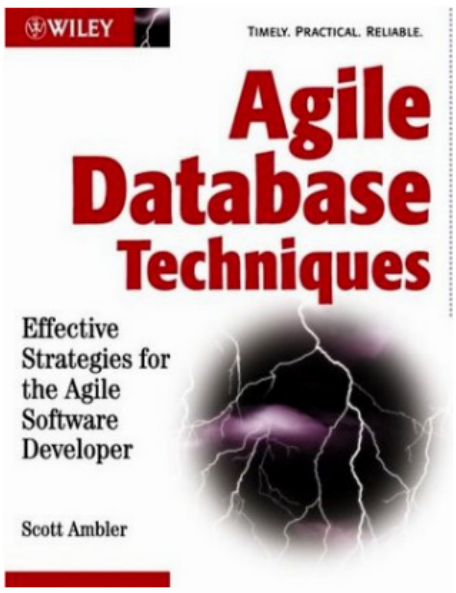

- 2002 年《人月神话》创下销售记录
- 2003 年《最后期限》、《人件》、《人月神话》分别排在 china-pub 年度总销售榜的 1、3、4 名



# 了解 UMLChina

软件以用为本。不能为客户提供所需价值的软件是可悲的。以前，开发人员以“专家”自居，客户心存敬畏，认为自己“不懂”，不敢提需求，不敢评判。最终的结果是产生了许多“鸡肋”软件——软件做出来了，也能运行，却不提供想要的价值！现在，客户的胆子大了！软件只不过是帮助我完成业务、提高效率、赚取利润的工具而已！我要的软件要能够为我的业务提供价值，运行架构还要符合我的实际情况，还要有足够的可靠性、灵活性、可用性…。软件组织的“好日子”不再有，不得不关心这些问题：如何捕获有价值的用户需求，如何组织需求，如何获得良好的软件结构，如何获得稳定的性能，如何提高软件的可用性…。

软件（技术）以用为本。不能为软件组织实际使用的技术是可悲的。为了提升软件质量，许多软件组织都在谋求技术的改进：需求的技术、分析设计的技术、增量迭代的技术、构件组装的技术……我们搜索，我们买书，我们看书，我们讨论，我们试用...但学到的技术如果不能最终用于自己的项目之中，将是极大的浪费。而这最后一段路最是艰难。

**相信【软件以用为本】。为此倾注全力，不断实践，苦心研究，采用一切可能的方式，帮助软件组织朝着这个目标进发——让软件变得好用，让技术变得好用：**

- UMLChina 训练——“聚焦最后一公里”，专注于上门为企业作 UML/UP 团队训练。训练完全和受训团队当前项目结合，【训练的过程就是指导团队用所授 UML/UP 方法实作团队当前项目的过程】。UMLChina 在团队训练方面已经有了丰富的积累。从东北到西南，已为超过 50 家企业提供服务，涉及各种类型的软件开发项目。
- 项目指导——是 UMLChina 训练的延伸。UMLChina 专家带领团队用 UML/UP 方法做项目，使团队顺利行走在 UML/UP 的路上。
- 专家讲座——请来国外国内的顶级专家，传播他们的知识和经验。著名专家有：Gerald Weinberg、James J. Odell、John Vlissides、Kent Beck、Alistair Cockburn、Martin Fowler、Scott W. Ambler、Bruce Powel Douglass、Alan Cooper、Roser S. Pressman、Kendall Scott...等。
- 新闻——全球如发生和 UML 相关的事件，都尽力为您及时报送。
- 《非程序员》——和《程序员》同年（2001）诞生，却一直保持自己的独特风格，为众多高阶软件人员所青睐。每月发行一期，杂志将长久保持免费下载的电子形式。
- 书籍——继 2002 年《人月神话》创下记录之后，2003 年《最后期限》、《人件》、《人月神话》分别排在 china-pub 总销售榜的 1、3、4 名。随后 UMLChina 把精力集中到 UML/UP 相关图书：《有效用例模式》、《UML 风格》、《探索需求》、《PEAA》、《敏捷数据》…
- 讨论组——2000 年下半年开始设立，目前注册用户已经超过 40000 人。

# 极限编程如何极限？在大型项目中使用 XP 实践

Lan Cao、Kannan Mohan、Peng Xu 著，李腾 译

查看评论

## 概要

快速的软件开发过程的需要，以及在整个软件生命周期中适应变化的需要，正在使诸如 XP 这样的敏捷开发过程变得流行起来。但是，这种方法论被认为只适合小型团队的中小项目开发。在本文中，基于对一个大型项目中所采用的改进后的极限编程的案例研究，我们指出了以往理论中的敏捷原则在适用于大型软件开发时的不同之处。根据这些不同点，我们对如何将敏捷方法进行改进以适用于大型项目提出建议以作为一般性的指南。

## 1 介绍

我们现在面对着一个飞速变化的业务和技术环境。在这样一个环境中，传统的软件开发方法所认为需求需要在项目初期分析清楚并且保持稳定的想法是行不通了。不能够持续地快速地将需求地变化融合到软件中就意味着对业务环境反映迟钝，将最终导致业务上的失败。一些敏捷开发方法，象极限编程（XP），Crystal methods, Lean Development, Scrum 和 Adaptive Software Development (ASD)，纷纷被提出来，以适应当今多变的业务和技术环境要求。

敏捷编程的目标是允许组织能够快速交付和修改项目[1, 5, 10]。虽然软件开发过程中所采用的具体敏捷方法会不同，但是他们大致都有一些共同的特征，诸如：迭代开发，开发过程中经常同客户讨论，快速的程序发布[1]。一种应用比较广泛的敏捷编程方法就是极限编程（XP），这种方法强调快速迭代，严格的代码测试，同客户的紧密协同工作[3, 4]。XP 已经在一些小型的软件项目中得到了成功应用[13, 17]。

XP 的支持者认为采用这种方法可以比传统开发带来更多的先进性，包括：降低管理费用，提高团队产出，愉快的客户关系以及缩短发布周期。但是，敏捷方法的适用性也受到其他因素的限制，包括：项目的大小和类型，项目组成员的经验值，以及客户信任度。Boehm[5]认为敏捷方法很难应用于大型项目，因为它缺少充分的架构设计，过于重视早期结果而导致低覆盖面测试。敏捷方法同时也被建议不要在关键任务的项目中使用。但是实际上，在现在易变的业务环境中，在大型项目中敏捷方法的应用也同样被需要，以解决其所强调的问题，例如变化的环

境，不清晰的用户需求，交付的里程碑等等。软件开发者不仅面临巨大的快速交付产品的压力，同时也必须保障产品的质量。单纯的敏捷方法或者是以质量为核心的计划驱动方法都不能满足目前的要求。对开发者而言，所被要求的交付和质量的能力比以往要更多[5]。

由于缺乏前期的设计以及文档工作，大型并且复杂的项目并不能直接采用敏捷方法[15]。但是，大多数专家仍同意敏捷方式和传统的开发方法在本质是统一的[16]。XP 的实践已经被映射到 SWCMM 模型，而后者通常被认为大型的项目和组织所适用的[14]。调整 XP 方法，使之应用于大型复杂的项目，以达到更快的开发周期，上述工作已经正在努力进行[8, 16]。但是除了这些实际的应用之外，关于如何调整敏捷方法以适应大型项目的原则或者指南尚未从理论上提出。在本文中，我们将研究一个实际中的案例，该案例采用了调整后的敏捷方法来进行大型项目的开发。我们将明确适用于大型项目的敏捷方法的最佳实践。我们也会比较这些实践同传统理论中的敏捷方法所比较。基于这些比较，我们会把这些实践进行总结，从而作为敏捷方法应用到大型项目时进行修改的一个指南。

