X-Programmer

Usage-Centered software

非程序员

主编：davidqql
审稿：think, davidqql
版面：lanzhou
投稿： editor@umlchina.com
广告： adv@umlchina.com

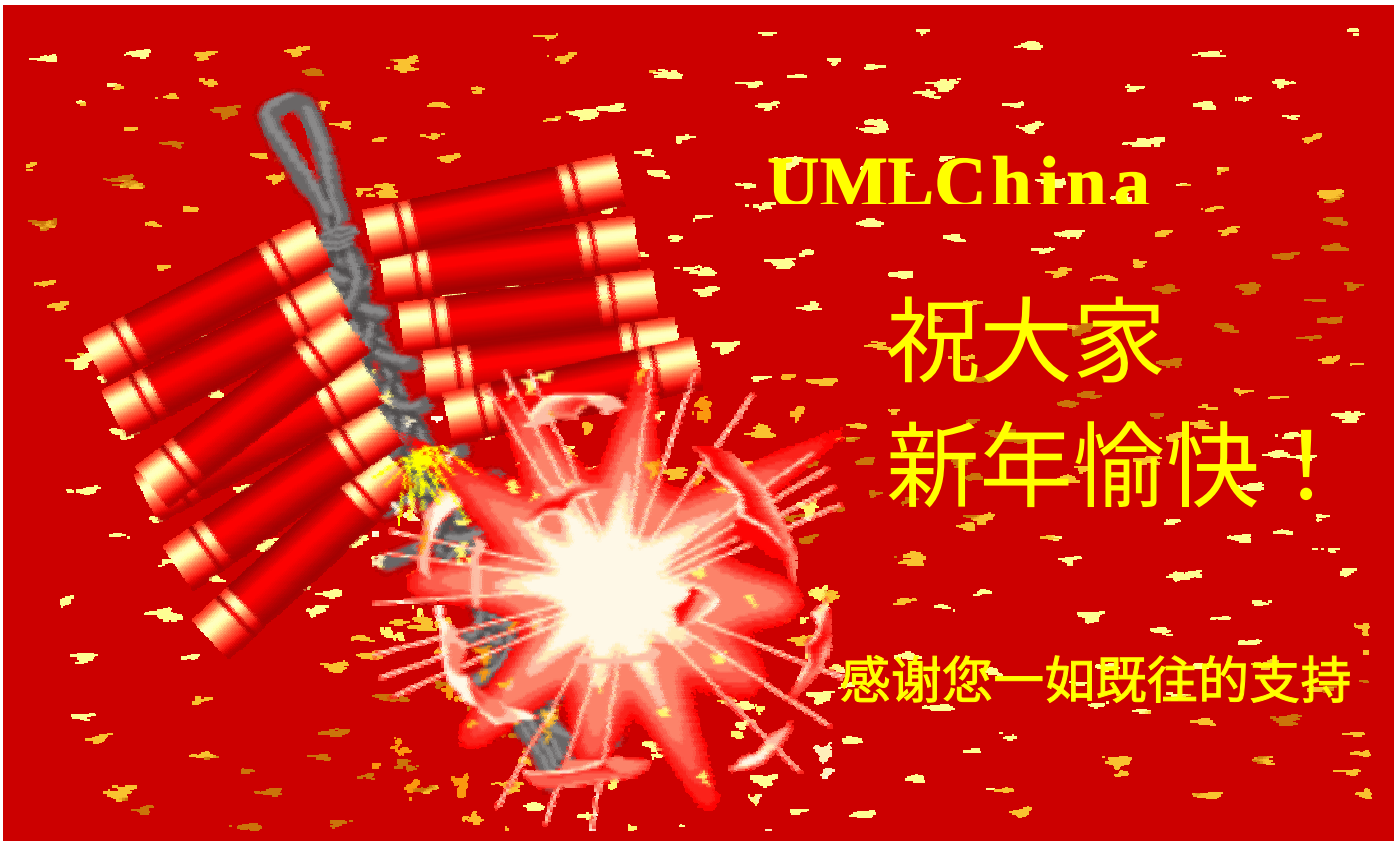
2002-2-10

UMLChina.com

征 稿

关于需求，设计，构造，测试，维护，配置管理，管理，过程，工具，质量...翻译或原创均可。（详细）

投稿请 EMAIL: editor@umlchina.com



UMLChina讨论组

昵 称： umlchina

密 码：

注册 忘记密码 µÇÂ¼

搜索UMLChina

ËÑË÷

最新>>

新增答疑专家 NEW

推荐《人月神话》 NEW

增加XP栏目

更新各栏目论坛精华

1月23日与Martin Fowler交流实录>>

最新招聘

广州某软件公司

北京软通动力科技有限公司

更多>>

导航

《非程序员》：UMLChina发行的软件工程杂志，下载量过万

杂志下载 征稿启事 NEW 来稿情况

专家解疑 ：世界级专家为您解疑，点击人名进入各专家的答疑板。提问请先知道 提问建议

Kent Beck Alistair Cockburn Mark Paulk Roger S. Pressman NEW John Vlissides 高焕堂 钱五哥 叶云文 .....

书籍推荐 ：UMLChina推荐的软件工程书籍

UMLChina推荐书籍 国内书籍信息 国外书籍样章

文档中心 ：大量软件工程资料下载

OO和UML 用例&需求 建模实例 NEW 模式 NEW 工具 过程XP和轻量级方法 组件 NEW 交互设计 测试 物件导向杂志 PMT评论

论坛精华：UMLChina讨论精华

行业动态 OO和UML 模式 工具 过程组件 交互设计 国内书籍 嘉宾交流

服务中心：

Architect编辑翻译组 OOAD培训，培训资料下载

Martin Fowler 访谈

编注：北京时间2002年1月23日上午9:30-11:30，Martin Fowler先生作客UMLCHINA讨论组的聊天室，畅谈XP。Martin Fowler在面向对象分析设计、UML、模式、软件开发方法学、XP、重构...方面，都是世界顶级的专家，现为ThoughtWorks首席科学家。Martin Fowler著有4本经典书籍：“Analysis Patterns：Reusable Object Models”、“UML Distilled: Applying the Standard Object Modeling Language”、“Refactoring: Improving the Design of Existing Code”、“Planning Extreme Programming”。以下是交流实录。由gigix翻译。原文链接。

关于测试框架

gigix：Fowler先生，您知道有什么C++的自动化测试框架吗？

fowler：我的一个同事 Bill Caputo 在我的网站上放了一篇关于自动化测试框架的文章。文章的地址是http://martinfowler.com/articles/ciWithCom.html。

gigix：用C++和用Java写一个测试框架，哪个更困难一些？

fowler：我发现，对于任何编程任务，用Java都比用C++要容易。现在我已经很少用C++了。

gigix：Fowler先生，我在编写一个C++的测试框架。您能否告诉我：在这个过程中，最重要的是什么？

fowler：你是在写测试框架还是应用程序的测试用例？

gigix：我在写一个名叫“CUnit”的测试框架。它看起来很象JUnit，但是用C++写成。您能给我一些建议吗？

fowler：当你把JUnit移植到别的语言中的时候，最重要的是遵循你自己语言的习惯用法，而不只是照搬Java中的设计。

gigix：我刚才看到Source Forge有一个叫“CppUnit”的项目，您知道吗？

fowler：CppUnit已经有一段时间了。我无法做什么评价，因为我已经好久不用C++了。

关于重构

tensile：请问“重构” 的精确含义是什么？

gigix：重构是一个改变软件系统的过程，它不会改变代码的外部行为，但是改善其内部结构。

fowler：对，这就是重构的定义。

gigix：我的一个朋友在重构的时候不使用测试框架，您认为这样做行吗？

fowler：只做重构不做测试是非常不明智的。重构而不测试，就好象走钢丝而没有安全网。

tomsmart：您好，Fowler 先生。请问我们在什么时候应该重构一个程序呢？

fowler：如果你想添加新特性或者修补 bug，而你的软件让你感觉有点难以下手，那么你就应该重构了。

extreme：您做了多少年程序员之后才意识到重构的重要性呢？

fowler：我从八十年代早期就开始编程了，而第一次接触重构是在 1995 年前后。

simoncn：Fowler 先生，请问在真实的项目中，应该在什么时候进行重构呢？

fowler：重构的关键是每天做一点。如果你使用的语言有相应的重构工具，那会对你有很大的帮助。

zhangxf：代码复审应该是怎样的？我们应该在复审的过程中重构吗？

fowler：代码复审中的重构会起到很好的作用。我在书中曾经说过：代码复审是重构的一个好时机。

viery：现在，重构的理论还不完整，所以它还需要更多人的经验。您是否能给我们一些指导？

fowler：别去等重构的理论了，现在就开始尝试着用吧。

mypine：重构应该由哪个部门来做？

fowler：每个程序员都应该重构。

mypine：我觉得重构很困难！不过我支持重构！

fowler：编程就很困难！重构只是任务的一部分，它能让你的程序变得更好。

mypine：结构上的重构是最重要的，是这样吗？

fowler：你说的“结构上的”是指什么？

mypine：请原谅，我的意思是说系统的结构（包括类体系及其关联）。

fowler：最重要的是要去掉重复代码。代码中的“坏味道”驱使你去重构，而不是你刻意去做哪些重构。

mypine：Fowler 先生，请问您一般选择什么时机开始重构？

fowler：每当我手上的代码开始散发出不好的味道时，我就开始重构。

mypine：“不好的味道”是指什么？

yhufo：请问我应该怎么发现源代码中不好的味道？

fowler：在《Refactoring》这本书里，我谈到了代码的坏味道。重复代码就是坏味道的典型。（编注：第 9 期《非程序员》里有文章“极端编程中‘坏气味’的发现与响应”）

gigix：“Bad Smell”是指不好的代码形式，在《Refactoring》里面列出了 21 种，首当其冲的就是大类、长方法、重复代码。“Bad Smell”标志着这部分代码需要重构。

Charity_Zhou：您是说，大类也是一种坏味道？但是在某些系统中，确实需要用许多小类来组成一个大类。

fowler：我没有把组合类看作大类——它是由许多小类组合在一起工作的。

gigix：“ 大类 ” 是指代码量过大的类，而不是规模大的类。

yhufo：为什么数据库应用程序的重构那么困难呢？

fowler：因为数据库通常没有封装，而且集成过程通常也不好。在重构的过程中你可能要搬运所有的数据，如果数据量很大的话，这也是很耗时间的。

yhufo：在 XP 里，我们用重构来改善代码。那么为什么不一开始就做好设计，从一开始就编写良好的代码呢？

fowler：预先做完整的设计实在太困难了，特别是在需求会发生变化的情况下。所以，你既需要预先设计，也需要重构。

yhufo：但是重构会造成更多的函数和子程序，从而使系统变慢。

fowler：大量的小类不会使系统变慢。通常，在现代虚拟机上，更多的小类会得到更好的性能。如果你的 profiler 告诉你需要把小类组合在一起，你就放心地这样去做。但是只有当 profiler 显示这样做有好处的时候，你才应该这样做。

wchenwu：我喜欢重构的方法，但是对我来说，它太难实践了。

fowler：找个学得比较好的人来教你。在重构的时候，首先关注代码的坏味道——特别是重复代码。

yhufo：请问您知道有什么 VB 的重构工具吗？

fowler：我还不知道有什么 VB 的重构工具。如果我看到了，我会贴在 refactoring.com 上的。

mypine：那么 Delphi 呢？有什么 Delphi 的重构工具吗？

fowler：也不知道。如果知道了，我也会贴在 refactoring.com 上面的。

shenborui：您能推荐一些 Java 的重构工具吗？

fowler：在 www.refactoring.com 上面，我开出了一张重构工具的列表，里面有好几种优秀的 Java 重构工具。目前我使用的是 IntelliJ（www.intellij.com），这是一个非常好的 IDE。

关于模式

timwap_cn：您好，Fowler 先生。请问在应用程序开发中使用模式很重要吗？

fowler：了解模式是很重要的，但是也并不是随时都要使用的。

timwap_cn：请问在模式和软件体系结构之间有什么关系？

fowler：模式是一种描述体系结构的方式。

timwap_cn：那么您能推荐一些有用的软件体系结构方面的网络资源吗？

fowler：我没有找到太多的软件体系结构方面的网站，当然我的网站上有一些东西——但是都是关于信息系统的。

gigix：Fowler 先生，您是否认为设计模式仍然有价值？它们是否鼓励过分设计？

fowler：毫无疑问，设计模式是好东西。你可能会过分使用它们，但是当你学会了正确的使用方法以后，你就能更好地利用它们。

extreme：关于设计模式，有什么好的网上资源吗？

fowler：http://hillside.net/

mypine：请问重构是模式的一部分吗？

fowler：重构与模式有关系。模式经常是重构的目标。

heey：请问在设计或编程的时候使用模式很重要吗？

fowler：对模式有一定的了解很重要。你不必随时使用模式，但是你应该知道。

yhufo：请问设计模式和分析模式之间有什么关系？

fowler：分析模式是用于系统领域建模的模式。它们会告诉你如何对某种常见的应用领域问题建模。设计模式则是关于如何保持设计的灵活性的。它们在软件的每个角落都有应用。

yhufo：是不是说分析模式是关于业务逻辑的，而设计模式是关于软件编程逻辑的？

fowler：这是一种不错的想法。

gigix：Fowler 先生，windy.j 非常喜欢您的《分析模式》。她也是第一个开始翻译这本书的人。

fowler：windy，谢谢你的翻译。

windy.j：很高兴在这里与您交流。我觉得您给我们的模式都很经典，可是有点难以把握。☺

fowler：所以我现在开始研究一些 ISA 的模式。也许我以后还会继续回来研究分析模式的。

yhufo：如果我们遇到了新的应用领域问题，我们应该如何使用分析模式来解决新问题呢？

fowler：看看是否有现成的分析模式与你的问题相关。如果有，就尽量使用现有的模式。

windy.j：请问，从业务的经验中获取模式，这是正确的方法吗？

fowler：我从别人的实践中获取模式。ISA 模式就来自于对 ThoughtWorks 公司的项目的观察。ISA 模式可以在我的网站上看到：www.martinfowler.com。

windy.j：新的模式给旧的模式增加了更多的内容，而且代码也是用 C++写的，真好。

fowler：新增加的这些模式更能反映我现在对分析模式的观点（但是代码实际上是用 Java 写的☺）。

关于 XP

simoncn：Fowler 先生您好，我发现在实际工作中很难实施成对编程（Pair Programming）。

fowler：你需要先找到愿意认真尝试这种方法的人。我没有觉得成对编程的开销是个多大的问题，只要人们认识到编程中最难的部分并不是敲键盘，他们就能接受这种方式。不过，人们往往需要先尝试一下成对编程，看它是否能给自己带来实际的利益。

zhangxf：您是否认为，在成对编程中，程序员的组合是至关重要的？包括程序员的经验、领域知识等等。

fowler：不同的组合都能生效。你应该根据任务的需要来选择组合形式。比如说，将两个新手组合在一起就会给他们带来困难。

extreme：我听说，如果真的要使用 XP，你就必须采用它建议的所有实践。这是真的吗，Fowler 先生？

fowler：如果要得到 XP 的效果，最好是采用所有的实践，至少一开始是这样。但是对于“ 持续集成 ”和“ 测试第一 ”，单独的实践也可以给你带来利益。

poirot：与某些重型过程相比，XP 是不是在美国更加流行？

fowler：XP 还是新事物，还很不成熟，但是它在不断成长。

wwabc：请问“ XP ”是什么意思？谢谢。

fowler：“ XP ”是“ 极限编程（eXtreme Programming）的缩写，请看 xProgramming.com。

poirot：在 XP 中，需求是不稳定的，软件总在变化。这是否意味着 XP 中必须有某种特别的机制来跟踪所有的变化？

fowler：简单的机制就是最好的，不然人们就必须花时间来学习它。

j2ee：在什么时候应该使用 XP 之外的其他方法，例如 RUP、TSP？为什么？

fowler：使用团队自己选择的方法。是团队选择了 XP，而不是你强加给他们的。

lzhihua：请问 RUP 和 XP 之间最大的区别在哪里？

fowler：RUP 是一个非常全面而复杂的过程框架。而 XP 更简单，并且是为某些类型的项目特别设计的。XP 非常强调纪律性，所以也更容易学习。

lzhihua：您能用一句话描述 XP 吗？就象对 RUP 的描述那样。

fowler：我不认为你真的能用一句话描述 RUP！我只能说：XP 是一种快速过程，它强烈关注技术性的实践。

lzhihua：RUP 这样说：体系结构第一、用例驱动、迭代开发。对吗？

fowler：没错。XP 也是以体系结构为中心的，但是它鼓励不断进化发展的体系结构。XP 的迭代性也强于 RUP，至少在实践中是这样。

lzhihua：您说 XP 是一个技术性的过程，那么 XP 中的管理过程又是怎样的呢？

fowler：XP 既有技术性过程，也有管理性过程，但是更多的细节偏向于技术。

yhufo：是不是在大型项目中应该使用 RUP，在小型项目中使用 XP？

fowler：不要在超过 20 人的任何项目中冒然使用 XP，除非你先在比较小的项目中得到了实际的经验。

j2ee：在美国，XP 方法得到软件公司的普遍认可了吗？

fowler：XP 还很新，还没有很多的认同者。

关于规范

tomsmart：当我们接到一个大项目的时候，我们是否应该更关注设计而不是编程？

fowler：我认为设计和编程是同一件事。

tomsmart：当然了，当我们把设计转换成程序的时候，我们会说它们都属于这整个项目。可是为什么您告诉我们它们都是同一件事呢？

fowler：在编程的时候，你总是要作出设计决策。在考虑设计思路的时候，你需要尝试用程序把设计表达出来。

timwap_cn：我在开始编程之前已经制定了计划。应该怎样跟踪它的变化呢？

fowler：不用跟踪，不值得。代码应该能够清楚地表达自己，根本不应该再保留额外的文档。只有值得写的文档才应该存在。

zhangxf：我认为，在大型项目中，文档还是很重要的，它是交流的载体。请问您怎么看？

fowler：绝对不要忘记：代码是你最重要的文档，其他的文档都只是代码之上的扩展，代码才是基础。可是很多人写的代码不够清楚。这样你有再多的文档也没用。

zhangxf：可是，问题是并非每个人都会去看代码。

fowler：任何搞技术的人都应该去看代码。

zhangxf：在某些项目中阅读代码是可行的。但是在我现在的项目中，有 2000 多个用例、5000 多个类，通过阅读代码来理解就不太可行，而且也太浪费时间。

fowler：在这种情况下，就应该用图和文档来表现代码。但是细节问题仍然要通过代码本身来表现。

zhangxf：当然。但是 XP 的某些部分很容易被误解，甚至引起错误的做法。确定何时写文档、何时不写，这是关键，但也是非常灵活的。

fowler：XP 并不是反对文档，而是只有当文档可以增加价值的时候才记录文档。只有 XP 的批评者才会说“ XP 里没有文档 ”。

timwap_cn：在项目开始的时候，我总是想得到一个好的设计结构，这没什么错吧？但是有时候，我不知道应该怎么去完成。在项目开始之后，我发现项目的过程完全一团糟。

fowler：你应该在开始编程之前先草拟一个设计，但是随着你对系统了解的加深，不要害怕修改它。

extreme：Fowler 先生，您是否有过这样的体验：您认为自己有一个很好的计划，但最后却陷入了混乱？请问应该如何避免这种情况？

fowler：最重要的就是要经常对计划作出评价，并且随时更新你的计划。计划绝对不应该是定死不变的东西。计划最好是一个理解变化的结果的平台。

j2ee：有两种编写代码的风格：“ 先编码、再复审 ”；“ 编写一点、测试一点 ”。请问您怎么选择？

fowler：我当然选择 “ 编写一点、测试一点 ”。实际上，我最喜欢的是 “ 测试一点、编写一点 ” ☺并且随时做设计。

gigix:首先编写测试，测试不能通过，然后再写代码，让测试通过。这是 Fowler 先生比较喜欢的开发过程

j2ee：但是有人告诉我：复审比测试更好。

fowler：复审也很好。但是小步的、渐进式的测试可以极大地改善代码的质量。很多人把复审称为 “ 大量编码，然后测试 ”。小型的编码/测试循环就完全不同了。

tensile：如何改进代码的质量？

fowler：要改进质量，很简单的方法就是发现并去除重复代码。

mypine：您用于测试的时间和编码的时间是个什么比例？

fowler：基本相等。

kenxia：您不认为设计比测试更重要吗？

fowler：“ 测试第一的设计 ” 也是一种设计技术。它让你认真考虑自己的接口，并且鼓励你对代码中的坏味道进行重构。

关于 CMM

extreme：我听说很多印度公司通过了 CMM 4 级认证，我想知道这是否有意义。请问您怎么看，Fowler 先生？

fowler：CMM 可以帮助你确定某些东西，但是别把它当成某种 “ 级别 ” 来看。

poirot：Fowler 先生，您认为 XP 是一个反 CMM 的过程吗？

fowler：敏捷社群的许多人都把 CMM 看作与 XP 这样的敏捷方法完全不同的东西。

extreme：您刚才说的 “ 敏捷社群 ” 是指的什么人？

fowler：“ 敏捷社群 ” 是指那些对 XP、Scrum、FDD 之类的敏捷方法感兴趣的人。

tensile：如何用 XP 来提高效率？

fowler：哪方面的效率？

tensile：公司和员工的效率。

fowler：我们发现 XP 的实践大大提高了我们的效率。

tensile：在我们公司里员工的效率比较低，所以我们公司想实施 CMM 或其他方法。

fowler：如果情况的确很糟，CMM 也是有帮助的。不过公司应该集中精力去改进过程，而不是想着达到哪个级别。

extreme：但是 CMM 似乎太复杂了。

fowler：CMM 并非一定要变得复杂的，这取决于你怎么实施它。通常是实施的方法让它变得复杂的。

gigix：但是 70%的中国软件公司都只有不到 50 个人。对于他们来说，CMM 是正确的选择吗？

fowler：我绝对不会从 CMM 入手。但是对于一个已经很糟糕的公司来说，实施 CMM 总比什么都不干要好。

lzhihua：我希望为我们公司引入 XP，但是看起来它对开发者的要求太高了。

fowler：XP 需要开发者要求应用它，并且有热情去学习它。

poirot：您是否认为 XP 能帮助软件公司达到较高的 CMM 级别？

fowler：请看 http://www.xpuniverse.com/XPandCMM.pdf。

关于著作

tomsmart：您读过《Code Complete》这本书吗？我最近读到了。尽管是 1993 年出版的，但是我还是觉得它对我非常有帮助。

fowler：那的确是一本精彩的书。

rantaiqi：Fowler 先生，请问您能在中国出版您的《UML Distilled》吗？

fowler：我相信《UML Distilled》已经有中译本了，至少我听说是这样。

gigix：您能为中国的开发者推荐一篇您的文章吗？我将很高兴为您翻译。

fowler：最适合翻译成中文的也许就是《持续集成》。地址是 http://martinfowler.com/articles/continuousIntegration.html。我的很多文章都被翻译成了日文。日文和中文很接近吗？还是有很大的差异？

