第 42 期（2004 年 10 月）

王晓昀

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/

Borland 为微软的 VSTS 提供 UML 支持

[2004/10/26]

软件开发优化（Software Delivery Optimization）方面平台无关解决方案的领袖公司 Borland 软件公司 (Nasdaq:BORL) 今天和微软及其它业界领先的公司结盟，宣布对软件工厂的支持及未来的计划。软件工厂是改善软件开发的新方法。

作为支持的一部分，Borland 还宣布将提供领域相关的建模解决方案：为微软的 VSTS（Visual Studio 2005 Team System）提供 UML 建模支持。

Borland

Borland 的首席技术官 Pat Kerpan 认为， “今天，软件开发已经是一种艺术，开发人员每天都要从事的艺术。尽管软件开发总会从艺术一样的途径中获益，Borland 同时也相信，更高效和可预测的开发过程将会带来巨大的商业利益。软件工厂方法将提供高度自定义的工具和内容，可以演进并自动化开发的过程，使得开发人员可以将更多的注意力集中在他们的艺术上。自动化也将带来巨大的利益。人、过程和技术的结合是 Borland 的软件产品提供优化愿景的根本所在。”

软件工厂包括领域相关的工具、过程和内容，该方法希望可以在软件开发中自动化可重复和可预测的过程。这使得开发团队可以更多地关注于优质程序的更快速的开发，并保证程序和设计时的业务目标保持一致，这一点是软件开发优化的重要原则。

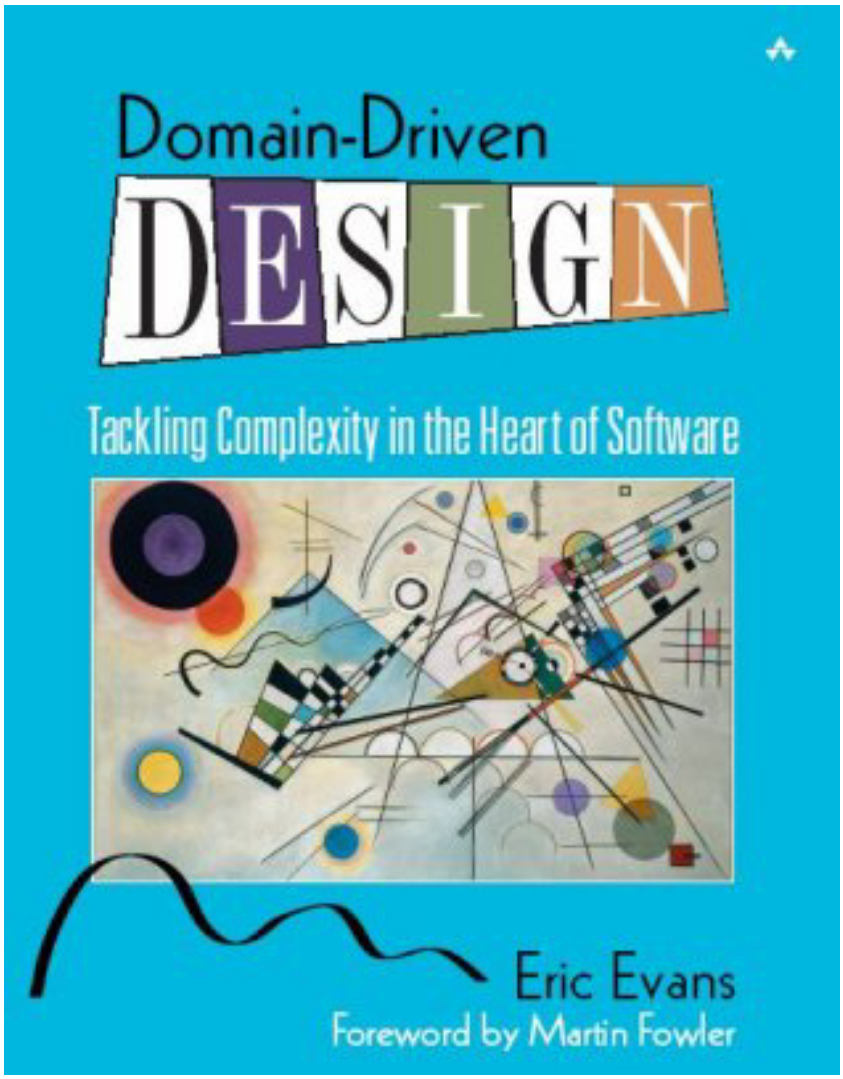
为加速软件工厂的创建，微软为 VSTS2005 引入了众多开发自定义设计工具方面的软件组件。Borland 计划 2005 年上半年之前首先在其已有的 Together for Visual Studio .NET 建模方案中支持 UML2.0。Borland 还希望在 VSTS2005 发布之后借助于微软最近宣布的一些组件来在 VSTS2005 中支持类似的功能。

作为 Borland 自己在推进可视化设计方面的努力，该公司同样于本周宣布了一系列考虑到架构师、设计人员、开发人员等不同角色的角色相关的建模解决方案。Borland 的 Together 建模工具产品线也通过提供基于角色的协作的高端建模能力，使得覆盖开发团队和整个开发声明周期的设计信息的检查、分析和交流变得更加容易。通过对 Borland JBuilder(R), Borland C++ Builder(R), Eclipse 和 Microsoft Visual Studio 的支持，Together 产品已经将 UML 变成了多种领域相关的语言。

微软 VSTS2005 的主管 Rick LaPlante 认为，“软件工厂为软件开发提供了比 UML 和其它建模语言可以提供的要更详细和更简单的方式。但是，还有些客户想要更严格的可视化设计途径，如 UML。对这些客户，Borland 提供了好的建模解决方案”。

了解 Borland Together 建模解决方案新产品线的更多信息，请访问 www.borland.com/together。

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



众多问题根源在于：领域模型深度不足，没有抓住业务本质！

《领域驱动设计》中译本即将由清华大学出版社出版，UMLChina 审稿

IBM Rational 新开发平台对非洲的影响

[2004/10/14]

很多新近涌现的企业都将很快可以通过开源的 Eclipse 开发框架初次使用到 IBM Rational 的软件开发平台。这将帮助他们通过无缝软件开发建立起竞争优势。

根据 IBM SA Rational 产品专家 Nadeem Malik 的观点，这个新平台将于下月发布，它特别考虑了约 2000 家从事构建、扩展、现代化、集成以及软件部署的南非公司的使用。

Malik 认为，无论是主流银行这样的大企业，还是小型中型的企业，开发人员都需要变得更加负责任、弹性，并通过整合他们的开发过程而集中各方面的精力。同时通过整理为最佳实践（best practice）的方式（以成为行业标准为目标）提升雇员的技能。

同样地，CRM 包（译者注：客户关系管理）中提供了整合销售、计价、顾客关怀过程（customer care processe）的工具以提供竞争优势，新的 IBM Rational 开发平台中也有这样的功能来保证企业可以整合软件开发过程。通过在免费的 Eclipse 工具集中嵌入新的 IBM Rational 软件开发平台的主要的功能，IBM 希望可以在新兴市场中得到大量的开发者团队，尤其是在非洲。

新的功能包括用以进行优先级排序、计划、管理以及 IT 项目度量的工具；帮助架构师设计应用架构的工具、基于最新的 UML2.0 的建模工具以及一个为业务分析人员设计的测试工具。

Malik 认为，通过提供紧密联系业务、开发以及操作的解决方案，IBM 展示了从 Rational 的第一个联合开发团队开始五年来在软件开发方面的显著进展。

（自南非 itweb，UMLChina 袁峰 摘译，不得转载用于商业用途）

Aonix 发布 High-Integrity Profile for Ameos

[2004/10/13]

Aonix 发布 High-Integrity Profile for Ameos；并在嵌入式系统上展示了相关 Demo, Demo 展示了 UML 建模和实时命令关键 Java 应用的代码生成能力；

安全和任务关键全套解决方案提供商 Aonix(R)今天发布了应用于实时和任务关键应用的高完全 High-integrity UML profile（HIP）。这个 profile 是对 Ameos UML 和 MDA 建模环境的扩展，和目标语言无关、并提供经过测试的用于实时通讯及分布式开发的常用模式的实时实现；该 profile 由 OMG 的 UML Profile for Schedulability、Performance and Time 规约而来。并在嵌入式系统上给出了 HIP 的演示，展示了符合 UML2.0 标准的 profile 包和代码生成器是如何提供基于标准 J2SE 源代码的实时扩展的。

HIP 通过预定义的版型（stereotypes）（如 HIPeriodic 和 HISporadic）简化了使用 Java 的分布式应用开发任务和远程方法调用 RMI（remote method invocation）的开发。为了满足任务和安全关键应用的开发人员的要求,HIP 的通讯模式，如黑板（balckboard）、缓冲区（buffer）、事件（event）等，都基于 ARINC-653 标准的。该标准是由 Aeronautical 通讯公司（ARINC）开发的用于进程或者线程异步通讯的标准。

Aonix 公司的产品市场主管 Michael Benkel 指出， “HIP 为实时市场提供了成熟的通讯模式。使用 HIP,开发人员为程序应用的模型会更加易于阅读，因此也更易于实现和维护。同样，我们也提供了一个强有力的转换引擎，以实现从模型到目标相关代码的转换，从而大大缩短了开发过程。”

通过 Ameos 的 UML2.0 Profile Editor，HIP 允许定义版型和属性，并分配给 UML 元模型的模型元素，并保证这些 profile 是设计良好的、文档化的、易于被整个项目团队所使用。Ameos 的转换引擎中用到模型驱动架构来提升模型的抽象层次、减少对目标平台的依赖。通过特定的转化规则，HIP 将 UML 模型映射到实时的 Java 代码，得到的 Java 代码是符合 RTSJ 的，并且可以在 PERC 上执行，PERC 是 Aonix 的 Java 实时虚拟机。

HIP 基于一个欧盟资助的项目中开发的多个 profile 和生成器。该项目中涵盖了实时 Java 工具链中的各种规约和实现，包括了实时的 JVM、建模工具和模型验证工具。作为命令和安全关键领域的领头羊，Aonix 的顾客必须遵循多种业界特定的标准。强大的 MDA 解决方案可以轻松地适应（adapt to）不同行业（例如航空电子工学、国防、汽车等）的各种标准和认证。

这个 Profile 中包含了 UML 的 profile 定义和应用于实时 Java 虚拟机的转换规则。HIP 是标准 Ameos 产品的一部分。包括了 HIP 的 Ameos 将很快可以应用于 Windows、Linux 和 Solaris 平台。

（自 tmcnet，UMLChina 袁峰 摘译，不得转载用于商业用途）

征稿

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

Metamill 公司发布 Metamill 4.0－支持 UML2.0 的 UML CASE 工具

[2004/9/29]

Metamill 软件公司宣布了其可视化 UML CASE 工具 Metamill4.0 版本的问世，该工具支持双向代码工程，并在这一新的版本中支持 UML 2.0。 (http://www.metamill.com)

4.0 的新功能：

- 支持 UML 2.0

- 对嵌套元素的更好支持

- get/set 方法的自动生成

- 对完整方式源代码（integral method source code）的支持

主要功能:

- 可视化 UML 建模，具有直觉而迅速的用户接口

- 对 C++, C, Java 和 C#的双向代码工程支持

- UML 2.0 以及 XMI 1.2 支持

- HTML 文档生成

- MetamillScript － 脚本语言

- 快速的 Windows 二进制程序 - 没有字节码

“UML2.0 是 OMG 的 UML 标准最近的更新。Metamill4.0 对 UML2.0 建模技术提供了强有力的支持－模型从抽象形式到接近实现的程序中，用 MDA 的术语来说就是 PIM 和 PSM 之间，涉及到很多细节，Metamill4.0 和这些实现细节紧紧绑定起来;最了不起的事情就是 Metamill 仍然是一个低价的工具，同时提供了和贵得多的其它软件一样的功能。”

适用性、价格和系统要求：

Metamill 要求 Windows 95/98/NT/Me/2000 或者 XP，仅售$125 每 license，另外大量购买和升级还有折扣。可以从 www.metamill.com 在线购买 Metamill。另外，网站上也提供有 30 天免费试用版本及屏幕拷贝。

关于 Metamill 软件公司：

总部位于 Luxembourg 的 Metamill 软件公司开发业务市场领域的软件工程产品。其旗帜产品 UML CASE 工具 Metamill 已经赢得了各大洲的众多用户。

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

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于：
领域模型深度不足，
没有抓住业务本质！

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

软件工程技术丛书

设计系列

企业应用架构模式

Patterns of Enterprise Application Architecture

(美) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校

UMLChina 训练辅助教材

连接三种建模技术——王晓昀访谈

think 文

讨论

2004 年 10 月 22 日，PowerDesigner 总设计师王晓昀在中国举办讲座“展望 PowerDesigner 11.0 最新功能”。笔者到现场聆听，并利用一起午餐的时间询问了王晓昀先生一些有关 PowerDesigner 的问题，整理如下。

●——笔者，★——王晓昀。

（右为王晓昀）

●PowerDesigner 目前在世界各地的销售分布如何？

★大约 40%在美国，40%在欧洲，还有 20%在世界各地。

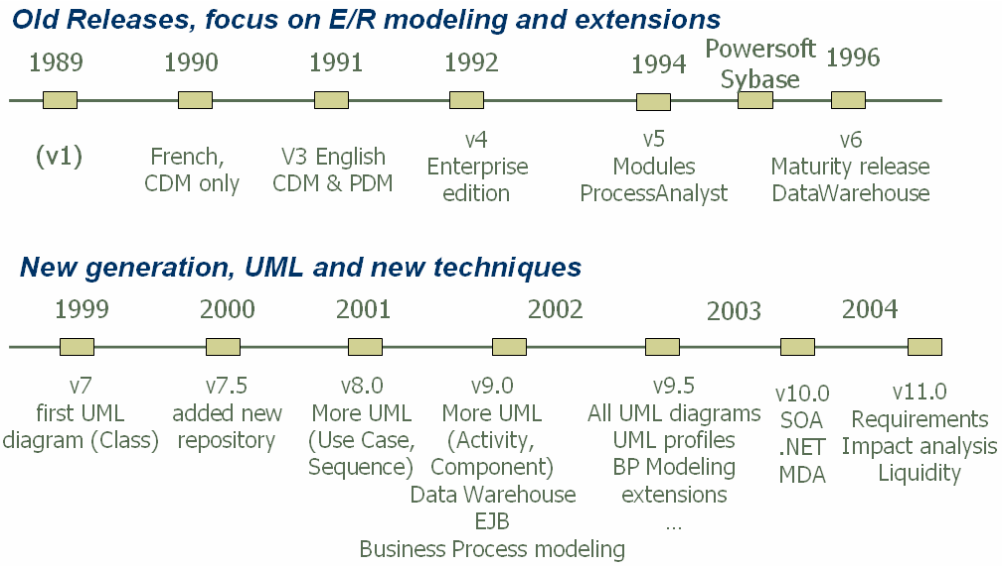
●其他工具厂商像 Rational、Telelogic、Borland 在工具背后会有像 UP 这样的应用开发周期的过程指导，PowerDesigner 的功能从业务建模到代码都有，但是似乎没有这样一个过程指南..... PowerDesigner 最适合什么样的方法学？

★可以把 UP 再加上一些企业流程建模的东西，还有数据库建模的东西。目前我们没有一个像 UP 这样的东西，主要还是没有足够精力分散来搞。但我们会通过写各种书，通过完整的开发实例，向 PD 用户展示 PD 在软件开发过程中的应用。

●PD 使用者觉得 PD 在使用的交互设计上比别的工具要合理，当初设计 PD 时在这方面有没有什么特别的考虑？

★我从 1989 年开发第一版时，是我自己给自己编的，我就是想怎样才会好用，像画 Class，可以连续一个个

往上放，然后一个 Right Click 就可以了。当时 Windows 还是 1.0 版本呢。后来经过大家使用不断反馈来修改。



●PD 是否打算开发多平台的版本？刚才讲座时我听您说到 Sybase ASE 获得 Linuxworld 的最佳 Linux 数据库大奖……

★我们也想，但我们的代码用 C++编写，好多地方用到了 Windows 的 API，如果有好的办法不用太费神就可以分离到 Linux 平台，那当然好。目前我们 95%的用户都是使用 Windows 平台，所以暂时还不是一个大问题。有一部分客户是 Linux 平台，但设计的时候还是可以在 Windows 机器上设计，或者在 Linux 上装 Windows。

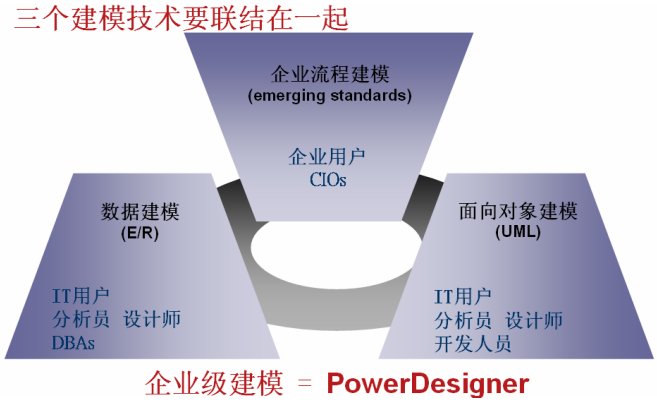
●现在越来越多的建模工具逐渐合并到 IDE 中，不再以单独的“建模工具”存在，PD 是否也要做这样的变化？

★现在已经有 Eclipse 3.0 plugin，可以在 Eclipse 上使用 PD。PowerBuilder 里也可以使用 PD。目前正在做 Visual Studio .Net 的。明年估计 5 月份左右发行。到时候在 VS.Net 里面都能够使用 PD 的各种模型。

●微软明年也要正式推出 VSTS，里面也有自己的建模工具，PD 如何应对？

★他们主要是针对 C#和.NET，对我们影响不是特别大。PD 不一定要和编码连在一起，可以做上层的分析，以后如果 UML 到达可编程的程度，也可以放到微软平台上。一般大公司里会有三个软件：用 Rose 或 Together 画 UML，用 PowerDesigner 或者 ERWin 做数据库设计，还有企业流程建模的工具。如果说只允许选择一种工具来完成上面三个任务，一般人都会选 PowerDesigner。因此微软的动作对我们来说影响不大，我们支持微软，不和微软竞争。像某些厂商那样集成度高有好处也有坏处，很容易造成重复购买。应该允许从这家买点这个，从那家买点那个，这样有选择的话，厂家就不容易提价。我们的客户在各个平台、各种语言都有，所以只要有可能，我们会尽力支

各种语言和平台。

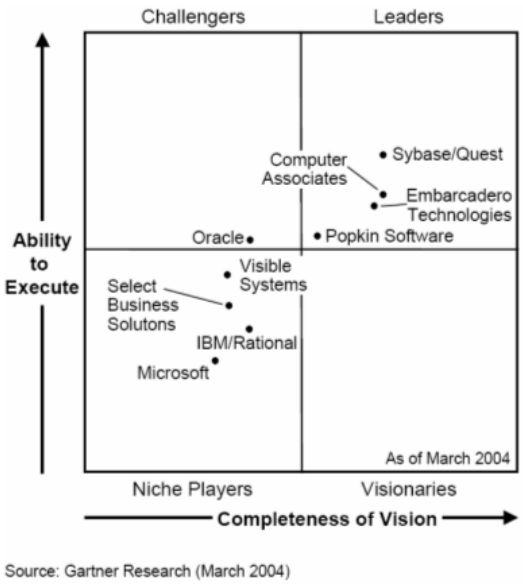


●PowerDesigner 在团队开发方面的考虑是怎样的？

★目前通过 Repository 来实现，还可以设定谁能改哪部分，还可以作对比，如果觉得你改的比我改的好，我可以选择合并你的。对于远程协作建模方面，目前还没有支持到对同一张图的实时修改。可以通过 Netmeeting 来协作。

●PD 是数据建模起家，现在以 UML 或 MDA 工具的形象出来，如何对待以前的形象？

★目前在 Gartner 上，数据建模方面 PD 排第一，是 Leader，但在 UML 方面，他们把我们放在 Visionaries 或者 Challengers 那边，我们问他们，要变成 Leader，我们还缺什么？他们告诉我们需要三个东西：Marketing、Marketing、Marketing。不是产品的问题，而是要花很多钱来宣传这个地方比 Rose 好，那个地方比 Together 好，要做好多的工作。我们不能作正面竞争，还是想通过数据建模上的地位来发展，慢慢往外扩大。



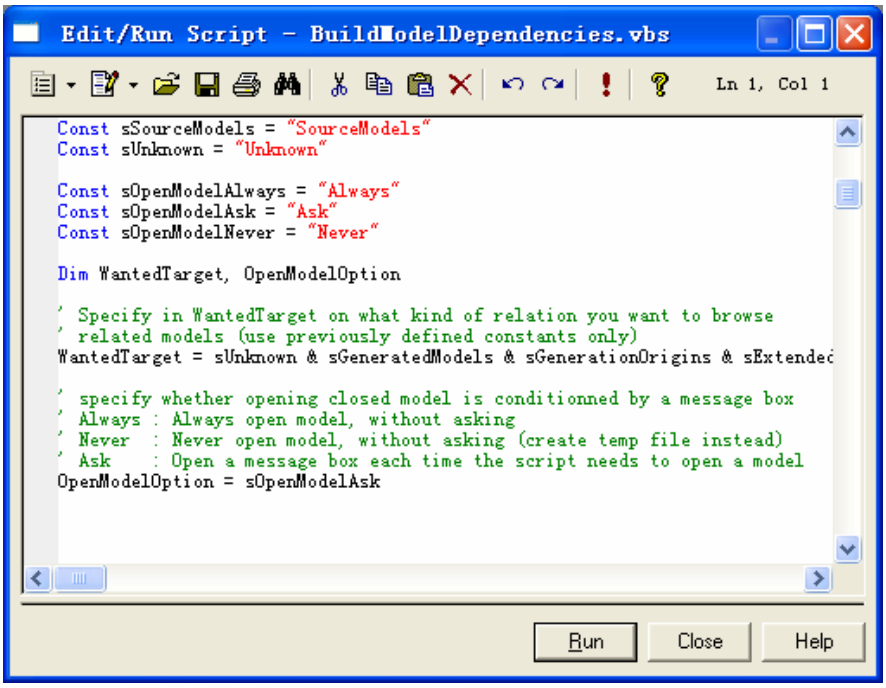
Source: Gartner Research (March 2004)

●PowerDesigner 打算什么时候支持 UML2.0？

★下一步。但也不会全部都加进来，这样可能太复杂了，我们会有选择地加其中一部分。

●对于模式的支持呢？设计模式或者针对行业的分析模式？

★目前还没有支持。我们也鼓励客户提供自己行业的模型，收集整理好以后可以放上去。这样，开发一个银行管理之类的模型，就可以在原有模型上面改，不必从头画起。现在也可以自己通过一些 VBscript 来扩展 PD 自己实现这方面功能。



- 2004 年 10 月 22 日王晓昀“展望 PowerDesigner 11 最新功能”讲座实录下载>>
- 2004 年 3 月 25 日王晓昀“PowerDesigner 与模型驱动开发”讲座实录下载>>