## 2 敏捷软件开发方法和大型项目

传统的开发方法强调项目的设计和文档，去控制不可预期的变化。但是项目的需求，范围以及技术的变更往往又并非开发人员所能控制的。问题经常是如何在项目的生命周期中应对不可避免的变更。而并非是试图使变更最少化[10]。敏捷方法正是采用了这种应变的战略，以降低因项目变更而产生的成本。

敏捷方法最大的优点在于可以尽早而且持续地向客户呈现有用的软件产品[1]。敏捷方法认可需求的变更，即使在开发的后期过程也是随时准备接受的。经常提交程序的可用版本，面对面的交流，同客户的密切合作，保持设计的简单性，这些方法都是敏捷方法所使用的手段，从而降低因为项目变更而产生的风险。这些手段强调一定程度的计划，但是重点被放在计划的过程而不是文档成果[5]。开发过程中知识库往往仅仅是在开发团队中共享。因此，上述的方法最适用于小型的开发团队，而且是客户可以专心投入的项目。

已经有一些敏捷方法发展起来，包括：极限编程，Crystal methods, Lean Development, Scrum, Adaptive Software Development (ASD)。在这些方法中，极限编程的使用是最广泛的。XP 是一种轻量级的方法，它摒弃了很多传统的开发包含的过程，包括冗长的需求定义和文档。它强调开发团队保持小型化和简单编码[4]。XP 定义的生命周期包括四个基本活动：编码，测试，倾听，设计[14].。XP 通过 4 个主要途径改进软件项目：交流，简单，反馈，激励。即：客户以及开发团队的持续交流，保持代码的简单，通过测试提供持续的反馈，主动解决问题[14].。有十二条核心的原则用来指导实现上述的目标。

XP 实践关注促进交流和团队工作。管理人员，客户，开发者都是这个团队中的一个部分。而团队的任务在于交付质量稳定的软件。开发者同客户间交流的障碍可以通过客户代表同开发者一起工作来解决，而每日构建，结对编程都促进了项目中的团队成员交流，从而降低了管理的开销。广泛的交流和及时的反馈也增强了客户和开发者之间的信任感[12]。

XP 同样也增强了开发人员对项目过程的控制[13]。采用 XP，开发人员可以把握项目的进行方向和进度。而且，经常性的测试使程序员清楚他们的代码是否可以完成预想的功能。而这种感觉又可以促进程序员的主动性。

尽管敏捷方法可以使软件项目获益，但是并非所有的项目都可以直接采用的。大型的，复杂的项目，其自身特征使其难以直接采用敏捷方法。这些特征主要包括 3 个：应用领域知识传播缓慢，需求不稳定和冲突，项目沟通存在障碍[7]。首先，大型项目中，开发团队深层次的开发信息在许多开发人员之间传递起来是非常缓慢和不充分的，而这些信息必须保持完整性，需要有坚实的设计，以协调对项目应用域的理解以及了解系统整体应当如何运行。其次，开发者对于应用的理解不充分，一些其他事件，诸如业务目标和策略的变更，都可能会引起项目中的需求变动和冲突。第三，对于大型的团队在开发过程中所面临的协调和交流困难，都必须花费很大的代价去完成定义，统一表达习惯，建设信息流系统。

敏捷方法缺乏前期设计和整个生命周期内的架构设计，主要依赖个人知识和非正式的交流。这些都导致了较高的风险。例如,团队可能会因为缺乏适当的设计而犯无法挽回的架构错误。开发者所容易接触到的客户对于需求的知识可能不够。非正式的交流对于需要处理大数量的参与者可能不够有效，而大量的信息又是大型项目的特征。传统的开发方法是计划驱动的，可以减少这些风险。例如：生命周期的架构设计，必要的详细文档，正式的外部评审，强有力的管理

.但是，大型的项目也同样面临项目需求变化和快速上市的压力。研究表明敏捷方也可以被映射到 SW-CMM 模型，一种传统的软件开发指南的模型[14, 16]。 为了降低快速上市和需求变更的风险，在一些大型项目中已经开始采用了 XP 方法。Elssamadisy[8] 和 Reifer[16]对采用了 XP 方法的一些大型项目做了研究，指出一些敏捷方法，例如迭代开发，经常性测试和反馈，小规模发布和重构，对于大型项目都是适用的。然而还有一些实践，例如例会，以及使用隐喻描述系统架构就不那么可行。研究发现如果没有全局的设计，当系统增长时要维护一个项目的整体视图是相当困难的。很多的使用者参与时，同开发者进行的交流也会成为问题。现在的理论并没有提供将敏捷方法改进到大型项目中的指南。在本文中，我们研究了一个案例，提出了如何将敏捷方法应用到大型项目上的一些实践。

## 3 大型项目的敏捷实践

现在我们开始研究一个大型开发团队进行大型复杂的金融应用的案例，以下简称 FinApp。本文中，我们将汲取其中的成功经验。

### 3.1 案例描述

FinApp 参与了一个复杂的企业级系统的开发，这个系统向银行，保险公司，信贷和经纪人提供金融服务。项目包含 6 大种类的 1000 多个业务对象。项目的开发人员有 22 个。项目的开始阶段采用了改善过的 XP 方法。一些 XP 的实践被修改过后在项目中采用。在这个项目中，已经有一些成功的应用交付给另一个大型组织，这些应用将和 FinApp 进行整合。

### 3.2 使用敏捷方法的需求

FinApp 充分认识到在开发他们复杂的金融系统时，XP 实践经过适当的修改是可以发挥其作用的。在关键的甲方人员看来，整个企业系统的关键任务的特性已经得到了很好的定义，在处理的流程中也进行了很好的坚持。但是，他们也认识到缺乏敏捷使完成需求的变更变得非常困难。而且仅仅依赖传统的交流方式，使开发团队成员间的交流遇到了困难。

在下面的章节中，我们将讨论 FinApp 如何将 XP 进行修改，以符合他们的开发环境，以及这些实践如何同理论上的敏捷实践相一致。

### 3.3 为大型系统改进 XP 方法

FinApp 的应用领域是相对成熟和稳定的。因此对应用的可靠性和安全质量有着同样高的要求。“区别在于，我们面对的是一个企业级的应用，这是最首要的。。。我们对 XP 进行了最大限度的修改，因为我们是企业级应用”，FinApp 的一位项目经理如是说。企业级应用需要抽象的设计，以适用于不同的最终应用。系统架构师花费了六个月的时间进行了业务建模和系统的架构设计。

FinApp 是一个大型的系统，大约有 7000 个程序文件。应用还包括了企业现金管理，一个非常复杂的子系统。系统的安全性也是非常关键的，程序的质量也是开发过程所需关注的。尽管用户的需求会经常变化，但是应用的业务对象仍然存在一个比较基本稳定的模式。基于这些考虑和项目本身的限制，FinApp 修改了 XP 实践，采用了一种谨慎的敏捷方法。

### 3.3.1 前期设计

XP 不强调项目前期的设计，因为它认为所有的东西都是会变化的。它采用“比喻”的方式来描述项目的基本元素和关系。但是，对于大型的项目，前期的设计被认为是必不可少的。而在 FinApp 项目中，前期的设计过程是可行的，也是必须的。