gigix：有非常非常大的差异。99%的中国人都看不懂日文。

simoncn：就象英语和西班牙语的差异。

fowler：谢谢你们让我知道这一点。gigix 要翻译，没问题，我只要你保留我的版权声明和到原文的连接。

yhufo：在中国我没有找到您的《Refactoring》，但是我真的很想看。您能给我一份电子书吗？

fowler：很遗憾，我也没有电子书。不过我相信你在中国也能找到这本书。这里的一部分人很明显是看过这本书的。

extreme：Fowler 先生，您能告诉我们一些您写那本著名的《Refactoring》过程中的事情吗？

fowler：我发现那些比我更渊博的人都忙着干别的事去了，所以只好决定自己来写这本书了。

关于 UML

extreme：那么，您通常用什么 UML 工具呢？

fowler：我绝大多数的 UML 图都是用 Visio 画的，使用 Hruby 的模板。

extreme：为什么您不用 Rose 呢？那可是 UML 工具中公认的 No. 1 呀。

fowler：Rose 太贵了，而且也不太符合我的习惯。所以我很少用。

extreme：那么，在美国最流行的 UML 工具是什么？

fowler：我不知道最流行的 UML 工具是什么，而且我对这个答案也不感兴趣 ;-)

choupixiong：您好，Fowler 先生。我在学习 UML，但是没有机会应用，请问我应该怎么办？

fowler：尽量尝试在你自己的问题里面画 UML 图，看看它是否能帮你观察自己的软件。

mypine：您怎么看待 UML？我觉得 UML 不是自包含的，所以象您这样的前辈应该再对他做些修改，不是吗？

fowler：我觉得 UML 已经够紧密的了，而且我也没有兴趣再去修改它。现在，我更喜欢研究模式的问题。

wwabc：Fowler 先生，如果 UML 推出新版本，那么会有什么新内容呢？

fowler：UML 最近不会出什么新版本（至少我不知道）。

关于过程和方法

extreme：您怎么看待 RUP？它太复杂了吗？

fowler：我想 RUP 的确太复杂了。它最大的问题就是它什么都想管。

fl_xyg：请问在设计模型和用例之间有什么关联？

fowler：不必过分操心模型的问题。对于有些编程任务，用例就可以形成程序的结构。用例可以用不同的方法实现，类图、交互图、CRC 卡……只要对你的团队有利，就去用。在为期一个月的迭代周期里面，两到三天的时间画图通常就够了。

fl_xyg：对不起，您刚才说“ 为期一个月的迭代周期里面，两到三天的时间画图通常就够了 ”，您能详细说明一下吗？

fowler：如果你这样做，那么对于一个月的迭代周期，大概两天的时间画图就很合适。你可以在整个迭代过程中逐步进行设计。XP 就推荐这样逐步设计。

timwap_cn：在软件设计和项目开发中有许多概念，请问应该怎样为团队选择正确的方法？

fowler：敏捷方法是我的首选，犯一些轻量级的错误总比犯一个巨大的错误要好。记住：软件开发的目标是要给用户提供有价值的软件。随时都要记住这一点。在开发过程中，要随时确保软件的高质量，这样你就能以适当的节奏发布可用的软件。

wwabc：在项目中创建业务用例和用例是必须的吗？请问它们之间有什么关系？

fowler：用例很有用，但是通常没有必要同时使用这两类用例。如果你想要更完整（也更好）的答案，我推荐 Cockburn 的书。

Charity_Zhou：我在把对象映射到关系数据库管理系统时遇到了一些困难。一个类对应一个表，完全面向对象，但是性能却又成了一个问题。

fowler：关于对象到数据库的映射，请看 http://martinfowler.com/isa/index.html。

zhangxf：您怎么看待 XML 数据库？它与 OODB 很相似，而且我也看到了一些真正的 XML 数据库。

fowler：XML 数据库是一个非常有趣的想法。对于很多问题，它们也许真是不错的解决方案。我们应该关注。

simoncn：从对象世界的设计到 RDBMS 的映射（以及反向映射），请问您的意见？

fowler：我喜欢先得到一个概念模型，然后从它派生出对象和数据库。如果问题领域逻辑比较复杂，就应该由对象模型来驱动。如果应用程序非常简单，你也可以不要对象模型。

serenaday：您认为哪种团队模式是最好的？

fowler：你说的“ 团队模式 ”是指什么？

serenaday：我是说团队的创建。

fowler：团队创建可是个很大的话题了，没有什么绝对的答案。我喜欢让团队找到自己的方法。

cyberp2p：我们大家都知道软件工程的重要性，但是一到了实际工作中，许多团队都陷入了混乱。我们要怎么解决这个问题？

fowler：你必须尽量提高团队的纪律性，慢慢来。

serenaday：我们的团队比较小，只有 4 个人。您能给我们一些建议吗？

fowler：你想要什么建议？四个人的团队非常合适。

extreme：请问您现在还编码吗，Fowler 先生？

fowler：最近我已经不太写代码了，而且写的代码也大半都是用在书中的例子。

mypine：请问您对全局方法怎么看？

fowler：很少需要全局方法。但是与其他东西一样，有时候它们也是正确的选择。一本好的 C++书应该会谈到这个问题，可惜我已经不用 C++了。在 Java 或 C#中没有那种“自由的函数”。

mypine：但是 Java 中有 static 关键字，那不就是全局吗？

fowler：Java 中的 static 和 C++的 static 是一样的——那是一个类方法。

extreme：通过 static 方法，我们在 Java 中也可以得到“全局”的函数，就象在 C++中一样。

fowler：这也是一种常见的组织方式，完全取决于你怎么使用它们。但是，static 方法总是属于某一个类的。

deborah.d：请问我怎么跟踪业务用例模型和用例模型之间的关联？

fowler：不用去做详细的跟踪，不值得。我建议你看看 Cockburn 的《Writing Effective Use Cases》。

serenaday：您认为哪种版本控制工具更好用一些？

fowler：我对版本控制工具没有什么特别的看法。不过 Subversion 看上去很不错。

simoncn：您怎么看待 CRC 卡？我觉得它有时候会引导我向过程设计的方向去思考，而且也没有什么好的工具支持。

fowler：CRC 卡是揭示对象之间交互的好技术。

xuxu1976：Fowler 先生，您怎样看待组件？

fowler：组件很有用，但是很难设计得好，特别是对于那些商业软件。

trybird：请问分析类图和设计类图之间有什么差异？

fowler：很遗憾，不同的人对这些词的用法也不同，所以对于你的问题，我很难给出一个唯一的答案。

serenaday：您怎么看待微软的.NET？

fowler：.NET 是很好的技术。它和 Java 将是未来的主流。但是很多开放源代码的脚本工具也很好（例如 Python 和 Ruby）。

zhangxf：您怎么看待 Web Service 以及工作流与 Web Service 的集成？

fowler：现在我们已经针对 Web Service 做了许多有意义的工作，我希望能写一本关于这个主题的书。

simoncn：请问您对来自 OMG 的 MDA 有什么看法？

fowler：MDA 是另一种编程语言，同时还包括图表。要让一种编程语言获得成功是很困难的。

gigix：您知道敏捷建模吗？您认为 UML 在敏捷过程中有什么样的地位？

fowler：我一直都在关注 Scott Ambler 的研究。我觉得敏捷建模是个很好的想法。

gigix：Fowler 先生，请问您对“ 软件体系结构 ”这个词有什么想法？

fowler：体系结构通常是指大尺度的设计问题。

unimap：Fowler 先生，请问我在项目中应该怎样做软件测试呢？

fowler：用 JUnit 之类的测试框架。

poirot：Fowler 先生，您推荐在编码之前先编写测试用例，这是不是 UML 没有涉及到的一种针对功能的语意规范？

fowler：没错，这是一种很好的想法。这也是“ 按照合同设计 ”的好例子。

fowler：非常感谢各位，今天聊得很开心。希望以后能在中国见到各位。

tensile：谢谢。

extreme：谢谢 Fowler 先生。

文档中心

所在位置： 文档中心

论坛精华

文档更新：模式(2/3)，组件(2/3)，建模实例(2/3)...

资 料	介 绍
行业动态	欢迎提供软工新闻，UMLChina最近活动
OO和UML	综述、Use Cases&需求
建模实例	实例教程，.mdl文件，文档实例....
模式	Patterns
工具	Rose...等各种软件工程工具介绍
过程	XP，RUP，CMM...
组件	COM，CORBA，EJB，N-Tier...
交互设计	人机界面的工程学
测试	
书籍	国内出版信息、英文书籍样章
OOTW	物件导向杂志（台湾）简体版
PMT评论	关注软件质量的电子杂志

下载说明

如出现“403...Forbidden字样”，那是因为由于服务商设置原因，zip文件同一时间只能一个IP下载，所以，请多试几次，或者用网络蚂蚁跟踪！我们也会争取在空间允许的情况下，把所有文件以pdf方式上载。

下载pdf文件打开后如果有CMap..错误。这是因为Arcobat Reade未安装中文字体，请下载 中文字体包 或安装中文版Reader 。

其它辅助工具：

PS阅读器，GSView。

缩印工具inePrint 可以把多页打印在一页上，最多可达32:1。

返璞归真：
通过简化用例来简化用户界面

Larry Constantine 著 harvey 译

我们常被问及精简那些最简化、抽象和通用窗体用例的重要性。到底有多重要呢？在以用户为中心的设计中，简化那些重要窗体的用例是获得成功的关键。它能够为开发者设计优秀的用户界面助一臂之力。通过消除不必要的或技术驱动的操作步骤，设计者可以使用户界面上那些最常用或最基本的操作变得简捷。

最近，在我们一家欧洲客户的培训课上有一个十分清晰的例子。问题涉及一个为“捶击者(thumper)2000”设计的触摸屏控制面板，“捶击者(thumper)2000”是一种测试工业原材料的机器，可以把被测试物件击碎（在这里我们不讨论那些无关的细节）。一个重要的用户角色被称为“标准化测试员（standardized Test Runner）”，基本上，一个机器操作员很少了解所测试的材料和测试本身，但是他能够机械地设置机器，初始化和监视一系列预定义的测试过程。在某些特殊的情况下，这类操作员可能会对标准测试的规则作一些小的调整。

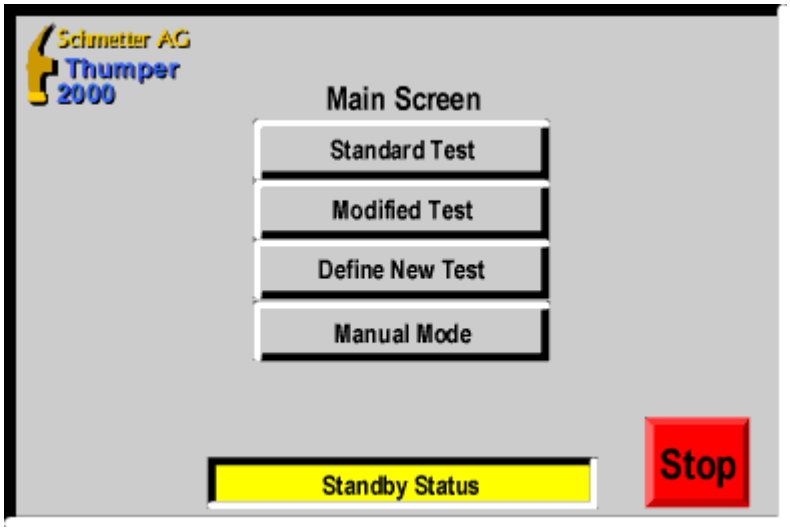
针对“标准化测试员”这个角色，有一个我们称为“执行预定义测试(running predefined test)”的重要用例。对这个关键用例可能的描述如下所示(左边第三步指明了一个用户发起（user-initiated）的可选扩展操作，这是另外定义的一个用例)。

执行预定义测试	
用户意图	系统响应
请求标准化测试	显示标准化测试列表
选择期望的测试	显示所选测试设置
可选项： [->调整测试设置]	机器为所选测试作准备 确认准备就绪
开始测试	执行测试 报告结果

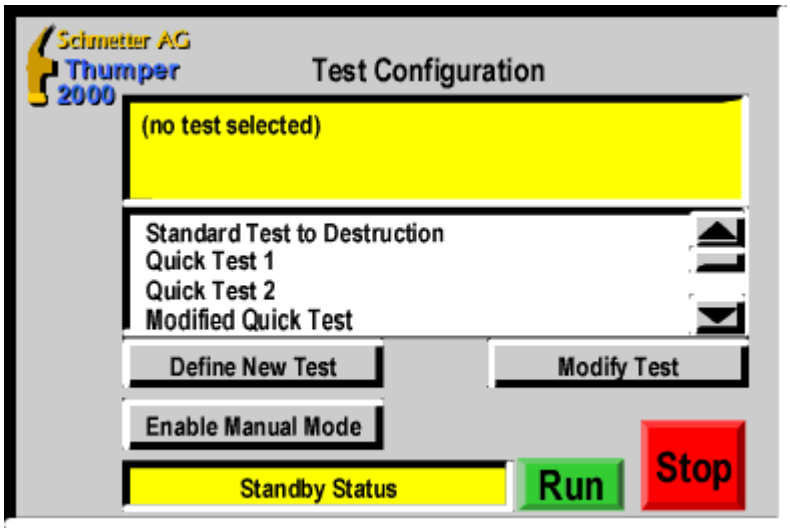
基本用例（Essential use cases）既可以开始于用户意图，也可以开始于系统响应，训练课上的一个设计小组指出，取消用户所做的第一步操作能够使这个用例更加简洁，这样的用例如下表所示。

执行预定义测试	
用户意图	系统响应
选择期望的测试	显示标准化测试列表
可选项： [->调整测试设置]	显示所选测试设置
	机器为所选测试作准备 确认准备就绪
开始测试	执行测试 报告结果

这里所争论的是：一个标准化测试员可能会忽略其他操作或者对其他操作不感兴趣，但为什么在开始某项操作前总是必须告诉系统是针对那一项测试？得出较复杂版本用例的小组是沿着表格右边那条线索得到的主界面原型。这是一种十分常见的用户界面窗体：开始于一整套按钮让用户在开始工作之前选择要执行的操作任务。与许多平庸的常规设计一样，它把额外的步骤强加给每一个用户。



简单一些的用例反映了一个不同结构的用户界面。使用简化用例的小组，设计了一个把标准化测试列表置于顶级主界面的原型。这样的设计源于表格左边那条线索，它允许直接执行那些最常用的任务而不用切换屏幕。



当然，不应该天真地把这里的想法应用到每一件事情上。如果走到一个荒谬的极端，只是为了简化用户操作步骤而把用户交互控制在一个界面显示，那将会把所有操作平铺在一个屏幕或对话框上。任何设计都有一个权衡的问题，窍门是决定在哪里划一条界线。在这个简单的实例中，这条线画在对最简单的使用所做的最简单用例旁边。更复杂的情况，比如定义新标准化测试（defining new standard test），隐藏在另外一个层次，这是相当合理的。

对于相对较小的设计问题，不管用例是否精简，也许一个经验丰富的开发者仍然可以开发出最简化的操作。然而，当用例数量的增长对系统的影响越来越大时，将用例简化到最精炼程度的优点就变得实实在在了。

(Larry Constantine, January 1999)

所在位置： OO/UML -论坛精华

论坛精华

对于模式的“十大误解”

Gigix 译文专栏

John Vlissides 著，透明 译

译者语：现在“模式”这个词真是非常流行。就象任何流行的东西一样，对它的误解也真是不少。甚至在一些发表出来的文章中，也存在着各种各样的误解，我想这会对读者造成非常糟糕的引导作用。早已想写一篇文章来澄清一些对模式的误解，却又因为水平所限难以成文。恰在此时，我看到 John Vlissides 先生的《十大误解》，于是我便乐得当文抄公了。

关于设计模式，下面有十种错误的观点——很多都是很流行的观点。且看 Vlissides 先生如何拨开这些迷雾。

最近，围绕着模式的讨论日嚣尘上，人们对模式的混淆、惊惶和以讹传讹已经不是一点半点。这也从一个侧面反映出对于主流软件开发者来说，模式是一个多么新鲜的领域——尽管严格说来，它并不是一个新的领域。同时，模式也是一个飞速发展的领域，留下了大片的空白。而且，没错，我们这些模式的支持者也应该受批评，因为我们没有把自己的知识完完全全地教给别人。尽管我们这样想过，但是却没有这样做[BMR+96，Coplien96，CS95，GoF95，MRB98，VCK96]。

因此，我觉得自己有责任来纠正一些模式方面越来越夸张的误解——我常常听到的那些误解都足以形成自己的模式了。甚至连我也曾经想以模式来组成自己的软件结构……后来我才明白：“把所有的东西都变成模式”，这种想法本身就是错误的！总之，请记住，这篇文章不是在给模式社群的人们看的。我想，绝大多数的模式专家都会同意：下面这些都是很常见的误解。但是他们也许不会同意我消除这些误解的方式。

这几年以来，我听到过许多人谈论模式。把听到的这些东西整理一下，对模式的误解大概有三类：对于“模式是什么”的误解，对于“模式可以做什么”的误解，以及对于支持模式的人群的误解。我的“十大误解”都落入这三类之中，所以我就把这些误解都组织起来。首先，我们来看看关于“模式是什么”的误解。

误解之一：“模式是某种场景下某个问题的解决方案”

这个定义来自于 Christopher Alexander[AIS+77]，所以如果把它称为“误解”，也许有人会把我视为异端。但是下面这个简单的反例能让你看清它的缺点：

问题：我获奖的奖券就快过期了，我怎样拿到奖金？

场景：离截止时间只有一小时，可是我家的小狗把奖券给吞了。

解决方案：把狗开膛破肚，掏出奖券，然后飞奔到最近的兑奖地点。

这是“某种场景下某个问题的解决方案”，但它不是模式。还缺了什么？至少缺三样东西：

1. 可重复性。解决方案应该对应于外部的场景。
2. 可传授性。一个解决方案应该可以移植到问题的不同情况上。（绝大多数模式的可传授性都建立在“约束”和“效果”的基础上。）
3. 用来表示这个模式的名称。

确实，很难找到一个令人满意的定义。“模式讨论”邮件列表（patterns-discussion@cs.uiuc.edu）上正在进行的讨论也证明了这一点。难以定义的一大原因是：模式既是一个事物，也是对类似事物的描述。要区分这两者，有一种办法：“模式”这个术语只用来指代模式的描述，同时用“模式实例”来指代模式的具体应用。

但是，术语的定义很可能是白费力气，因为一个定义可能对一个人（比如程序员）有意义，但是对另一个人（比如只能看到书面材料的项目主管）却毫无意义。当然，我不打算在这里提出什么最终定义。但是，任何对模式要素的规定，除了必须包括问题、解决方案和场景之外，都必须提及可重复性、可传授性和名称。

误解之二：“模式就是行话、规则、编程技巧、数据结构……”

我通常把这些误解统称为“蔑视”。如果你试图把某些不熟悉的东西简化为熟悉的东西，产生这种想法是很自然的，尤其是当你没有特别的兴趣去钻研这些不熟悉的东西时。另外，某些人经常拿新瓶装陈酒，然后大吹“创新”、“革命”一类的口号。保持警惕也是好的。

但是，这种蔑视通常不是来自亲身体验，而是来自肤浅的认识和一点点冷嘲热讽。而且，没有什么东西是真正“全新”的。人们的脑海中一直都存在着各种各样的模式，只不过我们现在刚开始为模式命名，并将模式记录下来。

来逐个说明这些看法：的确存在着模式的行话，例如“模式”这个词，例如“约束”，例如 Alexander 的“无名质量”，等等。但是模式是不能简化为行话的。与其他计算机科学领域相比，模式引入的新术语实在是少得可怜。对于听众来说，好的模式本来就很容易接受。在说明一个模式的时候也许有必要引用问题领域的行话，但是几乎没有必要使用什么特定于模式领域的术语。

没有哪个模式是能让你不假思索就使用的规则（组件则正好相反），同时模式也不仅仅是“编程技巧”，尽管模式中的“方言（idiom）”部分是特定于语言的。在我听来，“技巧”这个词带有轻蔑的意思，并且过分强调解决方案，而忽视了问题、场景和名称的重要性。

毫无疑问，你也曾经听说过一次创新被接受的三个阶段：首先，它被当作垃圾扔在一边；然后，人们会认为它不具可实施性；最后，大家都明白它的意义时，它也没有意义了——“我们一直都这样做”。现在，模式还没有完全走出第一个阶段。

误解之三：“理解一个，就理解了全部”

一刀切的规则往往是不公平的，而人们对模式却更加一刀切。模式的领域、内容、范围、形式、质量……这个领域大得吓人，只需要随手翻翻 PLoP 的书[CS95, MRB98, VCK96]就能看出。不同的人写出不同的模式，而模式的变化甚至比作者还要多。象 Alistair Cockburn、Jim Coplien、Neil Harrison 和 Ralph Johnson 这样的作者已经超越了最初大量记录不同领域、不同形式模式的阶段。只看几个例子就对模式做一个笼统的结论，这样的结论必定是错误的。

误解之四：“模式需要有工具或方法学的支持才会有效”

在过去的五年中，我记录、使用并且帮助其他人使用模式，还至少帮助完成了一个基于模式的工具[BFV+96]，所以我可以很有把握的说：模式的绝大多数好处都来自于原封不动的使用——也就是说，不需要任何形式的支持。

在谈到这个主题的时候，我通常会指出模式带来的四大好处：

1. 它们记录了专家的经验，并且让非专家也能理解。
2. 它们的名称构成了一份词汇表，帮助开发者更好的交流。
3. 它们帮助人们更快的理解一个系统——只要这个系统是用模式的方式描述的。
4. 它们使系统的重组变得更容易，不管原来的系统是否以模式的方式设计的。

过去，我一直认为第一项是模式最大的好处。现在我认识到，第二条起码也同样重要。请想一想，在一个开发项目的过程中，有多少字节的信息在开发者之间流动（包括口头的和电子的）？我猜就算没有 1G 也有好几兆。（在推出了《设计模式》之后，我已经收到了好几十兆给 GoF 的电子邮件。而且我们所描述的还都是小型到中型的软件开发项目。）由于有如此之大的信息交流量，所以效率上任何微小的提升都能大量节约时间。在这个意义上，模式拓宽了人们交流的带宽。我对第三、四条的评价也在逐渐提高，特别是在项目越来越大、软件生存周期越来越长的今天。