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

清华大学出版社

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

探索需求

设计前的质量

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

Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

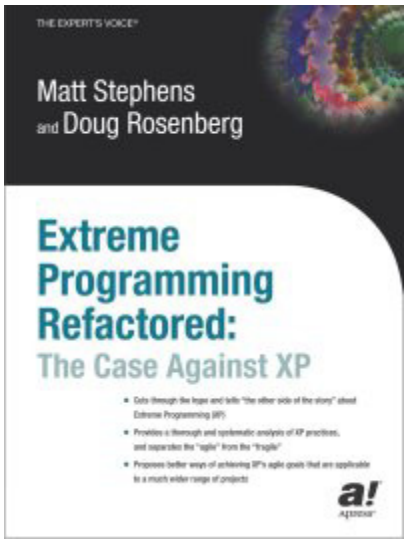
清华大学出版社

重构极限编程（上）

Mark Collins-Cope 著，张江 译

讨论

Mark Collins-Cope 采访了 Doug Rosenberg (《User Case Driven Modeling with UML》的作者) 和 Matt Stephens (与 Doug 一起合作编著《XP Refactored》，此书于 2003 年夏天发行)，关于他们对于极限编程的反对意见。



总论

[Mark Collins-Cope] 你好，Matt ，Doug

[Doug Rosenberg 和 Matt Stephens] 你好，Mark

[Mark] Doug，多年来你一直坚持对 XP 的反对意见，因此我相信对于您正在编著《XP Refactored》这本书，很多人都不会太惊讶。那么作为今天访谈的开始，我想请您首先谈一下这本书给我们带来哪些信息，然后我们再对其中的话题深入探讨。

[Doug] 首先，这本书的全称（即“XP Refactored – The Case Against Extreme Programming”）会让你对于 Matt 和我的工作有个大概的了解。从根源来说，我最早是几年前在 OTUG（the Object Technology User Group），主要是与 Bob Martin 和 Ron Jeffries，开始关于 XP 的争论。在热烈讨论克莱斯勒 C3 项目的下马到底是公司所声称的成功还是在我们很多人看来的失败时，这场争论达到了白热化。我随后的一些幽默讽刺被广为接受，特别是我在 UML

-World 主题演讲并且在最近的 Rational 用户大会再次演说的《Alice in Use Case Land》，在 SD West 发言的《Fragile Methods》，《Emperor's New Code》以及我儿子 Rob 和我改编的很多好玩的披头士歌曲，它们最后被编成《Songs of the Extremos》。

大约一年前，我偶然读到 Matt 的文章“The Case Against Extreme Programming“。在这篇文章里，他经常以极为好笑的解说，一步步一层层的揭穿 XP 的很多夸大宣传以及拐弯抹角的逻辑。当时他谈到关于“从来没有过结对编程就没有资格说不喜欢它”，当我读到其中一行时，突然忍不住的笑出了眼泪，并且意识到我们必须得一起写一本书。你应该去读这篇文章了解我所说的那一行，它以“我从不把我的头放进”开头。我和 Matt 今年初在英格兰相遇时，我告诉他，他所写的那一行将会改变他的生活。他告诉我他当时犹豫了很久，考虑是否该把这句话从文中拿掉。所幸最终还是把它留下了。

所以我们的这本书是基于 Matt 的”Case Against”一文中的很多观点，用了整章篇幅将 C3 项目中的美妙的谎言与现实情况对比（引用邱吉尔的名言“从来没有过一次沮丧的失败，产生了如此多的谎言”），并且尽可能使之充满幽默。为了幽默，有时我们仅仅是简单的把 XP 大师们的自相矛盾的话放在一起（有时这些话看上去的确非常好笑，比如“日程安排是客户的问题”和“极限编程者并不担心口头的文档”），有时我们会巧妙运用讽刺和挖苦。

但是之后我们又展示了一些 XP 真正非常薄弱的环节，这里我选取其中当前正在讨论的一点，比如，把一些像需求管理和项目进度这类次要工作的责任，从开发组中剥离出来，委托给“客户”承担。这样超负荷的客户就变成了失败的独立因素，程序员们就可以专心重构并且每天下午 5 点按时回家。另一个我们都注意到的很大问题，就是他们认为：项目开发之前的规划，某种意义上说都是浪费时间；预先规划的不足可以用小迭代以及重构来弥补。以上只是我们将讨论的议题的一部分。

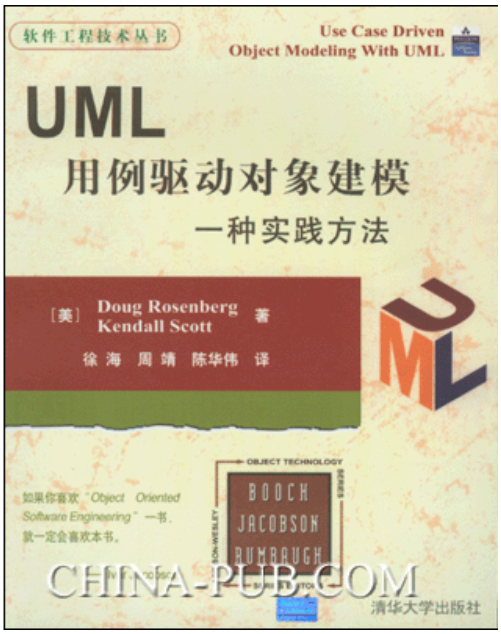
过程和符号

[Mark]在上一次的访谈中，我把下面这个问题给了 Robert Martin－现在我想知道您的回答…..“过程目前是软件开发社区中的热门话题。我们知道有 RUP， Iconix，XP 等等，它们之间看上去都有些矛盾的地方。那么什么是折中的软件开发者和管理者所该采取的方案？对于软件开发来说有没有绝对正确或者错误的方法？

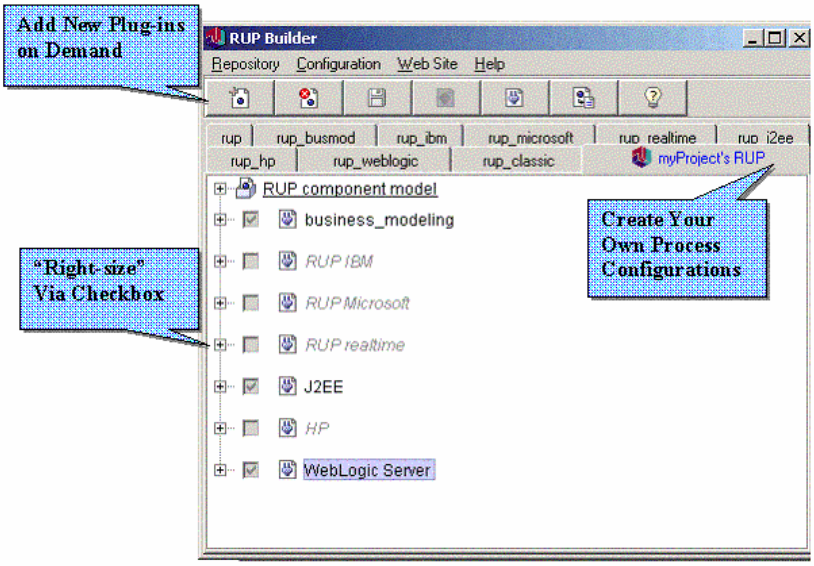
[Doug] 我们在 Rational 公司的朋友们会告诉你，RUP 的理论能够被裁剪定制，然后符合所有的需求，并且他们也正在努力使 RUP 更容易被裁剪。实际上，我们最近刚刚为 RUP 发布了一个“ICONIX 过程”的插件，它能够在十分钟之内，为你的 RUP 安装一个在我写的“Use Case Driven Object Modeling”提到的方法－ 我们叫它“QuickStart”.

而且，我相信 ObjectMentor 最近已经为 XP 开发了一个 RUP 插件。这种为 XP 制作 RUP 插件的概念非常有趣。我不太确定，但是我想这个插件很可能去掉了 RUP 中所有的分析和设计活动。我个人认为这并不会促使你所说的“折中的极限开发者”去购买 RUP，但是我猜想 XP 里 RUP 插件的存在，会让一些人心安理得的把 XP 作为一种“真正的过程”来接受。

[Mark] 因此 XP 现在是 RUP 的一个实例，ICONIX 也是 RUP 的一个实例…看上去 RUP 是无所不能的☺！这种放之四海皆准的过程有实际价值吗？



[Doug] 是的…整个世界都是 RUP 的一个实例，除了 RUP 本身之外。RUP 是一个过程框架，不是实例☺ 然而，严肃的说，我们开发插件是因为 ICONIX 过程是一个使用 Use cases 和 UML 的流水线路径。我们已经得到了很多使用 Rose 的客户，并且我们认为如果能够把“Use case Driven Object Modeling”中的过程“安装”进 RUP 的安装，这将是非常有益的。我认为一个每个成员都能通过网站访问的过程是很有用的，特别是对于较大型的项目来说。实际上，姑且允许我把我们的过程叫做 RUP 的实例－你可以很容易并且快速的把我们的过程安装进 RUP，并且应用插件技术，或多或少的改变了 RUP 的特性。这是一个非常好的概念，你为你的过程建立了一个 UML 模型（大多数都是不同阶段的活动图），在 RUP Builder 工具中运行，然后它产生了一个文件，你可以把这个文件放进 RUP 从而改变了你的 RUP 安装。我认为这个插件概念很有用，因为你能够建立你自己的过程，并且把 RUP 作为一个承载工具来发布它。是的，我认为这就是它的价值所在。



[Mark] 在过去的几年中，UML 标记符已经非常流行，现在有很多的开发商正在卖支持它的工具。在某种程度上，我认为 XP 是对于来自这些公司的日益增长的市场压力的一个回应－“你必须拥有一个工具来开发软件”。XP 是不是表明，你不需要昂贵的工具就可以开发软件？

[Doug] UML 不是必须依靠某种昂贵的工具。比如我们这些天用 Visio 教了很多班。在班上，我们在贴纸上用粗墨笔画了很多 UML 图，然后把它们贴满墙。有一个人负责把这些图全部画进 Visio 里。你必须先画出 UML 模型然后再编码，这是很重要的思路。没有人能让我相信，如果事先不规划更容易成功完成软件开发。

[Matt] 设计过程本身当然不需要任何软件工具，不管是贵的还是便宜的。但是当你要把设计表达出来就需要了，因为这是对设计的一种检阅－ 一种扩展的思维过程。你可能需要用 Word 和 Visio，或者 Rational Rose，或者根据你公司的预算选择其他任何工具。例如在我正在参与的一个项目里，我们大量的设计工作是用白板上潦草的线和方块来表示的。它们有些是 UML 图，有些只是一些胡乱的符号罢了，但无论它们是什么都符合当时的思维过程。因此无论纸和钢笔，还是白板和粗墨笔，都可以是设计过程的基本工具。但是在某些阶段，你需要用一种软件工具来把你的设计文档化，让别人能够阅读、理解并评论你要实现的东西。

实际上，在文档化设计的时候，我经常能够想起一些更好的设计，或者发现现有设计中的缺陷。在整个开发过程中应该有这么个东西，它能有效地帮助你在开始编码之前，做出良好并且简单的设计。不错，在编码的时候设计会得到不断的检验－但是提前的设计检验过程会减少很多后来的重构工作。我们能在编写恰当的设计文档的时候做这件事，这并不需要占用太多时间。有一个技巧是写刚好足够向前跟踪的文档。我们应该把大量时间花在设计上，而不是写文档。

[Mark] 但是“刚好足够”不正是 XP 所推崇的？

[Matt] 不巧的是，“刚好足够”是相当主观的标准。刚好足够什么？这一切取决于你自己的理解。做什么（或者不做什么）会增加软件风险，什么又会降低风险，什么是可以接受的软件风险级别？如果只是为了写文档而写文档，这就会增加风险，因为它需要其他的文档来对它进行维护，并且它可能对你根本就是无用的。当你编写你的设计的时候，你的心里应该有一套准则。比如，Agile Modeling（AM）以“有目的的建模”为原则－如果你处理得当，在文档化你的设计时所获绝对会大于投入。同样 AM 也建议你，为了让你的设计文档最大程度地满足读者，你应该了解他们。这里 XP 有一个错误－他们认为如果读者是一个程序员，那么源代码就是设计最清晰的表达形式。当然，源代码的确能最“真实”反映设计的文档，因为它“正是”要实现设计，但是它可能并不是最具表现力的，并且谁能说你的代码绝对正确呢？如果仅仅是代码的确做了点事情，又通过了一些单元测试，这并不能说明它正是吻合设计要求的。源代码过于复杂，不能清晰的表现我们的目的。我们需要比代码和单元测试简单些的东西，来定义我们的设计－－比如一份 Word 文档，或者一个类模型。

[Mark] 不是有一些很好的软件工程师不喜欢图形符号吗？对于他们来说 XP 不正是一种很好的方案吗？

[doug] 我相信存在这样一些人，他们能够写出很有效率的代码，却不喜欢图形符号。姑且让我称呼他们为“好的软件工程师”。我实际上也有一个工程学位（电气工程）。和我一样毕业于电气工程的人中，没有谁不能读懂并且画出电路图的。这些图对于在系统开发过程中交流想法，有着巨大的作用。软件中的 UML 图，就相当于电路中的电路图以及逻辑图。它们都是工作的一部分。

[Matt] 即便在一个 XP 项目中， UML 或者其他什么图形标识也都占有一席之地。区别仅在于你使用图形设计次数的多少罢了。正如已经谈到的，如果有谁不能画出设计却立志要成为一名程序员，我觉得这是一件不可思议的事情。当前业界甚至在朝着把建模工具作为实现代码的手段之一的方向发展。CASE 工具在生成源代码方面越来越强；TogetherJ 利用 UML 来实现 Java 类主框架已经有多年了。现在的区别是这些代码生成工具，比如 Sun 的 ACE 项目，声称要从图形的业务规格书中创建全部的运行代码。像 Javelin Software 的 JGenerator 这样的企业级工具，已经能够从文字标识中生成全部的应用程序。具有讽刺意义的是，这些需要事先大量收集需求、大量设计的工具，却正在使项目变得越来越敏捷而不是笨重。这是因为这些自动生成工具封装了整个架构。我不是可以用它们生成一个.NET 方案吗？接下来我又用它们改而生成一个 J2EE 方案…OK，可能我们还达不到这种级别上的灵活性，但是或多或少它是产业界发展的方向！ 如果一个人不能实现设计，他们就不应该是一名程序员－并且他们在今后多年里也不会有任何发展机会。

软件架构

[Mark] 您针对 XP 的批评中的其中一条，是它的渐进架构概念(emergent architecture) – 一种在项目中的各迭代阶段中逐步演化的架构。这对于软件架构不正是很符合实际的观点吗？

[Doug] 对我来说不是。可能这与我有一些航空业的客户有关。他们分别从事弹道导弹防御项目、航空局、哈勃望远镜、航空电子设备、直升飞机项目、军事飞行计划系统、指挥控制系统等等，我为他们提供培训和工具。我昨天刚刚和一个来自很大的喷气式战斗机项目的客户谈过，我当时建议他们采用渐进式的方法来生成电子控制系统的架构。我们最终对此都大笑了之。我认为如果在这类项目中人们能够先做好架构并在架构中进行设计，那么做业务系统或者电子商务系统的人也能够做到这点。

对于小型项目，我能够理解，在一系列的渐进阶段中对一部分的架构进行少量演化，或许的确是有效的策略。但是如果你阅读了“Wiki Web”，你会看到一段引述 Kent Beck 的建议 “系统越大，你越需要更多的渐进架构”。和其他很多这样的引述一样，它让我直摇头。

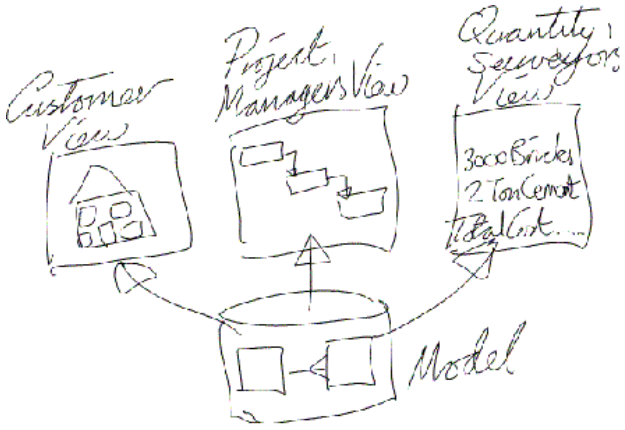
[Matt] 渐进式架构可能有效，但是像 Doug 所建议的，这是仅仅对于小型项目来说的。 或许你可以把一个项目的架构分解成一些小的子架构，然后对他们进行增量演化；但是这样这些部分之间就缺乏真正的结合。特别是当很多小组合作开发时，你最终实现一个字母排序的架构，各部分之间没有有联系的地方。 你可以通过增加特殊的进程和文档层，来避免这种情况的发生。但是这样你就失去了 XP 自身的优势。最好从一开始你就考虑全局的架构设计。

[Mark] XP 有一种思想，用系统隐喻(system metaphor) 来替代架构。相对于很多人采用的架构设计方法，“在电脑上画一些线和方块的漂亮的图”，这会不会是一种更好的办法？

[Doug] 就我理解来说，XP 的系统隐喻图很像我们所说的“领域模型”— 也就是说，在问题领域识别出一些最重要的名词（概念模型）。但是领域模型（简化的 UML 类图）比系统隐喻图更加精细。而且技术架构不单是从领域模型中演化而来，它更包含了一些与编程语言相关的细节，比如系统采用的界面开发包、底层数据库、通信协议等。

[Matt] 系统隐喻图对于非 XP 项目来说也是有用的。它在 XP 的书中并没有严格定义，所以你可能比如说把隐喻原则应用到领域建模中，提供一套一致的对象名称和描述。如果你遵循 ICONIX 过程，你就会这样做。所有的后续文档和代码都是在领域模型下完成。然后对于 XP 的隐喻而言，它自己还不足以完成项目中各部分的一致和

耦合。因此你必须要有一个合适的设计，并且把它写下来。



[Mark] 但是回过头来看很多我们的系统架构图，它们被用来描述架构（假设我们确实这样认为），却都是由很多缺乏语义的方块和线组成 。

[Doug] 噢，对于我自己来说，我非常高兴看到对于构架有用的用例图、领域模型、健壮图和其他任何补充图形。我不认为没有细致的语义的线和方块有什么问题。建筑设计师也没有明晰的语义，但我承认他们的价值。

设计

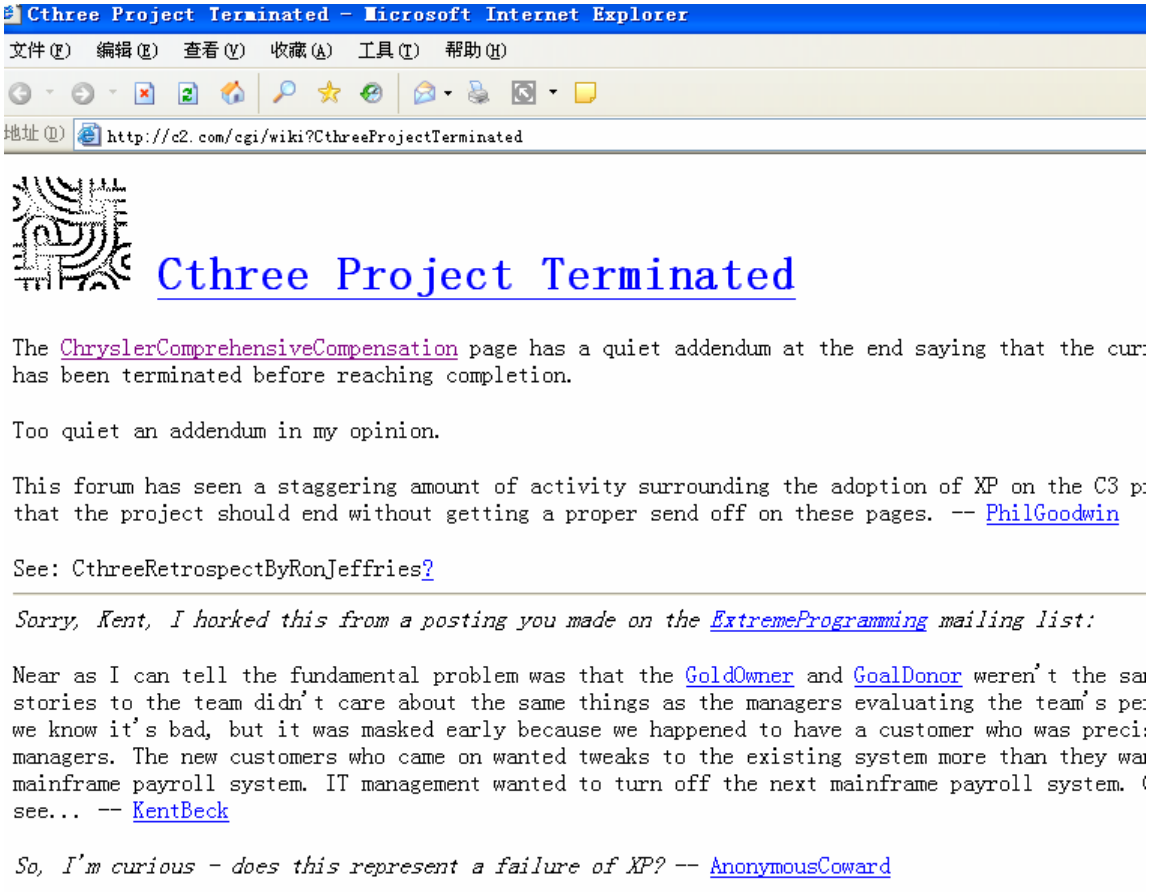
[Mark] “做可能有效的最简单思考”是 XP 的设计座右铭。 软件工程师们经常做过度的工程方案 － 会带来问题的方案－这条 XP 座右铭难道不是符合所有软件过程的吗？

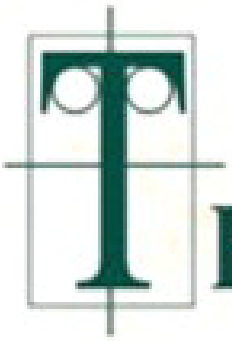
[Matt] 简单原则，但愿 EJB 专家们能够想到它。然而，有时候也有可能过度简单。对于 XP，你所需要做的就是一个原型逐步演化过程，你只需要简单的设计出软件现在需要做的事情。问题是，原型会导致不完整。就像用例一样，原型也是一个“皆大欢喜”的情况。那些很难判断的部分，比如发生异常怎么处理、用户操作错误或者进行不合理操作怎么处理，这些部分占到程序的 80%，却并没有在原型中体现。这些隐含的难点虽然用户不会看到，但是如果开发团队只是设法让系统成功运行然后向用户交货，从而遗漏它们的话，必定会因此遭受损失。

[doug] 嗯，这个观点有很多变种。KISS(Keep It Simple，Stupid) 就是其中之一。绝大多数情况下，“保持简单”当然是一个好主意。但是让我们来看看这句：“可能有效的最简单”。好好想想它意味了什么。这句话超出了通常“保持简单”的含义。这里有一个引用，来自一篇很老的论文“分布式计算”，关于 Chrysler C3 项目（这是提出 XP 的论文之一）。这篇论文中具体的解释了简单原则的目的：“我们不为将来未知的需求而在系统中做空泛的工作。我们做最简单的事情支持现有的特性”。

这是与仅仅“保持简单”非常不同的思想。它意味着“不要提前考虑其他需求，着重做好现在的事情”。我所遇见的大多数程序员都遵循着这个建议(我靠编码生活了 15 年)，结果证明是极其危险的。Extemos 想让你相信“编码后不断重构”可以解决一切问题。你可以用橡皮筋、泡泡糖、绑绳什么的先把系统构建起来再说，反正明天要对它进行重构的。呵呵，对我来说这不是一个好主意。