首先，银行业的环境是相对稳定的，系统可以被描述为有限的集中业务模式。项目经理认为：“这部分业务有一种基本的模式对应，那部分业务也有一种基本的模式。。。实际上，存在 6 到 8 种模式可以涵盖整个系统的业务”

其次，对于象银行系统这样一个复杂的，关键任务的应用。灵活架构和设计模式这样的最佳实践都是非常重要的。早期稳定的架构设计是支撑后期应用的基石。

对于关键任务应用，前期的设计可以确保系统的安全性和稳定性得到保障。“他们（客户）需要确信项目不存在任何后期的修补，因此，它必须是完全安全的，归档的，从起始阶段就非常明晰的”。客户需要确信开发人员没有在建造一个不断花钱的无底洞。因此，所有的设计和源码都会提交给银行。

早期的设计可以减少开发新功能的工作量。每一个新功能基于系统架构和设计模式进行。通过采用这种方法，开发新功能的时间和工作量都比重新进行开发大大减少。“每次我们开发新的服务的时候，我所做的仅仅是映射到现有的模式上去，不再需要技术说明。。。我们不需要技术说明，因为我们遵从已经建立起的模式”FinApp 项目在现有服务和设计模式上构建新的服务。例如，一项应用可以仅仅通过进行重新配置就可以满足不同国家的金融业用户的需求，而不是在重新开发用户的应用。

早期的设计也可以减少开发人员的培训时间，从而降低将新人融合到开发团队中的成本。“我能做到在任何一天将全部的团队重组，并加入新的成员。我可以让人礼拜一开始，礼拜二就可以产出，大多数企业要花 6 个月才能做到这一点，而我确实可以做到。因为你花一天半跟他讲解银行的业务模式，第二天你就可以根据这些模式创建自己的应用”

前期的设计可以控制变更产生的成本。对于大型项目，系统的根据要求而发生的改变，以及对原有错误的修正都会导致一些不可预知的后果，影响运行的稳定。设计模式通过使用集中的异常处理等机制来控制这些变化所造成的损害。

前期的设计为开发人员提供了一个清晰的系统整体视图，帮助他们理解功能如何加入到整个架构中去。XP 认为大多数的功能是独立的，但是在大型项目中并非如此。Taber 和 Fowler 也指出：当系统逐渐增大时，维护一个整体的已实现的功能说明蓝图和描述这些功能如何共同工作会变得越来越难。由于随着项目规模和复杂度的增加，开发和分析人员在不同的功能之间会迷失联系，这也使架构设计对大型项目而言变得更加重要。一种变通的方式是在过程中进行设计，而不苛求完整的早期架构设计。这种展开的设计方式通常会被视为会造成存储层和业务层间的紧耦合，从而导致在开发几个月后难以对代码进行重构。FinApp 通过使用详细稳定的前期设计来避免了这种情况的产生。它使用了稳定标准的接口和模式贯穿整个项目。有了坚实的架构，参照已经建立起的模式，新的功能就可以很简单地加到项目中。

### 3.3.2 快速的，分层的发布

快速（2—4 周）的发布周期和持续集成是敏捷方法所推崇的。对于大型应用，这一原则同样适用。“因为所有的东西都是模糊的……客户到底要什么？那么如果你给了我他们（客户）所想要的，那么他们会来看看，并且说，那个不错，你能做到么？我要些别的，等等。我的意思是程序的变更是正常的，在这种情况下，增加复杂的理解内容和存在模糊的概念是必然的。所以，为了完成这些协作，你必须有一个快速的，不断的版本发布。”

早期的体系结构设计一般都要花费比较长的时间，FinApp 的架构设计花费了 6 个月.类似于 XP 的快速发布实践，分层的方法在每次迭代之后都可以递交立即投入使用的产品。但是这种方法和 XP 的不同在于迭代的周期并不固定，而是基于层次和任务的特性。

一个 FinApp 的项目经理在谈到采用这种方法时说：“采用分层的递交，如果我们一部分的工作有了延期，我们仍旧有可以工作的部分，而不是说我没有任何可用的东西”关键在于编码时支持提供端到端的功能（supports functionalities end-to-end），而不是开发集成度高的模块。

### 3.3.3 客户代言人的参与

对 XP 而言，理想的客户是常驻现场的一个最终用户，他有能力和精神作出决定。客户的参与是 XP 项目成功的一个关键因素。但是实际上这种同客户的接触是非常困难的。大型项目的困难会变得更加突出当于应用领域的复杂性往往超越了仅仅少数客户代表的经验和专业知识。软件的开发范围需要涵盖不同种类的使用者。开发团队的组成变得更加复杂，包括的用户和整个组织的管理部门。还有一个问题就是可以接触到的用户经常不是系统的最终用户。

在 FinApp,真正的客户是银行和银行的最终用户或者其他的金融机构。对开发人员而言，客户并不是可以随时参与的。尽管有个理想的客户来参与开发过程很难保证，.但是 FinApp 认为客户在开发过程中有一定程度的参与还是非常重要。

我们没有接触真正的用户。所以，真实的现场客户不是我们可以选择的。于是由产品经理或者系统分析员代替了客户的角色，他们同客户直接联系。产品经理几乎每天都同开发团队进行沟通，讨论需求的变更，进行开发方面的决策。同很少同客户沟通相比，这样的实践会产生更好的项目产出。

“最成功的案例就是我每天上午 10 钟和产品经理开会，任何一个有疑问的开发人员都会参加，我们进行协商，解决问题，修改需求，修改规格说明，确实有一种 XP 那样客户参与的协作过程，那是非常棒的”。对于大型项目，客户的参与是个非常清楚的需要。而当直接面度客户不是那么可行时，使用客户的代言人，一个业务专家，就成为一个很好的折衷方案、

### 3.3.4 灵活的结对编程

虽然结对编程被认为是一种好的实践，但是并非在所有场合都是那么符合实际的。在 FINAPP，结对编程采取了一种更加灵活的方式，只有系统分析，设计，测试用例的开发和单元测试采用了结对。开发人员在编码的时候试图结对，但是大多数最终还是回到了单独编码。这实际上和开发人员的个性有关。结对的适用水平根据开发人员和其任务的状态决定。FinApp 的项目经理总结到这种将实践适应到企业文化的过程时说：我并不坚持人们共享一个监视器。那样做只会是独裁，我猜想我们很多的开发人员都是黑客的性格，他们真的只想编码。结对编程同激发开发人员的士气是密切相关的。如果把选择权留给开发人员，结对编程被认为会工作得更好

结对编程的益处是被充分认识到的，它们是：

－减少了开发时间。

在 FinApp 的报告中指出，开发人员结对编程时工作得更快 。根据一位项目经理的说法，两个程序员结对编程时完成项目时间大约仅是单独指派任务的 80%，“我对此有把握，因为两种方式（结对和非结对）我都采用过了”。两个开发人员之间进行的交流使交流文档的编制和管理变成不再是必不可少的工作。

－减少了培训时间。

通过将不同熟练度的开发人员结对在一起，他们能够相互学习。“你要么指导别人，要么得到指导。”“我能做到在任何一天把整个团队打散，把新的成员加入进来。我可以让新人在周一开始，到周四就开始有产出。而大多数企业要 6 个月才能做到。”

－改进代码的质量

“开发人员之间的交流被认为是保证代码质量的一个关键因素以降低开发过程中的不一致性和缺陷。”