至少在短期内，模式主要存在于大脑中，而不存在于工具中。如果有了方法学或自动化工具的支持，应该还有其他的收益，但是我相信这些都只是蛋糕上的奶油，而不是蛋糕本身，甚至都不能算蛋糕的一层。

****

到目前为止，我说到的误解都是关于“模式是什么”的。现在来看看关于“模式可以做什么”的误解。这里有两种不同的风格：夸大其词的和心存疑虑的。

误解之五：“模式可以保证软件的复用性、更高的生产力、世界的和平……”

道理很简单，模式什么都不保证。它们甚至有可能无法给你带来利益。在创造新事物的过程中，模式无法取代人的位置。它们只是带来一种希望，有可能让一个缺乏经验的、甚至是未入门的，但是有能力、有创造性的人尽快获得设计的能力。

人们经常说：好的模式会让读者有“啊！”的回应。实际上，只有当模式恰好拨动了读者的某一根心弦时，他们才会有这样的反应。如果没有，模式就只象森林里普通的一棵树。因此，什么是好的模式？不管它写得有多好，如果不能引起读者的共鸣，它又怎能算是好的模式？

模式只是开发者的工具箱中的另一件工具，对模式寄以过高的期望只会适得其反。准备充分，然后做最坏的打算——这就是防止受骗、消除对抗最好的方法。

误解之六：“模式可以‘生成’整个软件体系”

这个误解与上一个有点相似，不过侵略性稍微少些。

每过一段时间，模式论坛上就会讨论一次模式的生成能力。按照我的理解，“生成能力”是指模式创建最终行为的能力。很多人认为，模式可以帮助读者解决模式本身没有明确提出的问题。我读过的一些书里也有同样的观点。

对于我来说，“生成能力”的关键是模式的可传授性——约束及其解决，或者对于效果的讨论。在你定义、精炼一个体系结构时，这些观察是特别有用的。但是模式本身并不制造任何东西——任何东西都是

由人来制造的。而且，模式不可能覆盖体系结构的每个方面。给我任何一个有实际价值的设计，我都能找出其中模式没有涉及的设计问题。也许这些问题不常见、不具可重复性，或者只是还没有以模式的形式记录下来。不管怎样，你都必须负责填补模式之间的空白——用你自己的创造性。

误解之七：“模式是针对（面向对象）设计或实现的”

另一个极端则是过分的限制，就象这个误解。坦白说，任何人都完全可能相信它，我不会有丝毫惊讶。无数的人问过我这个问题，所以它也进入了“十大误解”之列。如果你觉得这种观点实在天真幼稚，跳过这一部分吧。

如果模式不能记述专家的经验，那它们就什么都不是。对于模式的作者来说，任何形式的专家经验都是可记载的。当然，在面向对象软件设计中有值得记载的经验，但是在非面向对象设计和分析、维护、测试、文档、组织结构……这些方面也同样有值得记载的经验。现在，不同领域中的模式开始浮出水面。至少已经有两本关于分析模式的书[Fowler97, Hay96]，并且每次 PLoP 大会都会收到新的模式类型。（特别有趣的是 1996 年的 PLoP 大会甚至关注了音乐作曲的模式！）

和大多数的误解一样，这个误解也是有原因的。看看人们用来描述模式的格式，有两种基本的风格：描述高层结构的 GoF 风格（用于《设计模式》中）；接近文学的 Christopher Alexander 风格——叙述性的，尽量少的结构图。由于已经用模式描述了面向对象设计之外的东西，所以我现在认识到 GoF 风格造成的偏差。对于我研究过的某些领域的专家经验，这种风格根本无法描述。为什么结构图看上去总是让人联想到 C++？对于音乐作曲的模式，“实现”应该是什么？“协作”部分真的有意义吗？

很明显，一种格式不能适应所有的需求。比较具有普遍意义的是模式的概念：模式是记载并传达专家经验的工具——不论是哪个领域的专家经验。

误解之八：“没有证据表明模式帮助过任何人”

这种观点曾经有一定的说服力，但是现在已经没有了。在 Software-Practice and Experience [Kotula96]这样的杂志上、在 OOPSLA[HJE95, Schmid95]和 ICSE[BCC+96]这样的会议上，人们不断的报告自己从模式得到的利益。Doug Schmidt 已经清楚的说明了在传授计算机科学时使用模式的收益[PD96]。尽管绝大多数的证据都只是定性的，但是据我所知，至少有一个组织得到了一些定量的结论[Prechelt97, PUS97]。

随着时间的推进，我们需要更好地掌握和使用模式的好处和缺点。尽管最初的反馈已经让我们看到了希望，但是我们还需要更多的实践经验来做全面的估计。但是，如果仅仅因为模式的好处还没有完全证实就拒绝使用模式，那你就真的是个大傻瓜了。

****

关于“模式能干什么”的谬论还真不少。下面，最后两种误解不是关于模式本身，而是关于支持使用模式的人们的。

误解之九：“模式社群是精英的小圈子”

我很想知道这种观念到底是从哪里来的。如果说模式社群的人们有什么引人注目的地方，那就是他们之间巨大的差异。从 PLoP 大会的与会者名单就能看出——人们来自世界各地，有大公司的也有小公司的，有分析员、设计者和实现者，有学生也有教授，有大名鼎鼎的作者也有初出茅庐的菜鸟。更让我惊讶的是，竟然有不少与会者都不是计算机科学工作者！这个社群仍然在不停变化，每年的与会者名单都与前一年不同。

如果将模式社群的背景公开出来，别人也许会觉得奇怪：很少有学院派人士。实际上，绝大多数 PLoP 的与会者都是实践者。软件模式早期的推崇者——包括 Kent Beck、Peter Coad 和 Ward Cunningham——都没有学院背景。GoF 中只有一个人（Ralph）是学院派，而且他也是我所认识的最实际的学院派。很明显，模式社群从根本上反对任何可能的宗派主义和精英论。

误解之十：“模式社群是自私的，甚至是阴谋家”

我不止一次的听人指责说：模式最大的用处就是为写模式书籍的人带来了源源不断的收入。他们甚至还暗示模式的发展有着某种邪恶的目的。

胡说八道！

作为 GoF 的一员，我可以肯定地告诉你：对于《设计模式》引起的反响，我们和其他任何人一样吃惊。对于它在 OOPSLA 94 上的初次亮相引起的暴风骤雨，我们也同样没有准备——甚至出版商都没有预料到会有这么大的销量。在整个写作过程中，我们最关心的就是尽量写出一本高质量的书，至于销售上的问题，我们根本没有时间去想。现在“模式”已经成了一个时髦的词汇，不可避免会有一些人将这个词用来谋取私利。但是，如果你认真读过顶尖的模式作者的作品，你就会感觉到他们共同的目标：将难以获取的经验、最佳的实践、甚至是竞争中的优势——多年实践经验的累积——分享给读者，而且讲解得一清二楚。正是这种帮助读者的热情激励着每一个真诚而踏实的模式作者。如果没有这种热情，作者就会弄巧成拙——这种不负责任的作者往往正是人们对模式所有误解的根源！

致谢

（略）

参考文献

[Beck+96] Beck, K., et al. "Industrial Experience with Design Patterns," Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, March 1996, pp. 103-114.

[Budinsky+96] Budinsky, F., et al. "Automatic Code Generation from Design Patterns," IBM Systems Journal, Vol. 35, No. 2 (1996), pp. 151-171.

[Buschmann+96] Buschmann, F., et al. Pattern-Oriented Software Architecture: A System of Patterns, Wiley, Chichester, England, 1996.

[Coplien96] Coplien, J. Software Patterns, SIGS Publications, New York, NY, 1996.

[Fowler97] Fowler, M. Analysis Patterns: Reusable Object Models, Addison-Wesley, Reading, MA, 1997.

[Gamma+95] Gamma, E., et al. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.

[Hay96] Hay, D. Data Model Patterns: Conventions of Thought, Dorset House, New York, NY, 1996.

[Hueni+96] Hueni, H., et al. "A Framework for Network Protocol Software," OOPSLA '95 Conference Proceedings, published as ACM SIGPLAN Notices, Vol. 30, No. 10, October 1995, pp. 358-369.

[Kotula96] Kotula, J. "Discovering Patterns: An Industry Report," Software---Practice & Experience, to appear.

[PDML] patterns-discussion@cs.uiuc.edu.

[PLoPD95] Coplien, Schmidt, eds. Pattern Languages of Program Design, Addison-Wesley, Reading, MA, 1995.

[PLoPD96] Vlissides, Coplien, Kerth, eds. Pattern Languages of Program Design 2, Addison-Wesley, Reading, MA, 1996.

[Schmid95] Schmid, H. "Creating the Architecture of a Manufacturing Framework by Design Patterns," OOPSLA '95 Conference Proceedings, published as ACM SIGPLAN Notices, Vol. 30, No. 10, October 1995, pp. 370-384.

[Schmidt96] patterns-discussion@cs.uiuc.edu, December 12, 1996.

[Tichy96] Personal communication, October 24, 1996.

重点书籍推荐！！

UMLChina推荐

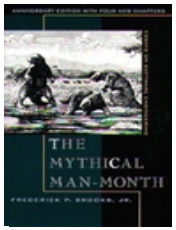
国内出版信息

英文样章链接

重点推荐

以下是UMLChina参与并重点推荐的一些书籍，书的评价好坏，可能各人各异，唯一能保证的是，做这些书的人是 用心 的。

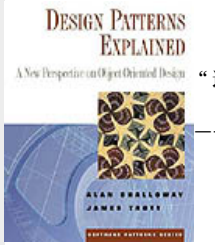
《人月神话》，Frederick P. Brooks, Jr 著，Adams Wang 译，清华大学出版社，预计2002年6月出版。序言，目录，第2章，第17章。留言评论>>



“《人月神话》一书被大量地引用，很少存在异议；相比之下，《没有银弹》却引发了众多的辩论，编辑收到了很多文章和信件，至今还在延续。他们中的大多数攻击其核心论点和我的观点--没有神话般的解决方案，以及将来也不会有。他们大都同意《没有银弹》一文中的多数观点，但接着断定实际存在着杀死软件怪兽的银弹--由他们所发明的银弹。今天，当我重新阅读一些早期的反馈，我不禁发现在1986年～1987年期间，曾被强烈推崇的秘方并没有出现所声称的戏剧性效果。”

——Frederick P. Brooks, Jr

《设计模式精解：面向对象设计的新视角》，Alan Shalloway, Jim Trott 著，透明 译，清华大学出版社，预计2002年3月出版。



“这本书最大的优点之一就是它用类推的方式将概念解释得非常清楚，而很少使用程序范例。我正在做一套关于OOP和软件开发的录音教材，这本榻彩龈拍畹姆绞蕉晕曳浅S衅舭 ！

——Bruce Eckel

Top

用户界面的交互模式

Martijn van Welie, Hallvard Trætteberg 著 James Zhong 译

本文讨论并展现了用户界面的交互模式。这些模式集中于最终用户在与系统交互时所产生问题的解决方案。模式从最终用户*的*角度总结出一种把可用性作为设计质量的关键的格式,。本文讨论了这种格式，并用 20 种模式编写并表现出来。

可用性原则

本文展现的模式是为描述让系统用户受益的成功解决方案而正作出的部分努力。因此，它们是大多数人熟悉的解决方案。其它集合如“ Common Ground ”（Tidwell 1998）或 Web 模式集合（Perzel 和 Kane 1999）并没有清晰地表明用户角度或设计者角度之间的差异。尽管那些模式中的一部分的确让用户受益，它们缺少适当的焦点和基本原理。例如，在 Tidwell 模式中问题的陈述形式通常是“ *物件如何最佳地显示……* ”，这与用户的使用并不明显相关。一般来说，每种以用户角度为中心的模式对设计者而言也是有用的，反之则不然。所以，我们的模式应该既让设计者也让最终用户受益。

由用户界面设计（UID）模式引发的兴趣可追溯至 1994 年（Rijken 1994，Bayle 1998），但即使存在多种模式集，也未能形成一套被接受的模式集合，即模式语言。似乎缺乏对 UID 模式的格式及焦点的一致意见。既然 UID 的模式语言必须在开发足够多的采用同样焦点或“ 视点 ”编写的模式后才能出现，它当然没有确立起来。我们认为 UID 的模式需要一种以*可用性*为中心的特殊格式。

站在用户的角度

当站在用户角度上时，将重心放在*怎样*和*为什么*改善可用性的争论上变得至关重要。没有这样的基本原理，不可能明白一个方案是不是或为什么是真正*好的*和*可接受的*方案。某些 UID 方案解决了设计者或市场人员而*不是*用户产生的问题。例如，横幅及闪屏是可接受的设计，但几乎不属于可用性原因。标识用户界面的模式并不困难，困难的是标识那些*真正*有益于用户的模式，并解释其可用性。

有众多糟糕并反复出现的用户界面设计的例子，使得标识良好的解决方案变得困难。另一方面，

糟糕的例子对刺激设计者使用模式非常有用，它的作用类似于反模式的思想。“ Interface Hall of Shame ”收集了这类糟糕例子并作了细致的说明。在其它情况下，对最大限度减少用户的可用性相关问题的解决方案加以描述是很诱人的。例如，在用户输入数据以后验证它并不总是最佳方案，尽管经常使用；在第一个位置就不应该允许用户输入句法上错误的数据。明确格式的模式可用来达到这个目标。

对用户问题分类

我们的模式是任务相关的，在此集合中我们根据它们所解决的使用问题的类型对它们进行分类。在（Norman 1988）中建议了若干用户界面的原则。这些原则指出了用户在与系统交互时可能遇到的问题和疑虑（McKay 1999）的类型。

- **可见性**。让用户能够通过看到某物而判断出如何使用它。嗯，我想这种特色可能会完成它……
- **可伺服性**。包括对象的感知属性和真实属性，表明对象是如何使用的。咦，这个对象如何工作？哦，我明白了……
- **自然映射**。在用户想完成什么和完成它的机制之间创建清晰的关系。要执行我的任务，我需要选择这个选项，输入那样的信息，再按这个按钮……
- **约束**。减少任务执行方式的数目和执行任务所必需的知识量，以使任务更容易解决。噢不，我需要在这里输入什么？哈，我已有这些选择了……
- **概念模型**。良好的概念模型在于，用户对某物如何工作的理解与它实际的工作方式相符。通过这种方式，用户能够自信地预料到自己行为将产生的效果。要执行这项任务，我提供了必要的信息，再发出这个命令……它看起来如我所料地在工作……
- **反馈**。暗示用户任务正在完成中以及任务正在正确执行。太好啦，它起作用了！

这些原则假定用户总是表现出完全理性的行为。实际上，用户会犯错误，并做他们并不真想做的事情。因此，有如下附加原则：

- **安全性**。需要对用户无意识的操作或错误加以保护。哎哟！我犯了一个错误，这里是如何纠正它的方式。现在我明白了，我会再试一试。

- **灵活性**。用户可能改变自己的主意，并且每个用户的做事方式可能互不相同。既然我考虑到它了，那个参数本应该是……取消它，我想改变一下顺序。

正如所观察到的，问题的类别很普通，并且在我们的模式中，场景描述对区分何时使用模式的情形是很重要的。另外，这些问题常常可以用多种方案解决，使解决方案的精确度和详细度显得尤为重要。

问题 **用户想要完成一个目标，但是在完全达到该目标以前必须做出若干对用户而言可能是未知的决定。**

可用性原则 用户指南（可见性）

场景 一个非专家级的用户需要执行一项频度较低的复杂任务，该任务由若干项子任务组成，每项子任务均需做出判断。子任务的数量必须较少，通常介于 3~10 个之间。

约束

- 用户想要到达总体目标，但是对需要执行的操作步骤并不熟悉或不感兴趣。
- 子任务可排序，但并不总是彼此独立。即，在能做下一项任务以前，需先完成某项特定任务。
- 要达到目标，若干步骤需要完成，但需要的确定步骤可能因前面步骤中所做的判断而不同。

解决方案 在整个任务的全过程中，引导用户操作每一步。让用户按步骤进行操作，并显示出有哪些步骤存在，以及已经完成了哪些步骤。

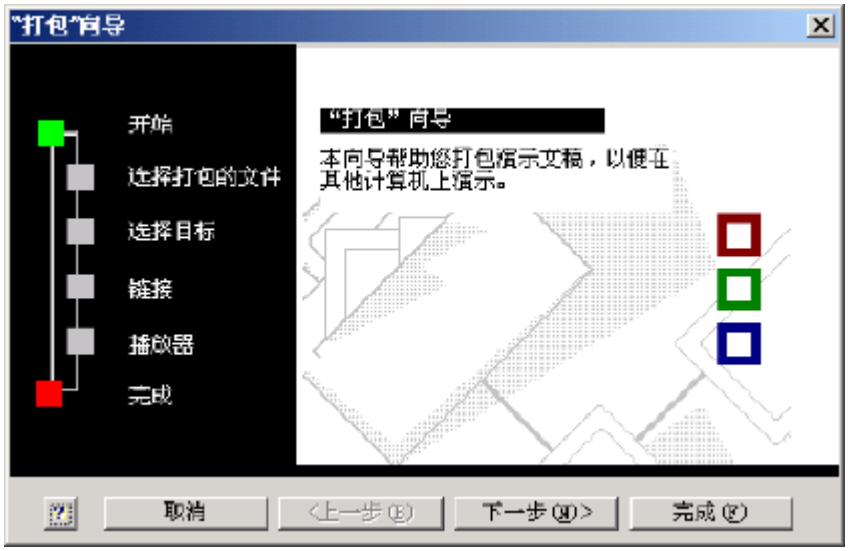
在复杂任务开始时，告知用户将完成的目标和若干需要作判断的事项。用户能够使用导航图形标志（如按钮）转向下一项任务。如果用户在完成当前操作前不能开始下一项任务，应提供反馈，显示用户不能在结束前继续（例如，禁用导航按钮）。当然，用户也能够返回前一项任务重新选择。

对每项任务的意图，都要给用户反馈；并让用户在全程中都知道他们所处的位置，以及操作序列有哪些步骤。当这项复杂任务完成时，应提供反馈，告知用户任务已完成，并且可选地显示结果已被处理。

知道缺省选项的用户可立即使用快捷方式以允许所有需要的步骤一次性完成。另外，在操作序列的任何位置都可通过选择 “ 退出 ” 而放弃任务。

基本原理 导航按钮提示用户他们正通向一条分步的路径。每项任务按一致的风格展现出来，以突出表明一系列的步骤正在执行。任务序列通知用户哪个步骤将需要执行，以及用户当前所处位置。任务的“可学习性”和“可记忆性”得到改进，但这样可能对任务的执行时间带来负面影响。当用户被强制沿任务的顺序前进时，用户错过重要事情的机会较少，因此能减少犯错误的机会。

示例



这是 PowerPoint 的“打包向导”。用户想要打包一份演示文稿以便演讲稿能放映在另外的计算机上。需要做一系列相关的决定，而向导帮助用户作出这些决定。绿框显示出任务序列的当前位置。

已知应用 Microsoft PowerPoint“打包”向导，InstallShield 安装程序

相关模式 空间导航，清单导航

网 络 布 局

问题 **用户需要快速理解信息并依据该信息采取行动。**

可用性原则 一致性，可预知性（概念模型）

场景 任何这样的情况：若干信息对象需要展示并被安排在一个有限的空间区域上。典型事例是，对话框屏幕、窗体和网页的设计。

约束

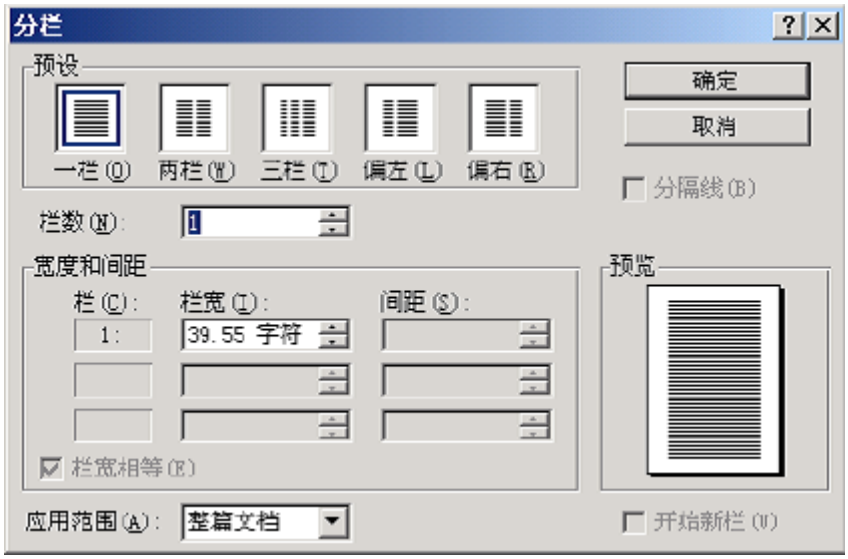
- 用户需要看见许多对象，但希望看见一个清晰的组织方式。
- 用户希望花在屏幕上浏览/阅读/察看对象的时间降至最低。
- 对象常常是相关的并且能从概念上进行分组。
- 界面必须紧凑，但是仍然清晰、令人愉悦并具备可读性。

解决方案 将所有的对象安排到一个网格中，使行和列的数目最少，同时使单元格尽可能大。

对象被安排在一个使用最少数目的行和列的矩阵中。同样类型的对象必须用同样的方式排列和显示。如果若干对象可被分组，则同时网格也适应于分组的级别。较短的元素可被拉伸，以网格边界为起始处。较长的元素可占用多个网格单元。特定的对象可能拥有固定尺寸，这会增加行和列的数量以便对象保持它们的固定尺寸。标准的响应按钮可能拥有预定义的位置，可考虑放置在网格之外。

基本原理 减少行和列的数目改善了需要浏览信息并采取适当动作的时间（Fitts 法则——该法则认为，指向一个对象的时间与网格单元中对象的距离的对数成比例）。另外，它的页面布局非常一致，视觉混乱最少，不会让用户感觉到任何的不适。阅读信息所需的时间减少了，任务的性能增强了。因此页面布局看起来令人愉快，提高了满意度。