更加美好的是，正是这篇论文宣称，C3 这个原计划 2000 年 2 月将取代克莱斯勒原有工资支付系统主框架的千年虫项目，将提前在 1999 中旬开始正式投入运行，并能够支付克莱斯勒 86，000 名员工的工资。其实还没真正实现，就宣称成功。最后的事实是，2000 年 2 月 C3 项目被取消时，它仅仅能用于 10，000 名员工的工资支付（这篇文章写完后一年内其实就达到了这个效果）。我推荐每个人都读一读 Wiki 网上的 CthreeProjectTerminated（终止的 C3 项目）这一页。希望在这个访谈最后我们能为你们的读者提供一些链接地址。



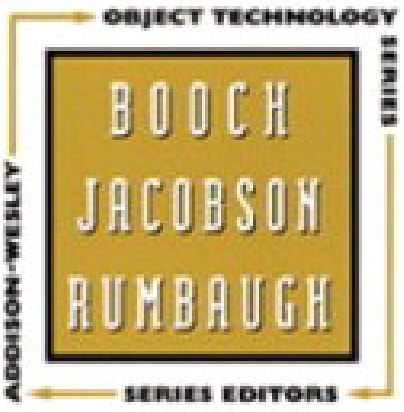


THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

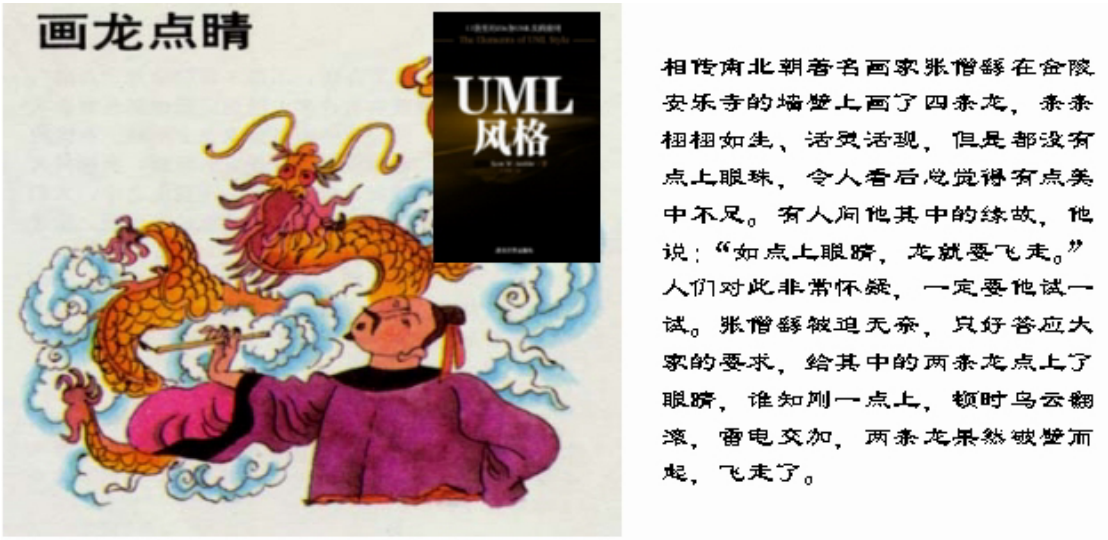
Covers UML 2.0

UMLChina 译
（王海鹏、汪颖）



CD-ROM Included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社出版



它不讲概念，它假设读者已经懂了概念。

它不讲工具，它假设读者已经了解某种工具。

它不讲过程，它假设读者已经了解某种开发过程。

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

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

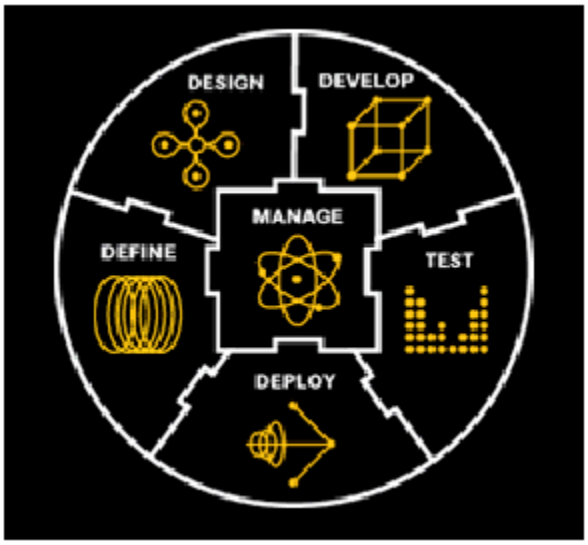
使用 Borland ALM 解决方案的统一过程

Tom Gullion 著，高天成 译

讨论

概述

本文对使用Borland ALM解决方案实现统一过程（包括IBM Rational统一过程即RUP）进行了综述，中心焦点是Borland软件产品如何为贯穿软件开发生命周期（SDLC, Software Development Lifecycle）的核心最佳实践和科目提供充分的支持。Borland软件产品加速了统一过程中的软件开发，从而可以真正更快地提交更好的软件。(*译者注：ALM即Application Lifecycle Management，应用生命周期管理。*)



读者

本文的内容主要面向对用Borland产品实现统一过程或RUP感兴趣的人，也适用于那些想总体了解这些产品如何支持统一过程的技术细节的人。本文的内容对那些使用不同过程的人也可以提供有益的信息，因为许多软件开发的最佳实践是有效的。

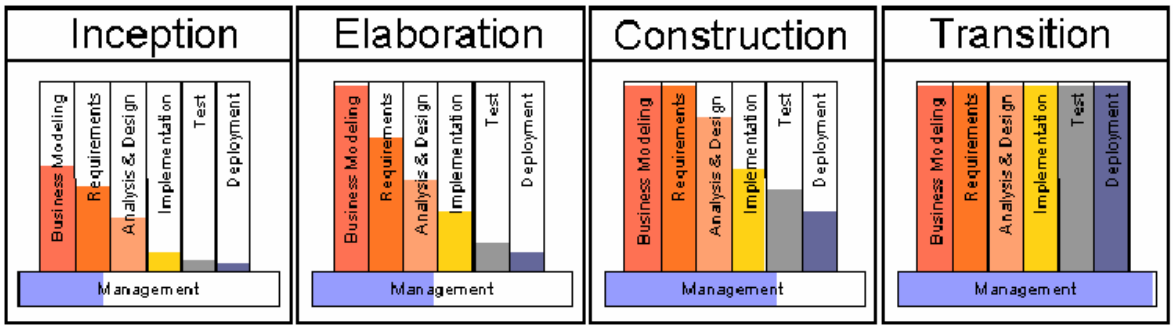
统一过程最佳实践

统一过程 (UP) 和它的一个广为人知的改进产品——IBM Rational 统一过程或称RUP，都包含一个最佳实践的集合。以下6个实践都来自于对成功开发团队的观察资料。[RUPBP]下面的部分将描述Borland ALM解决方案是如何在对这些最佳实践的支持中领先于同类产品的。

迭代的软件开发

迭代的软件开发是Borland ALM解决方案的核心。事实上，不支持这个基本最佳实践的现代软件开发工具是无法生存的。Martin Fowler曾经建议“你应该仅仅在你想获得成功的项目中使用迭代的开发。”Borland ALM解决方案完全采用这一原则。

使用统一过程关键的好处之一是每次迭代都是增量地建立在先前工作的基础上。这种方法的优点是解决方案的一致改进和演变作为问题域能够被更好地理解。选择软件工具支持你的开发时，对增量开发的支持是个关键。



上图描述了统一过程的四个阶段的详细演化级别[SPIRAL]。每个阶段都建立在前一个阶段工作的基础上。正如Richard Manard所说，“能够使团队在同一产品集上协作的工具可以提高团队的工作效率”。[RM]

Borland StarTeam是一个自动化的配置和变更管理系统。它的功能不仅仅是文件的版本控制。StarTeam给项目团队提供了一个集成的开发环境，在这个环境下可以处理需求、管理变更、追踪缺陷和线索讨论、并且可以通过管理中心任务实现对迭代软件开发环境的管理。

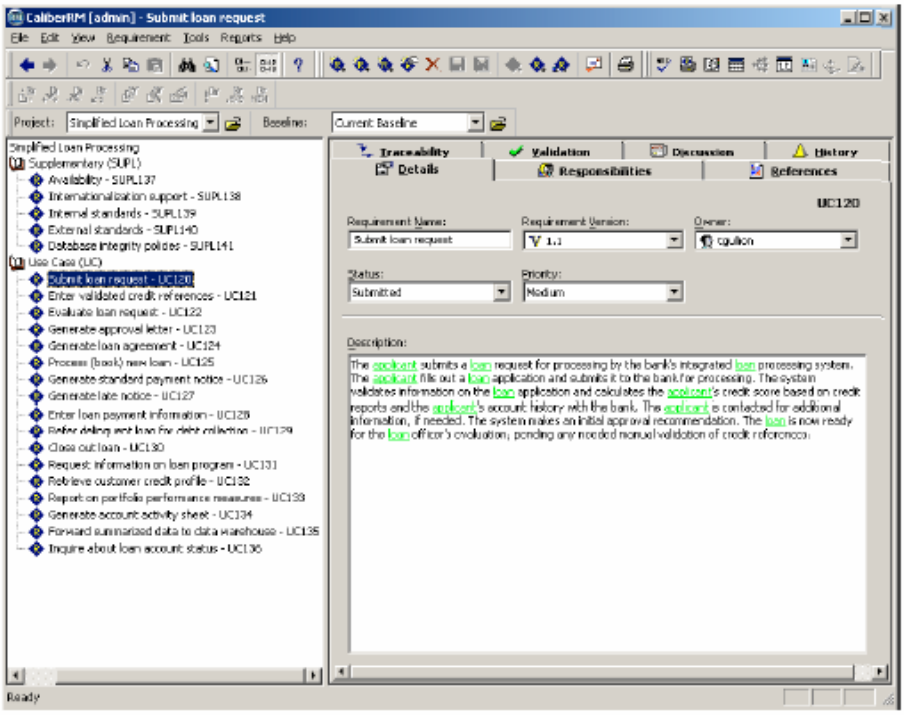
Borland Together的*LiveSource*技术 (如同步的双向工程) 从1992年以来一直走在迭代软件开发的最前沿。保持模型和代码的同步被证明对设计有着巨大的影响。同步模型和代码不会浪费你的精力，完全是自动进行的。

Borland ALM解决方案各个部件之间的集成实现了科目之间转变点的平稳过渡。这使得所有的项目成员都可以看到项目产品的实况，从而增加相互交流和工作的效率，减少潜在的错误（如使用了过时的设计文档）。

Borland ALM解决方案很好地建立在对迭代增量开发的基础上。这个平台提供了一个简单的结构，却又有着广泛的含义。项目成员可以在适当的环境中看到所有的项目产品，而且可以确定它们是最新的。这样，所有的开发成果就会保持稳定，因为它们都是建立在实际设计元素基础之上。

需求管理

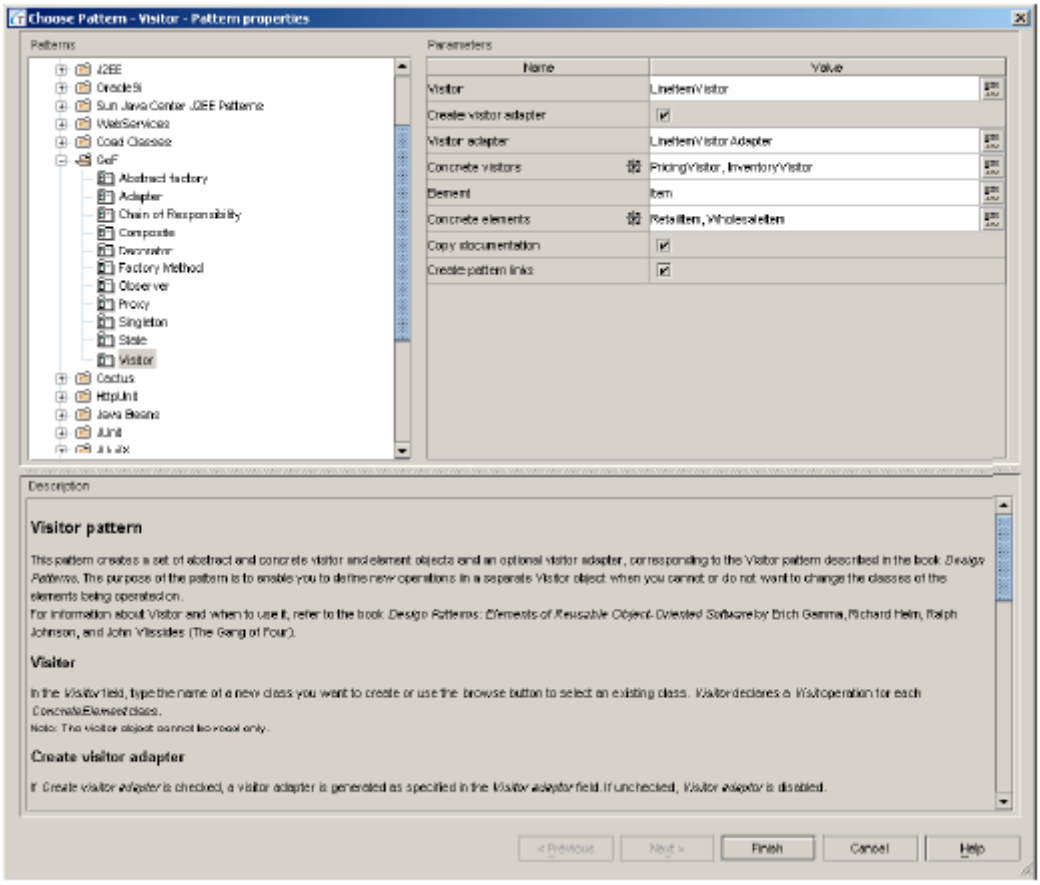
Caliber为管理需求提供了全方位的支持：一个与需求协同工作的的高效的用户界面；如果需要的话，需求属性可以定制；需求和其他项目产品之间可追溯的链接；版本历史（审计记录）；需求基线；完全特征报告能力以及丰富的SDK（Java或者COM接口）。



需求存储在Caliber的知识库中。这和其他开发商的方法有着细微的差别，其他开发商通常同时使用文档和数据库。Borland的这种体系结构将人们从维护文档和数据库之间同步的手工劳动中解放出来，从而使客户获得很大的好处。另外，需求基线的设置可以更早，管理只需要更少的精力，对贯穿整个软件开发生命周期需求的管理也简单化了。

使用基于构件的体系结构

Together为开发、使用和管理基于构件的体系结构提供了一个优秀的建模环境。例如，可扩展的设计模式支持使你可以使用现有的模式或者实现自己的模式以供日后重用。



Together和Jbuilder都提供了对实现和使用构件库的支持。你可以很简单地将构件加到项目中，享受CodeSense的全部好处，甚至将构件库文档集成到在线帮助系统中。

可视化软件建模

“在构造或者更新一个工业级的软件系统之前开发一个模型就像大型建筑的蓝图一样重要。好的模型对于项目团队之间的沟通以及确保建筑的稳固是很重要的。我们构造复杂系统的模型是因为我们无法完全领会任何这样的系统的全貌。”[UML1.3]

UML规范非常清楚地说明了可视化建模的好处。提供一种清晰的方法用于沟通、理解和维护演进的系统体系结构的完整性是极其重要的。

建模工具不应该仅有设计软件（常称为正向工程）的能力，还应当能够把现有代码中的设计表示出来（即广为人知的逆向工程）。Together长期以来都是这两个领域的业界领导者。它甚至创造了双向工程或*LiveSource*的范例。这种技术确保模型和代码之间始终是同步的。这对可视化建模真正有效地完成其交流的基本任务是基本适宜的。“不正确的内容是不可靠的，这种不可靠会妨碍内容的可信性，最终降低它的效率。”[SMD]

可视化建模的好处可以扩展到包含整个项目的模型，而不仅仅是UML模型。一个相当普遍的对用例驱动开发的错误使用是致力于很快地“分而治之”，即每个开发者拿到一个用例并完全实现它。这种例行公事的方法将会产生象意大利通心粉一样的代码，相同的功能用不同的方法在不同的地方实现。

可视化建模可以尽可能地避免这种类型的问题，特别是需求的可追溯性与模型内部超链接的结合。这种情况在一个完全集成的环境中是显而易见的，如在Borland ALM解决方案中，所有项目成员都可以直接看到需求（用例建模）、模型元素和源代码之间的连接。

验证软件质量

在RUP中，“管理质量贯穿于所有的科目、工作流、阶段和迭代中”。[RUP]当前软件开发业中正有一种提高质量的推动。测试驱动开发[TDD]特别强调测试，它建议在写代码之前先写单元测试用例。Borland ALM解决方案完全支持使用标准xUnit框架（如JUnit）的单元测试。

在需求科目内，Caliber提供了许多功能部件来管理质量。当需求被创建时，Caliber解析需求文本从而发现错误，甚至指出将来可能会认为是错误的不明确的术语。术语可以容易地加到工程的术语表中，从而保证业务领域的重要术语得到始终如一的使用。

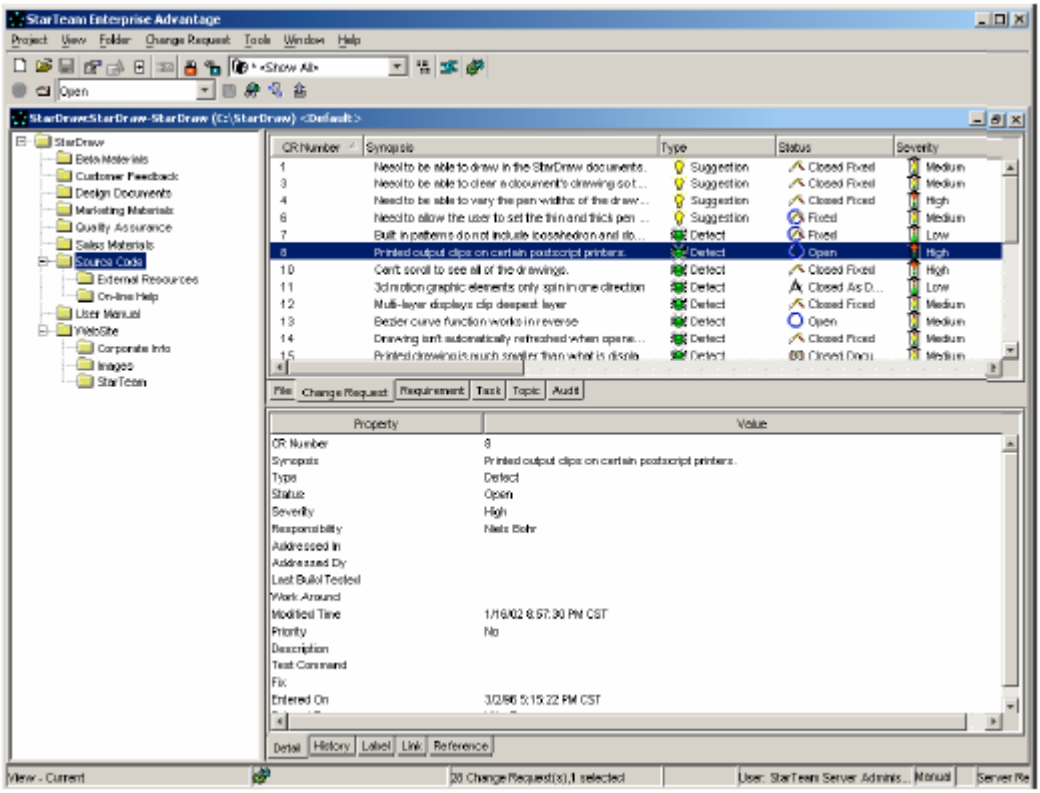
Caliber能够与Mercury Interactive TestDirector®集成。这使将需求直接链接到测试成为可能，可以确保应用程序正确地工作且满足需求规格。

Together提供了审计和度量的功能部件来保证代码级到模型级的质量。审计功能根据工业标准的最佳实践评定源代码，包括代码风格、临界错误、声明风格、文档、EJB细节、命名风格、性能、可能的错误、多余的内容以及人机界面构造细节。度量功能使模型的度量自动化，提供了关于类模型状态的有价值的细节。除了简单的静态数字报告外，度量还能生成图表以可视化的形式观看模型在迭代中的演化过程。度量包括基本要素、内聚性、复杂性、耦合性、封装、硬件抽象层替代、继承、基于继承的耦合、最大限度、多态、比率、用户界面以及审计违例。有对象建模经验的人会马上掌握Together的度量功能可以准确测度核心概念的好处，如内聚性、耦合性和封装。现在架构师可以确定他们的设计正在正确地演进。审计和度量框架是可扩展的，这样团队就可以部署自定义的审计和度量以适合他们的环境。

开发人员可以使用Borland的Optimizeit产品剖析他们的代码。这些强有力的工具捕获客户端和服务器端的性能瓶颈、线程问题、代码覆盖。Optimizeit完全集成在Borland的IDE中，可以最大地提高开发者的效率。

控制软件变更

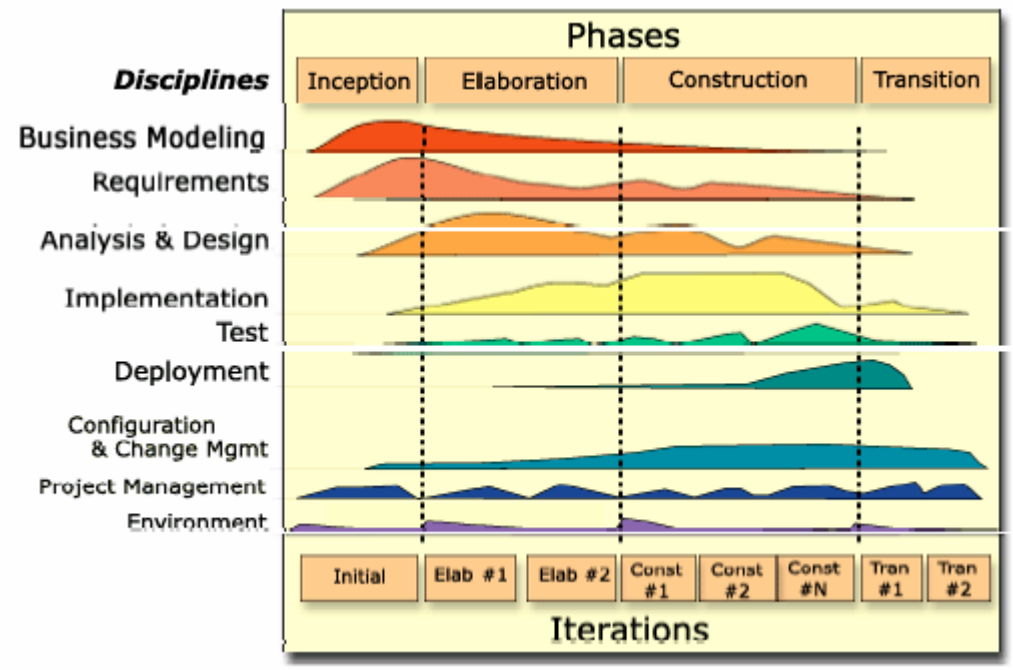
StarTeam位于Borland ALM解决方案的中心，目的是实现对变更的有效控制。它完全支持大型、小型组织包括广泛分布式的项目团队的变更和配置管理的需要。



变更请求在StarTeam中集中管理。将变更请求和实际项目产品一起存储在知识库中，可以提供完全的控制、可见性和权威的审计历史，也消除了手工同步外部需求文档和知识库变更的必要性。这在科目内部工作流的组织方面有着显而易见的好处。

阶段

统一过程将一个开发周期分为四个阶段：初始（Inception）、细化（Elaboration）、构建（Construction）、交付（Transition）。描述这种结构的最流行的图之一是RUP的“驼峰”图（如下所示）。[RUP]