在两个开发人员开始构建编码之前，他们必须彼此确信他们对构建的对象充分了解。因此这之前需要协同，最终需要的也还是交流。而开发人员往往又是在一知半解的情况下就开始编码的。如果你必须首先编写测试用例并向其他的人解释，或者让别人去买它的时候，他们会指出这其中的矛盾和问题。对我而言，这便是这种(结对)最大的好处。这同样也关系到集体精神，结对编程加快了团队中开发人员间的知识传播，从而是任何一个开发人员能够应对几乎所有的系统中出现的问题。不断的轮换结对人员增强了这种集体精神。“虽然我们是面向任务的，但是在我得安排下没有人，没有团队在一起工作超过 6 周，一般的周期是 2 周。”,这样做增强了开发人员的学习能力，并加强了集体精神

－创造协作和支持的环境

开发者在结对时总能可以找到人来帮忙。而且这种结对的团队总有一个团队的人员来学习。通过这样做，就创造了一个协作和支持的环境。

**3.3.5 识别和管理开发人员。**

FINAPP 认为用人对敏捷方法应用得是否成功是一个非常关键的过程。可以认识到基于模式的开发过程和贯穿整个系统的标准接口的重要性的程序员是项目成功的关键。开发人员对于架构设计和设计模式的知识也是很重要的。大约只有 15%应聘者可以满足这种项目环境的要求。

把握开发人员的士气对项目的结果是非常重要的。在 FinApp 中，认为采用激励是比较好的方式。

因此，激励计划和灵活的首创精神被用来激励开发人员。“它创建了一个零消耗的环境”“尊重他们的贡献，不要把他们认为仅仅是机器上的齿轮。我知道如果你把他们说成仅仅是机器上的齿轮的话会产生什么样的后果。他们会变得很糟。他们会变得情绪很糟，这时你手边只有很糟糕的人。”灵活的工作时间，远程工作，关注结果而非微观管理，这些被认为是影响到开发人员的士气和主动的关键因素。

### 3.3.6 向前重构的代码重用。

基于模式和结构的开发方法使重用的范围相当宽广。重构被认为是跨功能的重用的一项重要技术。然而，这里的重构同 XP 实践中定义的还有所不同。在 XP 中，重构是不改变程序功能基础上的代码设计改进。冗余的代码被去掉，代码被充足，变得整洁，通用的代码被移出到单独的类当中去。在 FinApp 中，不是改变现存的代码，“向前重构“被大量的采用。向前的重构是利用现有代码，而并非重新构建，来开发新功能的一种方法。原有的功能保持不变，新的功能在现存的基础上开发。例如，”我们做的只是在现有的帐户汇款的基础上开始，向前重构，拿来帐户汇款去构建电汇的功能。你在 XP 里面构建的话，只是构建一次以满足目前的需要，当拿到新的需求的时候，你把他们覆盖掉，结果发现这一块是不变的，那一块变化的。所以，这就是我要加入设计模式所要解决的问题，将变化从静止中分离出来。现有的设计和代码被向前重构来加入新的功能。总之，典型的 XP 实践中，重构是改进设计和使系统更加强健的一种方法。在 FinApp，重构被看作一种在前期设计支持下的重用技术。这种架构不仅仅局限在象银行这样的应用，也可以修改后在任何一个金融系统中使用。

### 3.3.7 有控制的赋予权力－组织结构

FinApp 的项目经理们认为深层次的分级组织架构是极大惰性和迟钝的。“按照 XP 的说法，我们没有多层的管理。我们不喜欢中间繁复的管理层次。这种情况最糟糕不过。我们把这些都干掉了。（因此）我们有了非常快，非常敏捷，非常灵活的环境。”将决策权下放给开发人员，这一点被认作为敏捷方法成功的关键因素。“从我的观点来看，XP 的概念，很多都来自于面向对象的管理，把决策下放，赋权给那些真正在工作的人们，激发他们的生产力。其中的一方面就是不要坚持让他们共享一个监视器（结对编程）”

通过使用标准的接口和设计模式，权力的赋予被以一种优雅的方式控制着。开发人员解决自己的问题的时候不能偏离标准的方法。

## 3.4 超越 XP 实践－有趣的发现

总之，FINAPP 的设计模式概念，前期设计，分层的递交方法，结对编程，这些被认为是开发大型项目的最重要经验。另外，我们的研究还发现一些有趣的经验，他们和 XP 方法不同，但是也没必要反其道行之。

### 3.4.1 前期设计对其他实践的影响

根据 XP 的实践，大量的早期设计对应对频繁的变更是不合适的。但是，通过这个案例，早期的架构设计实际上支持其他的敏捷实践，诸如结对编程，重构和快速发布等。

设计模式的支持使结对编程的成功成为现实。模式预先确定了系统的架构，使对整个项目的理解得到了加强。基于准确的理解和标准的模式，开发人员之间的交流变得更加有效。设计模式同时也减少了详细文档的需求和分层的管理。这些都减少了开发的时间，有效的支持了快速开发。最终要的是，设计模式加快了结对的程序员之间的知识传递。新人总是和对设计模式了解的老手结成对。FinApp 的项目经理这么说："我能让一个新人在周一开始工作，在周四就开始产出……因为你花一天半的时间跟他讲解银行的模式，到了周四他就可以在这些模式上构建新的应用了。只要花 2 个礼拜就可以学习完全部的模式"

在 FinApp,重构带来的重用也被设计模式所支持。因为所有的功能都是遵循设计模式而开发的，现有的功能在构建新的功能的时候被大量重用。 "对我来说，XP 和设计模式使一回事，还有重构"

**3.4.2.管理的支持和企业文化**

FinApp 的案例显示管理的支持对采用敏捷软件开发是非常关键的。在同一个大型的传统银行兼并之前，FinApp 在交付产品的时间和质量方面一直都是非常成功的。而新的企业强调结构化的流程，正式和详细的文档和层次化的管理。"概念上是如果我有了这些流程，把握了这些流程的细节，就不会再有什么遗漏，就不用再担心什么了。但是从某些角度来看又不是这么简单，因为这些流程太繁琐了。" 企业文化的改变结束了敏捷方法的使用。需要编制详细的文档。从轻量级的方法返回到重量级方法使同甲方交流变得缺乏，从而导致了由于误解而产生的产品缺陷数量的增加。

## 4 敏捷原则和大型软件项目的实践

经过讨论发现， Baskerville[2]描述了一系列的敏捷开发的原则。在对多个美国的互联网软件公司研究之后，他们确定了一些快速开发的实践方法。他们将敏捷原则同传统开发过程和快速实践的原则都做了比照，他们发现每一个快速开发的实践都是同敏捷原则相对应的。他们也将所研究的实践和原则同敏捷宣言(www.agilealliance.com)中的进行了比较，敏捷宣言阐述了敏捷方法的价值以及敏捷方法同传统开发方法的区别。

改进的 XP 实践，在 FinApp 的应用是成功的，它在大型项目的限制下提供了敏捷。这些实践都是以敏捷原则为指导的。表 3 显示了修正后的 XP 实践是如何适用于大型项目，以及如何遵循 Baskerville 所提出的敏捷原则的。[2].

| No. | Agile Principles |
|---|---|
| 1 | 接受多种有效的方法（Accept multiple valid approaches） |
| 2 | 接受需求的变化（Accommodate requirements change） |
| 3 | 客户参与（Engage the customer） |
| 4 | 在成功的经验上构建(Build on successful eXPerience) |
| 5 | 创建优秀的团队(Develop good teamwork) |
| 6 | 符合项目环境约束的高效开发工作<br>(Effective software development conforms to project environment constraints) |
| 7 | 对软件开发过程的创新做好应对不可预知后果的准备（Prepare for unexpected consequences from innovation in software processes） |