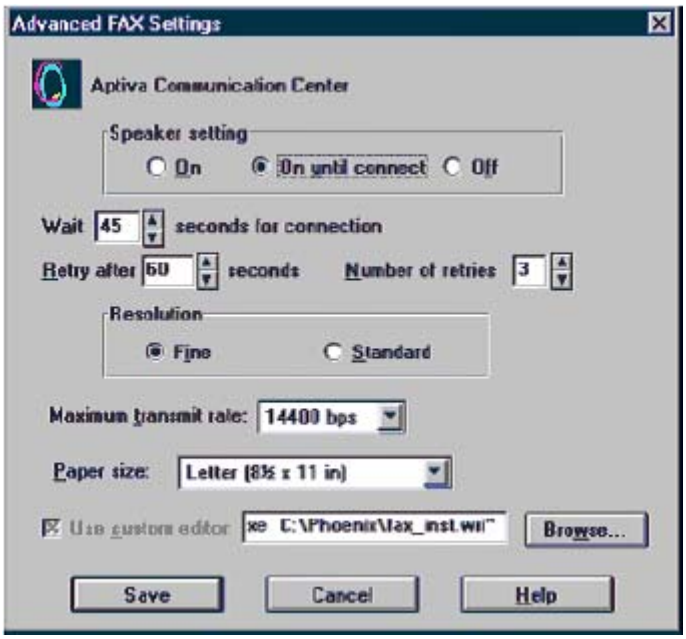
示例



该屏幕截图来自 Word 。若干对象放在一个对话框里。元素已在网格中排布好，对象均匀地对齐并调整好尺寸，以减少行和列的数目。

已知应用 Microsoft Word 选项对话框

反面示例



这幅图来自 IBM 的 Aptiva Communication Center，它表明开发者仅仅想要将东西放在屏幕上，而不是使人们更容易调整设置。屏幕上没有章法；你的眼睛仅仅是到处跳转，同时你的大脑试图抽取出一定的顺序。

进 度

问题 **用户想知道操作是否仍在执行，以及还需要等待多长时间。**

可用性原则 指南（反馈）

场景 系统任务：花费较长时间（通常超过几秒钟），并在下一任务开始前必须完成。

约束

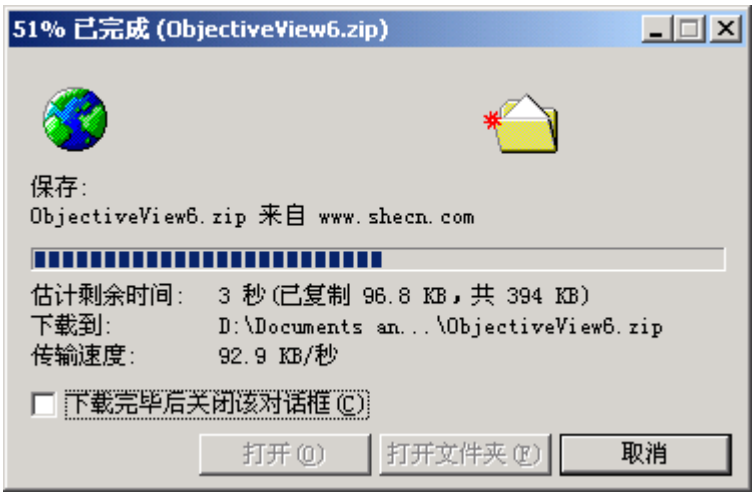
- 操作性能不总能由用户（或设计者）控制/避免，例如它依赖于一个可能失败、受阻或性能较差的外部系统或硬件。
- 用户不想等待，但需要清晰的进度反馈以及估计的完成时间。
- 用户可能不熟悉任务的复杂性。
- 在操作期间，用户可能因耗时太久而决定中断操作。

解决方案 显示出应用程序仍在运行，并给出进度指示。

以一定频率，提供反馈，比如每两秒使用一次动画，给用户操作仍在执行的印象。另外，提供有效的进度指示。进度的表示一般是剩余的完成时间，已处理单元的数目，或工作完成的百分比。进度可用图形标志显示，如进度条。进度条必须具有一个标签，说明相对的进度或度量的单位。

基本原理 通过以大约 1 或 2 秒的速率提供新的反馈，用户可看到应用程序是否仍在处理而没有不再响应。进度指示给出了在该状态下还要持续多长时间的反馈。这两方面综合起来，解除了用户的焦虑。省去二者之一将无法解决用户的难题。这种解决方案提高了满意度，因为用户知道什么正在进行以及必须等待多久。它增强了控制感。该模式亦避免了由于用户重试而引发的多余系统负载。

示例



当使用 Internet Explorer 5 下载一个文件时，显示给用户这个对话框。它显示进度的百分比，同时有已接收数据的千字节数。另外显示估计的剩余时间并且两秒更新一次。一张飞动的文件的动画表明下载还未停止。

已知应用 Netscape 的下载对话框，苹果电脑的文件复制

防 护 盾

问题 **用户可能会偶然选择一个具有不可逆（负面）效果的功能。**

可用性原则 错误管理（安全）

场景 具有不可逆（负面）效果或需要消耗很多资源才能撤消/逆转的功能。（负面）效果可导致不安全或极不希望出现的情形。通常它是产生严重后果的行为参数的组合，并且用户可能不会察觉，因为它通常是安全的。

约束

- 需要保护用户，但是不应该改变正常的的操作。
- 当试图避免错误时，用户正在追求速度。
- 某些（负面）效果比其它的更不希望出现。

解决方案 通过插入防护盾来保护用户。

在功能上添加额外的保护层以避免用户犯错误。要求用户确定要选择的意图，而缺省应答是安全选项。

基本原理 额外的层促使用户需要重复两个错误而不是一个。安全的缺省值减少了犯第二次错误的机会。这种解决方案增加了安全性，减少了错误，并提高了满意度。然而，它要求额外的用户操作，导致执行效率较低。

示例

文件的副本已经在指定的位置存在。覆盖它将导致副本的丢失。缺省值是“否”以防止着急的用户直接选择“是”。

这是另一个示例，来自 Sony 的 Mavica 数码相机。用户选择了格式化功能，并被要求确认该操作，缺省值是安全选项。

已知应用 Microsoft Explorer，Apple Finder

相关模式 报警

选 项

问题 **每个用户都是不同的并且比较喜欢做稍微不同的事情。**

可用性原则 适应性（弹性）

场景 应用程序非常复杂并且它的许多功能都可根据用户偏爱而调整。没有足够的关于用户偏爱的了解，无法假定适合所有用户的缺省值。潜在客户可能是新手也可能是专家。

约束

- 让用户组织用法能提高满意度，但同时使用起来变得更复杂。
- 用户可能不熟悉可能的调整选项。
- 用户需要知道调整结果。
- 很多的选项是可能的，但用户只希望改变一个子集。

解决方案

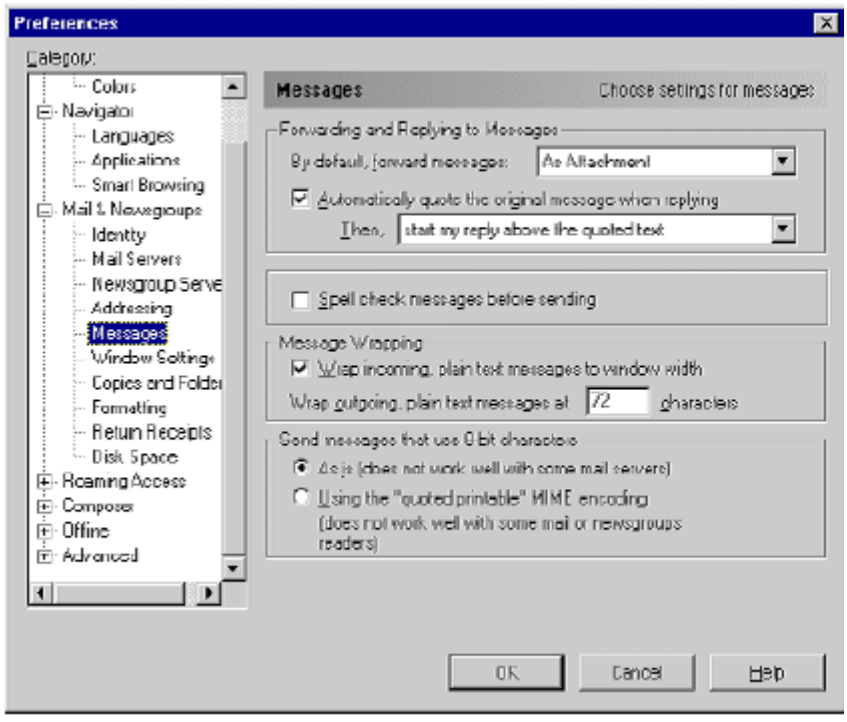
允许用户调整他们的偏爱。

不是强加给每个用户一种单一的配置策略，应允许用户更改他们的偏爱。为用户提供这样的选择，它们将成为该用户今后使用的缺省选择。选项常常可以分组。如果分组数较小，可为每组使用属性页；然而当分组数较多时，应使用树型结构。每个选项必须加以解释以让用户明白导致的后果。

基本原理

导航结构被用于选择调整的类别。树型能比 TAB 控件容纳更多的分类。同时显示所有的设置将带给用户“配置文件”的印象，此配置属于该用户。专家级用户可为达到他们的特定的目的而微调应用程序，诸如提高满意度以及可能带来的性能提升。这种解决方案改善了满意度及性能，但降低了可记忆性和可学习性。

示例



这是来自的 Netscape 4 的选项对话框。在各个区域，用户都能设置自己的偏爱选项。

已知应用

Netscape 参数设置，Internet Explorer 5 的 Internet 选项，Mac OS 9 的多用户特色

相关模式 网格布局，清单导航

环 境 菜 单

问题 **在任何时候，用户都需要知道他们的可能性是什么以便决定做什么。**

可用性原则 用户指南（可见性）

场景 一个应用程序通常包含许多功能，用户需要知道其在使用过程中任何一点的可能性。用户通常是初学者或偶然用户，功能不是经常用到。

约束

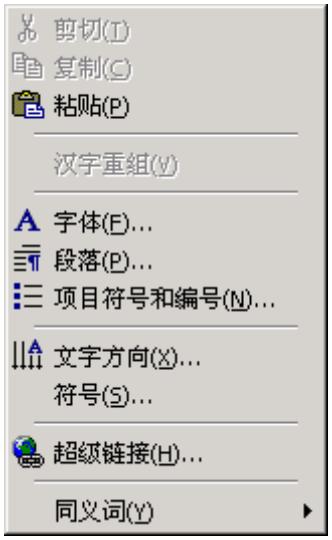
- 用户可能不熟悉可能性的含义。
- 可能性的数量是很大的，而用户需要定位需要的那一种。
- 不是所有的可能性在当前上下文中都有效，但用户可能仍然想知道所存在的可能性。
- 结合学习和使用，但不用打扰专家。

解决方案 将选择放在一个菜单中。

显示在当前上下文有效的所有功能的清单。使功能用单个动作即可访问。如果功能数较多（超过 7 个），对功能分类并使组可辨识。如果功能总数较少（通常为一个组或在一个组内），清单应该显示所有功能，并显示当前上下文中哪些是可能选择的。对于不熟悉功能标签的用户，应该有可用的描述解释该功能。功能应按一个或多个如下的标准排序：语义、相似性、使用频率或字母顺序。如果可能性清单在不同上下文中差异不大，清单则应显示所有的可能性并高亮显示当前上下文的可能性。

基本原理 功能清单给用户一个针对所有可能性的快速预览。人们熟悉这样的清单（例如宴会菜单）并迅速识别出它的功能。这种解决方案改善了可记忆性和满意度，但因为额外的动作而降低了执行速度。

示例



这是来自 Word 的“编辑”子菜单。该菜单显示了与文档编辑相关的所有功能。这些功能按语义分组，并只有当前上下文中的可能性被高亮显示。

已知应用 带有菜单的任何应用程序，Web 站点的菜单

相关模式 网格布局

焦 点

问题 **用户想要快速获知他们所看到的对象的信息，并可能修改该对象。**

可用性原则 指南（约束）

场景 其中有若干可视对象被操作的应用程序，典型地，如绘制包或浏览工具。

约束

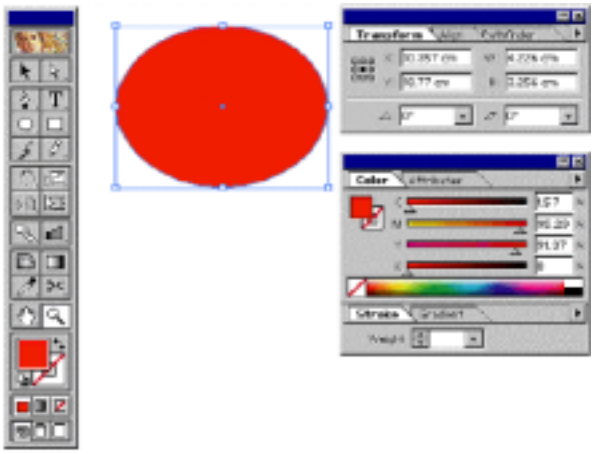
- 许多对象是可见的但用户通常一次只影响一个对象。
- 用户既需要一个对象集合的概览，也需要对象的属性和可用功能的细节。
- 用户可能想应用一种功能到若干对象上。

解决方案 在应用程序中引入焦点。

焦点总是属于界面中的一个当前对象。焦点亦即用户正对其工作的对象决定了上下文的可用功能。焦点对用户必须是可视地显示，例如更改它的颜色或沿它的四周画一个矩形框。用户可通过选择另一个对象而改变焦点。当一个对象拥有焦点时，它变成了与该对象相关的所有功能的作用目标。另外，当焦点改变时，包含相关功能的窗口被激活。

基本原理 人们习惯于作用于对象。焦点是“捕获对象”的等价物。因此，属于对象的功能被激活并展现给用户是非常自然的。这样降低了如下动作的数量，需要选择功能并针对特定对象执行。这种解决方案改善了执行速度和可记忆性。

示例



该屏幕截图来自 Adobe Illustrator。用户可用这个绘图应用程序绘制图形对象。当圆被选定时，对象特定的窗口显示了对象的状态。该示例显示了圆的颜色和尺寸。

已知应用 Adobe Illustrator，使用直接操作的许多其他应用程序，Microsoft 文件资源管理器，窗体

相关模式 属性设置

明 确 格 式

问题 **用户需要为应用程序提供数据，但可能不熟悉需要哪些数据或使用什么语法。**

可用性原则 用户指南（限制）

场景 必须输入结构化数据的任何系统。数据比如日期、房间号、社会安全号或序列号常常是结构化的。用于这些数据的明确语法可能因国家或产品而异。

约束

- 当数据以意外的语法输入时，数据无法被应用程序使用。
- 用户可能熟悉数据但可能不知道确定的所需语法。
- 追求录入速度，但也想正确输入的用户。
- 文化习俗决定了用户期待的语法是什么。例如，日/月/年在欧洲较普遍，然而在美国使用月/日/年。

解决方案 仅允许用户按正确的语法输入数据。

提供给用户针对结构每个数据元素的域。如果域的语义出现重复，用数据单元的名称标识每个域。域不允许输入错误数据。避免用户可随意敲入文本的域。另外，用一个示例或格式描述来解释语法。提供合理的缺省值给必需的域，不是必需的域应该避免或标识为可选。当使用可选域时，由此产生的后果必须向用户解释。

基本原理 主要思想是，通过不使输入错误数据成为可能而避免输入不正确的数据。通过显示所需的格式，错误的机会减少了，因为用户被给予了完整的信息。然而，用户现在必须给出多个数据输入而非一个，所以需要更多的时间来输入数据。这种解决方案降低了错误数并提升了满意度，但执行效率可能降低。

示例

该屏幕截图来自 Microsoft Windows 的日期和时间控制面板。输入日期被划分为三个输入区域。每个输入域都只允许有效输入。输入无效日期变为不可能。

已知应用 Microsoft Windows 日期/时间控制面板

相关模式 网格布局

空 间 导 航

问题 **用户需要访问那些不能在可用空间里放置下来的大量信息。**

可用性原则 用户指南（自然映射）

场景 拥有多种状态、功能和对象仅在组内相关的系统。

约束

- 大量数据需要非常多的空间，但可用的显示区域有限。
- 大量数据通常不是无关的，可以通过分类来和用户对数据的概念模型相匹配。

解决方案 在多个区域中显示信息，并允许用户在区域间切换。

将信息元素分组到单独的空间里。允许用户一次只选择一个空间。每个空间区域应以类别名称加以标识。所有独立的空间应该能从导航区域通过单个操作访问。导航区域应放置在空间的顶端或左侧并必须与余下区域相连。如果空间数较小（如小于 8），导航区域应放在顶端。当空间数目较大时，导航区域应采用树型结构放置于空间的左侧。

基本原理 将元素分组使用户更容易找到一个特定的元素。在顶端或左侧放置导航条减少了所需的屏幕空间。这样做的原因是因为标签常常是较宽和较小的文本。另外，在西方社会人们的阅读习惯是从左到右、从上到下。这种解决方案改善了执行时间。

示例

这是 Microsoft PowerPoint 的“选项”窗口。用户能使用 TAB 控件在各选项间导航。每个 TAB 显示了一类选项。

已知应用 Microsoft PowerPoint，Netscape，诸多 Web 站点

相关模式 向导

与现实世界相似

问题 **用户需要知道在界面中如何控制一个对象，就像对现实世界中相似的对象一样。**

可用性原则 类比（概念模型）

场景 使用现实世界比喻物并直接在界面上操作的应用程序。界面中的对象与现实世界对象相似，表明交互方式也应与现实世界的交互方式相似。

约束

- 交互操作由输入设备完成，但每种设备仅具备有限的一套处理能力，诸如移动、旋转、力反馈和自由度。
- 对象在现实世界中的操作方式导致了预期的交互发生方式。

解决方案 将使用的输入设备和窗口部件相匹配。

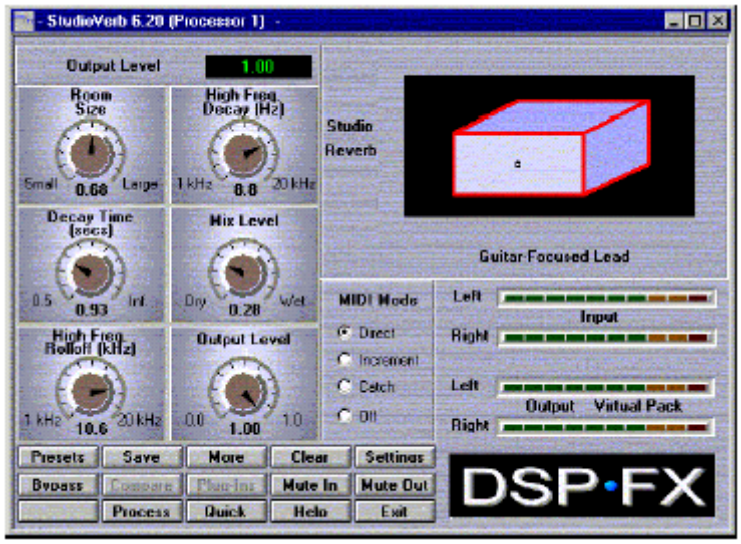
依据可能的动作将窗口部件与输入设备结合起来以互相匹配。动作包括自由度与反馈类型（如触觉或视觉）。例如，如果窗口部件需要绕 Z 轴旋转，则使用支持它的输入设备。如果不匹配，选取一种不同的输入设备或更改窗口部件。

基本原理 如使用现实世界的比喻物，那么该比喻物与它在现实世界中的使用方式相同是非常重要的。否则，用户可能开始认出了窗口部件，而后却发现了出乎意料的行为。这种解决方案减少了学习时间，改善了执行时间和满意度。

示例

此图显示了一种称为“幻影”的特殊输入设备是如何用于 3D 建模应用程序的。这与现实世界中雕塑家如何创作一座雕像相仿。“幻影”设备结合触觉反馈提供了很大的自由度，因此雕塑家可以真实感受到正在设计中的雕像的表面。

反面示例



该屏幕截图来自 DSP-FX，一个实时音效的软件包。在现实世界中，具备该功能的物理设备使用旋钮。这样做的原因之一是人们需要比滑动条更少的空间，尤其是当设备仅有 1 英寸高时。在此例中，使用了同样的比喻物。然而，旋钮需要旋转对人的手而言没有问题，但对鼠标而言却极不自然。鼠标无法旋转，用户只好以圆周方式移动鼠标来模拟旋转。从效率而言旋钮与滑动条同样地工作，然而在这种情形下，它们却并不节省屏幕空间。

已知应用 Worldbeat 系统，Urban Planning 工作台

预 览

问题 **用户在一个较小的集合中寻找某一项，并试图通过浏览集合来发现该项。**

可用性原则 兼容性（自然映射）

场景 在许多应用程序中，用户需要寻找一项如文件、演示文稿、视频剪辑或图片，对此视觉或听觉的搜索条件更有效，但集合的索引并非可听或可视（例如是一个文本标签）。

约束

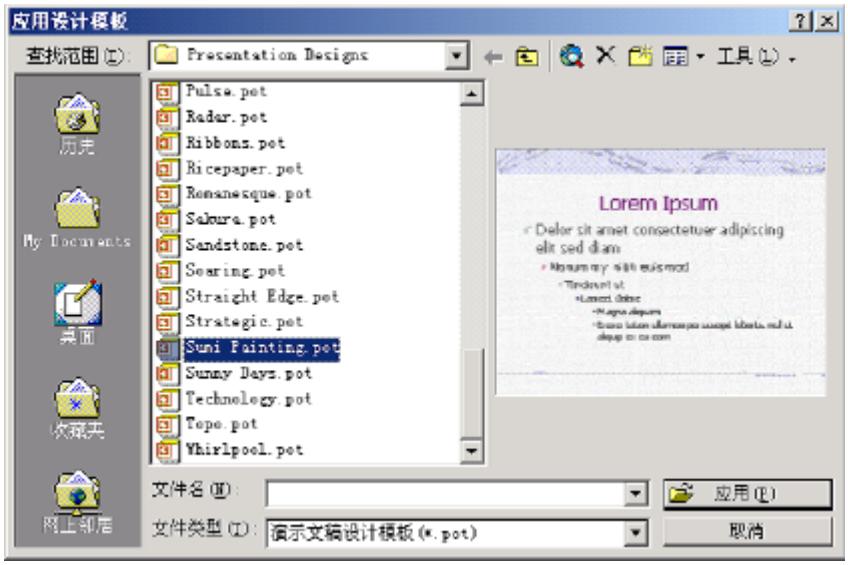
- 尽管用户只希望看到真实的项，但所需显示真实项的资源可能无效。
- 用户看到了索引名但可能无法仅通过索引名来辨别某一项。
- 用户正寻找某一项，但搜索结果只需要索引的名称以作其它任务之用。