这幅图展示了四个阶段（图的上方），每个阶段内的迭代（图的底部）并指出了划分每个阶段的里程碑（用点线表示）。这幅图也显示了每个阶段内努力的级别。当考虑软件产品支持统一过程开发项目的时候，非常清楚的是，协作至关紧要。

Borland ALM解决方案对协作的支持在本文中通过多种途径进行了描述。技术解决方案如跟产品关联的线索化的主题讨论提供了一个用于交流的强有力的框架，该框架允许在版本控制和集成的条件下在产品之间清晰地进行导航。

信息无论在形式上还是在位置上都是一致的，这就鼓励了协作。讨论的内容存储在项目的版本控制库中，成为项目的基础资源。

角色

今天软件开发的现实可以用“用更少的人办更多的事”很好地描述。在新型的着眼效率的团队内，项目组成员例行公事地被要求扮演多个角色。统一过程确定了许多角色，也指出个人应当经常设想自己在多个角色中的职责。

Borland ALM解决方案认可这种现实，并提供了广泛的支持。业务设计员和需求分析员使用Caliber创建用例，并建立用例之间可追溯的关联。系统分析员使用Together创建用例图，并将它们直接连接到Caliber中的用例。成果或者内容都没有重复。项目组成员可以彻底避免在应用程序和项目产品之间拷贝和粘贴文本而引起的种种问题。

这种通过集成在原有位置上浏览和编辑项目元素的能力消除了转换上下文的需要（当在应用程序之间切换时）。对脑力工作者而言，这种转换是破坏性的，可能导致长达15分钟生产力的降低。[PW]

科目

统一过程包括若干科目。这些科目原先被称为“工作流”，后来为了跟OMG的软件工艺元模型（Software Process Engineering Metamodel）相“协调”而改名为科目。[AUP] [SPEM]在本文中，我们使用Larman关于科目的定义，即“活动和相关产品的集合”。注意“工作流”仍然存在于UP的上下文中，不过现在是用来描述科目内部或者跨科目的活动序列。

业务建模

业务建模科目是为了发现或者澄清业务实际上是如何运作的。业务过程分析员将在Caliber 的术语表中捕获业务词汇。这保证了关键的业务术语能够共同、一致地定义和使用。业务建模元素（如业务工人、业务用例或者自动的需求）在Caliber中以文字形式输入，在Together中以UML图元的形式输入。这种分离可以方便地确定业务用例模型的结构。当然，所有的工作都会直接提供给后面的迭代过程和阶段。

需求

现实世界的项目必须管理贯穿整个生命周期的需求。如果需求存在于不同的位置（如知识库、文档和UML模型中），会变得更加难于管理。很久以前软件业认识到当试图分别维护文档和源代码时，这种分离的途径倾向于错误。解决方案当然是使用简单的办法如Javadoc[JD]将代码和文档集中起来。

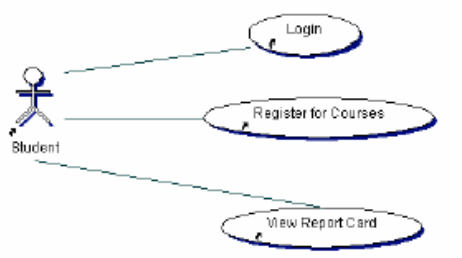
Borland ALM解决方案采用了在需求管理中创建全新范例的概念。需求现在不仅在Caliber界面中是可访问的，而且完全支持通过集成直接嵌入建模环境和IDE。所有项目成员都可以确信需求是正确的和最新的，因为他们是直接和“*live*”的需求协同工作的。毕竟，过时的需求文档“造成的危害比根本没有文档还要大” [PP]。

分析和设计

从需求集到一个完全可执行的系统的演进是这个科目的目标。Borland ALM解决方案比起其他任何方案来，

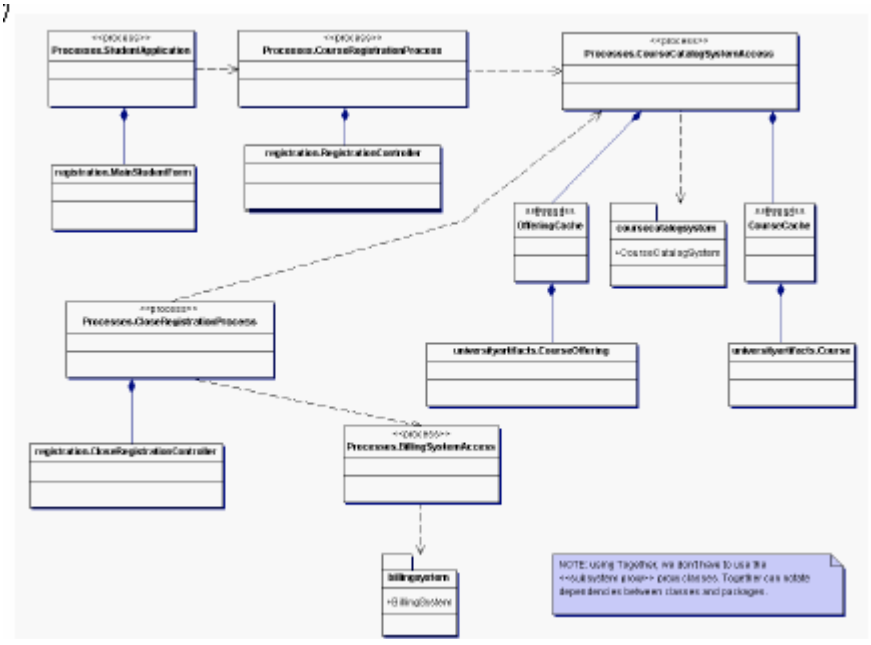
可以实现更多的内容。用例被映射成体系结构并在迭代中改进。Phillipe Krutchen论述了使用多种体系结构视图的重要性[4P1]。这产生了使用该方法设计体系结构的很长的一段历史。通过将设计模型组织到设计包，子系统和大量的体系结构视图都可以在Together中实现。不幸的是，由于篇幅所限，详细的解释已经超出了本文的范围。简言之，快捷方式和逻辑类图使这种方法成为可能。

快捷方式是一种在其它图中包含软件拷贝的机制，它提供了一种简单但有效的方法创建交替的体系结构视图。在下面的例子中，为清除起见，大的用例图中一个小的部分被分离出来。注意所有的元素都是作为快捷方式（用小的曲线箭头表示）包含进来的。快捷方式是指向原始元素的指示器，因而会始终反映原始元素的任何变化。



许多Together的用户除了物理包图以外从来没有用过别的东西。这些包图是为描述UML包（等同于Java包或者C++/C#的命名空间）的内容而自动创建的。用Together的说法，这些都被称为物理包图。

逻辑类图创建对象模型的虚拟视图，这些视图没有必要映射到任何的物理结构。它们主要用于说明性的目的。考虑下面引用的Together中的一个4+1模型的例子。这个“加工设计元素”视图包括多个类和两个包。在这些逻辑图中Together不仅包含了包的快捷方式，而且还能够自动标记依赖关系的连接。



然而，双向工程环境声望的上升导致现在的RUP版本建议“单个演进的设计模型与实现的一致性维护更加容易”。这将在下一节——实现科目中讨论。

实现

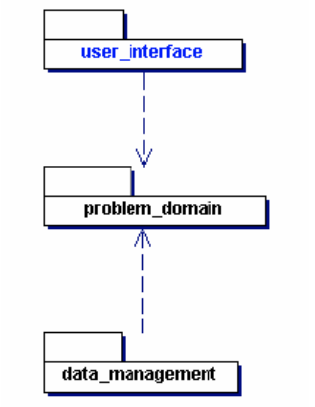
在实现方面，Borland长期领导IDE产品的历史占中心地位。其产品如Jbuilder、Delphi和新的C#Builder赢得了众多的赞誉，这些工具一直走在前沿并且将生产力水平提升到新的高度。作为Borland ALM 解决方案的组成部分，这些产品同样与StarTeam和Caliber实现了集成。这就保证了团队中的所有的生产力都被管理，而且会尽早满足指定的需求。

但是这并不是说我们可以把设计抛在一边，然后告诉编码者开始编码。设计必须被管理。Together的逆向工程功能部件特别适合这个任务。

正如在前面的科目中提到的，RUP指出“在双向工程环境下，设计模型演化到一个详细的水平，变成了代码的可视化表现。”确切地说，模型就是代码，代码就是模型。Together主要的好处之一是你可以保留选择的权力。不管你是否需要分层，是多视图方法还是“单个演进的设计模型”，Together都能够支持你的杰作。

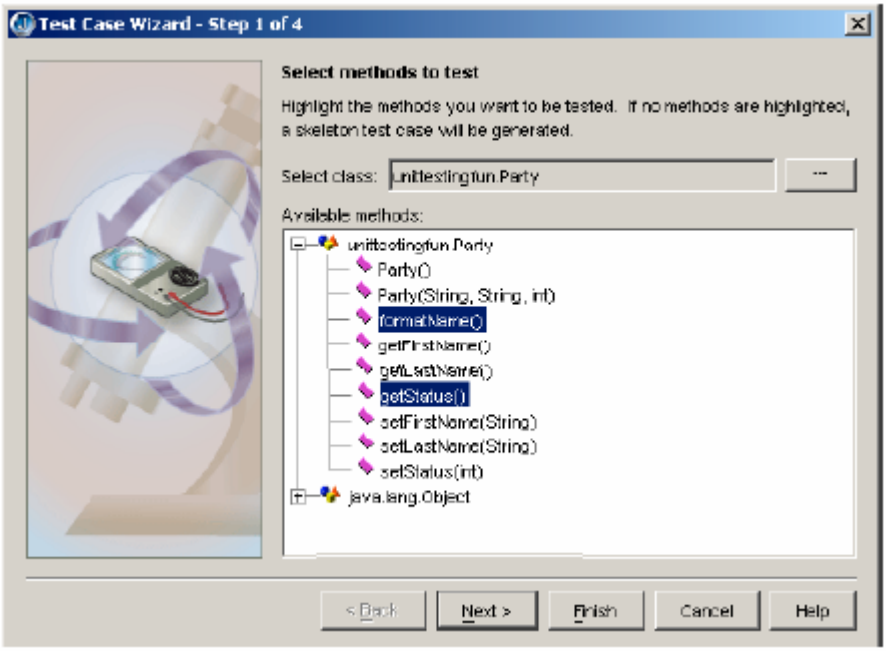
Together在支持演进设计方面胜过他人的途径之一是灵活、可配置的图的详细等级选项。类图，无论是物理的还是逻辑的，都可以被配置来展示过程阶段所需的所有细节。图可以设置到分析、设计或实现的详细等级。这种设置省略了可能会在下面的源代码中出现但是与当前显示的视图无关的细节。

自动依赖关系标记能够提供管理演进模型的自动化手段。建立一个如下所示的逻辑类视图以确保包依赖不被破坏是很简单的。主要优势是Together的分析技术可以正确描述模型的当前状态。在这个例子中，很清楚我们的体系结构是合理的：问题域层不依赖于用户界面或者持久层。



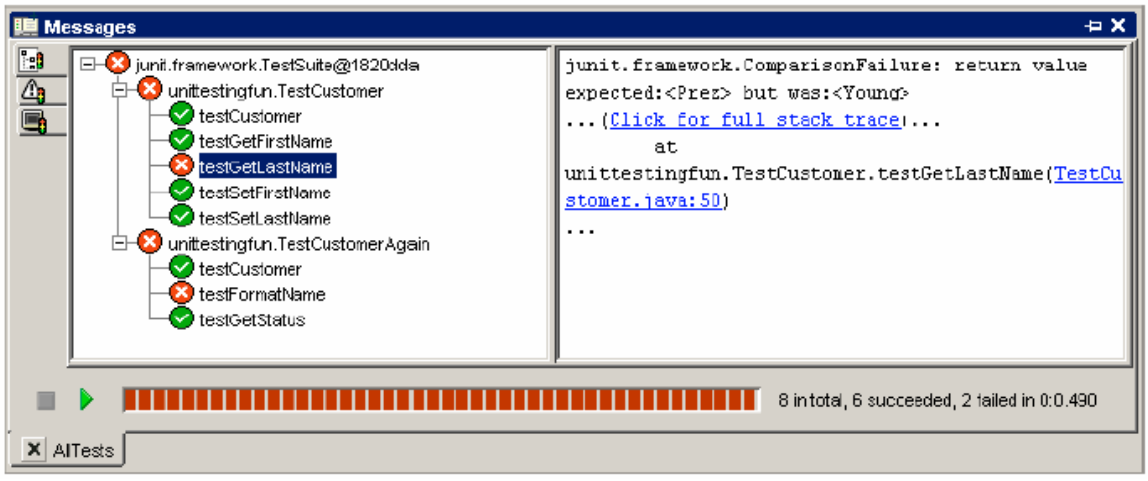
测试

单元测试长期起来为验证功能提供了一个的优秀的框架。Jbuilder提供了为已有类自动创建单元测试的向导。



最新的敏捷软件开发界的进步引发了新的单元测试实践。对那些采用测试驱动开发方法，在书写代码之前书写单元测试用例的人，Jbuilder也给你那样工作的灵活性。高级的代码审查和重置功能分析单元测试代码并提供向导来创建类、引进变量、添加方法和类。

Jbuilder同样包括对自动创建和执行测试集的支持。编译和运行测试往往只需要一个按键。



Caliber实现了与Mercury Interactive的TestDirector®的集成。这提供了又一级别的支持，可以在Caliber中的需求和TestDirector中的实际测试之间建立直接的连接。

部署

Borland在开放的、基于标准的产品解决方案上建立了它的声誉。Borland服务器产品如Borland Enterprise Server和Borland VisiBroker紧密追随J2EE和CORBA等规范。

无论你直接部署你的应用程序到应用服务器还是打包成可执行文件或档案包发布，Borland的工具提供了向导使这一过程自动化。团队能够变得更有生产力，因为他们只需要很少的时间准备用于部署的应用程序。

项目管理

项目管理包含多种技巧和实践。大部分的努力都花在管理人的行为和相互影响以及排除障碍为项目团队提供一个高效的工作环境上。

UP的商业版本，如RUP产品，为项目管理工作流提供了极好的概念上的指导。许多日常管理活动如监视项目状态、进度表和分配工作以及报告状况都可以使用Borland ALM解决方案的高级报告能力实现自动化。例如，StarTeam包括一个任务组件，允许将任务分配给团队成员去完成特定的工作并对实际成果进行跟踪评价。

配置和变更控制

StarTeam提供了一个强大、完整的配置和变更管理环境。它为分布式、远程团队开发提供了一个高性能的平台。多重视图能够限制仅在需要数据时才能访问项目内容。强大的分支功能支持基线和正在进行开发的项目。

变更请求在StarTeam中处理。手工管理变更请求的形式已经成为过去。StarTeam验证被请求的变更被处理并确保责任人被通知。

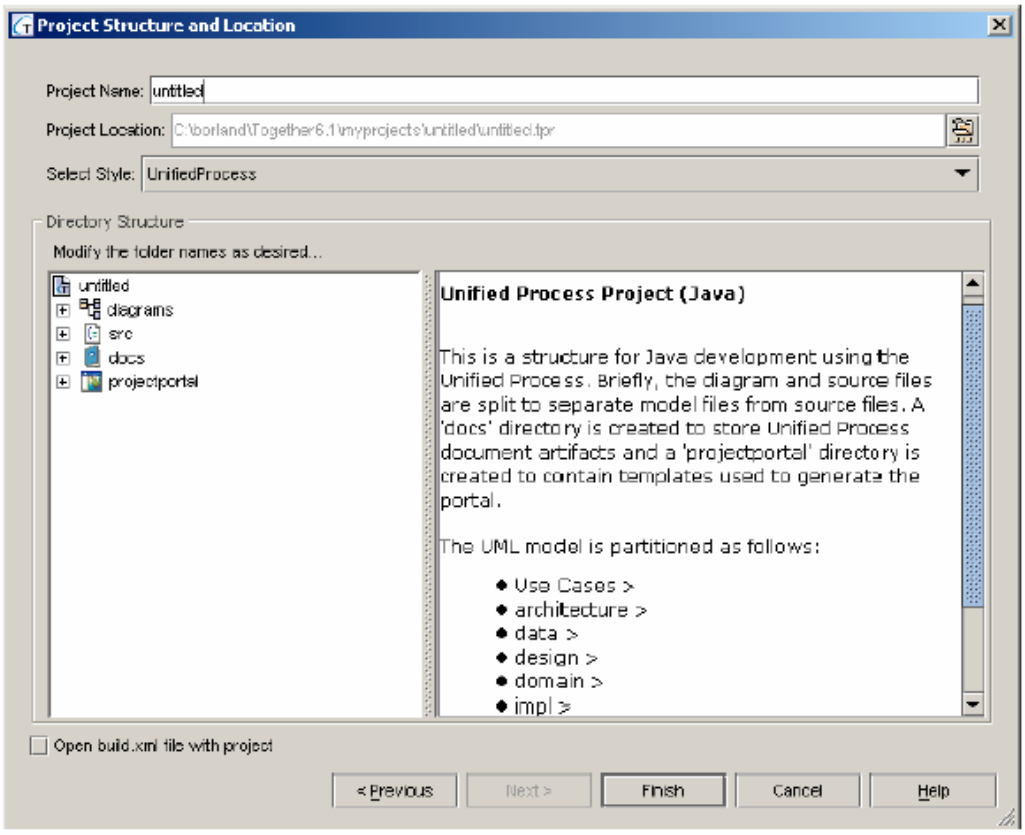
交流是变更控制之中的一个至关重要的部分。StarTeam的论题组件提供了问题的知识基础，团队成员可以通过集中讨论的论坛来访问。

环境

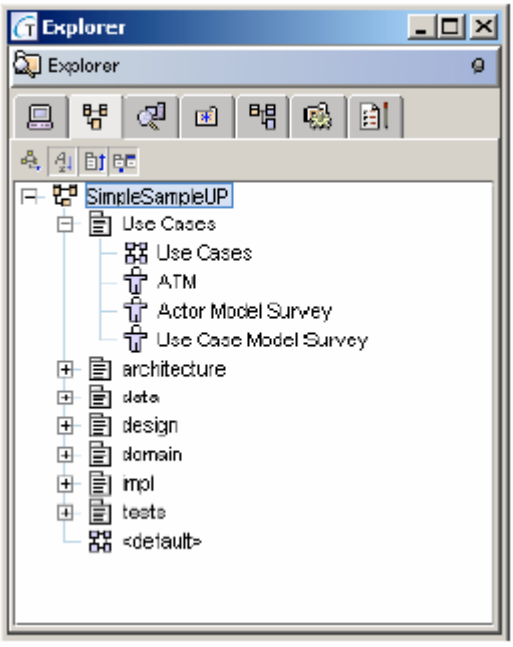
创建和维持支持开发团队的环境对任何项目的成功都是至关紧要的。开发团队需要他们项目的的共同、一致的视图。这使团队成员可以快速定位项目产品并分享知识。Borland ALM解决方案通过提供高度可用的用户界面和直接与实际项目产品协同工作的集成环境达到了这一要求。

该科目内更具挑战性活动之一是配置建模环境。对于如何组织和管理UML模型你可以有多种决策。

通过本地的Borland Professional Services组，统一过程项目向导提供了使用预先定义的模板自动进行项目配置的能力。在下面的屏幕快照中，选择了一个统一过程的模板。

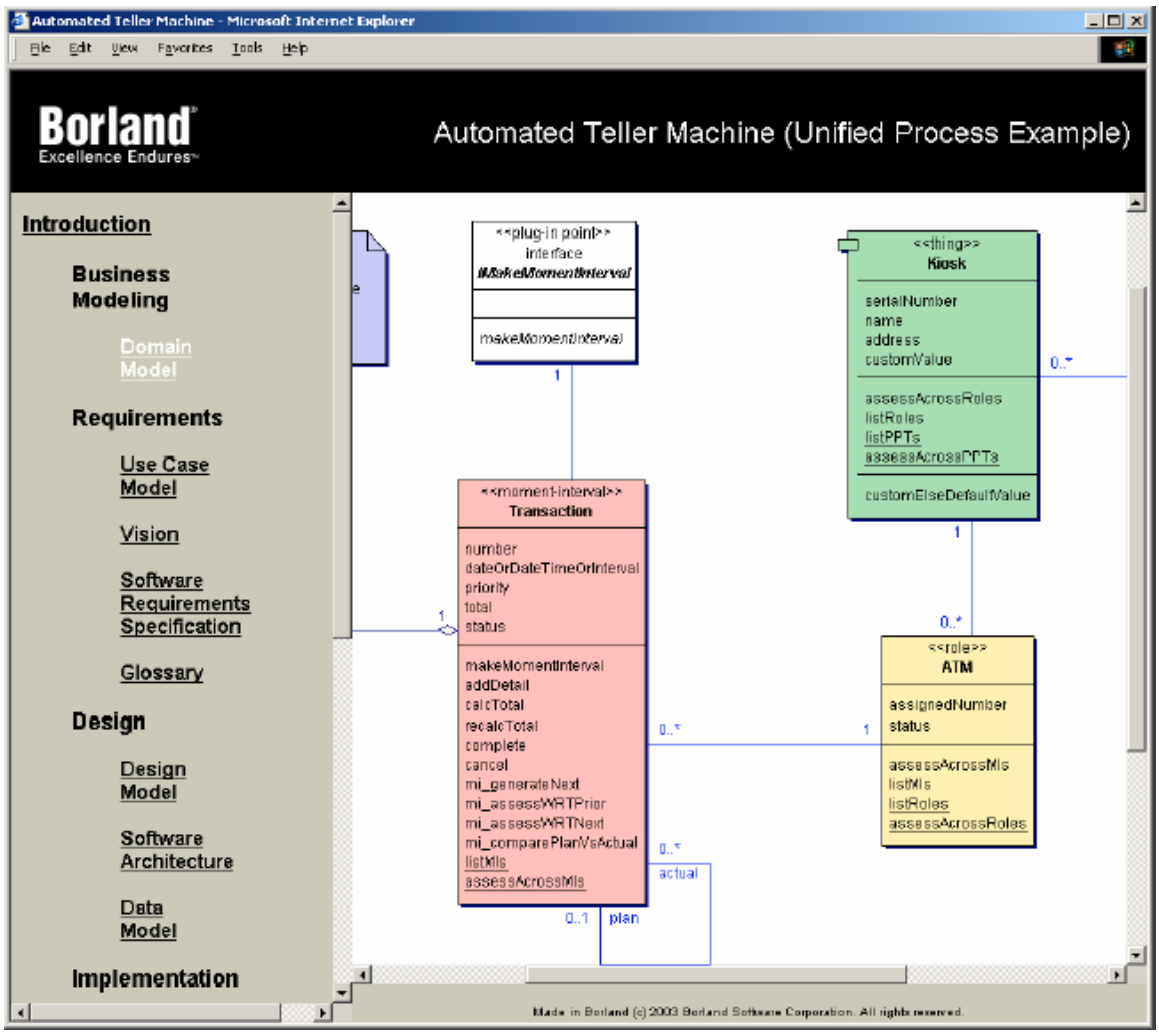


这个版本的统一过程模板按如下形式组织Together的模型。



正如UP推荐的那样，项目配置将模型分开，而且可以使你将整个项目作为一个整体来对待。创建需求跟踪连接、Together超链接甚至仅仅是在项目内部的浏览和搜索等诸多能力使其成为一个理想的项目结构。