表 2 敏捷原则(Baskerville et al 2003)

| 大型复杂项目的敏捷实践<br>（Agile Practices for Complex, Large-scale projects） | 敏捷原则（Agile Principles） |
|---|---|
| 前期设计（Designing upfront） | 1, 4, 6 |
| 短的发布周期，分层发布<br>（Short release cycles with layered approach ） | 2 |
| 客户代言人的参与（Surrogate customer engagement） | 3 |
| 灵活的结对编程（Flexible pair programming） | 1, 5 |
| 甄别和管理开发人员<br>（Identifying and managing developers） | 4, 5 |
| 基于重构的重用（Reuse with refactoring ） | 4 |
| 有控制地赋予权力<br>（Flatter hierarchies with controlled empowerment ） | 7 |

表 3 大型项目的敏捷实践同敏捷原则的比较

从表格 3 我们可以看出，FINAPP 的大多数敏捷实践是同敏捷原则相符合的 。这种符合一方面说明了这些实践的正确性，另一方面，它也建议了在大型项目中的敏捷原则的适用性，如下：

实践 1：前期设计。没有对 XP 实践全盘照搬。FinApp 使用了“改良的 XP“方法，它将前期设计同快速发布，结对编程，重构这些敏捷实践相结合。这个实践同原则 1 在“采纳多种有效的方法”的方面时吻合的，和原则 6 也时相符的。采用多种有效的方法，从另一方面来说， 也是环境的约束。FinApp 的功能是大而复杂的，也具有关键任务的特征。这种类型的系统开发需要一个仔细的结构设计

实践 2：快速的发布周期和分层的方法。FinApp 在每个很短的周期内进行端到端的功能（end-to-end functionalities）提交 。这样作是系统可以不断地适应新的需求变化（原则 2），递交的功能可以满足用户的需要，而不仅仅是死盯着规范的文档。

实践 3：加入客户的代理。这种实践对 XP 的客户代表方法进行了修改。 这符合了原则 3”使客户加入进来”，FinApp 不能直接接触到真正的客户，因此，它采用了产品经理作为客户代理的方法。

实践 4：灵活的结对编程。根据原则 5，强调“发展好的团队精神”，灵活的结对编程在 FinApp 中采用。 这个实践是 XP 的结对编程改进版本。同 XP 的“总是结对”不同，FINAPP 的开发人员在分析，设计和测试时是结对的，而编码是由程序员单独完成的。单独编程和结对编程的结合从结对受益的同时也克服了一些结对的缺陷（比如开发人员的抵制）

实践 5：甄别和管理开发人员。人员因素在敏捷开发中比传统的开发过程中更为重要。FinApp 强调为团队选择合适的人员，创造协作的环境以支持团队合作。开发人员在项目中不同方面的知识和经验是非常有价值的。

实践 6：向前重构的重用。这个实践对应着在成功的经验上构建这项原则。重构被用作一种提高重用的技术。象 XP 这样的敏捷方法并不强调重用部分的开发。他们更侧重速度，快速响应和随时编码。开发者往往仅关注满足目前功能的需要，而不是构建一个今后重用的部分。但是对于象 FinApp 这样的大型项目而言，前期的设计和设计模式就变得相当关键。系统的功能基于设计模式构建。处理业务的模块被仔细重构，进行泛化以便进行修改后可以处理不同的功能。通过这样做，代码可以重用，开发的进程也加快了。这种方法说明重用事实上和敏捷方法并不矛盾，而且应该包含在大型项目的敏捷原则之内。

实践 7：有控制的授权。在 FinApp,分层的管理被扁平的组织模式代替，它改进了交流，提高了生产力。开发者被赋予了自己决策的权力。另外一方面，对于大型项目而言，这种权力下放可能会导致意料不到的后果，诸如开发过程的不一致，不同的开发人员对产品概念不一致等等。 FinApp 通过几个步骤来解决这个问题（原则 7）。早期的设计和设计模式被用来标准化开发过程。比如异常的处理被集中化，以控制因为必要的变更所产生的缺陷。没有敏捷原则来处理组织架构，但是，我们的研究发现组织架构对于敏捷方法的成功采用是非常重要的。

## 5 结论：

目前软件开发需要以非常快的速度进行，为了及时满足需求的变化，轻量型的开发方法日益变得重要。但是这种方法并不能被直接应用到每个项目中，需要根据系统的性质和开发的环境进行修改定制。对于大型复杂的企业级项目尤为如此。根据上面的案例研究，我们提出了一些关于敏捷原则适用于大型项目的改进实践。我们将这些修改过后的实践同之前的研究进行了比较。建立稳定的系统架构是大型项目中敏捷实践应用同传统敏捷原则的一个非常大的区别。

本文对理论和实践有着以下一些意义：研究指出的关于对敏捷实践进行修改的原则可以帮助软件开发组织正确地采用敏捷开发方法。研究强调了在采用轻型的方法时所需要的一些注意的方法，分析了象 XP 这样的方法在特定环境下可能不会适用的条件。我们同时也指出了敏捷开发原则对大型项目的适用性。一般来说，XP 被认为适用于小型团队所开发的中小项目，但是我们建议对于大型项目而言，合理采用 XP 会使项目大获益处。

今后的研究会关注于确定轻型方法的理论基础，并将提供修改这些方法理论。进一步的案例研究会总结我们的发现，以确定修改开发方法时，所需要考虑到的项目特征，以及这些特征如何影响到我们的开发方法。

## 6 参考文献

[1]Manifesto for Agile Software Development. 2001, Agile Alliance:

[2]R. Baskerville, B. Ramesh, L. Levine, J. Pries-Heje, and S. Slaughter, "Is Internet-Speed Software Development Different?," IEEE Software, vol. Sep, 2003.

[3]K. Beck, Extreme Programming EXPlained: Embrace Change, Boston: Addison-Wesley, 2000.

[4]K. Beck and M. Fowler, Planning Extreme Programming, New York, NY: Addison Wesley Longman, 2001.

[5]B. Boehm, "Get Ready for Agile Methods, with Care," IEEE Computer, vol. 35, no. 1, 2002, pp. 64-69.

[6]A. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," IEEE Computer, vol. Nov, 2001.

[7]B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," Communications of the ACM, vol. 31, no. 11, 1988.

[8]A. Elssamadisy, "XP On A Large Project – A Developer's View," in Proceedings of XP/Agile Universe, Raleigh, NC, 2001.

[9]E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[10]J. Highsmith and A. Cockburn, "Agile Software Development: The Business of Innovation," IEEE Computer, vol. 34, no. 9, 2001.

[11]J. Little, "Up-Front Design Versus Evolutionary Design in Denali's Persistence Layer," in Proceedings of XP/Agile Universe, Raleigh, NC, 2001.

[12]R.C. Martin, "eXtreme Programming Development through Dialog," IEEE Software, vol. 18, no. 6, 2001.

[13]O. Murru, R. Deias, and G. Mugheddu, "Assessing XP at a European Internet Company," IEEE Software, vol. 20, no. 3, 2003.

[14]M.C. Paulk, "Extreme Programming from a CMM Perspective," IEEE Software, vol. 18, no. 6, 2001.

[15]J. Rasmusson, "Introducing XP into Greenfield Projects: Lessons Learned," IEEE Software, vol. 20, no. 3, 2003.

[16]D.J. Reifer, "XP and the CMM," IEEE Software, vol. 20, no. 3, 2003.