解决方案 允许用户预览各项。

显示一个项或多个项的典型和适当的预览。预览可比原来使用更少的资源，如屏幕、装载时间或质量。预览程度应足够到能辨别它。如果集合中的项的数目较小，预览可显示所有的项（如果在约束条件下可能的话，例如屏幕空间等等）。如果集合包含较多项或当标签对用户而言是重要的，只在项的标签旁显示单个项的预览。预览应位于所选项附近以保证它们之间的关联。当选择发生变化时预览也应立即更新。

基本原理 用户能“看见”或“听到”某一项是否找到，因此搜索时间减少了。否则，用户将需要在他们知道某项是否为正确项之前先打开它。因为预览比原来使用了更少的资源，它要有效得多。只要预览具有代表性，搜索变得更为有效。这种解决方案改善了执行时间和满意度。

示例



该例来自 Microsoft PowerPoint。用户可选择设计，设计定义文件的文件名展现出来。用户真正感兴趣的是根据设计的外观来作出选择。预览区域显示了应用设计以后一个标准页的缩略样例。

另外的例子是在一个 PDF 文件里的“缩略图”。Acrobat 的阅读器显示每一页的缩略图，以便在文档中可视地查找一页。网页经常也使用缩略图来浏览图片集。

已知应用 PowerPoint，ACDSee，Acrobat，Adobe Photoshop，网页

收 藏 夹

问题 **用户需要在包含大量项的集合中找到经常使用的某项。**

可用性原则 使操作量减到最少（弹性）

场景 用户在包含于大量项的集合中寻找一项。该项是重要的并且用户经常需要它。这些项通常是文件、颜色、网页或数据库记录，并且分别是文件系统、Web 或数据库等较大集合的一部分。

约束

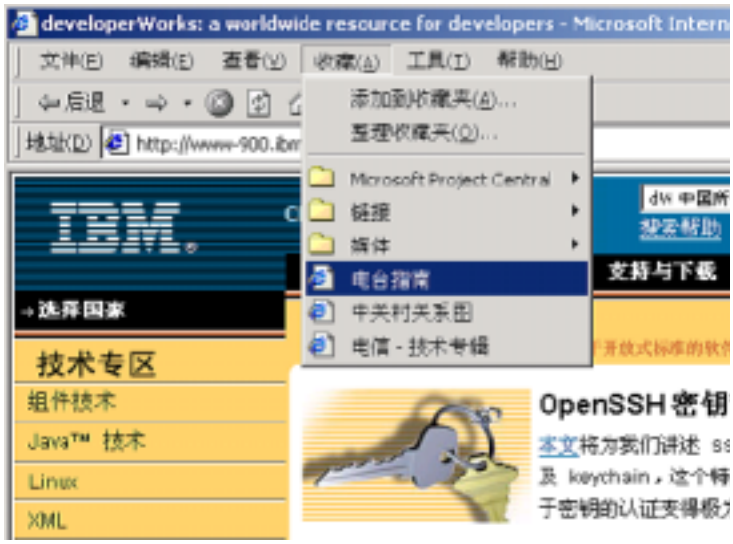
- 用户知道某项的存在并且经常使用它 ;因此这些项必须就在手边。
- 包含于间隔中的这些项的数目，影响着每次需要某项而搜索它所花的时间。
- 用户对某项感兴趣但可能仅在短时间中使用它。

解决方案 允许用户使用指向选项的收藏夹。

介绍一下创建及移除收藏夹的可能性。收藏夹是直接指向某特定项的标签。标签可以是文本型描述或其它可帮助用户识别该项的东西。应通过最少的用户操作访问收藏夹，通常是直接从菜单访问。然而，收藏夹的数目可变得很大。因此，应允许收藏夹在必要时分层排列。

基本原理 用户可能从阅读书籍时使用书签的经验得知了这个概念。收藏夹允许用户在无需记住某项的精确位置的情况下访问该项。在远小于整个集合的收藏夹集合中再次查找它的工作减少了，因此搜索时间也减少了。这种解决方案提高了执行速度和满意度。

示例



这是 Internet Explorer 5 的收藏夹菜单。通过选择标签可以进入一个网页。

另一个例子是调色板。这种情形下，项是一种颜色，用户可生成“ 用户定义 ” 颜色以快速访问一种颜色。

已知应用 浏览器（Netscape、IE、Opera），大量应用程序中的最近使用过的文件列表，调色板

相关模式 提示

命 令 区

问题 **用户需要知道在何处找到可能的命令以及怎样激活它们。**

可用性原则 一致性，用户指南（可见性）

场景 在每个应用程序中，各种功能需要让用户知道。

约束

- 某些功能在执行前需要用户设置额外的功能参数。
- 可用的屏幕区是有限的，主工作区应保持得尽可能大。
- 使用图形标识符的快速功能访问提高了交互的速度，但消耗了很有价值的屏幕区域。
- 显示出来的诸多不同的可视元素把整个屏幕弄得混乱不堪，并增加了待处理不同元素的认知负担。
- 某些功能比其它功能更常使用。

解决方案 将命令放在一个特定的可识别区域。

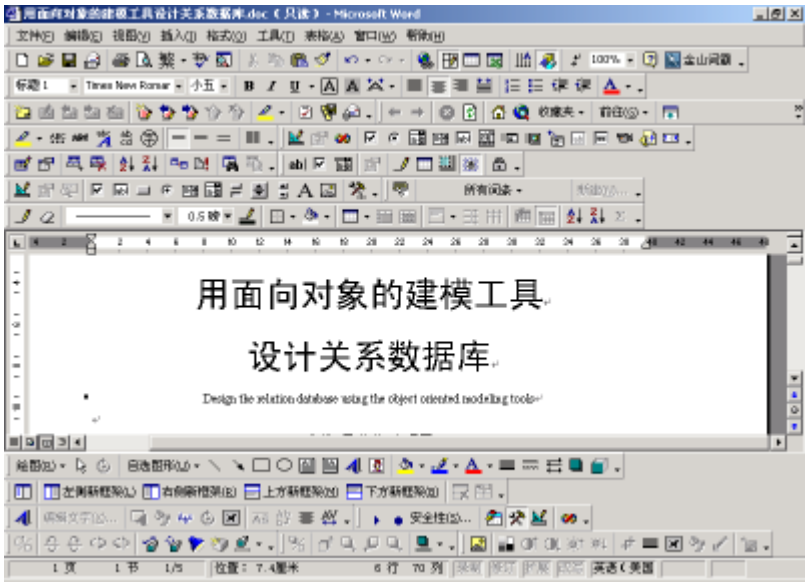
保留一块屏幕区域作为功能访问之用。该区域应位于顶端或左侧以保证良好的可视性。使用颜色、文本或其它可视特征使该区域与屏幕的其它工作区相区别。命令区域应伸缩至屏幕或窗口的整个宽度或高度。如果功能数目较多，则应按从概念上分组。命令区可作子分类以提供到一组命令的访问。子命令区应拥有相同的外观和行为。命令的访问应越直接越好，尤其是常用功能。命令区不应占据超过 1/6 的屏幕区域。功能入口的数目不应超过 60 个，以便用户不会产生视觉上的过量负担。

基本原理 在固定位置提供一块命令区为用户提供了一种找到功能的一致方式。规定直接访问区提供了对经常使用功能的直接访问，并方便了快速交互。区域的位置处于总是可见的区域。这种解决方案提高了可记忆性和满意度，但降低了交互的速度。

示例

该屏幕截图来自 Word 2000，仅显示了两栏工具栏及一栏主菜单。命令区域清晰可见却并不占据过多的屏幕空间。

反面示例



该屏幕截图为 Microsoft Word，并且几乎所有的工具栏都激活了。屏幕高度混乱，用户不得要领。命令区域占据了太多的空间，并包含了过多的直接访问功能。更糟的是，工具栏不仅包含命令的快捷方式而且包含了状态信息或属性设置。这些差别并不能从视觉上区分。

已知应用 任何基于 Windows 的应用程序或网页

容 器 导 航

问题 **用户需要在容器集中查找到一项。**

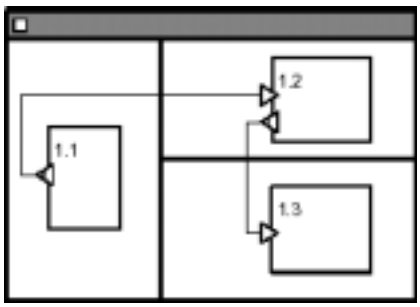
可用性原则 元素分组（自然映射）

场景 许多应用程序包含组合数据，用户必须浏览它们。很常见的，用户想要调用一项功能使其中一部分作为输入参数。

约束

- 用户同时查看所有容器和某一项是重要任务。
- 项的数目较大但在特定时刻只有各项中的一部分需要可见。
- 用户可能需要从一个容器切换到其它容器。

解决方案

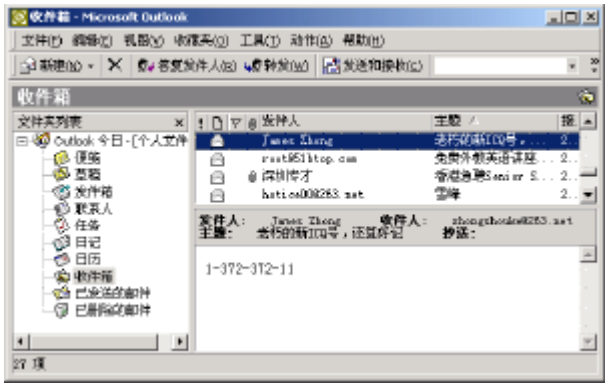


将屏幕划分为三个区域，显示容器及最终的选择项。

将窗口分为三个窗格，一个作为查看容器集，一个作为查看容器，还有一个作为查看单个的项。选择的容器应决定容器窗格的内容，选择的项应决定项的窗格的内容。选择可能用作调用功能的参数。每个窗格应针对待显示内容的类型而特制。例如，如果容器形成一个分层，则可使用能提供页节点选择的树状窗格。各窗格应可以单独滚动和调整大小。

基本原理 通过按西文阅读方式（从左到右、从上到下）配置窗格，支持基于选择的相应关系。通过提供选择，也支持导航外的其它功能。页面布局应有助于理解窗格之间的因果性。每个窗格可根据域和用户进行裁剪。各个可调大小的窗格给了用户自由。

示例



这是 Outlook 的邮件阅读屏幕。左窗格中显示了容器（这里是 Outlook 主功能导航）集合。其它窗格显示了所选的收件箱中的邮件清单，以及所选择的邮件。

已知应用 Microsoft Outlook，Netscape Mail/News，Eudora Mail

相关模式 网格布局

属 性 设 置

问题 **用户想要查看当前工作对象的属性，另外还需要知道如何修改它们。**

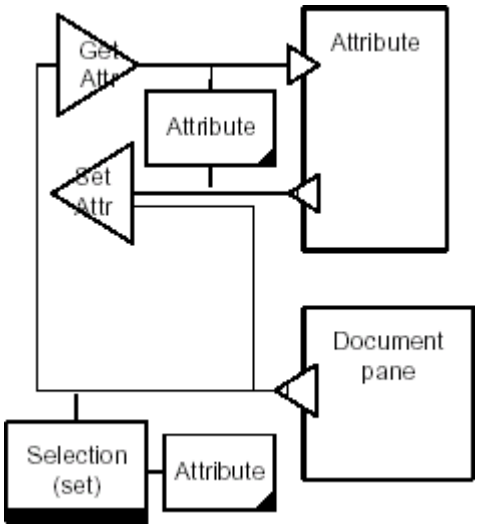
可用性原则 用户指南（可见性）

场景 在许多应用程序中，文档可拥有许多不同的对象，这些对象带有许多不同的属性。

约束

- 某些属性在文档中部分可见，但在文档上下文中通过直接操作对象而作出修改是困难的。
- 设置属性需要的控制与查看属性不同。
- 对象经常有若干类型，拥有不同的属性集。
- 一些属性对一类对象是唯一的，然而其它属性可能为各种对象所共有。

解决方案



创建显示属性值的特殊窗口部件。

对文档窗格采用对话框元素分帧，以显示不同的属性类型。只包含最常用元素，并让用户定制可用的元素。通过这些手段提供到对话框元素的访问，以设置相同的属性类型。对话框元素用当前选择对象的属性值填充，或在多选情形下的公共属性值。将对话框元素包含而选择对象不支持的属性变灰。当访问一个对话框元素并设置它的值时，对选择对象的相应属性值作了设置。

基本原理 许多属性在主文档窗格中是部分可见的，但与一个特定对话框元素所能提供的并不精确地相同。通过为每个元素提供特定的对话框元素，改善了精确度。如果提供弹出或下拉风格的对话框元素以供输入，也节省了空间。通过为每项属性类别提供特定的对话框元素，这些值查看和理解起来也应该更容易了。针对特定选择的未变灰对话框元素集，有助于了解每类对象的可用属性。选择总体的元素，让用户控制了焦点。通过让用户使用同样的对话框元素设置属性，为用户提供了直接操作的感觉。

示例 字处理器与风格（字体、对齐方式、标尺）属性，画图应用程序与图形（字体、颜色、画笔）属性，电子表格与单元格（格式、字体、颜色）属性。

已知应用 Microsoft Word，Microsoft Visio，Microsoft Excel

相关模式 命令区

报 警

问题 **用户可能无意间引发了一个问题的情形，并需要解决。**

可用性原则 错误防护（安全）

场景 用户执行一个操作，而该操作可能无意间导致一个问题。

约束

- 如果操作全部完成则工作可能出现丢失。
- 系统不能或不应自动解决这种情形，因此用户需要考虑。
- 出现频率。
- 问题可能解决的方式的数目。
- 用户特意完成某任务的可能性，例如用户希望这样做。
- 某些操作难以或不可能从中恢复。
- 用户可能不明白为何一个操作可能有害。
- 用户可能不明白后果或选项。
- 如果问题发生那么它的严重性如何？

解决方案 在继续任务前警告用户，并为用户提供放弃任务的机会。

警告应包含如下元素：

- 问题摘要。
- 已触发此警告的条件。
- 询问用户是否继续操作或执行其它操作的问题。
- 为用户提供两个主要的选项，确定选项及取消选项。

警告可能已包含更为详细的状况描述以帮助用户作出适当的判断。选项的陈述应包含动词指出希望采取的操作。不要使用“是/否”作为选项。选项是所询问的问题的答案。

在某些情形下，可能会有超过两种的选择。增加选项的数目可能在某些情形下是可接受的，但要尽量减少选项的数目。

基本原理 通过陈述触发条件，用户可理解为何警告会出现。一旦理解了问题，就留下两种选择给用户。仅提供两种选项，对用户而言选择显得较为简单：继续还是取消。更多的选项使用户作出判断的困难度增加了。通过在选项中使用一个动词，用户立即知道自己正在选择什么；反之，“是/否”选项需要用户明确记住问题是什么。这种解决方案减少了错误，提供了满意度。

示例

该屏幕截图来自于 Microsoft Project，当你试图退出程序时。它显示出甚至三种选择都是可接受的。

已知应用 Eudora，InstallShield 安装程序（当退出时）

相关模式 防护盾

提 示

问题 **用户需要知道如何选择功能。**

可用性原则 增加启迪性（可见性）

场景 应用程序中，功能可以由多种方式访问，例如通过菜单、使用键盘快捷方式或通过工具栏。该模式可用于以一种非显而易见的方式让用户知道其它的可能性。

约束

- 可用的屏幕空间可能有限，因此没有空间用于多余的视觉提示。
- 用户需要某种方式揭示和了解这些可替代或可能更高效的方式，通过一种非显而易见的方式。
- 用户可能知道或可能不知道访问某功能的其它方式。
- 功能激活方式的数目决定了可能的提示数目。

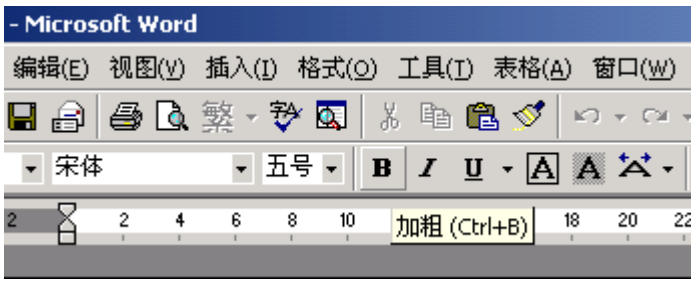
解决方案 **为用户提供用其它方式访问同样功能的提示。**

当通过一种方式访问一项功能时，提供访问同样功能的其它方式的提示。一种可能性是使用多标签；每个标签对应访问功能的一种方式。例如，如果功能具有键盘快捷方式，显示按键组合。如果有功能的图标快捷方式，显示图标。总是显示主要标签，并且在可能的约束条件下立即显示其它标签。

其它可能性是使用帮助助手或延迟消息来响应用户操作。例如，当用户将鼠标停留在一个窗口部件上约两秒钟时，工具栏提示显示出来。

基本原理 用户熟悉采用至少一种方法访问某项功能的可能性很高。通过显示其它标签例如键盘快捷方式或图标，用户将获知功能访问的更多联想。在某一点处，用户可能在工具栏中“看到”图标并使用它，而非使用菜单。同样的方式，用户可能相对鼠标更偏爱键盘访问的方式。这种解决方案增强了可学习性和可记忆性。当用户真正开始使用其它的功能选择方式时，执行速度也可能得到提高。

示例



这些屏幕截图来自 Word 2000。它们显示了该模式的可能实例：一种使用工具栏提示，另一种使用带图标的菜单。

在菜单中，有空间包含图标和快捷方式但工具栏图标不允许这样做。在那种情形下，信息在短暂延迟后的工具栏提示中显示出来。通过那样的方式，高级用户不会被不停弹出的窗口打扰。

已知应用 工具栏提示，Office 2000 菜单

相关模式 命令区

光 标 样 式

问题 **用户正在创建或修改一个对象，并需要知道哪种编辑功能处于被选择状态。**

可用性原则 即时反馈（反馈）

场景 在许多直接操作的应用程序中，用户首先选择一种工具/功能，这样就进入一种特殊的样式/状态，然后在工作于一个对象。这样的应用程序经常提供许多功能来创建或修改对象。

约束

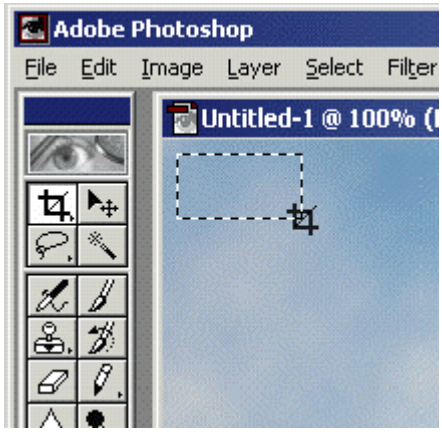
- 不是每种功能都有一个图标或形状。
- 完成一项功能可能经过好几种中间状态，它们也可能需要显示出来。
- 用户需要对所选功能的即时反馈，即系统所处的样式/状态。

解决方案 **在光标中显示界面的状态。**

界面状态在交互期间发生多次改变，例如当一项功能被选中或当一个操作如拖动被执行时。因此，通过改变光标向用户显示当前状态。光标改变为一个图标或其它形状，以给出关于当前界面状态的反馈。当功能完成或不处于激活状态时，将光标恢复至正常状态。

基本原理 光标提供了关于活动功能的额外反馈。用户在执行功能时会观察光标，因此它是屏幕上提供反馈的最合适位置，用户不需要再观看屏幕的其它部分。这种解决方案提高了满意度，并可能减少错误。

示例



该屏幕截图来自 Adobe Photoshop 5。用户已选择裁剪功能，如左边的功能面板所示。光标已改变并与功能图标具备同样的形状。

已知应用 Adobe Photoshop，Adobe Illustrator，Microsoft PowerPoint

清单导航

问题 **用户需要浏览或处理集合或清单的若干项。**

可用性原则 使操作量减到最少（自然映射）

场景 在许多应用程序中，用户需要检阅清单的各项。例如，当在新闻网站上阅读新闻时，或者当浏览数据库查询的结果集时。用户通常从一张清单中选择一个初始项，然后想要移到其它项去，一般是下一项或前一项。通常地，用户想要按顺序、往后或向前阅读清单的若干项。

约束

- 用户想要查看集合/清单的概览以及其中的至少一项，但屏幕空间可能不足以让二者同时显示。
- 用户需要能选择单个的项，同时能处理多项。

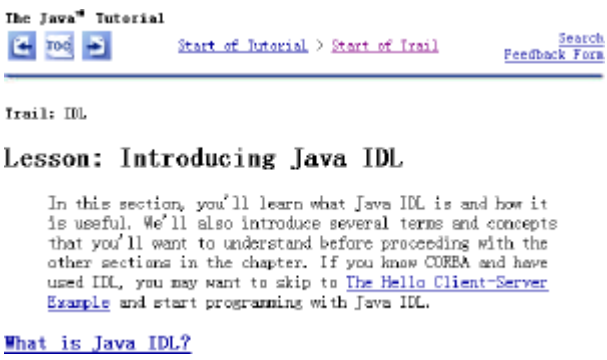
解决方案 寻找一种自然的排序方式，并允许用户直接从一项浏览至下一项及前一项。

基于对任务的了解，为集合/清单中实施一种排序方式。顺序可基于一种等级评定，如可检索性；或者基于字段值，如日期/时间。甚至任意排序都有用。

对呈现给用户的每一项，导航图标允许用户选择清单中的下一项或前一项。排序条件应为可见的（最好可以由用户配置）。要支持方向性，当前项以及一部分的索引清单应清晰可见。如果不考虑增加的复杂性这一问题，那么当前项的数字指示器可以是支持跳转至任意项的对话框，还可增加跳转至顶部或尾部的对话框元素，同时可增加能切换至完整的清单/索引/目录视图的元素。

基本原理 通过给用户一个简单的导航模型并允许用户直接到达下一项或前一项，用户不需要到返回到索引去选择下一项。这种解决方案提高了执行速度和满意度。

示例



示例是 Sun 的 Java 在线教程。用户无需返回索引再进入下一主题，页面上方有导航条。更佳的是索引也在那里，至少是部分的（显示了当前项的上下文环境）。

已知应用 Atlas F1 新闻网站，NT 事件查看器

相关模式 向导，空间导航

连 续 筛 选

问题 **用户需要在一个有序的集合里寻找一项。**

可用性原则 即时反馈（反馈）

场景 该模式允许用户通过连续筛选方式，并根据即时反馈而动态缩小搜索范围。

约束

- 用户正在有序的较大集合里搜索一项，但可能不熟悉确切的项，也不确定该项是否存在。
- 用户搜索某项，但搜索条件可能导致多个结果。

解决方案 提供筛选组件，通过它用户只对自己感兴趣的数据项进行实时筛选。

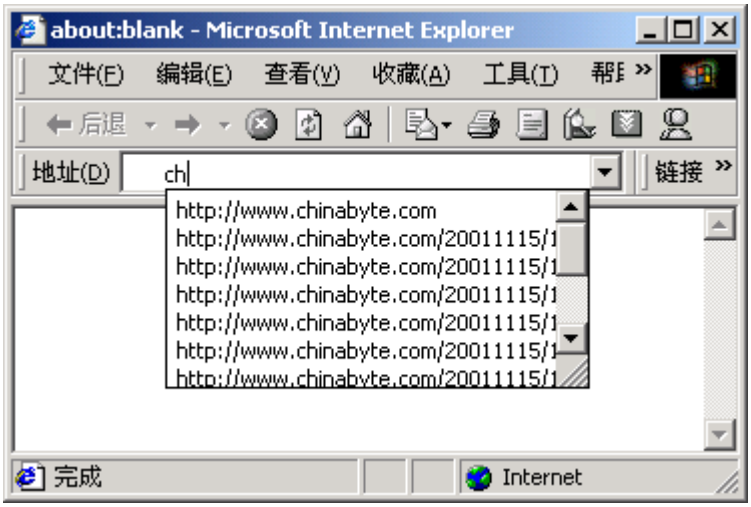
从用户开始输入搜索条件那一刻起，筛选出的集合根据搜索条件同时显示出来。如果相关，则最接近的匹配项被高亮显示，同时一些靠前或靠后的项可能也显示出来。

基本原理 因为用户获得了对搜索条件的即时反馈，用户的搜索非常有效，甚至还可能看到相关的项。因为筛选出的项显示出来了，用户可实时调整搜索词，甚至无需完成整个搜索词就可以直接获取想要的项。这种解决方案改善了执行时间和满意度。

示例

该屏幕截图来自 Microsoft Project 的帮助窗口。在索引功能中，用户键入字符的同时，清单中显示对应的条目。

示例



该屏幕截图显示 Internet Explorer 5 里的 URL 历史记录。当你键入 URL 时，它显示出匹配未完成 URL 的可能清单。

已知应用 帮助系统（Cakewalk 9，Microsoft Project 2000，Visual Studio 6），Internet Explorer 5，Visual Studio 代码编辑器的 IntelliSense 功能

相关模式 收藏夹

UMLChina讨论组

昵 称： umlchina

密 码：

注册 忘记密码 µÇÂ¼

搜索UMLChina

ËÑË÷

最新>>

新增答疑专家 NEW

推荐《人月神话》 NEW

增加XP栏目

更新各栏目论坛精华

1月23日与Martin Fowler交流实录>>

最新招聘

广州某软件公司

北京软通动力科技有限公司

更多>>

导航

《非程序员》：UMLChina发行的软件工程杂志，下载量过万

杂志下载 征稿启事 NEW 来稿情况

专家解疑 ：世界级专家为您解疑，点击人名进入各专家的答疑板。提问请先知道 提问建议

Kent Beck Alistair Cockburn Mark Paulk Roger S. Pressman NEW John Vlissides 高焕堂 钱五哥 叶云文 .....