这种项目方法的又一好处是可以提高生成文档的质量。Borland ALM解决方案中的文档自动生成功能部件考虑到了项目入口（Project Portal）的创建，这是组织项目信息的一种非常有效的方法。Craig Larman 建议，“项目文档应当只由项目站点在线记录的数字产品组成。” [AUP]项目入口接受了这一建议并且更进一步真正实现了文档生成和管理的自动化。如果一个项目在自动构造的过程中包含了“生成项目入口”这一步骤，就能从实质上消除过时文档的问题。



Borland Project Portal

总结

Borland ALM解决方案为整个软件开发生命周期提供了完全的支持。Borland已经努力为市场提供既能支持我们客户的过程主动又给他们选择自由的解决方案。

支持统一过程及其流行的改进产品RUP，本质上是充分地描述核心科目内的关键最佳实践和活动是如何被提供的。本文高度贴近这一主题并提供了简要的介绍。Borland的技术在应用于统一过程的内部结构时是突出的。

Borland ALM解决方案提供了在你的软件开发过程之内加速应用开发的一个完整的最佳品种的产品集。使用Borland解决方案的团队能够更好地交流、更快地开发跟更好的软件并且在预算之内交付它。（本文经Borland中国公司授权刊登）

参考文献

4P1	Krutchen, Phillipe. *The 4+1 View Model of Architecture*, IEEE Software, 12 (6), November 1995, IEE, pp. 42-50.
ASD	Martin, Robert. 2003. *Agile Software Development*: *Principles, Patterns and Practices*. Prentice Hall PTR, Upper Saddle River, NJ.
AUC	Armour, F., Miller G. 2000. *Advanced Use Case Modeling: Software Systems*. Reading, MA.: Addison-Wesley.
AUP	Larman, Craig. 2002. *Applying UML and Patterns. 2nd Edition.* Prentice Hall, Upper Saddle River, NJ.
JBR99	Jacobson, I., Booch, G., and Rumbaugh, J. 1999. *The Unified Software Development Process*. Reading, MA.: Addison-Wesley.
JD	Javadoc Tool Home Page http://java.sun.com/j2se/javadoc/

PP	Hunt, Andrew and David Thomas. *The Pragmatic Programmer*, Addison Wesley Longman, Inc. 2000
PW	De Marco and Lister, *Peopleware:Productive Projects and Teams 2nd Ed.* Dorset House Publishing
RM	Richard A. Manard, http://www.rambyte.com/succeeedrup.htm
RUP	IBM® Rational® Software. IBM® Rational® Unified Process® product.
RUPBP	*Rational Unified Process: Best Practices for Software Development Teams*. http://www.rational.com/media/whitepapers/rup_bestpractices.pdf
SMD	Ouchi, Miheko L. Software Maintenance Documentation, SIGDOC'85, Ithaca, New York, USA, ACM Press
SPEM	Object Management Group. *Software Process Engineering Metamodel*. www.omg.org/
SPIRAL	Royce, Walker. *The Rational Unified Process: A Commercially Available Spiral Model Implementation*. http://www.sei.cmu.edu/cbs/spiral2000/february2000/Royce/
TDD	Astels, Dave. 2003. *Test Driven Development: A Practical Guide*. Prentice Hall PTR, Upper Saddle River, NJ.
UML1.3	Object Management Group. *OMG Unified Modeling Language Specification, Version 1.3*. www.omg.org/

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

机械工业出版社即将出版

Scott Ambler

Charles Simonyi 的新方向

北京飞箭软件 译

讨论

软件越来越不能承受它自身的复杂性。什么是 Charles Simonyi 的解决方案？他要让编程工具变得那么简单，以至于非专业编程人员也能使用它。

--Claire Tristram

很少有软件专家能像 Charles Simonyi 对计算的发展产生如此革命性的影响，也很少有软件专家像他那样由于自己的努力而得到如此丰厚的回报。20 世纪 70 年代作为 Xerox 公司设在 Palo Alto 的研究中心的一名科学家，Simonyi 发明了第一个文字处理软件 Bravo，它在屏幕上准确显示出文件打印的效果，即通常所说的“所见即所得”（WYSIWYG）。后来，Simonyi 加入了微软。那还是微软创办初期，只有三十几个人。在微软，他是公司的主要设计者，领导了 Word 和 Excel 的研发。在这个过程中，他也成了亿万富翁，《福布斯》杂志最近列出的美国最富有的人中，他位居第 209。

去年，Simonyi 突然离开了他在微软 21 年的任职，开办了自己的意图软件公司（Intentional Software）。到目前为止，这个公司全部由他本人投资。从某种意义讲，意图软件是个不大的公司，目标也并不宏伟。Simonyi 不指望有一个产品卖它几年。但从另一种意义讲，他的目标极为宏伟，甚至“雄心”二字都不足以描述所需付出的努力。简单地说，Simonyi 要把软件从它自身的复杂性中解放出来，使得真正的计算机技术的潜能最终得以发挥。

Simonyi 深知他孜孜追求的是什么。他 17 岁时离开匈牙利到海外学习，然后就再也没回去。虽然他仍略带口音，但他不择词令所表达的看法毫不含糊，尤其说到他最感兴趣的话题。

他说，“我们一直在关注软件领域不断改进的过程，然而这种改进的效果并不明显，因为它永远不能解决人类自身不完善带来的问题。”

Simonyi 则另辟蹊径，在如何写软件、甚至在如何认识软件方面提出了一个革命性的改变的思路。他说：“人们通常所做的改进工作已使人们忘记了软件到底是怎么回事。”

为什么软件需要一个革命性的改变？因为今天软件技术已面临危机，它的复杂性已超出了我们驾御它的能力。当一个程序超过几百行，你就几乎不明白这个程序是怎么回事了。而如今台式电脑的软件包含几百万条代码,我们不理解就不可能改正程序的错误。因此 2000 年在美国有 25 %的商业软件项目被取消，这意味着仅此一项造成美国当年经济损失达 600 亿美元。

尽管软件已难以承受它自身的复杂性，我们却还没开始去探索如何解决。Simonyi 认为，挑战的关键在于要找到一种编程方法使编程人员和用户都能读懂。Simonyi 的解决方案是什么呢？要发明一种既简单、功能又很强的编程工具，使得软件好像由软件自身写成，就像 Excel 软件那样自动添加数字栏目，或象 Word 软件自动把文件格式化。

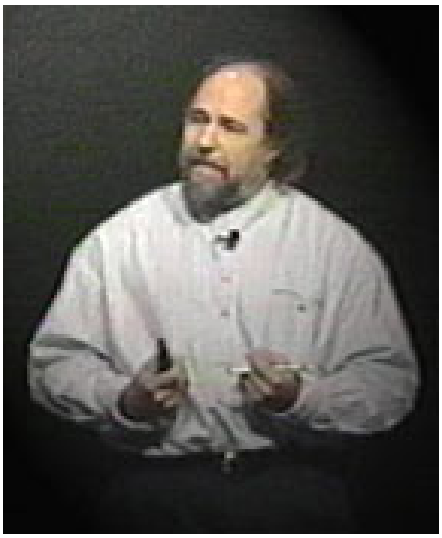
Simonyi 强调说：“软件要像 PowerPoint 演示那样容易编辑”。也就是说，要给它一个同样直观的界面。

Simonyi 企图解决软件开发的一个最根本的问题，即“让代码看起来象设计”。如果他获得成功，用户就能做出高水平的设计并让程序实现。这种设计更像流程图，而不是一行行的代码。程序代码是从设计自动生成的。

作为第一个“所见即所得”界面的发明者，Simonyi 似乎是唯一能迎接挑战的人。但这的确是很难的。有人说这是不可能实现的：从编写软件以来人们就一直在试图创建代表性的模型来自动生成那些复杂的代码，迄今为止他们只取得非常初步的成功。

也有人说我们没有其它选择。确实，IBM，Sun Microsystems 以及另外一些企业和科研机构也在做着类似的工作，使得编程简单化和自动化。

Grady Booch 是 Simonyi 的朋友，UML 的发明者。与 Simonyi 提出的方案类似，这个软件语言能让编程人员只需考虑软件做什么，而不用管真正代码的细节，不用管它是采用 Java，C++ 还是其它今天使用的语言。去年 12 月，IBM 花了 21 亿美元收购了 Rational 公司，就在这个公司 Booch 和他的同事们开发了 UML。Booch 说：“我们正在用软件解决从来没有过的更为复杂的问题。”



Booch 引用纳米微处理器这样的新技术为例（这种新技术将使计算机更为强大，只要我们能找到有效的方法为这种计算机编写软件）指出，“如果看一下我们将会走到哪里，那是永无止境的”。换句话说，软件必须赶上硬件。而今天我们编写软件的方式是做不到的。

让意图重见天日

我们如何使代码看起来像设计？Simonyi 主张，首先要了解目前编程中的问题是什么。

他说“目前，编程正好和开采钻石过程相反，在开采钻石时你挖出大量的泥土,而找出一点点昂贵的钻石。可编程时，你从一个有价值的真正意图开始，随后却把这种意图埋在泥土中”。

依 Simonyi 的观点，软件开发者是注定要失败的,因为他们同时要做三件事情——而仅有其中一件事适合他们。首先他们要理解用户复杂的需求，他们可能来自保险专家,会计师，或飞机设计师,软件是为他们做的。编程人员必须尽可能吸收用户多年积累掌握的专业知识。做到这一点即使不是完全不可能，对他们的专长来说也是不合适的。

其次，编程人员要把客户的需求翻译成计算机系统能理解的算法和界面。Simonyi 认为这是编程人员的核心任务。但目前这项任务完成得很差：软件完成后，用户无法修改，甚至无法理解软件如何反映了他们的意见。

第三，为使计算机正确执行命令，编程人员必须写出与机器一样精确的完美无缺的代码。尽管软件公司都在宣称他们的软件开发过程优于他们的竞争对手，事实上没有错误的编程是不可能的，因为人不是机器。最近(美国)卡内基·梅隆大学的一项研究发现，编程人员编写的程序平均每 1000 行代码就有 100 至 150 个错误。

Simonyi 希望为编程人员解脱上述第三项工作的全部及第一、二项工作的大部负担。他设想的不只使像工蜂那样的编程工作实现自动化，而且要让编程界面如此直观，以至于保险专家、会计师，或飞机设计师都能看到他们的作用，并能借助他们自己的专业知识改进程序，而无需通过程序编写人员的介入。一旦编程人员从那些压在他们身上的不适当的任务中解放出来，他们就可以集中精力完成他们应做的事情，即程序设计。

“真正的问题是，我们要用这个软件去做什么？”Simonyi 说，“这正是意图编程使我们有可能集中精力去做的事情。当你要创建一个非常好的卫生保健系统时，你就应该专注于卫生保健问题，考虑如何解决这些问题。但按我们现在编写程序的办法，却顾不上去理解问题本身，因为编程人员更关心如何把数字分类，如何把数据存盘。”

用图编写程序。

如果你让 Simonyi 解释怎样使像工蜂那样的编程工作可以自动完成，从而消除由于人为差错造成的程序错误，他会给你举出喷气发动机的例子。他说，拿涡轮叶片来说，它们必须做得非常精确。即使由很细心的熟练工人加工叶片，仍然不可能达到你要求的精度，而必须另造一台机器来加工叶片。其中会有人干预这个过程吗？当然，制造、维修和调整机器必须由人来完成。机器也会出错，机器一旦出错会很显著，你能马上发现，并改正它。程序编码也是如此。不需要人去接触编码。否则程序易于出错。人能制造这种机器。

机器来编写软件会是什么样呢？这种机器本身也是软件。但它的功能不是解决最终问题—执行某种新的家庭或办公室任务。它会是一个软件‘生成器’，由它来编写特定的一段软件。告诉生成器写什么程序，通过一个容易理解的界面，有时叫做‘模型语言’来完成。

目前最广泛采用的模型语言是 UML。它是由 Booch 和他的同事 James Rumbaugh 和 Ivar Jacobsen 在 Rational 公司做的工作开发出来的。目前，开放源软件群体正在对它开发，其中包括 IBM 和称为对象管理组织（Object Management Group）中的工业联合体的其他成员单位。UML 是一种建造图表的系统。它的目的是要使大型软件项目的管理人员能看到他们的设计，在程序员坐下来用程序语言 Java 或 C++ 编写程序前确保设计能满足用户的要求。

Simonyi 的观点，也是启动他的公司的动机，在于提出模型的概念，并进一步把模型和编程紧紧连在一起，最终合而为一。程序员和用户可以从许多不同的角度察看他们所创造的模型，只需修改模型就可随意修改程序。这有点像建筑师只要画出蓝图，就象魔术师那样产生蓝图中的结构，如果修改蓝图也能自动重建结构。

与 Simonyi 共同创办意图软件公司的 Gregor Kiczales 说，它其实就是原来的‘所见即所得’的概念。去年在这之前他离开了在 Polo Aito 研究中心的兼职位置，但后来他又回到在温哥华的 University of British Columbia 当他的终生计算机科学教授。Kiczales 赞成被称为面向方面编程（aspect-oriented programming）的方法，它能让程序员即使在很复杂的程序中，也能迅速编辑所有场合的相关命令，就像文字处理程序能找到和替代每一个拼错的单词。这是一种支持意图编程的基础技术。Kiczales 说，意图编程的思想，已用于文字处理，是否对其他程序也这么做？目前他仍是意图软件公司的顾问。

Simonyi 描述生成程序的新模型时说，它看起来很像 PowerPoint 的调色板，任何人都可用它制作幻灯片，把文本、图表或图像粘贴到一个直观的但又是虚拟的工作空间的不同位置。但是意图软件公司深信不同形式的问题可能采用不同的界面以满足不同用户的需要。例如，从事计算机模拟的科学家可能要利用程序生成器让机器把数学表述转换为相应的代码。

Simonyi 所想象的那种软件不仅不再要求未来的编程员像机器那样干活，而且可使特定领域的专家——保险，会计，卫生保健——十分方便地修改他们的软件，无需不断地去麻烦编程员。如果你是一个公司会计，使用常用的财务软件，当一个新的税法出来，Simonyi 断定，你自己就可以编辑你想做的事情的说明，然后再运行生成器，你并不需要打搅编程员。生成器以计算机的速度运行，比人要快约十亿倍，而精确性提高十亿倍。

软件的再生

即使一直很乐观的 Simonyi 表示，从这些工具的开发，到他的公司把产品投入市场至少需要两年。这些工具给编程所带来的简易并非唯一的好处，还使编程员创建的软件所能达到的复杂程度用现在的方法是无法达到的。

关于如何采用模型来拯救软件增加的复杂性，Simonyi 的同事和竞争对手有非常不同的想法。例如 Booch 希望通过建立覆盖面更宽的 Kiczales 的“aspects”的版本，充实 UML，使得例如象安全和认证的功能自动覆盖到整个软件系统。他还致力于‘找到方法来表达越来越强的抽象能力’——或者说，超越单个程序的模型，达到巨型软件项目的总体结构模型。

Booch 说，编写软件本质上是一个非常复杂的问题，而且越来越糟糕。我们刻板地书写应用软件，现在它们的代码已有几百万行。为此，你必须向简单化方面思考，模型能帮助我们。

James Gosling 是 Sun 实验室的一名研究人员，Java 语言的发明人。他是另一个近期支持这种模型的人。但是他研究的重点不同。认识到软件业常常再用老的程序，他建议找到一种方法把现有的程序插入一种图解程序的模型工具， 而不是要求编程员盯着数以几百万行的文本。理论上我们能对所有老的没有错误的程序提供这样的工具，考察它们所表达的设计，识别它们逻辑上的缺陷。

Gosling 说，像大多数编程员一样，我喜欢从头做起。但人们几乎从来没有那种奢望。我们正在创建一种工具，它能读很大的系统，例如 2-3 百万行，并能使它易于理解。

以代码命名的项目 Jackpot 还很小，不在 Sun 实验室的项目开发计划之内。但它是 Booch 和 Simonyi 所热衷的，因为它是模型正在成为推动产业的一股力量的又一个标志。

什么时候我们能从模型单独生成复杂的代码？虽然意图软件公司还没有投入市场的产品，它计划在 2004 年具体发布。IBM 购入 Rational 公司，承诺 UML 的研发，以及 Sun 实验室对此概念的兴趣，这些都表明这些思想正在达到关键阶段。

这些预言家所面临的挑战是巨大的。但是他们的成功也是必然的。因为他们所建议的事情的意义，远远超出那些相信我们能从几百万手工制作的程序段（任何一个程序段的一个小缺陷都会毁掉整个程序）做成一件完美的东西。这正是我们企图对今天的软件要做的。

Simonyi 说，软件是那样灵活，期望是那么宏大，和我们明天将要实现的目标相比，我们今天已习惯的普通改善是微乎其微的。瞧，硬件研发人员努力按摩尔定律所做到的，现在该轮到软件了。

本文原载于《MIT Technology Review》2003 年 11 月 3 日。由吴邦贤、林继玲翻译。

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于：
领域模型深度不足，
没有抓住业务本质！

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

软件工程技术丛书

设计系列

企业应用架构模式

Patterns of Enterprise Application Architecture

(美) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校

UMLChina 训练辅助教材

XP 和 FDD 的比较

Serguei Khramtchenko 著，李晨光 译

讨论

摘要

本文比较了两种软件开发方法：极限编程（XP）和特性驱动开发（FDD）。主要讨论他们在学术环境和受控环境中的实用性。本文比较了在项目管理中使用极限编程和特性驱动编程的不同方面：从收集用户需求直到实现整个程序。

本文要求读者至少熟悉两种方法中的一种。因为这不是一篇描述任何一种方法的文章，而是为了指明他们之间的异同点。

绪论

本章对极限编程和特性驱动开发进行了一个简单的描述。对这两种方法都很熟悉的读者可以跳过 1.1 和 1.2 小节，直接阅读 1.3 小节，那里描述了接下来几章将要讨论的作者的观点。

在过去的 10 年中出现了几种“敏捷”软件开发方法。这些方法将自己定位为在传统的“瀑布”开发模型之外的可选模型。在瀑布模型中，整个软件工程被划分为几个阶段，典型的阶段划分为：

－需求分析

－设计

－开发

－测试

－发布

关于**阶段/活动**的一个更完整的列表可以在 CSCIE-275 演讲材料中找到。瀑布模型假设每个阶段必须在下一个阶段开始前 100%地完成。这种方法的一个主要缺点就是设计中的错误往往直到发布的时候才被发现，这时项目已经基本完成，挽救错误的成本通常很昂贵。

特性驱动开发和极限编程是两种敏捷软件开发方法。敏捷开发通过迭代开发尽量避免“瀑布”模型的主要缺点。每一次迭代周期都很短（1 到 3 个星期）并包含以上的每个阶段。这样就能保证在开发过程中，设计错误会被尽早发现。

极限编程

极限编程是一种尝试简化和改进软件开发的方法。1996年，Kent Back在Daimler-Chrysler开始了一个项目，这个项目以他的软件开发思想为基础。他们当中的一些人激进地改变了开发方法。后来他把他的思想在一本名为《极限编程解析：拥抱变化》的书中发表。因为极限编程强调用户满意度，所以大部分的软件项目可以被描述为对一系列用户需求的详细实现。

极限编程从收集用户的“story”开始。每个story由用户撰写的一段非技术性文字组成。撰写这些story的目的是为了评估系统某个部分的复杂度和实现所需要的时间，而不是为了描述一个场景的所有细节。在story开始实现之前，所有的细节问题都应该及时和用户澄清。

下一个阶段是发布计划。发布计划描述了在系统的某个发布版本中应该实现哪些用户的story。每个发布版本由许多次迭代组成，每次迭代实现一部分在该版本中计划实现的用户story。每次迭代包括：

规划：选择要实现的story，澄清细节

编码：实现用户story

测试：每个类至少有一个单元测试

验收测试：如果成功，新功能就算完成了。否则就要到重新开始这次迭代。

上边描述的步骤很简单，没有表达出多少 XP 精神。XP 通过严格的项目计划来实现它的主要思想。可以在极限编程的主页【2】上找到这些思想。

简单：提出能够运作的最简单的解决方法。在很多项目中，由于开发者花费大量时间设计为了适应将来用户需求和应用平台改变的功能而导致项目偏离焦点，而这些功能可能从来不会被需要。

沟通：包括和开发团队的沟通和与客户的沟通。沟通是成功的关键。当用户需求晦涩难解、不够完全或者由于开发者的误解，经常会发生开发者实现的功能和用户需要的完全不一样的情况。

测试：自动化测试是极限编程的基础。极限编程的一个实践常常是这样的：任何一个开发者可以在需要时修改任何代码，如果引入了任何 BUG，单元测试都会马上捕捉到。

特性驱动开发

特性驱动开发是由Jeff De Luca和Peter Coad发明的敏捷软件开发方法。这个方法到1997年才有了名字。FDD的追随者在FDD的主页【1】上讨论该方法和过程。FDD声称在软件工程领域获得了多次成功。他比XP有更多的正式的需求和步骤，同时增加了对过程的精确跟踪。

FDD开发主要有以下两个主要阶段：

- 挖掘要实现的特性列表
- 一个特性一个特性的实现

挖掘特性列表是一个关键过程。这一步的质量在很大程度上决定了项目跟踪的精确度以及代码的可维护性和可扩展性的程度。这个过程需要客户的全程参与。这一步的输出是问题域的 UML 图，如果使用了双向开发工具，可以使用目标语言的可编译代码代替 UML 图。

特性列表来自 UML 图，特性使用一种开发者和客户都能明白的语言描述。例如：想象一个典型的购物车的例子，购物者登录一个在线商店购买货物。UML 图会包含一些类如 ShoppingCart，Item，Shopper，产生的特性列表会包括：

- 为 Shopper 创建新的 ShoppingCart
- 添加新的 Item 到 ShoppingCart
- 列出 ShoppingCart 中的所有 Item

- 计算 ShoppingCart 中的 Item 的总价格

这个特性列表为客户（资方）提供了一个量化的概念，因为它直接反映了要在软件系统中实现的功能。列表对于开发者来说包含了很多小的工作任务，因为每个特性都很小，它的开发都可以在一个很短的迭代过程中完成。把相关联的特性分组成工作包来开始实现工作。每个工作包在一个迭代周期内完成，大约 1 到 3 个星期。一个完成的工作包提供一个能部分工作的软件。每个迭代包括：

- 为工作包召开会议：工作包中包含的特性的细节将被澄清。
- 设计：必须的类/方法/文档将被建立
- 设计评审：接受或者否决提出的设计
- 开发：实现特性并创建单元测试用例
- 代码评审会议：进行相互代码评审
- 发布会议：实现的特性被发布到创建过程中

观点描述

这一节讲述了剩下章节将要讨论的作者的观点。FDD 和 XP 都是轻量级软件开发方法，这些方法认为这些年来在开发过程中存在管理过度的情况，有太多的规则需要遵守。为过程添加新的步骤不但不会改善过程，反而使它更难遵守。FDD 和 XP 都是从传统瀑布模型需要的步骤中选择了一个子集。然而这些子集是完全不同的，他们决定了每种方法的使用范围。

- XP 看上去更适合不稳定的项目，用户需求晦涩难动或变更频繁。XP 能够更好地处理这种项目，因为它有意识地不去做在当前实现周期中不是迫切需要的任何事情。
- XP 看上去不如 FDD 更有可扩充性。XP 过分依赖于团队内的沟通，这在团队变大时将会变得很困难。
- FDD 在项目的需求比较稳定时，能够提供更好的可预测性。
- FDD 有过程跟踪方法，这在商业环境中更具有吸引力。