[17]B. Rumpe and A. Schr der, "Quantitative Survey on Extreme Programming Project," in Proceedings of XP2002, 2002.

[18]G. Schalliol, "Challenges for Analysts on a Large XP Project," in Proceedings of XP/Agile Universe, Raleigh, NC, 2001.

[19]C. Taber and M. Fowler, "An Iteration in the Life of an XP Project," Cutter IT Journal, vol. 13, no. 11, 2000.

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

# 探索需求

## 设计前的质量

需求之于开发，就像婚姻之于人生

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

# Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社

The Addison-Wesley Signature Series

# Patterns of Enterprise Application Architecture

中译本即将上市

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

UMLChina 训练教材

# 糟糕界面集锦——应用程序点评

Pixelcentric 著，雷立辉 译

查看评论

## MSN Messenger 点评

MAC OS X 版的 MSN Messenger 也是微软即时消息交流的客户端（当然它是一个巨大的 MSN 和 Passport 服务系统结构的一部分）。尽管它与 Windows XP 版的 MSN Messenger 程序是独立的程序，但它的功能是非常相像的。这利弊参半。


Mac 版的 MSN Messenger 美学观点显然被 Windows XP 版所影响，但幸运的是它还有一些 Mac OS X 风格。比如说，工具条上的图标看起来有点像 Aqua 那种透明塑料的感觉而不是 XP 的那种硬塑料的感觉。这一点符合这个程序的所有控件。

“发送”按钮是一个巨大怪异的东西，它的高度竟然和文本输入框是一样的，甚至更大一些。回车和点击按钮都会发送输入的消息，所以让人觉得微软可能想用这种形式来强调这个按钮。想到苹果即将发布的 iChat，这个软件就根本没有“发送”按钮，回车也和 MSN Messenger 一样是可以发送输入的消息的。这就足够好了，不过如果添加一个小的“发送”按钮可能会更好一些，因为这样会消除一些用户找不到“发送”按钮的可能性。

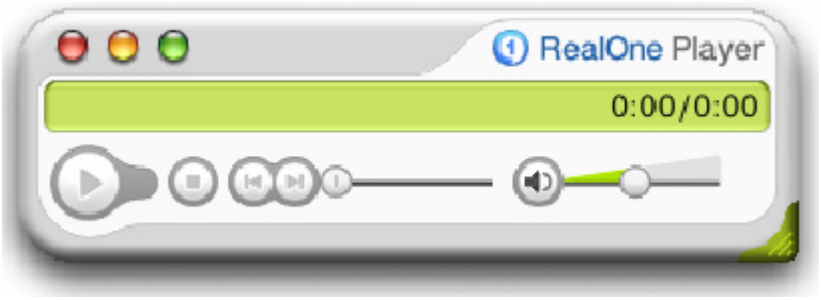
MSN Messenger 另外一个不足之处是对于表情图标的管理。点击这个表情按钮会弹出一个含有一组斜角按钮的菜单。这错误不仅在于它错误的使用了斜角按钮（bevelbuttons）（因为它们的行为像一个 menu woud），而且，这样会使得无处安置对于表情图标的解释文字。而 iChat 则有更好的解决方案。文字解释赋予这些表情图标一个更明确的含义，在发送端和接受端都不会引起误解。当然，这假设通讯双方都看过对表情菜单的解释。

在这个表情图标菜单的右下角的“…”项也会迷惑用户，因为用户会认为它本身就是一个图标，其中的图案就是那三个点；然而，点击这个按钮会打开一个对所有图标的浏览窗口，包括所有没有出现在这个菜单中的图标。为什么微软要做一个这样的图标浏览窗口呢？或许，微软就像 iChat 一样，加一个文字菜单而上面的“…”换成“More…”会更加合适一些。

最后值得一说的是，MSN Messenger 虽然只含有四个项目，但却比 iChat 的含有四项的工具条占据的面积大 9 倍。尽管 MSN Messenger 工具条的图标也具有形象的含义，但它们仍然有解释标签。但它的图标是 32x32 象素大小，比起 iChat 的 16x20 象素的按钮费空间多了。

## RealOne Player 点评

尽管 RealOne Player 具有非常简单的操作界面，但它却不尽人意，它与 Mac OS X 的好多界面风格传统相抵触。




RealOne 主窗口如此的不标准以致于让用户觉得无处下手。窗口的四角比标准的 Aqua 窗口更加圆。有趣的是其它的三个角都是圆角但右下角却相对尖一些（尽管 Aqua 也有点太圆了）。和 QuickTime 一样的毛病，它缺乏明显的窗口标题栏让有些用户感到有点迷惑。加之 RealOne 窗口标题栏没有显示标题文本而只显示了程序 logo 文本，这更加造成混乱。

为了它的“超炫”特点(wow factor)，如果将光标移动到上面，RealOne 播放器窗口大小改变控件将会加亮为离子绿色。对于其它控件也是如此。这让界面看起来有点紧张，尽管它相当简单。

非常明显，按钮不是标准的，而且设计得很差。如果它们处于失效状态，会让一些用户不能看到它们的存在。第二个按钮起的那些按钮（就是播放按钮相邻的右边那些按钮）太小了，它们应该更大一些。理由便是它们周围的空白太多。有意思的是，音量滑动条标题图标也是一个按钮，正如它周围的灰色边缘所示。但是，当这个按钮被点击时却不明显。实际上，点击这个按钮会使播放器静音，即使这个图标依然表示声音依然从喇叭放出。RealOne 看起来将这个表示颠倒了。

这些奇异之处和那个复杂的参数控制面板只能降低产品的竞争力，也就一起表明了 QuickTime…或者 Windows 媒体播放器的成功。看起来 RealOne Player 设计目的是为了显摆，而其对手的产品却从使用性角度来考虑。


*QuickTime 播放器或许不是纯 Aqua，但它至少看起来很干净和直观*


*有意思，微软的这个却是三个之中最符合 Aqua 风格的*

## QuickTime Player 6.3 点评

QuickTime Player 5 和 6 比起版本 4 要更加有条理和布局得当一些，但它的视频和音频播放控件操作性仍然不尽如人意。


为了使用音频控制功能，用户必须点击一个看起来更像是图片的按钮的非标准控件。这个问题控件就是在右面黄色工具栏上的音量显示控件。当这个控件被点击的时候围绕在其周围的边框才会出现。而正常情况下这个黑色边框是不可见的，也就是说，这个功能不是很明显。

而视频控件就根本不能通过点击窗口中的一个控件来打开。命令 K 或者点击菜单“显示视频控件…”才能显示出一个半透明条目，上面包含了一个控件。为了改变不同视频设定，用户只能通过点击一个非常小的灰色箭头来控制，而这个小东西非常不起眼（就在现在控制选项名称的右面）。每一次只能显示一个选项，这有个好处便是能让你很容易的看到你现在播放的视频。但仍然有一点需要提出，这个控件首先不应该放置在那儿，而应该是一个浮动的窗口；而且这个窗口有几个分割的部分，把这些视频控件（对比度，饱和度及亮度等）都放置在同一个窗口里，可以让用户一下子都能访问。

QuickTime 界面使用了金属质地，这种风格本来是 Mac OS X 专门预留给这些诸如媒体播放，音频输入等这些数字设备管理程序使用的。QuickTime 播放器并没有使用系统 API 来生成这种金属纹路界面，而使用了程序自身所带的位图图片。这就是说如果像 Mac OS X 10.2.3 版本中设置系统界面为金属风格时，QuickTime 播放器将和其它所有的金属界面程序表现形式及行为有一些差异。这也就意味着如果用户改变了系统显示主题风格而使用了其它非金属界面风格，QuickTime 将不会适应于新的设定。