书籍推荐 ：UMLChina推荐的软件工程书籍

UMLChina推荐书籍 国内书籍信息 国外书籍样章

文档中心 ：大量软件工程资料下载

OO和UML 用例&需求 建模实例 NEW 模式 NEW 工具 过程XP和轻量级方法 组件 NEW 交互设计 测试 物件导向杂志 PMT评论

论坛精华：UMLChina讨论精华

行业动态 OO和UML 模式 工具 过程组件 交互设计 国内书籍 嘉宾交流

服务中心：

Architect编辑翻译组 OOAD培训，培训资料下载

糟糕界面集锦—控件篇

iarchitect 整理，bucher 译

在图形界面中，控件就是程序与用户之间沟通的桥梁，而这座桥梁的好坏则取决于如下两个方面：

· 控件是否符合需求

· 控件之间是否风格一致

《Visual Basic Programmer's Journal 101 Tech Tips for VB Developers》的编辑正在减少给他们读者的技术提示，“ 技巧 ” 第 100 条展示了如何使用代码来使复选框代替单选框：

复选框和单选框很相似，但是它们之间有一个很重要的区别：对于单选框，用户一次只能选中其中一个，而复选框则没有这个限制。

只要一小段代码就可以使用户一组复选框中一次只能选中其中的一个。如果你想使用复选框来代替单选框，这是一个很好的方法。

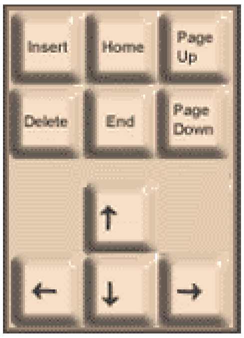
我们认为在这段文字下面还应该加一句：

如果你想让你的用户摸不着头脑，这是一个很好的方法

单选框用来给用户选择一些互斥的选项，复选框用来给用户选择一些允许重复选择的选项。复选框的外形就告诉了用户 “ 这是可以多选的 ”。**绝对不要**在需要用户选择互斥选项的时候使用复选框。

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

读者 **Cory King** 问了一个很好的问题：为什么 Adobe Acrobat 要修改导航键的定义？习惯上我们认为按下<PgUp>,<PgDn>后应该进行上下翻页，按下<↑><↓>键应该使页面上下移动。奇怪的是这个软件却把这两组导航键的功能弄反了。



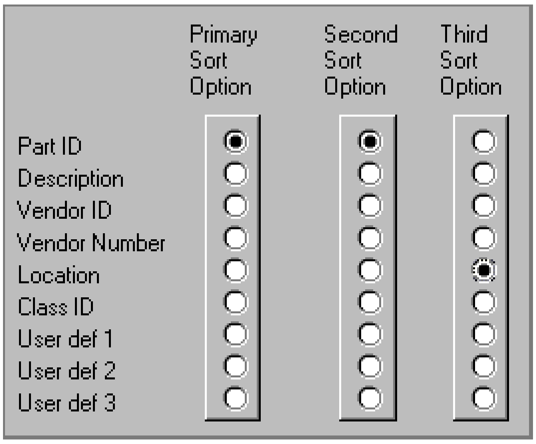
【译者：我用 Acrobat Reader 5.0，这个问题已经被修正。】

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

“ 这是智力测验吗？ ”

在回顾一份关于 *better-left-unnamed* 公司出品的软件性能评测报告时我们发现了这样一个奇怪的设计。这个对话框是用来让用户选择报表中字段排序的顺序。从图中可以看出，用户可以选择 3 个排序的字段。问题是用户必须指定 3 个排序的字段，因为你没有选择余地。正如图中显示，对结果进行"1.Part ID 2.Part ID 3.Part ID" 排序，这种高明的点子看来只有这位天才程序员想得出来了。

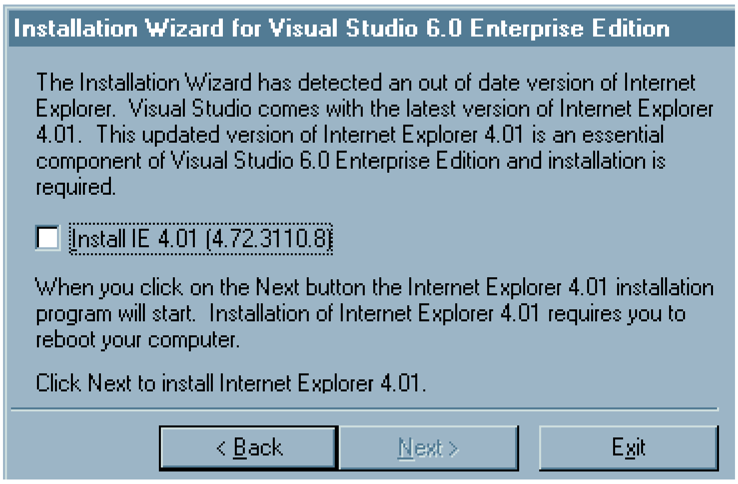
BTW：有谁还记得初中时代体育馆里的那种攀岩训练壁？



【译者：我倒觉得像弹珠台】

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

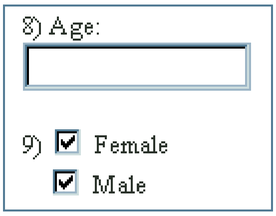
Grrrrr.在安装 VisualStudio 的时候，安装程序提示该软件依赖于 IE,必须安装 IE4 才能正常使用。不过我们觉得它提示的方式有点......愚蠢！既然必须安装 IE，为什么还要使用复选框来询问是否需要安装 IE？**你没有选择的机会！**不选这个复选框就会使 NEXT 按钮变灰，使我们无法继续安装。



【译者：我认为根本不用给用户选择，直接安装即可。有谁会买了 VS 却拒绝安装 IE 的？】

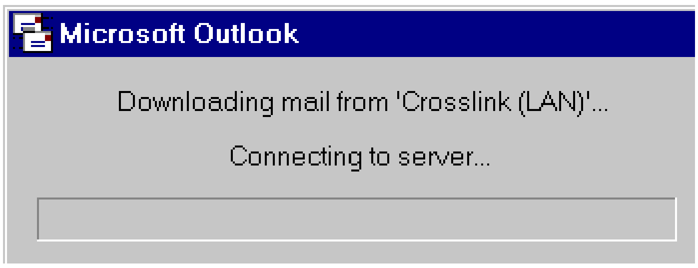
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Ryka 家族，一家女鞋生产商，他们希望能够抓住每一个潜在客户。有趣的是，尽管他们可以用单选框来让客户选择自己的性别，他们却使用了复选框。这样造成了潜在客户有了 4 种性别：男性、女性、双性、无性。我们觉得这和他们的座右铭 “ 女性出品、专为女性 ” 相映成趣。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

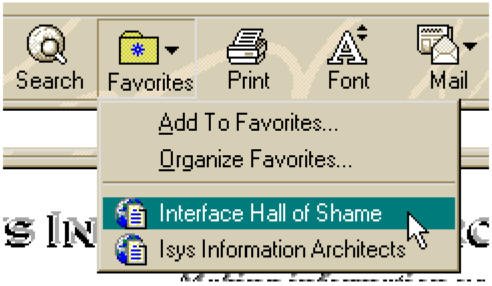
John Winters 给我们发来了一些关于在 Outlook 中错误使用进度条的图片。当接收 1 条消息（邮件）的时候，程序显示的进度条看上去像是工作指示灯：它指示了程序是正在工作、或者程序已经停止工作。用 Modem 接收一个 340KB 的附件时需要耗时 1 分 45 秒。在这段时间内，进度条一直处于 “ 满 ” 的位置，而不是显示取数据的进度（进度条常做的事情），用户看到这个界面最有可能的反应就是计算机死机了，需要使用 “ 三指礼 ” 来关闭程序。



【译者：OfficeXP 已经完全解决了这个问题】

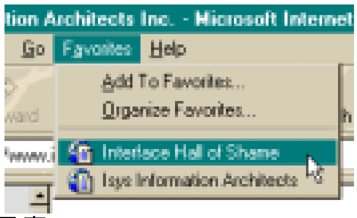
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

这是微软 IE3（从工具条背景可以看出是 IE3）的一个别出心裁的功能：在我们将站点添加到收藏夹的时候提示我们把这个站点添加到某一个分类？！**愚蠢的设计！**工具条就是用来快速访问经常使用的菜单项，微软却把它用来显示……菜单？！



【译者：我认为这是 MS 的一个创新，现在大家想必早已习惯了这样。】

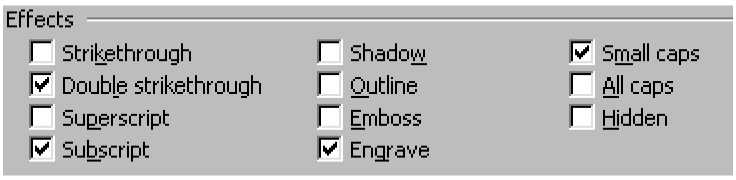
如右图所示的收藏夹菜单，收藏夹工具条按钮图片是与菜单上相同的。他们（微软程序员）这样做并没有给使用带来方便。事实上，由于破坏了控件规则：“ 工具条按钮不再启动另一个动作 ”，反而降低了工具条的功用。某些 “ 特殊的 ” 工具条按钮需要用户从一大堆五花八门的按钮中做出选择，而且可能需要用户作出后续操作。这么做对设计毫无益处，我们称之为…哦，愚蠢。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

非常感激 **Gordon Allison** 提供这个样本。

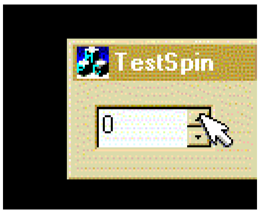
很高兴通知你们各位，微软的界面设计原来是由艺术家而不是界面设计师来完成的。



在 **Word 97** 的字体对话框中用户可以用一堆复选框来选择字体属性。这没什么问题。但是！其中有 4 对选项是互斥的：删除线、双删除线，下划线、双下划线，阴文、阳文，小型大写字母、全部大写字母。这些控件看上去是复选框但实际上却是单选框。显然，使用单选框将影响控件群的整体美感（所以那些家伙放弃了布局的合理性）。

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Mark Otway 发现微软的 VC++ 5.0 提供了一个有趣的微调控件。微软的这个控件彻底颠覆了用户以前的经验：用户必须点击下箭头键来增加数值，同样的，点击上箭头键来减少数值。这迫使那些尽职的程序员写一个程序去反转这个控件的行为。Mark 提道："微软所提供的这种'帮助'扰乱了那些开发人员对于规范界面的理解"。

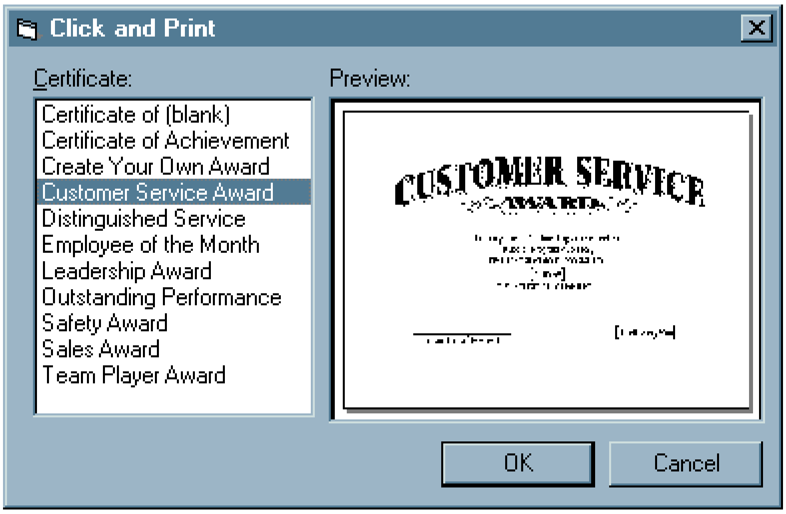


※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Click & Print Certificates 是一个用来打印证书和奖状的共享软件。该程序提供一个"样式伙伴"来帮助用户从一大堆证书样式中选择一个打印。如果你使用它，你会发觉有许多问题：(使用简介是右对齐、垂直对齐的。取消按钮在确定按钮之前。说明中提到的"OKEY"按钮实际上应该是"OK"按钮。还有那根恶心的绿线……)，还有一些严重关于选择样式的设计错误。

滚动水平条可以在各种样式预览之间切换。每单击一次可以切换一种样式。这里面有两个大的问题：首先，我们不知道共有多少种样式可供选择，只能一个一个看过来；其次，要想选中一个样式只能通过单击鼠标一次一次地翻页来实现，举例来说：要想选中"Team Player"就必须单击滚动条 9 次，等待它读入 9 次你不想看的预览画面。

我们设计了下面的界面，我们相信这个界面可以解决以上的问题。

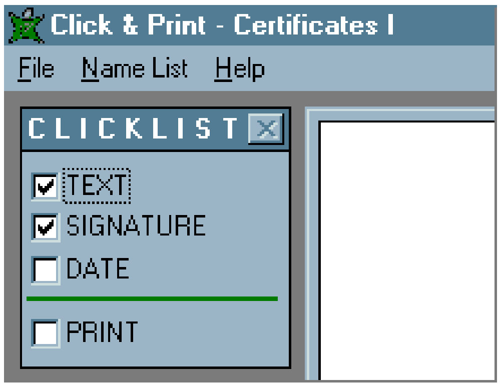


※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

复选框通常用来提供多重选择，*Click & Print* 把复选框当作状态指示器和命令按钮来用。当用户在 *Click & Print* 上单击一个复选框时，会弹出一个对话框要求用户输入要在证书上打印的信息。只有在用户单击 OK 按钮后该复选框才会被选中。

设置证书日期更是与众不同，程序默认证书日期是当天，但是如果你想选中日期复选框只能首先选中它，然后在弹出的日期选择对话框中单击 OK。

这个界面设计中最突出的问题是当用户单击"Print"复选框时会弹出一个错误警告："还有选项没有完成，你确定要打印吗"？



这是从 GUI 101 中得到的教训：

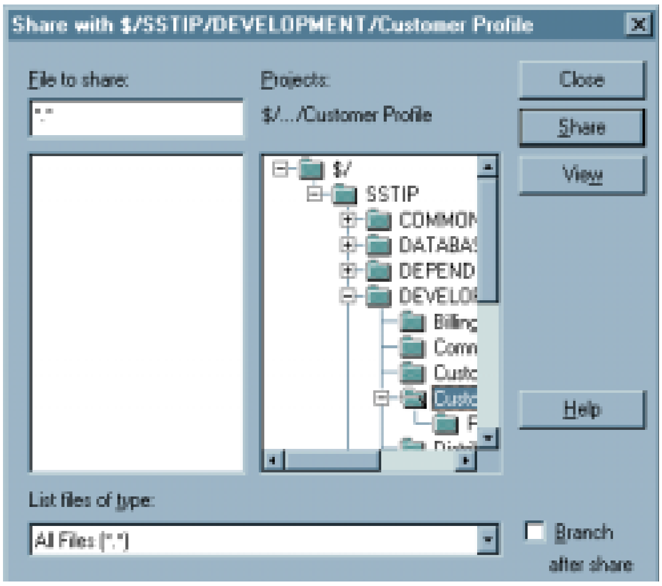
- 复选框应该用来设置某一个选项的"开"和"关"
- 命令按钮应该用来触发某一个动作

不要因为你对界面控件理解的混淆而把用户的脑子搞乱

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Microsoft's *Visual SourceSafe 5.0* 的开发者似乎对于 Windows95 中的通用文件对话框不甚满意，于是他们自己写了一个基于 Windows 3.1 的通用文件对话框。不幸的是，新的对话框结合了这两者的缺点。

如右图所示，当用户扩展目录树时，他（她）必须用鼠标把目录列表右滚来看清目录结构。这样就使得用户无法看清当前的目录层次。本来似乎通过 Project 标签可以看到当前的目录层次，但是开发人员在这里隐藏了目录路径，只显示最后一层的路径。为了确定当前的目录层次，用户必须使用鼠标左滚目录列表框。由于目录列表尺寸太小，开发人员迫使用户在选择目录时反复地把鼠标拖来拖去。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

这里是一个使用错误的工具来工作的经典案例，用户不得不与程序中的种种不便做斗争。*OzWin II* 是 Compuserve Information Service 的离线阅读器。该阅读器有一个不错的功能：你可以把你的签名加入到你的 Email 信息中。就像这样：

Brian Hayes

Isys Information Architects Inc.

http://isys.home.ml.org

不幸的是，OzWin 只提供了一个单行文本框来输入签名。如果你想要在签名中换行，就必须使用 OzWin 提供的特殊字符。在文字中使用 **^M^J** 作为换行。当然，你必须首先阅读帮助文件才能知道 (a)签名中可以换行 (b)如何在签名中换行

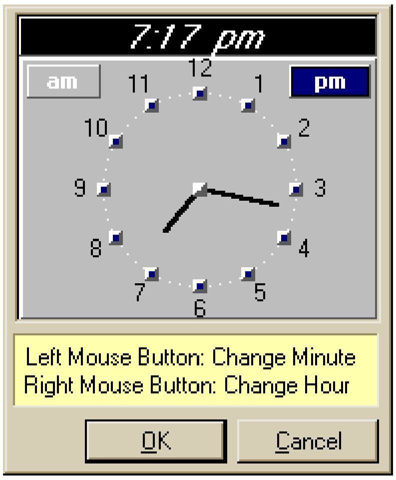
如果使用多行文本框就可以完全解决这个问题。

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

这里大多数人都已经使用电脑一段时间了，大多早已习惯了使用键盘来输入日期。出乎意料的是，在 Automate Pro 中试图输入日期时，我们发觉尽管我们可以看到日期数据，也可以选中它们，但是控件就是不理会我们在键盘上的输入。

Automate Pro 让用户单击 Set Time 按钮，然后在他们自己特别制作的“时钟”控件上选择日期来取代直接使用键盘输入。要不是窗体上有说明，恐怕没有人会知道如何使用这个奇怪的控件。

Automate Pro 强制用户使用他们自己的方法来选择时间（比如使用这个时钟控件）。如果可能的话，用户应该可以自己选择输入时间的方法--他们早已习惯的那种方法。

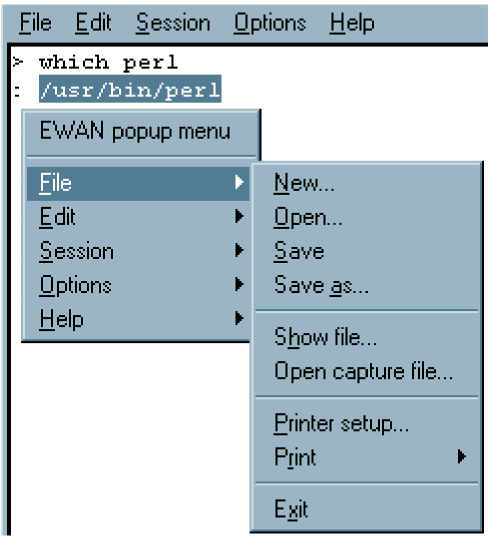


※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Ewan 是一个我们的 ISP 提供的虚拟终端软件（共享软件）。下图展示了“Ewan Popup Menu”（感谢上帝，开发人员在菜单上面放了一个标题，否则的话我们真难说出这到底是什么)在主窗体上用鼠标右键单击就会弹出这个菜单。问题时，就如我们所见，该菜单提供了过多的无关的功能，这使得用户很难找到他们真正想要的功能。