收集用户需求

这一章讨论为软件项目收集用户需求的过程。传统的用户需求被写成文档，详细的描述了系统需要实现的内容。这样的文档是过程中下面几个时期的基础，因为：

- 实现过程就是要完成文档中列出的所有需求
- 验收测试就是比较文档记录的内容和实际实现的内容的过程。

用例（use case），使用情景（usage scenarios），非功能性需求和其他描述人机交互的需求常常伴随着用户需求。这种传统的收集用户需求的方法的主要缺点是太死板。一旦完成了需求文档，用户就不再参与项目，或者开始扮演旁观者的角色。用户需求中的错误，疏忽以及开发者对需求的曲解常常累积到项目的最后阶段，这时候修改成本很昂贵。

XP 和 FDD 都抛弃了传统的收集用户需求的方法并用自己的方法替代。然而，他们在这个时期的重点不同。

XP：收集用户 Stories

XP 在定义用户需求上投入最小的努力。用户需求以撰写短小 story 的形式收集，这些 stories 的目的是为了了解项目范围，尽可能少犯错的估计完成时间。收集 story 的过程依赖于具体项目，一般不超过 2 个星期。一次发布 60 到 100 个 stories 就够了。每个 story 的细节要在迭代计划会议上和用户进行澄清。这样一种轻量级的收集用户需求的方法能够更容易适应变化。

FDD：建造问题域

收集用户需求是使 FDD 项目成功的关键过程。该过程如此重要以至于拥有一个值得自豪的名字“Process One“。这个过程的输出是 UML 图和特性列表。UML 图定义了软件系统的问题域。如果实现的好的话，覆盖了业务的所有相关领域并且在后续的发布中能够容易地增加特性。特性列表成为项目 timeline 的基础，用来进行过程跟踪。在这方面来说 FDD 和瀑布法一样依赖用户需求。意识到这一点，FDD 改变了收集用户需求的方法：用户和开发者交互式的创建需求。

参加这个过程的有熟悉 UML 的开发者，用户代表，问题领域的专家。建模使用彩色 UML。这是由 Peter Coad 在【4】中应用并描述的一个 UML 版本。附录 1 有简短的介绍。建模首先要为问题领域专家简单介绍 UML 语言。虽然他们不需要成为 UML 专家。他们还是能很快的学会彩色 UML。

建模被分成多次迭代完成。每次迭代从一小块业务的用户 Story 开始。整个团队分成几个建模组：每个组由开发者和问题域专家组成。每个组创建反映了用户 Story 的 UML 模型。由团队对模型进行讨论。在迭代的最后通过选择/合并得到最好的模型。建模过程迭代直到满足所有 story 的整个问题域模型被建立。

好的问题域覆盖了用户业务。这样被认为更稳定：因为用户对应用程序的需求会变，然而业务通常是不变的。“Process One”除了建立了面向对象的业务模型，还间接的指出了开发时期的持续时间。根据 FDD 规则：开发团队每 2 个星期建模后要花费大约 6 个月进行开发。

设计和文档

XP 几乎完全省略了这个步骤。来自 XP 社区的建议是：“我们建议你应该写一个能够满足需求的程序，文档尽可能少”。

FDD 也不需要创建设计文档。在 Process One 的最后，开发者创建一个 UML 图的描述，用来记录一些被否决的可选方法以及该决定的原因。这些文档以后会有用：在一个耗时很长的项目里，人们可能会忘记最初决定的细节，文档可以作为提醒。如果用户要求，正式的用户需求可以在这个文档的基础上撰写。

因为 FDD 需要创建许多正式文档，所以它尽量使项目信息公开化。内部网站是一个发布所有项目信息的理想的地方。这些信息包括：开发者列表，主办方和领域专家，UML 模型和注释，论坛，编码约定，用到的工具和库的列表，单元测试报告，进度报告等等。

开发和测试

FDD 和 XP 开发都在短期（1 到 3 周）的迭代中进行。本章描述了两种方法中迭代的内容，同时还描述了两个实践例子。关于 FDD 实践的描述主要基于作者的经验。对 XP 的描述是个人经验与 XP 站点建议的结合。

FDD 和 XP 的开发团队有不同的组织结构。FDD 需要选出一定数量的主程序员（CP），CP 拥有更多的经验从而作为团队的技术领导。他们还承担额外的责任。在团队层次结构中，CP 位于开发人员和项目经理之间。XP 开发团队不需要这样的层次。

迭代计划

迭代过程从计划会议开始。虽然两种方法都使用这个会议开始迭代，但是他们进行的活动不同。

XP 计划

客户选择一个 story 集合进行实现。团队和客户进行交流以澄清 story 的细节。使用开发者的语言重新撰写用户 story 并估计实现需要的时间。上一个迭代过程中的单元测试 failed 和 bug 也要被包含到当前迭代的任务列表中。

FDD 计划

CP 将适当数量的相关特性分组到工作包中，准备迭代计划会议。通常一个团队有多个 CP，他们独立引导迭代。每个迭代不一定包含整个开发团队，相反的，要为每一次迭代建立一个迭代小组。CP 从可用的人中选择迭代小组的成员。每个开发者会收到特性列表的一个子集。迭代小组要检查一遍特性列表，这个列表在 Process One 就熟悉了。如果需要，小组会和客户联系以澄清一些模糊的特性。在复杂案例中小组会建立序列图。计划会议也会为迭代里程碑设置时线（timeline）。

比较不同之处

主要不同在 XP 中，迭代包括整个团队，而 FDD 为每个迭代建立一个由 3 到 5 个开发人员组成的小组。这个事实可以作为解释 FDD 具有更好可测量性的原因：因为迭代小组都很小，沟通不是问题。

XP 迭代看起来能够更好的自适应。下一次迭代要修复上一次的 bug 和失败的单元测试，如果某一次迭代因为要实现太多的用户 story 导致不能产生高质量代码，那么在下一次迭代中实现的用户 story 会自动减少。

FDD 有两种调节机制：

1) 依靠主程序员的经验

2) 如果某些开发人员没有准时完成任务，不会使整个团队慢下来：不忙的开发人员会建立其他的迭代小组。

XP 看上去在迭代中不能在细分：在整个迭代过程中每天的活动都是一样的。FDD 分成以下几个里程碑：

- 迭代计划会议
- 设计阶段

- 设计评审会议
- 编码阶段
- 代码评审会议
- Promote to build moment

FDD 对迭代过程进行更好的跟踪，特别是迭代过程比较长的时候（3 个星期）。

设计和评审

XP 没有正式的设计时期。设计可以在迭代的开始进行或者在任何需要的时候进行。

FDD 有设计时期和设计评审会议。在设计时期，开发者声明实现特性需要的类，方法和属性。他们必须象 Java-Doc 那样使用代码注释的形式很好的文档化。设计完成后，每个团队成员会收到一份打印出来的其他成员代码的拷贝进行评审。在设计评审会议上，团队成员在彼此的设计上添加注释。注意：为了使评审会议简短，实际的评审工作要在评审会议之前完成。

FDD 设计评审会议能够在迭代早期发现错误的决定，这样能够避免整个迭代走向迷途。一旦每个成员的设计都被接受，团队便开始进行开发。为了避免过量的评审工作，迭代小组必须很小：包括 CP 不超过 5 个人。从 FDD 得到的好处是它能够产生充足的 code document。

XP 不信仰 code document。因为 XP 没有专门的设计时期，由每天会议来监控开发过程。另外，结对开发降低了错误设计的风险。

代码演习

两种方法都很重视开发中的这个阶段，虽然他们的优先级不同。XP 和 FDD 都强调了编码规范的重要性。规范能够使互相阅读代码更容易。遵照规范的代码会比较少有错误而且能够被更有效的评审。

单元测试是两种方法的基础部分。单元测试代码可以在主程序代码编写之前（测试驱动风格）或者之后编写。XP 坚持单元测试代码先于实现编写，然而 FDD 没有说明。基本要求是每个类至少有一个测试。

XP 的编码

编写代码是 XP 的主要焦点。XP 为了编码时期的好处省略了项目中的很多活动。最值得注意的是编码是结对进行的。两个开发人员同时坐在计算机前，一个负责打字，两个人一起思考。表面上看，好像是在浪费时间（两

个人打字可以产生更多的代码)。实践证明了相反的结果，可以如下解释：

- 有挑战性的编码工作在讨论中进行（头脑风暴工作方式）
- 编码和代码审查同时进行
- 这种方式使缺乏经验的开发人员更快的进步
- 人们会花费很少的时间在无关的事情上，如阅读邮件，浏览网页等。

XP 提倡重构。重构是在不破坏已有功能的前提下，将代码修改的更好的过程。重构过的代码有更高的质量。

XP 反对代码所有权的思想。尽管由指定开发人员创建了最初的实现，然而任何人都可以修改他的代码。浏览/修改其他开发人员代码的实践方式使系统知识作为一个整体扩展。如果有人离开了团队，由于损失了知识导致的负面影响很小。集体代码所有权有一个自然的约束：任何人不能修改他/她不理解的代码。这个约束在大的工程中会出现。

FDD 的编码

FDD 的编码过程不像 XP 中那样具有激情和挑战性。因为到编码时，特性已经在 Process One，迭代 Kick-off 会议，设计评审会议中被广泛的讨论过了。现在，类和方法已经设计好了，他们的目的也在 Code Document 中描述。编码通常成为一个机械过程。

不像 XP，FDD 强烈抵制重构。这里反对重构的主要依据是它花费了时间而且没有给客户带来任何价值。代码质量会在代码评审会议中保证。

FDD 鼓励强烈的代码所有权。主要思想是每个开发人员了解自己的代码并能够更好的意识到改动会导致什么后果。FDD 从不同的角度对待开发人员离开团队的问题：

充足的 Code Document 使理解别人的代码变得简单。

开发人员知道别人的代码做了什么，因为他们评审了设计。

开发人员会在代码评审时查看彼此的代码。

代码评审

代码评审有几个重要的目的：

- 找出代码中的错误：代码评审可以发现单元测试能够发现的所有错误，特别是象用户界面代码这样很难进行单元测试的地方。
- 加强编码规范。
- 能够培养缺少经验的开发人员，共享最好的编程经验。
- 使开发人员熟悉彼此的代码

在 BRE-275 项目的实现中使用了 XP 开发方法，我们没有进行代码审查。审查被认为在结对工作的时候已经做了。在学术环境中结对编程很困难甚至是不可能的。

FDD 有正式的会议，在迭代时间表中被记做“代码审查会议”。迭代小组的每个成员打印出自己的代码并在组员间分发。实际的代码审查在会议前完成。会议用来讨论发现的问题并指出解决的最好方法。

两种开发方法对于代码审查各有优劣：

- XP 的结对编程很自然的执行了代码审查，不管写了多少代码，都会被审查。
- FDD 将代码暴露在更多的眼睛下。实践表明经常只有一个迭代小组中的成员能够发现一些特别的 BUG。如果只有两个人在看代码，代码中的问题可能不会被注意到。
- 在 FDD 中代码审查从编码过程中分离出来。如果迭代过程很长（3 个星期）或者迭代小组有 5 个以上开发人员，那么代码评审过程会很低效。在一个长的迭代过程中，小组成员可能写出几百页的代码。在一次代码审查中审查这么多的代码很困难：过程会很长，人们会失去耐心导致 BUG 不会被注意到。

部署

在瀑布风格的开发过程中，部署通常是项目周期中的最后一个阶段。在所有的编码和测试工作完成后进行。部署阶段不依赖于开发方法：大公司常常拥有完全独立的开发，测试，产品环境。

FDD 和 XP 都把部署纳入到自己的迭代循环中。因为上边标出的原因可能不是真正部署到产品环境中。然而应用程序应该运行起来，并能够给用户使用。用户使用实现的特性并提供反馈信息。

有时实际管制环境会抵制持续部署的精神。例如：美国政府管制保健卫生工业。规章要求在部署成为可能之前要执行正式确认的步骤。确认的目的是为了保证所有需要的功能都依照用户要求实现。确认小组直到项目结束前所有的特性都实现才会开始工作。中间发布的版本不能被确认，也不能使用产品数据。这些都会使客户不能看到部分实现的系统。

XP 要求客户代表应该成为团队成员，在项目进行中使用系统是客户代表的一个职责。然而 FDD 并不强烈要求用户参与，保证用户及时反馈才是 FDD 的精神。

过程跟踪

软件项目的过程跟踪很重要。只有小项目和内部的开发不需要任何过程指示。大型的项目或者外部项目必须进行跟踪。粗略的过程跟踪是瀑布式开发的特性。项目公共的 5-6 个阶段都会给出他们的进度比例，所以项目经理很容易看到项目的状态并报告给更高一级管理人员。新的开发方法会提供不差于已有方法的过程跟踪能力。

XP 中的过程跟踪

XP 在过程跟踪领域没有提出太多。XP 只是保证进度可见。每个新的迭代增加新的功能，这些功能要马上能够接受用户的验收测试。但是，很难回答这样得简单问题：“项目什么时候能够完成？”一个估算的方法是比较完成的和剩下的用户 Story 的数目。然而这不是一个精确的答案：实现一个用户 story 和用户进行沟通可能会导致产生新的用户 story。

XP 非常敏捷。它最适合于用户需求没有很好的定义或者变化频繁的项目。在这样的环境下项目完成日期这样的问题变得没有意义，所以没有人会费心回答它。

FDD 中的过程跟踪

FDD 提供精确的过程跟踪，基于这样两个事实：

- “Process One”完成后，就知道了特性的总数。
- 每个迭代有定义好的声明周期，这个周期中每个步骤都要给出完成的百分比。下面是 Nebulon 的百分比：

Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to Build
1%	40%	3%	45%	10%	1%

图 6.1.1　FDD 迭代过程里程碑的比例

这些数据能够用来计算项目完成日期和创建跟踪报告。附录 2 中提供了报告的例子。如果手工创建这些报告，成本很高，FDD 项目依靠项目管理工具产生这些报告。

FDD 也面对项目完成日期的问题。Jeff De Luca 提到 FDD 能够承受项目需求达到 10%的改变而不会超过最后期限。这里的需求改变指的是对软件系统的需求，而不是业务逻辑的变化。因此问题域应该仍然有效描述了业务逻辑。新的需求应该不影响已经实现的大部分特性。

摘要和结论

这一节概括了这篇论文不同章节的概念的适用性。我的使用 XP 和 FDD 管理项目的投入使我得出了以下的结论：

a) 每种方法都成功利用迭代开发避免了在瀑布模型中错误最后才能被发现的弱点。

b) 每种方法都是轻量级的，XP 更轻：它不产生编码文档并且保持代码审查不拘于形式。

c) XP 更适合于变化频繁或者用户需求定义不足的项目。

d) FDD 不能很好地击中活动的目标，然而，当用户需求相当稳定的时候，它能够比瀑布提供更多的成功次数。

e) FDD 更可度量。它的开发团队有一定的层次性，即使在项目组很大的时候也能够保证迭代小组很小。

f) 作为方法的一部分，FDD 提供了过程跟踪和报告的能力，这一点有利于管理者并对大的公司更具吸引力。

g) FDD 项目可以手工管理，但是项目管理工具减少了维护特性数据库和产生报告的管理费用。XP 看上去不需要任何特殊的工具。

h) FDD 更像是为那些具有不同经验的开发者组成的团队准备的。具有更多经验/更高生产能力的成员成为主程序员。然而，在由开发水平相等的开发人员组成的小团队中，一些资源会不能充分利用：作者观察到当 CP 准备下一个迭代的特性列表时开发人员无所事事。

i) 两种方法都需要纪律并且不能代替好的项目经理。

j) 两种方法都提供了非常有用的技术，这些技术能够在各种开发方法中应用，即使在瀑布模型中。其中我比较喜欢的两种是单元测试（Unit Test）和代码审查（Code Review）

参考资料

1. Feature Driven Development web site: http://www.featuredrivendevelopment.com

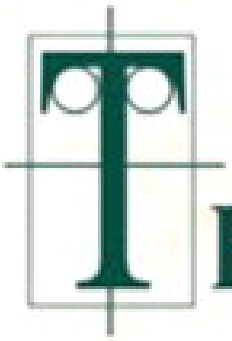
2. Extreme Programming web site: http://www.extremeprogramming.org

3. CSCIE-275 Lecture materials: http://www.people.fas.harvard.edu/~robinson/cscie275/materials/lectures/L04-20040301.pdf

4. Java Modeling In Color With UML : Enterprise Components and Process By Peter Coad, Jeff de Luca, Eric Lefebvre, 1999

5. Extreme Programming Explained: Embrace Change by Kent Beck, Addison-Wesley, 1999.

6. Web site of Nebulon Ltd.: consulting practice of Jeff De Luca: http://www.nebulon.com/fdd/index.html

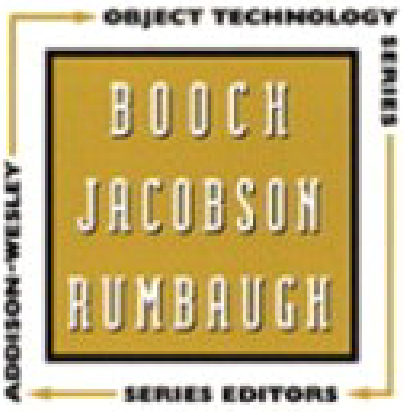


THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

Covers UML 2.0

UMLChina 译
（王海鹏、汪颖）



CD-ROM Included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社出版

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

探索需求

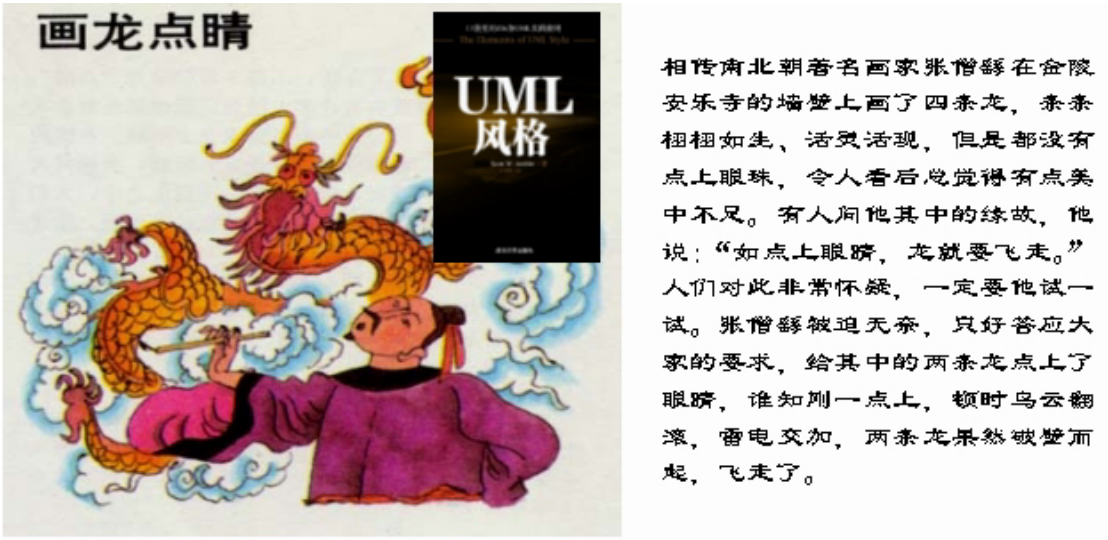
设计前的质量

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

Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社



它不讲概念，它假设读者已经懂了概念。

它不讲工具，它假设读者已经了解某种工具。

它不讲过程，它假设读者已经了解某种开发过程。

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

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

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

开源软件的可用性

David M. Nichols、Michael B. Twidale 著，droplet 译

摘要

开源社区已经成功地开发出大量的开源软件，但是大部分的计算机用户依然只使用商业软件。开源软件的可用性被认为是阻止它被广泛使用的一个原因。在这篇文章中，我们审视关于开源软件可用性的证据，并讨论开源软件的开发方式是如何影响其可用性的。我们同时描述怎样通过现有的人机交互技术来帮助分布在网络上的开发者、用户去解决软件可用性的问题。

简介

开源社区已经成功地开发出大量的软件。这些软件中的大部分都被一些技术用户使用，一般用于软件开发或者作为大型计算设施的一部分。尽管开源软件的使用量在增长，但是普通的计算机用户一般只使用商业软件。有许多原因导致这种情况：其中一个解释是开源软件缺少可用性。这篇文章将审视开源软件的开发过程是如何影响其可用性，同时指出了哪些提高可用性的方法可以被用于基于因特网社区开发的软件开发过程。

对这个话题的讨论基于以下两个常识：

- 开源软件的开发者是那些使用这些软件，同时为此软件开发做贡献的人
- 用户为中心的设计试图通过特定的方法（可用性工程，用户参与的设计，人类学）来弥补程序员和用户在软件开发和使用之间的鸿沟

在人机交互领域中，以用户为中心的设计方法的一个基本原理就是强调程序员很难为典型用户设计软件。简单的说，它强调开源社区很难在桌面环境中做到与商业软件那样满足典型用户的需要（Raymond,1998）。但是，就像我们在这篇文章中讨论的那样，实际的情况很复杂，这里有许多潜在的因素比如态度，实践和技术等。

在这篇文章中，我们首先回顾现有的关于开源软件可用性的证据。然后，我们概括出开源软件开发方式的特点在那些方面影响着它的可用性。最后，我们讨论怎样在分布网络社区环境中使用现有的人机交互技术来提升开源软件的可用性。

开源软件的可用性存在问题吗？

开源软件已经在可靠性，效率和功能方面赢得了声誉，这使得软件工程界的许多人都为之惊叹。因特网帮助并协调分布在世界各地的志愿开发者以提供开源解决方案，某些方案已成为该领域的市场领先者（如Apache Web服务器）。但是，大部分的开源软件用户都有一定程度的技术背景，同时，普通的桌面用户还是在使用具有商业版权的软件（Lerne and Tirole,2002）。对这种情况有这样几种解释：习惯、互操作、与现有数据的交互，客户支持，有组织的销售策略等。在这篇文章中，我们只关注其中一个可能的解释：那就是（对大多数用户来说），开源软件缺少可用性。

可用性通常用五个方面的特性来描述：学习的难度，使用的方便程度，记忆的难度，错误频率和严重程度，还有主体的满意程度（Nielsen,1993）。软件的可用性与软件的用途不同（它关注软件是否具有某些功能），与其他特征也不同，如可靠性和费用等。一些软件，如编译器和源代码编辑器，通常由程序员使用，它们都不会对开源软件的可用性造成大的影响。我们下面的讨论集中在这些软件上（如文字处理器，e-mail客户端或Web浏览器），这些软件的使用者代表了大多数用户。

开源软件可用性的问题很大程度上不是由软件本身造成的，所有的交互类软件都存在这个问题。问题是：开源软件的开发过程制造出的软件与其他方式制造出的软件相比可用性如何哪？不幸的是，很难组织一个受控的试验去比较不同软件工程方法。但是比较通过不同软件工程方法制造出的同类软件的相同功能却是可能的。我们可以了解到的这类研究是Eklund et al.（2002），他使用Microsoft Excel与StarOffice做比较（这种特殊的比较带来了一些问题，因为StarOffice过去是商业软件）。

这两个软件有许多不同点可能会影响这种比较，比如：开发时间，开发资源，软件的成熟度，先前存在的同类软件等等。有些不同点是开源开发方法与商业开发方法间的区别，但是大多数不同点却使判断这种比较是否“公平”变得比较困难。最后，用户测试，就像Eklund et al.（2002）所说，必须是刻薄的测试。但是，就像Mozilla项目（Mozilla，2002）所展示的，开源软件可能需要好几年才能据有这种可比性，起初负面的比较结果并不代表整个过程的缺点。另外，开源软件工程开发的特性意味着软件从一开始就是可见的，相反，处于胚胎阶段的商业软件是被严格保护的。

关于开源软件可用性的出版物非常缺乏，除了Eklund et al.（2002）的研究之外，我们只能找到以下几个项目的研究材料：GNOME（Smith et al.,2001），Athena（Athena,2001），Greenstone（Nichols et al.，2001）。开源项目的特点是强调持续增量开发，这种特性使它不太适合采用传统的正式的试验研究方法（当然文化因素也可能对此有一定的影响，这一点我们会在下一节中讨论）。