即使用户选择了隐藏文件扩展名，QuickTime 播放器窗口标题也同样会显示文件扩展名（如截图所示）。

让人觉得不可思议的是 QuickTime6.3 依然不支持保存包括扩展名在内超过 31 个字符的长文件名的文件。

这个截图是使用芬兰区域环境。有趣的是，音频控件并没有本地化。而其它如平衡，低音和高音控件文字还不是芬兰语。

## Adobe GoLive 6 点评

尽管这个程序是 Mac 操作系统的重要程序之一，它仍然不是一个顺从的孩子，因为它没有遵守苹果操作系统人机交互设计指导原则（Mac OS Human Interface Guidelines）。所以，它仍然以界面设计缺乏一些必要的修琢而让人大跌眼镜。


程序界面中几乎所有的地方的字体都是那么的小，即使有足够的空间让它们以大字号摆放。操作系统对于菜单字体的推荐大小为 12 个象素，但这个菜单仍然我行我素：两行小字儿外加一条分割线。同时我们也要注意到它使用斜面按钮作为列表框的标题。这样的错误在这个程序中不至一处。当然我们都清楚，Mac OS X（及其所有 Mac 操作系统版本）都提供了标准的列表框，而且标题栏还提供比如像正序排列，倒序排列，调整各列的相对位置等等丰富功能。（也就是说这个程序弄巧成拙，不用系统提供的标准控件而使用了性能很差的自定义控件。译者注）


和程序界面其它地方一样，在可用服务器窗口的字体也是异乎寻常的小。文本域的文字应该是右对齐而它却左对齐；图标的使用毫无意义所以应该删掉从而给其旁边的标签文字让出更大的空间；在列表框下面的“新建”和“删除”按钮应该变成全尺寸的按钮。这两个按钮太小了所以用户可能不能容易的发现它们，而且即使发现了它们，用户也不能很清楚的知道它们的用途。此外，在列表框标题栏仍然使用了可恶的斜面按钮。

下面是一个在 GoLive 和其它 Adobe 程序中使用的 Adobe 风格的属性页。为什么在整个窗口中只有一个属性页而它还要以属性页显示？直接将这个属性页的标题变成子窗口的标题不就行了吗？如果再有新的属性页添加进来后再变成属性页也不迟啊。同样也要注意到他们空间利用率太低。文字太小而且尽管这是最小化状态而窗口还是有许多无用空白。

当然这个程序不是一无是处。比如说，它提供了一个非常方便的创建链接的连接工具（就是窗口最上面文本框的左边的小按钮）。只要按住这个按钮然后将光标移至想要的地方松开就会生成一个地址行。然后你就可以将这行文字拖放至站点管理器或者别的文件窗口就会生成一个 URL 地址，而避免了使用键盘输入冗长的地址。更精彩的是，将这个地址拖拽时，光标后面的窗口将会逐个交替的弹到最上面，所以就是被别的窗口所覆盖的窗口也有机会显示在最上面，这就避免了创建链接时先要手工重新安排窗口。

## Mozilla 1.3 点评

*注意：根据 Mozilla 开发总部的计划，他们将在 1.5 版时使用 Mozilla Firebird 模块来代替现在的浏览器模块，所以这段评述的有效期不会太长。*

Mozilla 浏览器的排版引擎（Gecko）是最有兼容性的引擎。但糟糕的是与所有正式出版的浏览器相比，它的用户界面友好性却是最差的。Mozilla 的用户体验是如此之差，使得那些讨厌使用这个浏览器的用户尖酸地说这个浏览器不是为普通用户设计的而是为那些网络好手特制（fringe browser）的边缘浏览器。这当然是我们不能接受的事实，因为这个浏览器的目标便是为网络界设计一款标准而兼容的浏览器，而特制浏览器的设计目的与此背道而驰。实际上，一些后来的 Mozilla 批评者开始声称 Mozilla 既不是给最终用户设计的，也不是为网络好手设计的 ，而只是夹杂在其它应用程序中间的一个能工作的普通应用程序。这当然是不足为信的，因为 Mozilla 引擎是几个面向终端用户的浏览器的核心支撑部分，最有名的应该是 Netscape 导航者浏览器了。这些声称 Mozilla 浏览器不是为终端用户设计的人显然是将 Mozilla 和排版引擎 NGLayout（Gecko）概念混淆了。实际上在别的浏览器产品上使用 Mozilla 引擎还是相对轻松的。

Mozilla 浏览器界面提供了两套风格的主题：一套是多平台统一风格的现代界面，一套是尽力模仿所安装操作系统界面风格的传统界面。大多数情况下，传统界面工作得较好而现代界面却完全失败了。但是传统界面也是不尽完美。下面所说的便是这两套风格界面的缺陷。请注意下面的界面截图是笔者在 Mac OS X 上故意使用用户自定义主题而 Mazilla 使用传统主题界面。这样会让我更加简单的指出 Mozilla 并没有使用 Mac OS X 界面显示管理器所设置的颜色设定，而它只是简单的假设了用户使用 Aqua 风格（当然苹果自己的软件 iTunes 在这方面也是犯了同样的错误）。

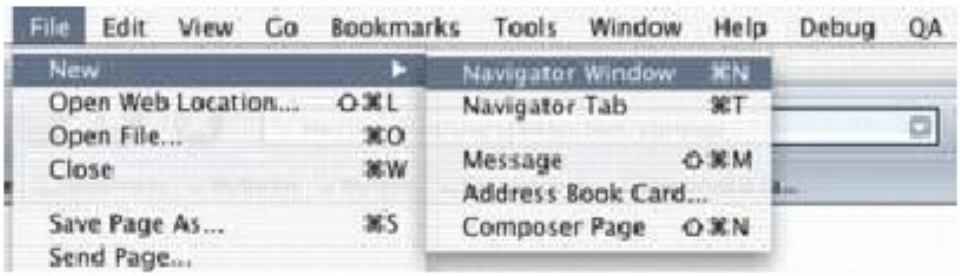
当 Mozilla 在传统风格模式运行时，你最先看到的便是这个 Netscape 4 式样的工具条。这和下图的现代风格界面的工具条一样，表面看是一个可以定制但实质上是不可定制的。虽然它的地址输入栏额外支持 Quartz antialiasing，但它并不是标准的 Mac OS X 控件。糟糕的是，它和 Mac OS X 文本输入框行为稍微有点不一样。Mozilla 其它地方的界面文本输入框如在网页中显示的文本输入框，也和 Mac OS X 中的文本输入框行为有点不一样。（也就是说，你在程序中使用的标准界面元素行为最好和做在操作系统的界面元素行为完全一样。译者注）。

界面中的图标和 Netscape 4 中的图标是一样的，这让人觉得这个程序并一个不是标准的 Mac OS X 程序而仅仅是可以运行在 Mac OS X 上而已。不过，在窗口标题栏右边的工具条开关按钮设计样式则是一个迈向 Mac 风格的进步。


当使用现代风格界面时，用户会觉得它们的确面对的是一个和 Mac 控件风格完全不一样的界面。

奇怪的是，“Home”按钮从浏览工具条上消失而出现在书签工具条上。甚至更为奇怪的是工具条的位置不再是可以调整的了。用户可以隐藏工具条上的按钮（例如“Home”按钮），但工具条不可移动而且按钮位置不可交换。这根本不像程序 Mail，Finder 或者 Chimera 那样其工具条是可以完全定制的，更别说它是一个标准的 Mac 风格工具条。