对于主菜单来说这个弹出菜单根本是多余的，它需要你反复的移动鼠标来对准选项。我们不能理解用户怎么会忍受这个。

本次实例中，我们提到的程序是 1995 年 1 月 17 日的。我们希望 Ewan 的开发人员能够提供更多的上下文关联菜单。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

Maxis 出品的*模拟城市 2000* 毋庸置疑是一个让人着迷（或者说，上瘾）的模拟游戏，但是对于新上手的玩家来说有一个地方实在让人心力憔悴。工具条上的部分按钮（仅仅是一部分！）居然包含有子菜单。而且只有在用鼠标在按钮上按住一段时间后才会弹出。

在快速上手指南中并没有提到这一点，而且工具条上的图标也没有什么特殊的地方。当我的同事试图建造一个电厂的时候就遇到了这个问题：帮助文件仅仅告诉他单击“ Power ”按钮，他照着做了，结果却发觉这样只是往地上放了一些输电线。在经过一系列的失败和尝试后，他终于发现如果在这个按钮上按住一段时间鼠标，就会弹出一个包含“ 电厂 ”选项的子菜单。

这个设计中的一个最大的失误就在于在工具条按钮上一次普通单击会执行上一次的操作，但是工具条上并没有任何图示表明上一次是什么操作。结果导致用户一疏忽就容易操作错误，或者每次操作都打开子菜单来选择，最终降低了操作的效率。

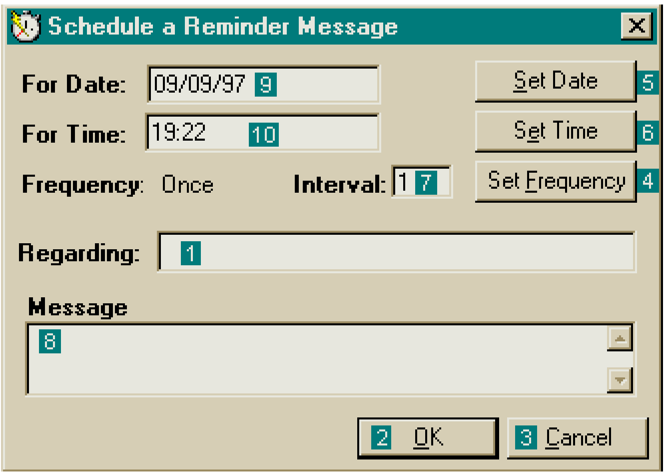
【译者：以前我也遇到过与这位作者的同事一样的问题，当时还是付钱在 SFC 上面玩。】

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

对制作一个图形界面程序的步骤还有疑问吗？在 *Unisyn's Automate Pro* 中这个问题特别明显：开发人员显然忘记了 TAB 键的作用是在控件之间遍历这个常识。作者使用了开发工具自动生成的控件次序(Tab Order)。程序的界面如下图，需要说明的是，开发人员显然首先创建了 Regarding 文本框，然后加入了 OK 和 Cancel 按钮，接着是 Set Frequency 按钮……。

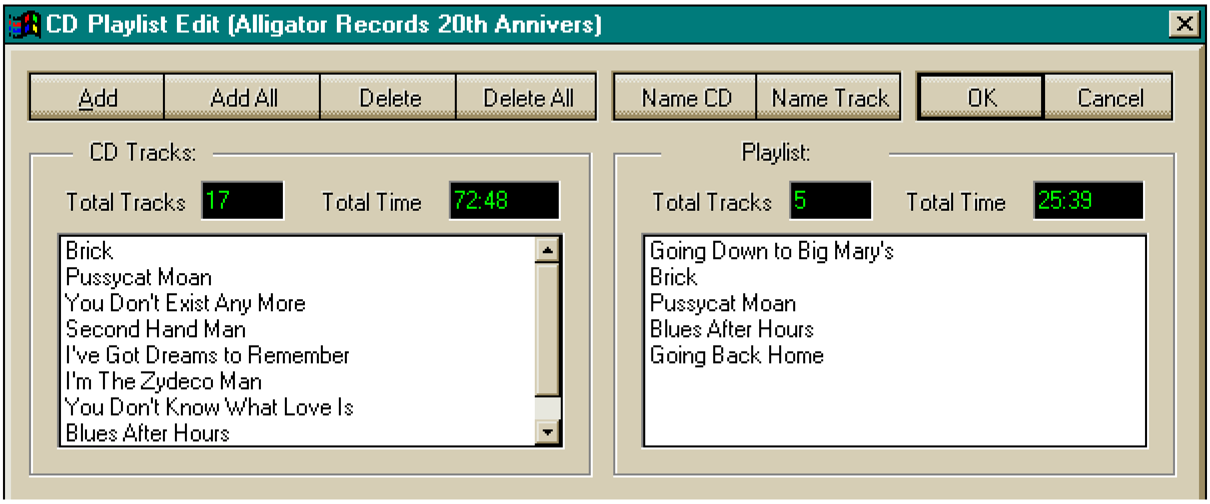
这样设计的结果就导致用户必须按 TAB 键 7 次才能从 Regarding 文本框移到 Message 文本框。

也许他们会在 3.8g 版中修正这个问题



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

在界面设计中，“ 一致性 ” 是首要的原则。IBM 的一个 CD 播放器 *AudioStation* 显然就违反了这条原则。该程序有一个播放列表功能，用户可以在这里面定义歌曲播放的顺序。上图就是播放列表的截图。它有两个列表控件：CD 音轨列表（左），播放列表（右）。



在该设计最突出的问题是两个播放列表行为不一致。例如：在音轨列表中双击一首歌曲会把它添加到播放列表中，可是，在播放列表中双击却不会删除该歌曲。另外还有，音轨列表可以复选，播放列表却一次只能选择一

次只能选择一首歌曲。以上这些使得该软件得以入围我们的糟糕界面评选。

在播放列表中还有不少设计问题，我们已经把它们列在下面了。换一种方式看看这张图片，看看你能看出几种（其中一些问题只有在使用中才会发现，IBM 真该在发布软件之前运行一下程序）。

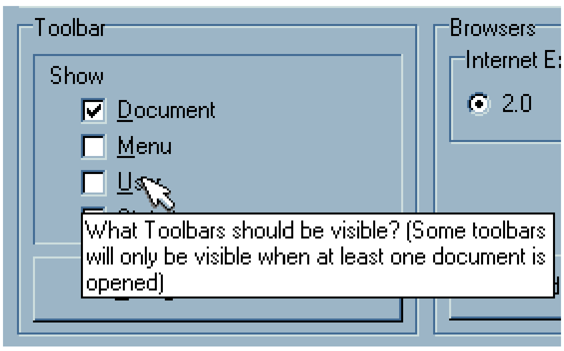
用一面镜子就可以看到这里究竟写了些什么：

1. Command buttons placed at top of window rather than the bottom
2. The Add button is the only control that has a mnemonic access character
3. The Add button only works when the Track list has focus, but is not disabled otherwise
4. 3D font decreases readability
5. No means by which to reorder songs within the Playlist.
6. Name CD function accepts more characters than it will display, then truncates input unnecessarily.
7. Add button does not become the default when clicking on Track List item to indicate the double-click action.

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

通常来说提供上下文关联帮助是一件好事情。但是在 *WebEditPro* 中，这种帮助却帮了倒忙。从好的方面看，*WebEditPro* 几乎为所有的控件（还有一些其他界面元素）都提供了工具提示。但是在另一方面，这些帮助总是自动弹出，无法关闭。这种帮助对于第一次使用该软件是有益的，但是对于熟练用户来说这样无疑令人恼火。用户不得不不停地移动鼠标来去掉那些帮助。

界面设计很重要的一条就是必须让用户感觉他们能够随心所欲地使用（控件）。*WebEditPro* 剥夺了用户支配控件的权利。结果使得用户感到恼火并且开始使用别的软件。没人喜欢一个唠叨的软件，特别是在他（她）不希望被打搅的时候。提供帮助是一件好事，但是应该在用户提出请求之后。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

此处展示的控件，在类型选择上是合理的，但是在单位设置上有所欠缺。在 Microsoft *Internet Explorer* 3 中设置历史文件尺寸时，用户可以使用一个滑动条来选择存放缓存文件的尺寸。问题在于：缓存文件的尺寸必须是一个占总磁盘空间的百分比数。几年前，这不是一个问题，但是自从我们从 **Ross Cormier** 处收到了如下的抱怨后，我们认为 Microsoft 有必要考虑一下这个问题了。

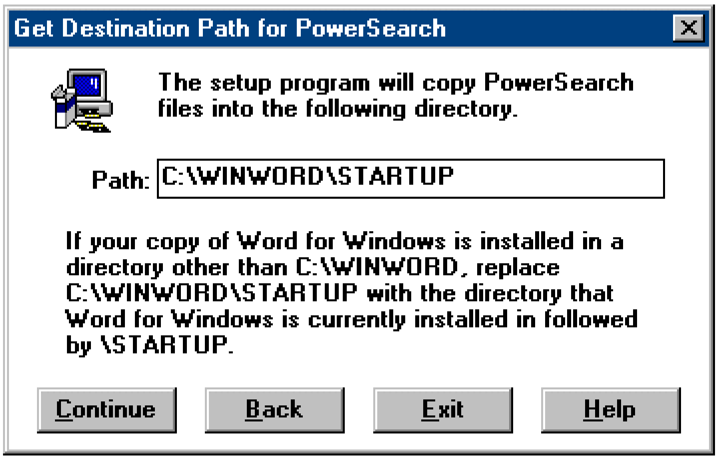

在 IE3 种设置缓存文件的尺寸时只能设置成磁盘总空间的百分比，最小的是 1%。我有一个 4G 的硬盘，我不想缓存文件有 **40MB** 大，多谢了！

用户控制是一个需要考虑的要点

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

安装 *PowerSearch* 时会要求用户输入程序的安装路径。用户通常来说总是选择默认安装路径，然后安装程序可以继续安装。不幸的是，如果用户希望把程序安装到其他目录，他（她）将不得不面对一系列问题：由于该界面没有提供“浏览”能，用户不得不把目录名记在脑子里，或者使用其他的软件来得到路径。不管用哪种方法，用户都必须手工输入路径，增加了错误的可能性。

这个界面能够入选我们这辑糟糕界面的一个原因是：这个程序**必须**安装到 Word 的 Startup 目录下；如果用户选择了其他目录，*PowerSearch* 插件就无法运行。这就要求用户必须知道确切的安装目录，因为程序并不会校验用户指定的目录是否有效。其实只要稍微加上几行代码，程序就可以自动探测出目标目录，避免用户输入错误路径。



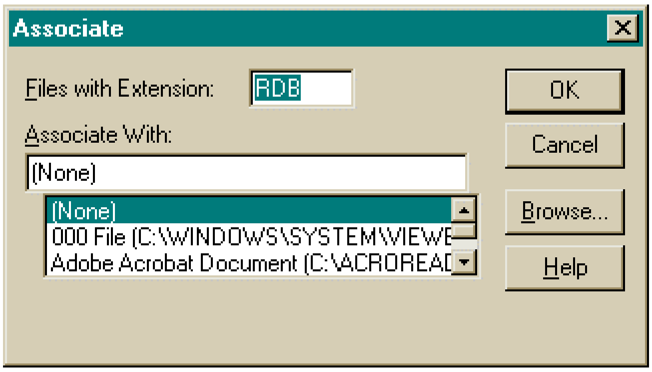
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

不管我们学习了多少关于人机界面的知识，微软总有办法发明出新的降低控件效率的办法。这是在 IE 的"认证"功能中使用的列表框。人们在阅读时，“ 从上到下 ” 的阅读速度要快于 “ 从左到右 ”。但是尽管这是人人皆知的道理，而且在 Windows 中一直使用垂直列表框。微软还是决定在这里使用水平列表框。与垂直列表框不同的是，水平列表框重新定义了列表框，单个项目在水平列表框中的卷动发生了很大的外观上的变化，导致用户必须不得不 “ 竖着看 ”。最不幸的还是别的开发者看到微软在使用这种列表框，就认为既然微软在用，那么就肯定是好的。错！

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

目前我们还无法确定该把这个问题划到哪个类别中，但是如图所示，在一个列表框中只显示 3 个项目无疑是愚蠢的。

这是微软的文件管理器提供的文件关联界面，用户需要把某一个特定类型的文件与一个应用程序关联时会弹出该对话框。如示例：在列表中共有 133 个项目；一次单击滚动条只能滚动两个项目。我们不明白为什么开发者不能把列表框做得大一点，或者把窗体做得大一点？由于开发者对列表框设计的随意导致该控件几乎无法发挥作用。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

也许你认为这个界面看上去并没有什么不妥，不过看来微软找到了降低列表框效率的方法。很明显开发 VB5 的 “ 以用户为中心 ” 的开发者特地避开了下拉列表框，他们使用了两列的列表框来代替。

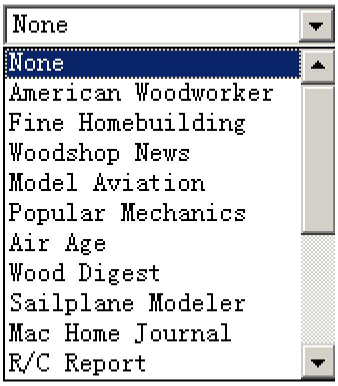
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

在浏览一个网页的时候，我们第一次看到不恰当地使用控件。下面还展示了我们第一次看到的不恰当使用控件的桌面程序。

请选择您的爱好：

（请单击测试）

看看我们自己做的样例：



（我们发觉有的浏览器显示控件的样子和我们当初设计的时候不一样。这里有一张这种情况静态图片）

想象一下卷动这个有 30 项的列表，一次一项！

有趣的事，两种方法都在同一个窗体中出现。

问题在于：当然啦，第一个（“ 走马灯下拉框 ” ？）需要很多的用户操作，迫使他（她）在这上面花上比一个标准下拉框更多的力气，去选中他（她）想要的东西。普通的下拉列表框方式有利于我们人类快速的查找信息，因此我们可以快速的找到我们感兴趣的东西（不必把精力放在不感兴趣的东西上）。

之所以使用这种设计，估计可能是有些浏览器对于包含大量数据的列表框支持不好（我记得 *Netscape for Unix* 就是这样）不过我们觉得使用像右边这种标准的、能够限制一次显示项目的数目、普通列表框倒是一种折衷的方法。

我们发觉 “ 走马灯下拉框 ” 居然可以复选。我们建议你试试（复选），再比较一下使用列表框复选，你觉得哪一个直观？

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

想要让自己的程序别具一格，正是出于这种被误导的动机。IBM 的 *Aptiva Communitations Center* 开发者决定不使用 Windows 自己的控件，用自行开发的控件取而代之。他们非常成功地做到了这一点：该程序看上去与其他 Windows 环境下运行的程序完全不同。不幸的事，对于那个程序而言，下图所展示的仅仅是该程序的一小点“特别”。

不同于我们的高中生涯，在图形界面设计中，“从众”是非常重要的。如果所有的程序外观和操作都类似，用户就能够很快上手（因为他们可以借鉴别的程序的操作经验）。当你的程序外观和操作都与众不同，你的用户不得不花更多的时间去学习使用它。刻意求新从来不是什么好事情。只需回忆一下你坐在租来的车里，在这辆奇怪的车里满地找变速的情形（现在的汽车工业对此类问题解决已经非常完善，在七十年代你绝对不会知道变速在哪个角落里）。

【译者：Control 原指控制器的意思，由于我不知道“Control”在汽车行业中的意思，所以这里暂时翻成了“变速”。如果有谁知道确切意思，还请指正。】

用钻石型来代替圆形的单选框只会让用户感到更多迷惑。甚至于会造成更大的麻烦：单选框看起来就像命令按钮；单击一个（单选框）会跳到另一个对话框。更令人困惑的是，这个程序混淆了标准单选框和自定义单选框（从外观到行为）。

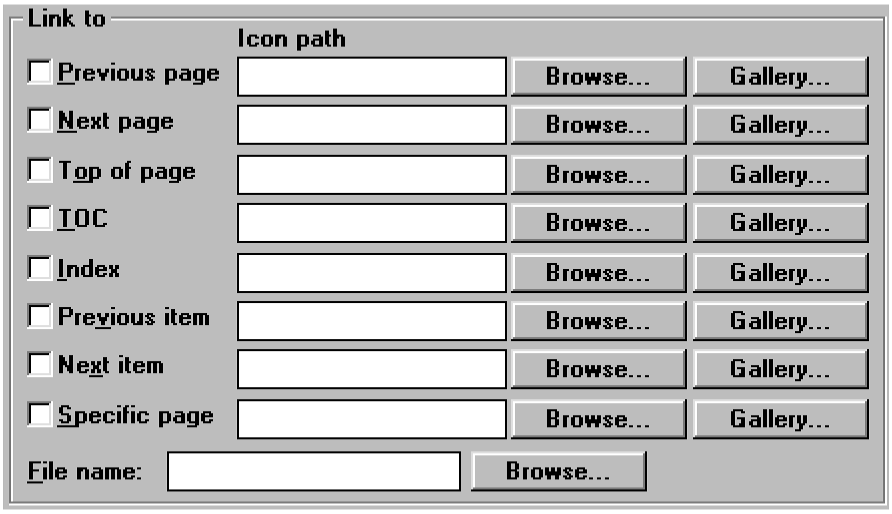
让我们做得更好，而不是更特别！

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

HTML Transit 的作者显然对于命令按钮有一种偏好。到处都是命令按钮，整个窗体看上去就像一个兔子养殖场。甚至有一个窗体，其中用了 15“Color…”按钮。

下面窗体是用来选择 WEB 页上导航按钮图片。所有的“Browse…”按钮都是让用户指定一个文件路径。同样，全部的“Gallery…”按钮都是用来在程序的剪贴画辑中选择。

也许我们不该说这个，不过当你使用该程序时。看到 25%的界面都被命令按钮覆盖，这也太多了吧。在同一个窗体上不应该出现两个相同的按钮，9 次也不行，15 当然更不行。



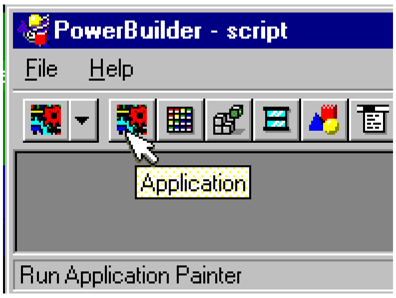
※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

GUI 的一个设计基本准则是：如果控件被禁用，那么就应该表现出来。这样不管是菜单项、命令按钮、列表框、下拉列表，我们只要一瞥就可以知道是否可用。奇怪的是为什么微软在按钮被禁用时不把它显示出来？这幅图片来自 *Office95* 系列中的 *Word*，但是这个问题出现在整个 *Office95* 中。

我们都知道微软的开发人员有能力去改变工具栏的按钮图片；修改按钮图片也不会影响什么。我们估计可能是微软的美工不喜欢图片变灰的样子，而不管这样会给用户带来额外的（该按钮被禁用）信息。对于用户来说很不幸，微软的界面设计似乎更像是由图片美工而不是界面设计者来完成，所以不必对微软经常能够入选我们的糟糕界面选集而感到惊讶。

※※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

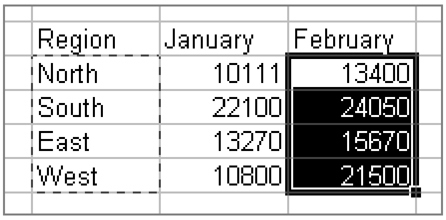
PowerBuilder 的开发界面上为所有的工具按钮提供了工具提示，这很好。它也为所有的工具按钮提供了状态栏提示，这也很好。工具提示正如它本来的那样，是延迟出现的。问题在于状态栏的提示也是延迟出现的，程序首先在工具提示显示部分的提示然后再在状态栏显示完全的提示。按照规定：工具提示应该是延迟 1-2 秒出现，但是状态栏应该是实时更新的。



※※※※※※※※※※※※※※※※※※※※※※※※※※※※※

从“编辑”菜单中选择“剪切”后会发生什么？选中的文本或者对象会被删除（复本被存放到剪切板中）。这条规则如此的根深蒂固，以至于“剪切、粘贴”应该被当作文化传统来尊重：几乎每个 4 岁以上的人都知道是怎么回事。

不幸的是：Microsoft *Excel* 的开发这似乎并不熟悉这条规则。对于他们来说，“剪切”意味着“让它留在那里”，或者至少是：“在我粘贴之前，让它留在那里”。在选择“剪切”之后，选中部分周围会出现一个选择框显示该部分被选中。直觉告诉我们，什么也没有动过！在“粘贴”或者别的什么操作之前，“剪切”并没有删除任何东西。在 Excel 中，“剪切”仅仅意味着“划出你准备移动的部分”。而且事实上，在完成一次“剪切、粘贴”操作后，对应的撤消命令是“撤销移动”。



Region	January	February
North	10111	13400
South	22100	24050
East	13270	15670
West	10800	21500

由于修改了基本规则，每个 Excel 新手（或者只是偶然用用）在使用“剪切”时都遇到了麻烦。通常就是用选择了“剪切”。

？？？？没反应

我再“剪切”

我又“剪切”

我“剪切”……

却不知道其实早就完成了操作。

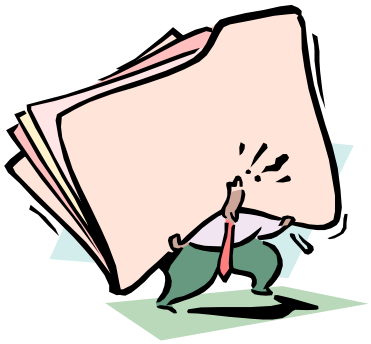
“雷德蒙（微软总部）里的那些家伙正因为这个被人嘲笑”

感谢 Steve Bliss 提醒我们 *Excel* 的这个特性。

(原文最后更新于 1999 年 9 月 19 日)

UMLChina.com

文档中心 ▶▶



资 料	介 绍
行业动态	欢迎提供软工新闻，UMLChina 最近活动
	综述、Use Cases&需求
建模实例	实例教程, .mdl 文件，文档实例....
模式	
工具	Rose...等各种软件工程工具介绍
过程	XP
组件	COM, CORBA, EJB，N-Tier...
交互设计	人机界面的工程学
测试	
书籍	国内出版信息、英文书籍样章
OOTW	物件导向杂志（台湾）简体版
PMT 评论	关注软件质量的电子杂志

UMLChina 书籍推荐

《设计模式精解：面向对象设计的新视角》，Alan Shalloway，Jim Trott 著，透明 译，清华大学出版社，预计 2002 年 3 月出版。

“这本书最大的优点之一就是它用类推的方式将概念解释得非常清楚，而很少使用程序范例。我正在做一套关于 OOP 和软件开发的录音教材，这本书讲述概念的方式对我非常有启发。”

——Bruce Eckel

UMLChina讨论组

昵 称： umlchina

密 码：

注册 忘记密码 µÇÂ¼

搜索UMLChina

ËÑË÷

最新>>

新增答疑专家 NEW

推荐《人月神话》 NEW

增加XP栏目

更新各栏目论坛精华

1月23日与Martin Fowler交流实录>>

最新招聘

广州某软件公司

北京软通动力科技有限公司

更多>>

导航

《非程序员》：UMLChina发行的软件工程杂志，下载量过万

杂志下载 征稿启事 NEW 来稿情况

专家解疑 ：世界级专家为您解疑，点击人名进入各专家的答疑板。提问请先知道 提问建议

Kent Beck Alistair Cockburn Mark Paulk Roger S. Pressman NEW John Vlissides 高焕堂 钱五哥 叶云文 .....