尽管关于开源软件可用性的正式研究很少，但有许多意见指出开源软件的可用性存在问题（Behlendorf,1999；Raymond,1999；Manes,2002；Nichols et al.,2001；Thomas,2002；Frishberg et al.,2002）：

> 如果这[桌面环境和应用程序设计]仅仅是个技术问题，那么其结果就没有什么疑问了。但是，这不是，这是一个人类环境学的设计和界面心理学的问题，而黑客从一开始就不擅长它。所以，尽管黑客非常擅长给别的黑客设计界面，但是他们并不擅长掌握另外95％人们的思维方式并为他们设计界面使得像J.Random 这样的最终用户和他的婶婶Tillie都乐意付钱购买它。（Raymond,1999）

> 传统的开源软件用户都是专家，最初的使用者几乎就是开发者的同义词。当开源软件随着Caldera和Corel公司发行明确面向普通桌面用户的LINUX发行版而进入商业主流后，就开始强调可用性和界面设计。相对于开源对非专业用户的吸引力，他们更愿意选择开源产品在价格，质量，品牌和支持方面的优势。（Feller and Fitzgerald,2000）。

Raymond强调，以用户为中心设计的要点（Norman and Draper,1986）是：开发者需要特定的外部帮助以满足普通用户的需求。HCI社区已经开发了若干工具和技术来达到这个目的，它包括：可用性检查方法，界面指导，测试方法，参与设计，以及多学科交叉的小组等。对开源软件的可用性给予越来越多的注意力，表明开源软件进入到与80年代商业软件相类似的时期。

由于软件用户的种类越来越多，并且越来越缺少技术背景，软件制造商开始采用以用户为中心的方法开发软件以使他们的产品更容易被新用户接受。但同时依然有很多用户在使用软件时遇到困难。因此软件公司雇佣HCI专家以改善用户的体验。

随着开源软件的用户群的扩大而包含那些不是开发者的用户时，开源项目需要采用HCI技术，以使他们的软件能够被普通的桌面用户使用。最近的证据(Benson et al., 2002; Biot, 2002)表明一些开源项目开始使用一些以前在商业软件中使用的技术，如提供给应用程序员明确的界面指导等(Benson et al., 2002)。

开源软件是否存在可用性的缺点是个难以回答的问题。现存的问题不能表明开源软件的界面不好或者开源软件的界面一定难以使用，而只是说明它的界面应该并且可以做的更好。一些评论家的意见和一些公司的行动，比如SUN在GNOME项目里所做的工作都说明了这个问题，而学术界(e.g. Feller and Fitzgerald, 2002)却大多对此保持沉默（Frishberg et al. (2002) and Nichols et al. (2001)是个例外）。但是，为了将HCI技术运用于开源软件开发，我们有必要了解开源软件的开发过程中那些对软件可用性有影响的方面。

可用性与开源软件开发

“他们不想为愚蠢的人去做那些无聊的事情”（Sterling, 2002）。

要理解当前开源软件的可用性，我们需要审视当前软件的开发过程。在以用户为中心的设计中一个基本要素就是开发活动可以反映到开发好的系统中。从两份材料（Nichols et al., 2001; Thomas, 2002）中我们可以发现开源软件开发中的一些特点导致它的可用性较差。需要特别指出的是，有些特点是商业软件开发同样具有的，这就可以解释为什么开源软件的可用性不比商业软件差，但也好不到哪里的原因。

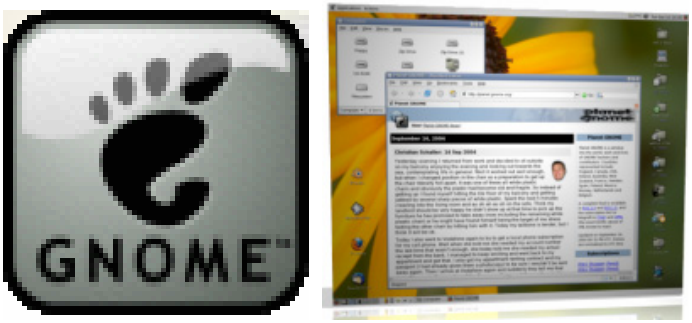
下面列出的特点并不意味着我们发现了全部的问题，但至少是探索这一主题的开端。我们同时指出，要证明这些假设是否正确有很多困难。

开发人员不是典型的最终用户

这是Nielsen(1993)的主要观点，这一点同样适用于商业软件开发者。在教授计算机学科的学生关于可用性主题的时候，根据我们的经验，主要是帮助他们试着以其他人的方式使用和观察系统，而不是他们自己或同伴的方式。实事上，很多高级的开源产品，开发者就是用户，所以那些高深的界面对没有技术背景的人根本没法使用，而这些人比拥有技术背景的人多得多。可能开发一个复杂的，难以学习的软件界面会是一件值得骄傲的事情。这些产品有一个特点就是难以使用，这使得那些使用它的精英可以把自己和普通用户区分开。Trudelle(2002)

指出“产品[浏览器]应该面向他们（开源软件开发者）认为的无知的新手”

但是当为这些非技术用户设计产品时，传统的可用性问题就出来了。在Greenstone研究(Nichols et al., 2001)中，一些命令行的使用习惯，如正确的执行没有提示会使用户迷惑。使用名词’man’(UNIX命令行)来代表帮助系统，在GNOME界面中使用名词’regexp’（正则表达式）都是典型的将开发人员的术语呈现给最终用户的案例。



开源软件方法在解决可用性问题时失效了，因为查看界面的都是一些“错误的眼球”，他们不能发现可用性问题。从某些方面来说，开源软件可用性的问题重现了早期商业软件开发中的一些问题：最初，大多数应用是计算机专家为另一些计算机专家开发的，但随着时间推移，系统的使用者中的大多数变成了非专家用户。向非专家用户转移，开源软件和商业软件走过了相同的轨迹，只是开源软件晚了几年。

商业软件和开源软件的不同之处在于：商业软件开发意识到了这些问题并且能够雇佣HCI专家来改变原来开发团队的组成结构，并在开发时优先考虑用户的使用习惯(Frishberg et al., 2002)。但是，以志愿者为主的开源软件组织没有能力来雇佣HCI专家以拥有在以用户为中心的开发中所需的技术。另外，在商业软件开发中，HCI专家也有足够的权力去决定优先考虑用户使用习惯。

可用性专家没有参与开源项目

有证据表明，参与开源项目的人员中，很少有具有可用性经验的人；其中一个例子就是Mozilla项目中（Mozilla,2002）“鼓励UI设计人员参加开源社区” (Trudelle, 2002)。开源软件来源于黑客文化（O'Reilly, 1999)。这种文化可以被其他黑客接受，他们来自不同的国家、组织、时区，通过Internet联系。但这种文化并不能被非黑客接受。

良好的可用性设计来自许多不同的学科，包括但并不仅限于：心理学，社会学，图形设计，甚至戏剧研究等。多学科交叉团队在解决可用性问题时非常有效，但是需要特别的技术去创建和支持这个团队。结果是，现有的开源团队可能缺少一定的技巧去解决可用性问题甚至是邀请外部人员来帮助解决可用性问题。黑客缺乏沟通技巧对创建多学科交叉团队来说绝对不是一件小事。

更进一步，一个成功的，多产的开源项目开发人员所需的技巧和态度没有什么要求。有许多黑客可以参与一个项目，所以开源项目有多种方法去评选其中水平最高的人，并给予他们更多的控制权和责任。如果这些方法同样适用于那些潜在的可以解决可用性问题的人的话，将会使开源软件中可用性专家短缺的问题更加严重。

以下几个原因可以解释为什么在开源项目中HCI专家那么少：

- 可用性专家比黑客少，没有那么多人参与
- 可用性专家没有兴趣，或者他们工作不能得到黑客们的肯定
- 可用性专家在开源社区中不受欢迎
- 习惯：传统的软件开发不需要可用性专家。当前在开源项目中程序员多而可用性专家少就是受这种传统的影响。
- 在开源项目中缺少可用性专家使得可用性专家没有机会参与开源项目并得到同行的赞赏。

在开源项目中增加功能的愿望比改进可用性的愿望大

是不是开源软件的开发人员没有兴趣设计更好的界面？由于在开源项目中工作的都是志愿者，他们都按自己的兴趣选择相应的工作，这些工作中大多不包括向初级用户提供的功能。这些志愿者参与开源项目的几个重要的动机已经被整理出来了(Feller and Fitzgerald, 2002; Hars and Ou, 2001)。它们包括从同行中获得尊重和解决难题的乐趣。增加新的功能或优化代码使得一个黑客的才能得到展示。如果一个开源开发者认为增强软件的可用性比较不重要，比较简单或者没有什么乐趣，他们就不会选择在这个领域中工作。这些志愿者有两类：选择全程参与或者选择那些问题比较多的部分参与。在这种比较中，可用性问题根本排不上号。

更有甚者，当参与一个开源项目时，系统软件的开发比应用软件的开发更受重视。“差不多所有知名的成功的开源项目都是在有一个技术需求但是没有相应的商业软件或开源软件实现这一技术的情况下开始的”。

Raymond把这种动机叫做“满足个人的愿望”(Raymond, 1998)。显然大多数的开源项目创建者都是需要一个高级的应用程序，开发库或系统框架以满足个人的需求，而不是去开发一个简单的程序去满足非技术用户的需求。

> “从开发者的角度来看，解决一个可用性问题不是一个值得骄傲的经验，因为解决这个问题并不需要特殊的编程技巧，新的技术或算法。同时，解决可用性问题也许并不会对程序的行为有大的改变（但是从用户角度来看或许有大的改变）。这种程序行为的改变很微妙，但是对典型的开源开发人员来说，它没有增加特性或解决BUG那么有趣。（Eklund et al., 2002）

“个人的愿望”这个动机使得开源软件与商业软件有很大的区别。商业系统开发是为解决另外一个群体的需求。商业软件是要把软件卖给用户来赚钱的，所以用户通常会非常谨慎，因为他们本身没有开发技巧。捕获这些用户的需求是软件工程中的一个难题，相应的技术被发展出来解决这个难题。相反，许多开源软件缺少用户需求捕获过程，甚至是正式的规格说明书（Scacchi, 2002)。它们依靠最初的发起者或一个小的社区来了解需求。它们被非正式地支持，它们通过不断演化的程序代码来实现需求，也有可能这种实现和最初的需求完全无关。

与使用相关的问题说明，具有讽刺意味的是，开源软件比闭源软件（CSS）更自我。一个私人的愿望表明软件设计是为满足私人需求的。明确的需求就不是必须的。开源软件在一个有相同想法的社区中使用，改进或者优化，并使这个社区的人受益。相反，闭源项目是为那些具有不同特点的人设计的，它的一个出发点就是增强软件的可用性，特别是软件要容易上手，这样就可以卖出更多的软件。

可用性问题比功能性问题更难描述和发布

功能性问题比可用性问题更容易描述，评估和模块化。它的特性使得它比较容易解决。一些（但不是全部）可用性问题比较难以描述，因为它涉及整个屏幕，交互和用户的使用习惯。给界面臭虫的打补丁可能不如给功能臭虫的打补丁那么有效。解决可用性问题可能需要对整个系统进行检查，而不仅仅是对现有设计的一个小改动，它涉及到超过一个设计师的界面设计，特别是当他们独立工作时，这将导致设计不一致而使得可用性降低。与此相反的是在一个高度模块化的应用中修改功能性臭虫就不会碰到这些问题。模块化会使改动只影响本模块。重要的（或高优先级的）重构可以发生在正在开发的项目中而对用户保持不可见。但是，界面改动的影响却是全局的，因为界面需要保持一致。

开源软件的模块化使得它们可以避免Brook's的法则。不同的部分可以在下一版中被更高级的模块替换掉。但是，对可用性来说，一个重要的因素就是设计的一致性。在不同模块或不同版本间微小的差异都可能激怒、迷惑用户，使得整体的可用性下降。它们全局的特性意味者界面不适合采用黑盒的方式，增量并且分布式的开发。

我们必须指出，开源项目成功地解决了特定类型的可用性问题。一个常见的类型就是开源软件的界面设计常采用“皮肤”：不同的界面布局会大幅改变整个应用程序的外观，但是对底层交互影响却很小。另一个类型就是软件的国际化，软件的界面文字（或是与语言相关的图片）都被翻译成当地的语言。这两种类型对模块化程度高的开源项目比较适用，但是如果解决的是深层的交互问题，就不太容易通过重新设计交互顺序而使项目的可管理性更好。其原因就是解决深层的交互问题可能涉及的不光是整个界面的改动，而且可能涉及到不同功能元素的改变。

另一个主要的，开源项目解决可用性问题比较成功的就是软件安装（主要是GNU/LINUX系统）。既便是专业人员有时都很难安装好早期的GNU/LINUX系统。Debian项目(Debian,2002)开始创建一个比较好的发行版以使安装更加容易，其他的项目和公司跟随了这种潮流。这些项目用传统的开源软件开发方法解决了这种可用性问题。复杂的需要手动进行的操作都被自动化了，最终用户不太想知道的信息都被隐藏了。当然，这是个开源项目，如果有人愿意去查看这些信息，项目还是提供了许多接口供他们使用。

可用性设计应该在编码之前进行

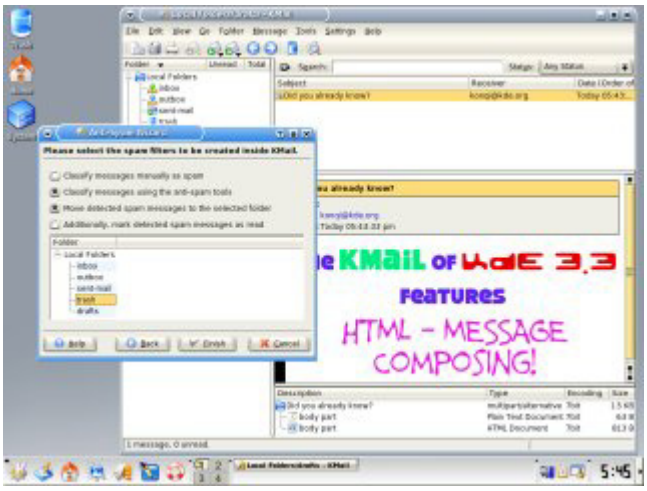
从某些方面来看，开源软件开发过程非常成功，它打破了传统软件开发过程中的一些既定规则。一个组织良好的项目意味着先制定计划，捕获需求，并确定哪些事情应该在编码之前进行。相反，开源项目尽可能早的开始

编码，它依靠不断的代码审查去精化，提升整体的设计：“你开始的开发社区需要一个可运行，可测试的程序去试用”（Raymond,1998）。相同的，Scacchi（2002）的研究也表明“在软件工程教科书中建议的正规的需求捕获，项目分析和规格定义等活动在开源项目中很少见到”。Trudelle（2002）也指出，Mozzila的设计是在修改BUG的过程中进行的，而这些BUG是在一些早期发行版中出现的。

这种方法在一些项目中适用，但在另一些项目中，项目协调者和主要参与人员之间需要有一个明确的计划并有共同的愿景。但是，界面设计工作最好在编码之前进行才会有好的效果。如果连编码的计划都没有，界面设计就不可能展开。开源项目的计划通常由项目发起人在很多人参与之前撰写。我们可以想象，开源软件的参与者通常对将要实现的功能比较清楚（这一点可以不用和传统软件开发中的预先计划相同），但他们通常对将要实现的界面没有什么共通的想法。除非项目发起者碰巧有很好的交互设计经验，否则，可用性问题会在项目初始阶段被忽视。以上我们所说的案例并不表明闭源软件就一定做得很好。我们只是想找出开源实践中的问题，以便我们能够解决这些问题。

开源项目缺少解决可用性问题的资源

开源项目是志愿性的，并且预算较少。雇佣外部专家如技术作家和图形设计师就显得不太可能。就像前文所述，这使得由志愿者组成的开源项目组中缺少此类人员。可用性实验室和大型的试验设备对大多数开源项目来说太贵了。在KDE（KDE Usability 2002）的可用性邮件列表上考虑请求可用性实验室捐献一定的使用时间以便在他们的设备上研究KDE的可用性。近期，有一些公司开始参与几个开源项目以解决其可用性问题，比如Benson et al.（2002）,即使它们在这些项目上的投资还不能和一些大的商业软件相比。除非解决开源软件可用性的资源有所增加或者有其他途径可供选择，否则开源软件的可用性问题还会因为资源缺少而难以解决。



商业软件成为了可用性的标准，开源软件只能跟进

不管商业软件的可用性是否优越，它们在最终用户中有非常大的影响，使得以后的创新设计都需要遵循这些习惯性的东西。为了使开源软件被最终用户接受，它们也开始遵循主流软件的界面设计。因此在（Eklund et al.. 2002）的测试中，Star Office的spreadsheet，calc就设计了与Microsoft Excel类似的界面以使学习使用它的时间更短。结果就是它只遵循的Excel的设计理念，而没有考虑是否能够在此基础上改进。

这并不是保守主义的原因，而仅仅是为了与闭源软件竞争以使它们的用户使用等同的开源软件。另一个可能就是，目前典型的开源开发者都比较喜欢开发有创新性的功能，而没有兴趣去开发有创新性的界面。最后，商业软件的底层代码是不可见的，它需要开源软件对手采用逆向工程去开发相同的功能。这项工作需要很大的创造力。相反，软件的界面是人人可见的，做出一个相同的界面也比较容易，或者干脆照搬就行了，要修改只不过是版权声明之类的内容。我们可以想象，如果没有这些因素的话，开源软件在功能和界面上进行巨大创新非常容易，因为它们并没有短期资金方面的压力。

开源软件比商业软件更易变得庞大

许多商业软件因为日益庞大的功能而受到批评，因为功能越多，购买所需的钱就越多，而且所需内存也越多。商业压力使得它们增加功能来让用户购买最新的升级版本。事实上，增加新的功能可能会降低软件的可用性，使得用户在使用一个小功能时都不方便。

同样的压力也存在于开源软件中，但原因却不相同。开发人员的习惯就是增加新的功能，而不愿意删除功能，特别是删除一些有问题的功能可能会激怒另一些开发人员。更糟的是，开发人员之间的尊重是参与的动力。所以宁可增加功能而使一些最终用户不愿意使用，也不会用更好的代码替换另一块不好的代码。项目维护者，为了让参与者高兴，可能会保留所有的功能，即使这个功能不太好用，如果有两个相同的功能，就把两个功能都保留，而让用户根据自己的需求去选择用哪一个。这样，参与该项目的人都得到所需的信誉。但是这种杂凑的设计方法值得商榷。

“多发布，早发布”的方法可能导致一个蹩脚的特性被用户接受。用户花了时间和经历去学习它们并且在工作中使用它们。当一个改进的版本发布了，即便它有更好的界面，使用原来界面的用户也不愿意使用新的界面。就算这个界面比旧的界面更易使用，但是因为旧的界面已经融入他们的工作环境中，他们不太愿意去重新学习新的界面而改变自己的工作环境。这种情况下，项目维护者就会在最新版本中保留了多个历史界面。保留旧的界面，取悦了旧的用户，吸引了更多开发者，却使最终产品更难使用。

开源软件倾向于更强大而不是更简单

大家公认“肿件”是不好的。但是，经过深思熟虑的，在系统中增加的多个可替换的选择总比那些不公平的取舍更好一点。我们可以想象，提供多种选择，对开源开发者来说，是一个期望的特性（即使从美学角度来说也是这样）。最终结果就是，应用程序提供了许多配置选项供高级用户去裁减，但是对普通用户来说，这简直是恶梦。在GNOME的桌面中有五个不同的时钟就是一个典型的例子。另一个不好的倾向就是开源软件的配置界面越来越多。

开源软件变得越来越复杂，这对普通用户来说，软件可用性会降低，但是对高级用户却意味着软件的功能变得强大。专业开发者很少碰到程序在缺省配置下不能使用的情况，所以他们很少关注这些配置，但普通用户却依赖这些缺省配置来使用程序。当然，商业软件的复杂性也在增长，但是至少有一些方法可以限制这种增长，比如开发额外功能所需的开销，或者是可用性问题受到越来越多的关注等。

解决开源软件可用性问题的一些方法

以上描述了造成开源软件可用性低下的一些因素。当然，有些因素可以使开源软件的可用性更高，即使这些因素在现在的开源项目中还显得不太重要。

一个重要的因素就是需要最终用户参与开源项目的开发。这种参与可以在撰写详细的需求时，测试时，撰写文档时，报告臭虫时，或者提出新需求时等等。这种参与同样是HCI专家的建议，比如Shneiderman（2002），并且在商业软件中已经采用（Kyng and Mathiassen,1997）。问题在于如何鼓励非专业用户和HCI专家参与这些与传统软件开发过程不同的开源软件开发？

我们描述了几个潜在的在开源开发中可以提高可用性的途径。

商业途径

一种方法就是开源软件提供功能强大的软件而商业公司提供较好的用户界面。值得指出的是，在开源软件开发中，由大公司领导的项目比那些由志愿者领导的项目有更多的设计经验和资源（Smith et al.,2001;Benson et al.,2002;Trudelle,2002）。但是，仅仅在开源项目中使用与商业软件相同的HCI方法并不能对软件可用性有多大的帮助。这是否意味着，提高开源软件可用性的唯一方法就是给它包装一个商业软件的界面？当然，这是一种办法，Apple OS X服务器就是一个很好的例子，另一个例子就是GNU/LINUX的商业发行版（它们面向的不太专业的用户）。Netscape/Mozzila的共享代码设计是另一种模式。但是，就像Trudelle（2002）指出的那样，商业伙伴和开源

开发者可能在对界面开发的方向上存在分歧，他们都会按照自己的兴趣去设计界面。

技术手段

一种解决HCI专家缺少的方法就是采用界面自动评估。Ivory and Hearst（2001）提出了一种复杂的自动可用性评估方法。自动评估有这几个优势：花费低，连续性好，并减少对评估员的需求。比如，Sherlock工具（Mahjan and Shneiderman,1997）能够自动检查应用程序中的图像和文字的一致性，它使用了简单的方法，比如它比较应用程序中的文字是否一致，控件的宽度是否一致等。如果应用程序的界面和其他代码可以很容易分开，就比较适合采用这种方法去检查。

一种有趣的了解用户使用习惯的方法是使用“用户代理”（Hilbert and Redmiles,2001），它允许开发人员把他们的设计期望作为程序的一部分。当用户执行了非法的操作（它会激活用户代理），程序的状态会被收集起来并发送给开发人员。这是应用程序的辅助功能，它主要收集用户操作行为（比如用户填写表单的顺序），它并不增加程序的价值。辅助工具对闭源开发者来说是一个重要的提升程序品质的工具（Cusmano and Selby,1995）。

学术界的参与

值得注意的是早先发现的一些问题已经在高等教育中有所体现（Athena，2001；Eklund et al.,2002;Nichols et al.,2001）。在这些课程中，学习HCI的学生参与或研究开源代码。这些活动对软件开发者是个喜讯，虽然它最初的目的只是教学实践。但他们实践和发现问题，解决问题的愿望要比完成作业的愿望更大。

这种模式表明个人，团体或班级可以参与下列的开源活动而不是全部的可用性分析和设计，它们包括：用户研究，工作环境研究，设计要求，受控试验，正式分析，设计草图，原型和实际的代码建议。为了支持这种参与，开源软件的支持软件需要做一定程度的修改。