现代风格界面中的按钮也和程序“sihouette”的按钮一样，这可以认为是这些愚蠢设计的副产品（by-design stupidity）。工具条图标的设计原则应该是图标之间尽量要以形状及颜色相区分。而这几个图标则都没有做到，它们都不起眼，且都是平面的和圆形的。

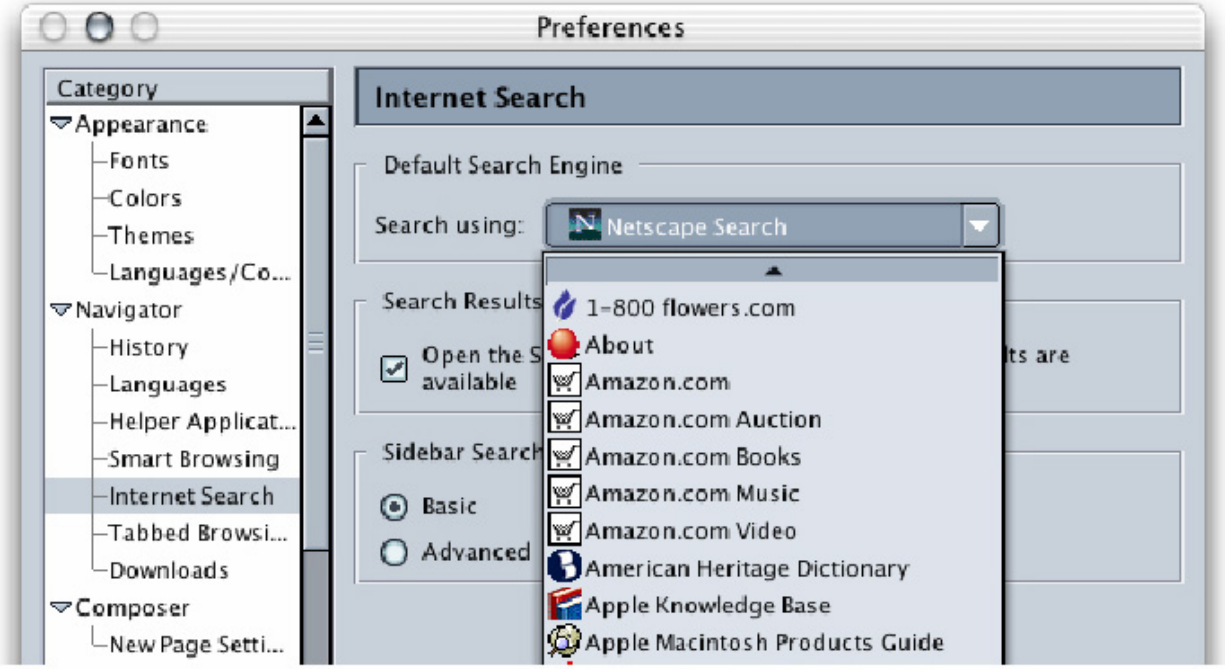


Mozilla 的菜单结构也是一塌糊涂，尽管现在比以前进步了一些。非常常用的命令如“新建一个窗口”，“新建一页(New Tab)”被放到了子菜单里面，这和通常程序中将新建命令菜单放到文件主菜单的前几项完全不一样。同时，像“发送页面”这样的不常用命令却在一级菜单中堂而皇之占据了一个位置。其它主菜单项的情况大致相似。对于一些初级网络用户根本不会用的菜单项却放置在一级菜单里面。

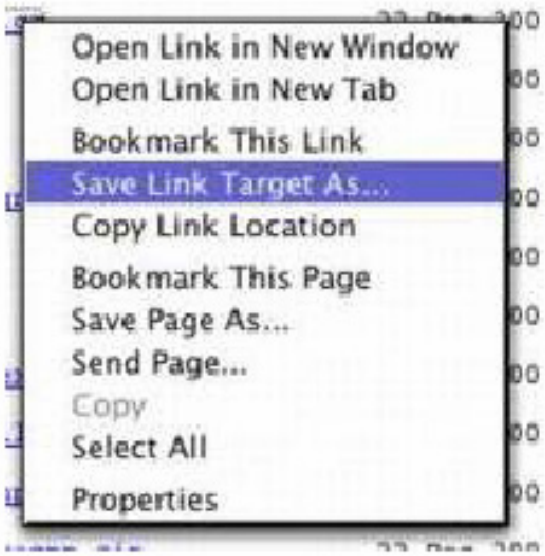

对于与之具有竞争关系的产品像 Internet Explorer 都把“邮件”这样的按钮放到主工具条上以使它更加容易发现和操作，Mozilla 却将它放到了左下角与状态栏相邻的地方。这样会让用户觉得这个“邮件”图标和其它子程序图标都是像状态栏指示一样，点击“邮件”按钮不能启动发送邮件窗口。幸亏这些按钮具有工具条提示，但是如果用户鼠标不移动图标上面是不会显示这些提示的。而糟糕的是这些图标本身就很不起眼。

这个参数配置窗口的设计可以说是糟糕透顶了。这些选项分成几个大组，每个大组里面又有几个小组，而且如果把它们都展开，它们的高度将超过列表高度的两倍。就像其它的 Mozilla 窗口组件一样，这个窗口使用自定义控件。这些控件操作方式和 Mac OS X 的标准控件大体差不多，样子也符合自定义风格。不幸的是，这个弹出菜单操作方式却和操作系统的标准菜单操作方式有所不同。

如果用户使用传统风格主题（如上图），弹出式菜单风格类似于 Windows 的弹出式菜单。但它的可滚动列表让我们想起列表框，而不是那种当用户将鼠标向上移动或向下移动时菜单项会自动滚动的菜单。




如果用户用的是更加新异的现代风格界面（如上图），这个弹出式菜单则更像 Mac OS 中的标准菜单风格，但工作方式仍与之相比有细微差别。为了将这个列表向上（下）移动，用户必须将光标移动到这个“箭头”项上（就是上图中的高亮的部分）。只能这样移动菜单项。而 Mac OS X 的标准菜单只需要将鼠标箭头移动到菜单接近顶部或底部菜单项就能滚动菜单项，而且它的滚动速度与光标箭头和菜单上下中线距离成正相关。Mozilla 的这个菜单却缺乏这样的特性。




最后，传统风格界面中的上下文关联菜单就像是一股复古之风（a blast from the past）：这个菜单样式选用了 Mac 系统 7 的菜单的白色背景而使用了 Mac 8 中的蓝色高亮风格，即使用户将操作系统主题颜色风格设置为青灰色也是如此。这些菜单不会沿用 Mac OS X 用户设定的颜色风格，以致于造成了这些菜单不伦不类，十分显眼，即使用户不使用自定义颜色设定也是如此。这是相当奇怪的，因为 Mozilla 的大多数控件在使用传统风格界面时都是因显示管理器（Appearance Manager）的设定而变化的。当然，对于这个毛病的快速解决方法就是给这个菜单加上灰色的外边框。这样至少会在 Mac OS X 的缺省蓝色主题下显得稍微溶入一些。从长远来看菜单应该采用与 Mac OS 菜单一样的行为和显示方式。或许 Mozilla 的 Mac 版本应该像 Camino（一个 Mac OS X 的浏览器）一样在 Mozilla 的 NGLayout 引擎中实现一组 Aqua 风格的上下文菜单控件。