最终用户参与

截至本文撰写之时，Mozzila的臭虫数据库Bugzilla已经收集了150,000个臭虫报告。大部分的臭虫报告都是关于功能性的（很少有关于可用性的），而且这些报告都是由专业用户或开发人员提交：

> 普通用户的报告都非常少见，我的一个估算是大约10%的用户知道新闻组是什么（而且他们中90%的人都是在打发时间），这就意味着少于10%的人提交了臭虫。这就意味着我们没有倾听到90%用户的需求，除非我们试图联系他们。

一般来说，大部分的开源用户都比较被动。比如99%的Apache用户就比较被动（Nakakoji,2002）。

用户不愿意参与的一个原因是参与所需的花费比参与所得的好处大。注册到Bugzilla（它是提交臭虫报告先决条件）并学会使用它的WEB界面所需的时间和精力不少。Bugzilla中的使用界面和使用习惯对大多数人来说也是个障碍。相比之下，Mozilla和微软的Windows XP里面的系统崩溃报告工具就不需要注册。更进一步，这些工具是应用程序的一部分，它不需要用户去使用另外的WEB站点。

我们认为，集成的可让用户报告可用性问题的工具是解决开源项目可用性的一个重要途径。在这种情况下，用户一碰到问题，就可以立即报告。现有的HCI研究（Hartson and Castillo,1998；Castillo et al.,1998;Tompson and Williges,2000）表明，在小范围内，用户的报告有利于解决可用性问题。这些工具是应用程序的一部分，它们没有专业词汇，易于使用，而且可以返回客观的程序状态信息，还有一些用户的评论（Hilbert and Redmiles,2000）。这些客观的和主观的数据组合使我们可以发现用户在使用软件时犯错的原因（kassgaard et al.,1999）。除了这些用户报告，应用程序也可以引导用户参与得更深（Ivory and Hearst,2001）。但Kaasgaard et al.（1999）指出，很难预测这些附件程序对整个系统使用时的影响。

另一种方法是提供一个可用性测试包可以让任何人随时测试。用户的测试结果返回给开发人员。Tullis et al.（2002）和Winckler et al（1999）描述了在测试网站时使用了这个方法；一个独立的窗口用于指导用户在主窗口中的操作顺序。Scholts（1999）也描述了同样的方法，他们把导航作为程序的一部分。试验室的测试与远程网站的测试表明，它们在收集用户使用习惯时的效果时差不多的，虽然实验室直接观察更加有效，但是大量的远程用户参与可以弥补远程网站测试中不能直接观察这个缺陷（Tullis et al.,2002;Jacques and Savastano,2001）。

这两种测试方法都不需要用户学习一些专业词汇。所以它们可以很好地应用于开源开发，它们允许参与者根据自己的实际情况来参与，这一点对比较分散的开源社区非常适合。虽然这种方法缺少在实验室中做可用性研究的那种可控性，但是它们得到的数据更真实，因为这些数据来自用户的使用环境（Thomas and Kellogg,1989;Jacques and Savastano,2001;Thomas and Macredie,2002）。

如果能够鼓励用户更进一步的参与，我们就可以非常容易地找出用户测试报告中的问题或结果的产生的原因。Bugzilla是一个共用的程序，它对开发人员来说是合适的，但是对普通用户来说，他们需要一个更简单的，不需要了解那么多技术细节的工具。Shneiderman建议的方法是如果用户参与了，他们可以得到相应的奖励，比如在购买软件时给予一定的折扣。如果是开源软件，可以在程序中声明比如这些信息“你最近的四个报告使我们可以解决臭虫X，它在新的1.2.1版的软件中已经解决”，这些信息可以让用户觉得自己的工作很有价值。

创建一个讨论可用性问题的基础设施

Bugzilla工具对报告功能性臭虫来说比较好用，但是对报告可用性问题就比较麻烦。如果我们希望得到那些关心HCI的人的臭虫报告，我们可能需要提供一个轻量级，不提供许多技术细节界面的Bugzilla。特别是那些建立在代码管理系统上的系统可以很容易地处理文本信息，对图形就不太适合了。

当收集到用户的报告和测试结果，这些数据需要重组，分析，讨论并据此采取行动。许多关于可用性的讨论都使用图形来表达，它们适合用草图和注释方法来表达。在Mozilla的臭虫报告中就有许多用ASCII字符画的图来表示界面元素应该在的位置。Hartson和Castillo（1998）分析了包括视频和界面快照等图形技术，它们可以作为文字图形的一个补充。比如，在应用程序的臭虫报告中可以包含一个界面的快照，并且这个结果界面可以在以后的在线讨论中被引用。虽然它们看起来可能有一些不同，但是根据可用性研究的一个关键结论是：细节不是很重要，它们可以在后来用户的参与中逐渐明确（Nielsen,1993）。

把可用性分析和设计分成多个部分

我们可以考虑使用一些轻量级的解决可用性问题的方法，它们与上面所描述的在学术机构和商业组织中需要采用正规的试验设备和正式的参与者不同。最终用户可以自愿地描述他们自己对软件的使用体验。有可用性经验的人可以提交他们的分析。更进一步，这些志愿者可以在一个用户的样本空间上做用户研究并提出报告。从这些小数量的研究中可以得到很多关于可用性的信息（Nielsen,1993）。

开源软件也可以采取同样的方法，把开发任务分成多个独立的部分，这样世界各地的人可以选取一个部分做个体的可用性研究，然后再把这些结果综合起来做分析。协调并行的小范围的研究需要可裁减的软件，但是它正好适用于开源软件的分布开发的特性。远程可用性研究表明采用必要的分步式开发是可行的，未来的工作是如何协调好这些项目并从收集的结果中做出正确的分析。

专家参与

HCI专家参与的一个主要问题是他们参与的动机。我们已经指出，开源软件有大量的人参与，而且开源软件中存在许多可用性问题，有这些专家参与，有关设计的取舍可以被更高效的讨论。另一个好处就是，不同的解决可用性问题的方法可以被采用，而不是仅使用文字图形的方式，这样，交流成本就会降低。我们可以想象，对某些HCI专家来说，参与开源项目有一些困难，因为他们建议的设计和改进措施在传统的以功能性为中心的开源项目中受到抵触。这怎么办？清晰的可用性分析和设计规则可能会有所帮助。如果缺少这些东西，可用性专家的建议可能会被忽视。

可用性专家的另一个角色可以是最终用户的中介者。这涉及到分析用户的报告，并运用他的专家知识去创建一个精简版的报告，这个报告集中描述开发人员关注的地方，而去掉那些带有偏见性的和没有代表性的描述。这些专家可以以最终用户的身份参与开发讨论会，以改正传统开源软件中只关注开发人员的个人需求而忽视目标用户需求的做法。通过鼓励最终用户参与，用户专家参与所带来的结果也可以被记录下来并变得可跟踪了。

教育

商业软件开发商也意识到软件可用性是他们需要重点考虑的问题，因为可用性问题会对产品的销量有很大的影响，开源软件也是一样。所以开源软件有足够的理由去学习如何提高软件可用性的技术。当然，这并不仅仅是把软件卖出去的问题。Nickell（2001）指出，开发人员还是希望他们的软件能够被大多数人使用的，他们也乐意改进那些会促使用户使用他们软件的因素。

仅创建一个技术的基础设施对鼓励可用性专家和最终用户参与的帮助不是很大，因为这里面缺少与之相应社会基础设施。新的开源项目的参与者不应受到冷遇，即使他们缺少传统黑客所具有的技术能力。那些如“无知的新手”和“愚蠢的用户”之类的词句应该去掉，或者至少把它们转移到更加技术化的区域去。除了能够使开源社区的组成人员更加多元化，开源社区还能够在这些人的帮助下变得更具亲和力。因为有多个学科的参与，这些学科包括商业软件开发中的心理学家和人类学家，需要采取一定的手段以使参与者能够高效的合作（Crabtree et al.,2000）。

讨论和未来的工作

我们不能说开源软件开发完全忽视了高可用性的重要性。最近的活动表明，越来越多的开源软件开始意识到可用性的问题。这篇文章指出那些影响开源软件可用性的障碍并探索可以解决这些问题的方法。有些方法是直接从前述的问题中得到：在缺少可用性专家的团队中采用自动化评估或者鼓励更多的最终用户和可用性专家参与项目，并从普通用户的角度提出问题。如果传统的开源软件开发捕获的是开发者自己的需求，那么可用性就是要更多地关注他人的需求。

对本文主题更进一步的研究可以通过多种方式进行。开源软件开发的一个巨大的优势是它的开发过程是可见的，并且被很好的记录。通过研究项目的文献（尤其是那些界面元素很多的项目如GNOME，KDE，MOZILLA），可以验证原来软件界面中的假设和声明，从而能够更好的理解现有软件中的可用性问题并改进之。其他问题包括“HCI专家如何成功地参与开源项目？”，“是否可用性问题并不像说的那么大”，“是什么因素使得开源软件在功能性和界面方面的创新与众不同？”。

上述的方法需要更进一步的检查和试验以确认它们是否可用于开源项目而不与传统的已证明是有效的以功能为中心的方式相抵触。这些方法并不仅适用于开源软件，有些方法对闭源软件也适用。其实，参与可用性工程送折扣软件和参与设计都是闭源软件的良好实践。但是，它们也许更适合在开放的，由志愿者组成的开源社区中使用。

大多数HCI研究更集中在软件发布前的正式设计技术而对发布后的研究却很少（Hartson and Castillo,1998;Smilowitz et al,.1994）。在HCI的教科书中，参与设计比参与使用占据更重要的位置。在这方面，开源开发不应该仅仅学习闭源软件的方法，而是考虑如何鼓励用户使用软件以便在后续版本中重新设计它。学术界（Shneiderman 2002;Lieberman and Fry,2001;Fischer,1998）呼吁应该鼓励用户更多的参与而不仅仅是在设计阶段参与（比如可用性测试，原型评估和工作环境调查等）。值得注意的是，这些在商业软件中被忽视的方法，现在开始在开源项目中被采用。

Raymond（1998）指出“调试是并行的”，我们也可以说可用性报告，分析，测试也是并行的。当然，特定类型的可用性设计看来是不太容易并行化。我们相信在将来的研究和开发中，可以帮助实现设计，测试，组织的并行化的技术会成为一个重点。特别的，将来的工作将会检验在本文中所提出的方法是否只是空谈（它们来自于开源软件开发）或者是对开源开发模型有洞察力描述。

结论

改进开源软件的可用性并不是说这种软件就可以完全替代同类的商业软件，因为这有许多因素，比如习惯，支持，立法和遗留系统等。但是改进开源软件的可用性有利于它的传播。我们相信本文是第一个详细讨论此问题的论文。

Lieberman和Fry（2001）的预测指出“解决一个软件臭虫将会以一种交互式的过程进行，这个过程涉及最终用户，系统和开发人员”。一些开源开发人员已经是这样做的了，如果这个用户基础可以扩展到整个最终用户群，那么软件开发的实践将会有巨大的改观。

有许多HCI技术比较容易，而且便宜，它们可以被开源开发人员采用。而且有些方法在分步的用户和开发社区里面使用效果更好。如果开源项目可以提供一个简单的框架让用户提交非技术性的，有关软件使用的信息，那它就可以吸引更多的用户参与其中。

Raymond（1998）指出“如果有足够多的眼球，所有的臭虫显而易见”。对可用性臭虫来说，传统开源社区中所具有的“眼球”可能是错误的“眼球”。如果鼓励更多的可用性专家和最终用户参与，我们可以说：如果有足够的用户体验报告，所有的可用性问题都可以迎刃而解。通过鼓励更多的最终用户参与，开源软件可以创建一个社区去解决可用性问题，就像它们已经采用同样的方法成功地解决了功能性和可靠性的问题。

关于作者

David M. Nichols是新西兰Waikto大学计算机科学系的讲师。

Michael B. Twidale 是Illinois at Urbana-Champaign大学信息与图书馆系研究生院的助理教授

致谢

我们感谢以下人员就此话题的讨论和评论：Damon Chaplin, Dave Dubin, Ian Murdock, Havoc Pennington, Walt Scacchi, Matthew Thomas, Stuart Yeates, the GSLIS writing group at UIUC, the HCI Group in the Department of Computer Science at the University of Waikato and many people who commented via Slashdot.

注释

1. Raymond and Steele, 1991, p. 364.

2. Feller and Fitzgerald, 2002, p. 114.

3. Feller and Fitzgerald, 2002, p. 139.

4. Feller and Fitzgerald, 2002, p. 85.

5. Feller and Fitzgerald, 2002, p. 101.

6. Nielsen, 1993, p. 12.

7. Feller and Fitzgerald, 2002, p. 93; Raymond, 1998.

8. A Bugzilla comment at http://bugzilla.mozilla.org/show_bug.cgi?id=89907#c14.

9. Shneiderman, 2002, p. 29.

参考资料

Athena, 2001. "Usability Testing of Athena User Interface," MIT Information Systems, at http://web.mit.edu/is/usability/aui/, accessed 28 November 2002.

Brian Behlendorf, 1999. "Open source as a business strategy," In: M. Stone, S. Ockman, and C. DiBona (editors). Open Sources: Voices from the Open Source Revolution. Sebastopol, Calif.: O'Reilly & Associates, pp. 149-170, and at http://www.oreilly.com/catalog/opensources/book/brian.html, accessed 28 November 2002.

Calum Benson, Adam Elman, Seth Nickell, and Colin Z. Robertson, 2002. "GNOME Human Interface Guidelines 1.0," The GNOME Usability Project, at http://developer.gnome.org/projects/gup/hig/1.0/, accessed 10 October 2002.

Sebastien Biot, 2002. "KDE Usability — First Steps," KDE Usability Project, at http://usability.kde.org/activity/testing/firststeps/, accessed 28 November 2002.

José C. Castillo, H. Rex Hartson, and Deborah Hix, 1998. "Remote Usability Evaluation: Can Users Report Their Own Critical Incidents?" Proceedings of the Conference on Human Factors on Computing Ssytems (CHI'98): Summary. New York: ACM Press, pp. 253-254.

Andy Crabtree, David M. Nichols, Jon O'Brien, Mark Rouncefield, and Michael B. Twidale, 2000. "Ethnomethodologically Informed Ethnography and Information System Design," Journal of the American Society for Information Science, volume 51, number 7, pp. 666-682.

Michael A. Cusmano and Richard W. Selby, 1995. Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People. New York: The Free Press.

Debian, 2002. Debian GNU/Linux, at http://www.debian.org, accessed 28 November 2002.

Susanne Eklund, Michal Feldman, Mary Trombley, and Rashmi Sinha, 2002. "Improving the Usability of Open Source Software: Usability Testing of StarOffice Calc," presented at the Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002), Minneapolis, Minn., at http://www.sims.berkeley.edu/~sinha/opensource.html, accessed 28 November 2002.

Joseph Feller and Brian Fitzgerald, 2002. Understanding Open Source Software Development. London: Addison-Wesley.

Joseph Feller and Brian Fitzgerald, 2000. "A Framework Analysis of the Open Source Software Development Paradigm," In: W.J. Orlikowski, P. Weill, S. Ang, H.C. Krcmar, and J.I. DeGross (editors). Proceedings of the 21st Annual International Conference on Information Systems, Brisbane, Queensland, Australia, pp. 58-69.

Gerhard Fischer, 1998. "Seeding, Evolutionary Growth and Reseeding: Constructing, Capturing and Evolving Knowledge in Domain-Oriented Design Environments," Automated Software Engineering, volume 5, number 4, pp. 447-464.

Nancy Frishberg, Anna Marie Dirks, Calum Benson, Seth Nickell, and Suzanna Smith, 2002. "Getting to Know You: Open Source Development Meets Usability," Extended Abstracts of the Conference on Human Factors in Computer Systems (CHI 2002). New York: ACM Press, pp. 932-933.

Alexander Hars and Shaosong Ou, 2001. "Working for Free? — Motivations of Participating in Open Source Projects," Proceedings of the 34th Annual Hawaii International Conference on System Sciences. New York: IEEE Computer Society Press, pp. 2284-2292.

H. Rex Hartson and José C. Castillo, 1998. "Remote Evaluation for Post-Deployment Usability Improvement," Proceedings of the Conference on Advanced Visual Interfaces (AVI'98). New York: ACM Press, pp. 22-29.

H. Rex Hartson, José C. Castillo, John Kelso, and Wayne C. Neale, 1996. "Remote Evaluation: The Network as an Extension of the Usability Laboratory," Proceedings of the Conference on Human Factors in Computing Systems (CHI'96). New York: ACM Press, pp. 228-235.

David M. Hilbert and David F. Redmiles, 2001. "Large-Scale Collection of Usage Data to Inform Design," In: M. Hirose (editor). Human-Computer Interaction — INTERACT'01: Proceedings of the Eighth IFIP Conference on Human-Computer Interaction, Tokyo, Japan, pp. 569-576.

David M. Hilbert and David F. Redmiles, 2000. "Extracting Usability Information from User Interface Events," ACM Computing Surveys, volume 32, number 4, pp. 384-421.

Melody Y. Ivory and Marti A. Hearst, 2001. "The State of the Art in Automated Usability Evaluation of User Interfaces," ACM Computing Surveys, volume 33, number 4, pp. 470-516.

Richard Jacques and Herman Savastano, 2001. "Remote vs. Local Usability Evaluation of Web Sites," In: J. Vanderdonckt, A. Blandford, and A. Derycke (editors). Proceedings of Joint AFIHM-BCS Conference on Human Computer Interaction (IHM-HCI'2001), volume 2. Toulouse: Cépaduès-Editions, pp. 91-92.

KDE Usability, 2002. KDE Usability Project, at http://usability.kde.org, accessed 28 November 2002.

Morten Kyng and Lars Mathiassen (editors), 1997. Computers and Design in Context. Cambridge, Mass.: MIT Press.

Josh Lerner and Jean Tirole, 2002. "Some Simple Economics of Open Source," Journal of Industrial Economics, volume 50, number 2, pp. 197-234.

Henry Lieberman and Christopher Fry, 2001. "Will Software Ever Work?" Communications of the ACM, volume 44, number 3, pp. 122-124.

Rohit Mahjan and Ben Shneiderman, 1997. "Visual and Text Consistency Checking Tools for Graphical User Interfaces," IEEE Transactions on Software Engineering, volume 23, number 11, pp. 722-735.

Stephen Manes, 2002. "Linux Gets Friendlier," Forbes (10 June), pp. 134-136.

Mozilla, 2002. Mozilla Project, at http://www.mozilla.org, accessed 28 November 2002.

Kamiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida, and Yunwen Ye , 2002. "Evolution Patterns of Open-Source Software Systems and Communities," Proceedings of the Workshop on Principles of Software Evolution, International Conference on Software Engineering. New York: ACM Press, pp. 76-85.

David M. Nichols, Kirsten Thomson, and Stuart A. Yeates, 2001. "Usability and Open Source Software Development," In: E. Kemp, C. Phillips, Kinshuck, and J. Haynes (editors). Proceedings of the Symposium on Computer Human Interaction. Palmerston North, New Zealand: SIGCHI New Zealand, pp. 49-54.

Seth Nickell, 2001. "Why GNOME Hackers Should Care about Usability," GNOME Usability Project, at http://developer.gnome.org/projects/gup/articles/why_care/, accessed 28 November 2002.

Jacob Nielsen, 1993. Usability Engineering. Boston: Academic Press.

Donald A. Norman and Stephen W. Draper (editors), 1986. User Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Tim O'Reilly, 1999. "Lessons from Open-Source Software Development," Communications of the ACM, volume 42, number 4, pp. 32-37.

Eric S. Raymond, 1999. "The revenge of the hackers," In: M. Stone, S. Ockman, and C. DiBona (editors). Open Sources: Voices from the Open Source Revolution. Sebastopol, Calif.: O'Reilly & Associates, pp. 207-219, and at http://www.oreilly.com/catalog/opensources/book/raymond2.html, accessed 28 November 2002.

Eric S. Raymond, 1998. "The Cathedral and the Bazaar," First Monday, volume 3, number 3 (March), at http://firstmonday.org/issues/issue3_3/raymond/, accessed 28 November 2002.

Eric S. Raymond and Guy L. Steele, 1991. The New Hacker's Dictionary. Cambridge, Mass.: MIT Press.

Walt Scacchi, 2002. "Understanding the Requirements for Developing Open Source Software Systems," IEE Proceedings — Software, volume 148, number 1, pp. 24-39.

Jean Scholtz, 2001. "Adaptation of Traditional Usability Testing Methods for Remote Testing," Proceedings of the 34th Annual Hawaii International Conference on System Sciences. New York: IEEE Computer Society Press.

Jean Scholtz, 1999. "A Case Study: Developing a Remote, Rapid, and Automated Usability Testing Methodology for On-Line Books," Proceedings of the 32nd Annual Hawaii International Conference on System Sciences. New York: IEEE Computer Society Press.

Ben Shneiderman, 2002. Leonardo's Laptop: Human Needs and New Computing Technologies. Cambridge, Mass.: MIT Press.

Elissa D. Smilowitz, Michael J. Darnell, and Alan E. Benson, 1994. "Are we Overlooking Some Usability Testing Methods? A Comparison of Lab, Beta, and Forum Tests," Behaviour & Information Technology, volume 13, numbers 1 & 2, pp. 183-190.

Suzanna Smith, Dave Engen, Andrea Mankoski, Nancy Frishberg, Nils Pedersen, and Calum Benson, 2001. "GNOME Usability Study Report," Sun GNOME Human Computer Interaction (HCI), SunMicrosystems, Inc., at http://developer.gnome.org/projects/gup/ut1_report/report_main.html, accessed 28 November 2002.

Bruce Sterling, 2002. "A Contrarian Position on Open Source." presented at the O'Reilly Open Source Convention, Sheraton San Diego Hotel and Marina (22-26 July), San Diego, Calif., at http://www.oreillynet.com/pub/a/network/2002/08/05/sterling.html, accessed 28 November 2002.

John C. Thomas and Wendy A. Kellogg, 1989. "Minimizing Ecological Gaps in Interface Design," IEEE Software, volume 6, number 1, pp. 77-86.

Matthew Thomas, 2002. "Why Free Software Usability Tends to Suck," at http://mpt.phrasewise.com/discuss/msgReader$173, accessed 28 November 2002.

Peter Thomas and Robert D. Macredie, 2002. "Introduction to the New Usability," ACM Transactions on Computer-Human Interaction, volume 9, number 2, pp. 69-73.

Jennifer A. Thompson and Robert C. Williges, 2000. "Web-based Collection of Critical Incidents during Remote Usability Evaluation," Proceedings of the 14th Triennial Congress of the International Ergonomics Association and the 44th Annual Meeting of the Human Factors and Ergonomics Society (IEA 2000/HFES 2000), Santa Monica: Human Factors and Ergonomics Society, volume 6, pp. 602-605.

Peter Trudelle, 2002. "Shall We Dance? Ten Lessons Learned from Netscape's Flirtation with Open Source UI Development," presented at the Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002), Minneapolis, Minn., at http://www.iol.ie/~calum/chi2002/peter_trudelle.txt, accessed 28 November 2002.

Tom Tullis, Stan Fleischman, Michelle McNulty, Carrie Cianchette, and Margaret Bergel, 2002. "An Empirical Comparison of Lab and Remote Usability Testing of Web Sites," presented at the Usability Professionals Association Conference (July), Orlando, Fla.

Hal R. Varian, 1993. "Economic Incentives in Software Design," Computational Economics, volume 6, numbers 3-4, pp. 201-217.

Marco A.A. Winckler, Carla M.D.S. Freitas, and José Valdeni de Lima, 1999. "Remote Usability Testing: A Case Study," In: J. Scott and B. Dalgarno (editors). Proceedings of the Conference of the Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia (OzCHI'99), Wagga Wagga, New South Wales, Australia, pp. 193-195.