书籍推荐 ：UMLChina推荐的软件工程书籍

UMLChina推荐书籍 国内书籍信息 国外书籍样章

文档中心 ：大量软件工程资料下载

OO和UML 用例&需求 建模实例 NEW 模式 NEW 工具 过程XP和轻量级方法 组件 NEW 交互设计 测试 物件导向杂志 PMT评论

论坛精华：UMLChina讨论精华

行业动态 OO和UML 模式 工具 过程组件 交互设计 国内书籍 嘉宾交流

服务中心：

Architect编辑翻译组 OOAD培训，培训资料下载

用 UML 描述工作流管理系统规约 摘 要

Pavel Hruby 著 *wind.deng*

统一建模语言（UML）为描述面向对象系统定义了一系列的标准符号。使用 UML 增强了领域专家、工作流专家、软件设计者和其他不同背景的专家之间的交流联系。UML 可以在普遍的场合使用，对工作流系统的用户而言很直观。 除了这些，UML 符号具有准确的语义，也就是说可视化的工作流描述可以作为软件规约。这一章侧重讨论了如何使用 UML 来描述工作流管理系统，如何跟踪从业务流程到面向对象软件设计的描述信息，如何用 UML 可交互工件来结构化项目知识库。

绪　论

这篇文章按下列方式组织：第一章，讨论了工作流产品的软件设计者和用户对一种通用语言的需要，第二章，介绍如何使用统一建模语言（UML）来描述一般的工作流概念，第三章，概述了如何把面向对象软件规约与工作流系统的描述联系起来。

下面描述了一个企业在实现新工作流管理系统时的通常情形：顾问与用户一起描述一个软件解决方案所支持的企业业务流程。开发队伍获得顾问的描述，但是他们很难理解业务术语，发现其中描述了太多的信息以至于难以用来实现此系统。开发者从技术观点来撰写系统规约，当把这个系统规约呈现在用户面前时，由于过于专业，以至于用户难以理解。然而为了进行下一步的工作，双方被迫接受了这个系统规约。

这种方式很容易导致系统无法达到用户的需求，因为用户、顾问和开发者通常都不是使用同样的语言，这样的交流问题导致难以保证业务流程所有部分都很好理解，并能转换为技术性的软件规约。另外，因为技术性的系统规约很难被使用此系统的实际用户所全部理解，使软件系统变得难以使用。

其挑战性在于能用一种既准确又友好的方法来为业务流程和业务系统建模。用来描述业务流程的每一个符号对用户来说很直观，并有确定的语义，因此开发者可以用它作为一般的描述，甚至用来精确的描述软件系统规约。

UML有着丰富和复杂的符号来描述软件系统。这些符号也许太丰富以至于不直观、不友好。然而，恰当地用UML来描述工作流管理系统有两大好处。首先，UML是软件界公认的符号标准；第二，UML也可用在不需要实现细节的一般场合。在下一节显示的UML图与那些领域专家已经在使用的图在直观上很相近，另外，它们的语义有精确的定义。如有必要，可出于软件设计的目的给同样的图增加详细的实现细节。

业务系统的描述由流程和静态结构的描述组成。流程最直观的模型就是一个活动或任务的序列，按照顺序完成以到达某个目标。因此，UML的序列图和活动图很适用于友好、准确、详细地描述业务流程，如组织图之类的静态结构，没有实现细节，可以用UML的静态结构图描述。图形化的实现细节往往会误导那些不精通UML的人。例如，导航箭头经常错误地表示方向，最好是仅用UML表示选项的某一特定子集。例如，把元素互相嵌套来表示组装比用实心菱形表示关联要好。用文字来描述各种属性，而不用UML符号，例如《refine》就比带三角形的虚线好理解得多。

工作流概念映射到UML概念

这一节介绍了用 UML 来描述工作流概念的例子。这仅仅提供了一个一般的指导，如何把工作流概念映像成 UML，详细的细节很容易从 UML 的语义和符号指南[7]得到。工作流管理系统的每个结构都能用合适构造型的 UML 符号来描述。

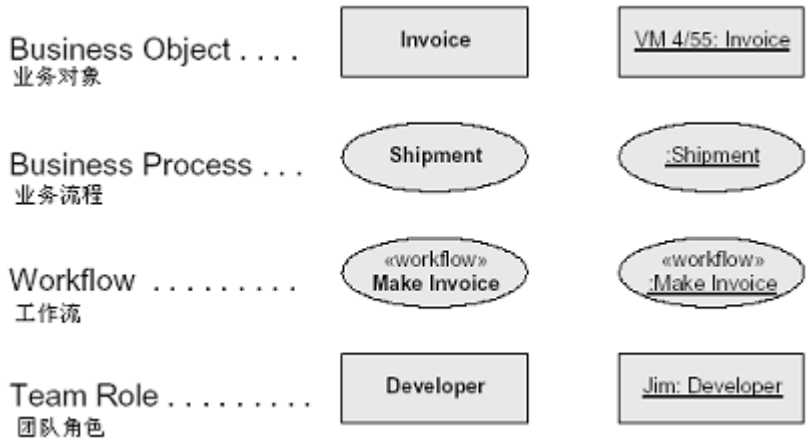


图 1　用 UML 表示业务对象、业务流程、团队角色

图1显示了用UML描述业务流程、业务对象、团队角色。业务对象在UML中用类和对象表示。类描述没有特性（identity）的业务对象，如发票(invoice)；对象描述有特性的业务对象，比如编号为VM4/55的发票(VM 4/55: invoice)。业务流程[1]用用例和用例实例来描述，用例根据目标、职责、前置条件和后置条件来描述；用例对象是具体的事件序列。工作流是自动化的业务流程，用带有构造型<<workflow>>的用例或用例实例描述。在UML中用类和对象描述团队角色（team roles），用类来描述团队角色的类型，对象描述担任该角色的具体工人（worker）。所有的符号可以用相应的构造型来修饰，如<<business object>>、<<business process>>和<<team role>>。每一个构造型都可以用文字或一个特定的图标表示。

注：1 本章介绍中，假设业务流程是在业务系统中的业务对象、角色和其他的实例之间协作完成的。请参看详细介绍"跟踪业务流程到软件设计"。

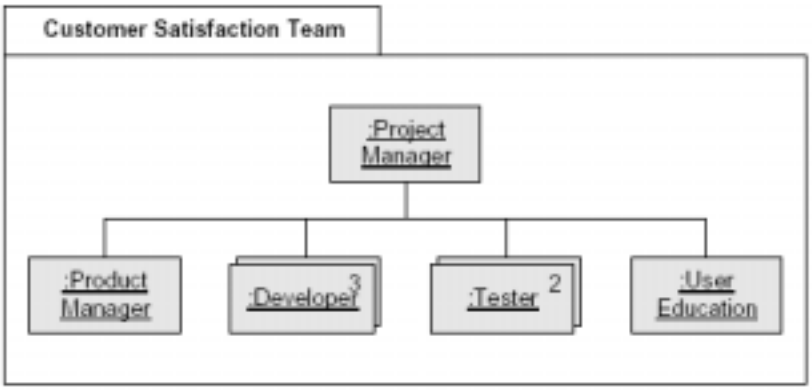


图 2　　UML 的静态结构图描述团队结构

图 2 显示了一个团队结构的例子。团队的角色用对象实例表示，为每个角色指定了雇员数量。一个客户满意团队（Customer Satisfaction Team）有三个开发员、两个测试员、一个产品经理和一个人担任用户角色。角色组叫做客户满意团队（Customer Satisfaction Team），用包符号表示。角色组也可能被表示成对象——作为角色的组装（composition）。如果团队表示为对象，团队和角色之间的关系为组装关系，那么根据 UML 元模型概念，一个特定的角色实例不能同时属于多于一个团队。如果团队表示为包，特定的角色实例可以同时属于几个团队。

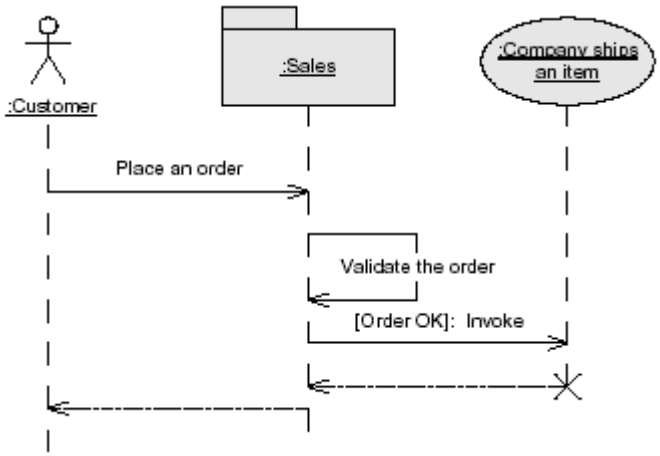


图 3　UML 序列图描述业务流程的实例

图 3 描述了业务流程的实例。角色客户(customer) 下了一份定单，然后销售(sales)部门中的某个工人确认此定单。如果定单有效，此工人调用另一业务流程“公司运输物品（company ships an item）”的实例。这个类型的图在 UML 表示法指南中没有明确的提到，然而，它符合 UML 的元模型。在对象生命线顶部的符号代表分类器角色，如图 3 中的角色、对象角色和用例角色。

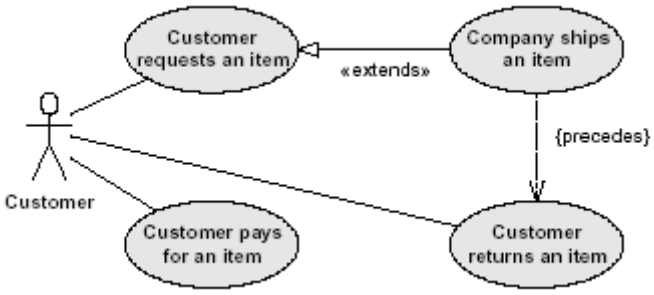


图 4　UML 用例图描述业务流程之间的静态关系

图4是UML用例图，描述了业务流程之间的静态关系。业务流程描述组织(organization)与角色客户（customer）的协作。注意在UML的1.1版本中，用例不能相互联系而总是由角色发送信号触发。这给建模环境带来困难，一个用例在运行期间，当特殊条件出现时，另一个用例也开始启动。在这种情况下，角色通过与另一个用例的联系初始化此用例而不需发出任何特定的开始信号。例如，如果客户的请求被评估为有效，用例公司运输物品（company ships an item）被组织中的对象触发。这个用例实例不直接由客户触发，希望下一版本的UML将减少用例间有关联系的限制。

UML用例图不容易表达出用例实例的顺序，例如，首先客户请求一项物品，然后公司将传送此物品，最后客户付款。一个解决的方法就是在用例间使用约束{precedes}或依赖关系<<precedes>>。类似的关系同样存在于OML（OPEN modeling language），参看[3]，Robert C. Martin建议使用关键字follows替代precedes，参看[6]。这样替代的原因是依赖关系<<follows>>与依赖关系<<preceds>>的指向相反，依赖关系<<follows>>指向通常的依赖方向——从依赖元素指向独立元素，至于哪一个更直观仍是个未解决的问题。然而，带约束或依赖的图仍然是静态结构图，并不描述特定场景。

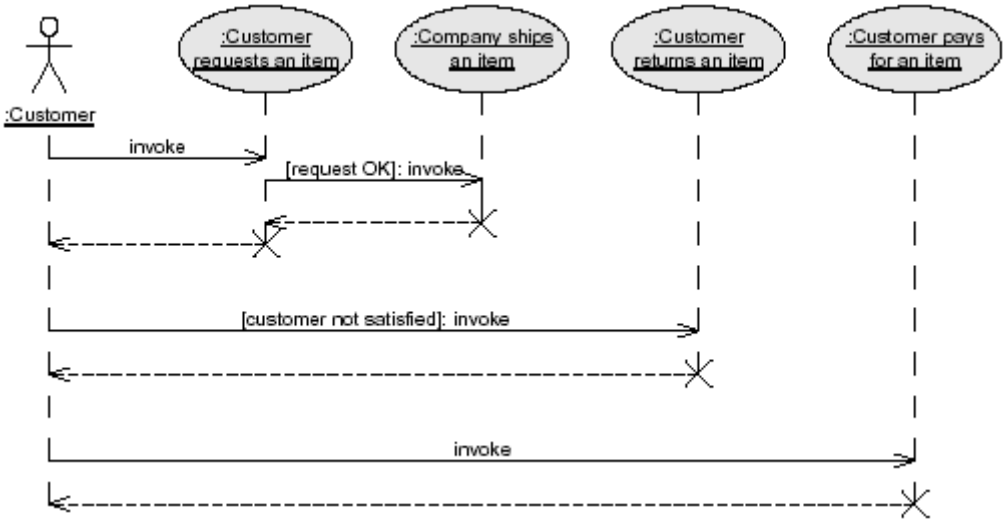


图5　UML序列图描述业务流程和执行者（Actor）之间的交互

角色可以通过特殊顺序启动用例的方法来使用系统。像这样的场景——用例实例序列——可以用顺序图或协作图描述，参看图5和图6。对照对象交互图，场景被描述为消息序列，用例交互图把场景描述为用例序列。这个图仅仅是由其他场景的实例组成的一个场景的UML图。在图5中消息调用（invoke）表示用例构造器和映射为从角色到用例的信号。根据每个用例的最开始操作，如调用请求（invoke request），调用运输（invoke shipment）和调用付款（invoke payment），可以命名这些消息，除了这些消息之外，用例交互图能表示角色与系统间其他消息的交互，并描述了用例与角色的全部交谈。

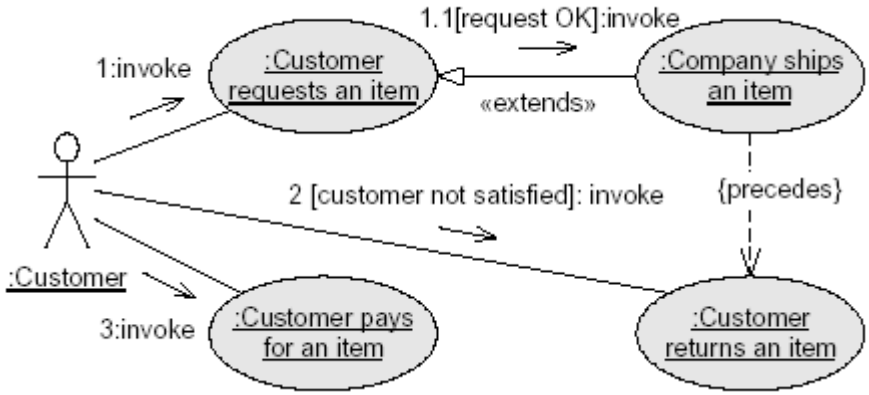


图 6　UML 交互图描述业务流程和角色之间的交互和关系

用例协作图也描述业务流程实例组成的一个场景。不象用例序列图，用例协作图描述用例实例和角色实例之间的用例关系和消息交互。用例协作图如图 6 所示。

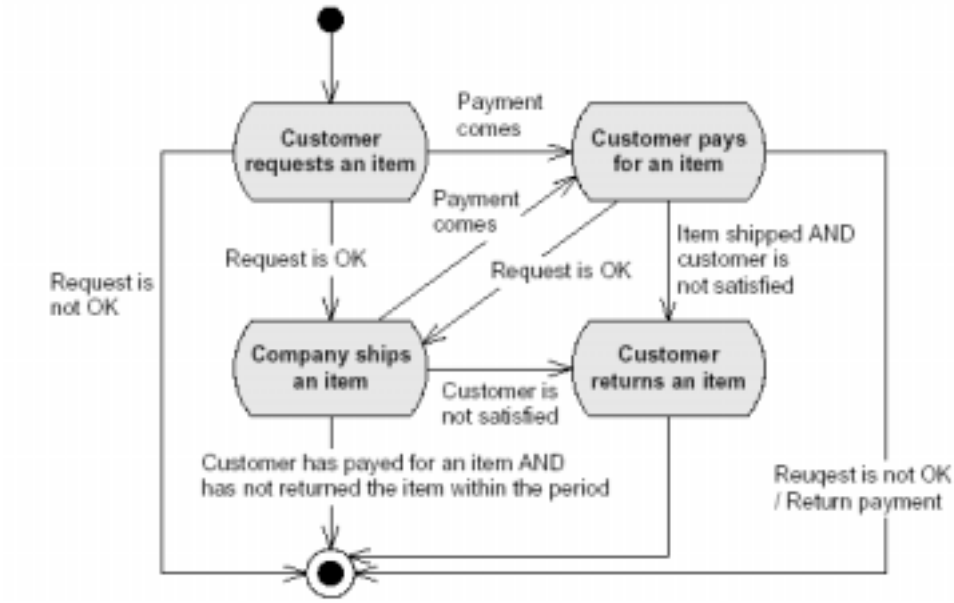


图 7　UML 活动图描述业务流程的允许的顺序

用例交互图描述的仅仅是由用例实例组成的一种典型场景。因此它不能表达用例实例所有允许的顺序，属于用例包的用例实例所允许的顺序可以在用例包生命周期内详细说明。用例包的生命周期可用静态图、Backus-Naur 范式（BNF）[2] 的活动图表示，在这些状态中，活动状态或 BNF 声明映射为用例包中的用例。用例包生命周期是用例包行为的准确描述，然而它难以正确表达，尤其在复杂的用例中。用例交互图很容易表达，但它描述的仅仅是由包中用例组成的某一典型场景。

注：[2] 请参看[4]如何使用 BNF 指定生命周期。

图 7 是 UML 活动图，描述了用例包订单管理（order management）的生命周期。在活动图中的活动与图 4、5、6 中的用例相对应。

注意 UML 元模型没有定义任何从状态或行为状态到用例实例的映射，这种映射可以在开发过程进行，与 Martin 和 Odell 方法相似，其中子系统的每个状态都指明子系统中的一个候选类。参考[5]。然后，其他开发过程可能以其它方式定义用例包生命周期。例如，用例包生命周期的目的在用例包的范围内可被指定为子系统接口操作允许的顺序。

用例交互模型和用例生命周期还有一个显著的区别——它们在项目知识库中的位置不同，并且与其它设计工件的关系不同。工件用例交互模型与工件用例模型相关，工件用例包生命周期与工件用例包相关，后者的抽象级别比相对应的用例模型和用例交互模型高。见图 8 和下一节的讨论。

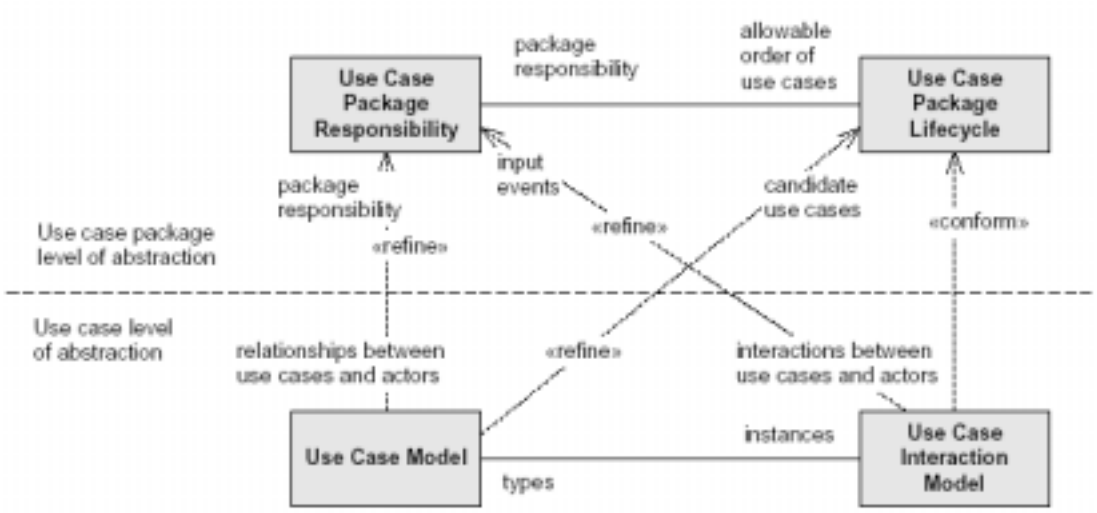


图 8　在项目知识库里的用例视图中工件之间的关系。符号用 UML 表示，为了更加清楚，依赖关系用 role ends[3] 表示。

项目知识库的结构化

在开发流程中，软件构架师、设计者和开发者确认软件产品的某些信息，如用例、软件构架、对象协作和类描述。这些信息可能十分抽象，如产品前景，也可能非常具体，如源代码。在本文中，我们把这些信息叫作软件产品设计工件（design artifacts）。

我们必须认识到设计工件与它的表达之间的区别：设计工件决定业务系统的信息，而表达决定如何把这些信息表示出来。有些设计工件用 UML 表示，有些用文字或表格表示，例如，类的生命周期可以用 UML 状态图、活动图、状态转移表或 Backus-Naur 范式表示。类库用文字来表示。

工作流管理系统的规格说明是定义在精确定义的设计工件基础上的，而不是在图上。这一节和下一节将讨论设计工件的结构，此结构可以明确业务流程、业务规则、软件构架和业务系统设计之间的关系，并且很容易扩展到覆盖业务系统的其他方面，如团队结构和项目计划。

业务系统可被描述为多级的抽象和多种视图。多级的抽象如组织级、系统级、构架级、类级；多种视图如逻辑视图、用例视图、分析视图、测试视图或文档视图。在抽象的每一级和在每一视图里，软件产品可以用四个设计工件来描述：分类器之间的静态关系、分类器之间的动态交互、分类器职责和分类器生命周期。每个工件可用 UML 图或文字来表示。如图 9 和图 10。

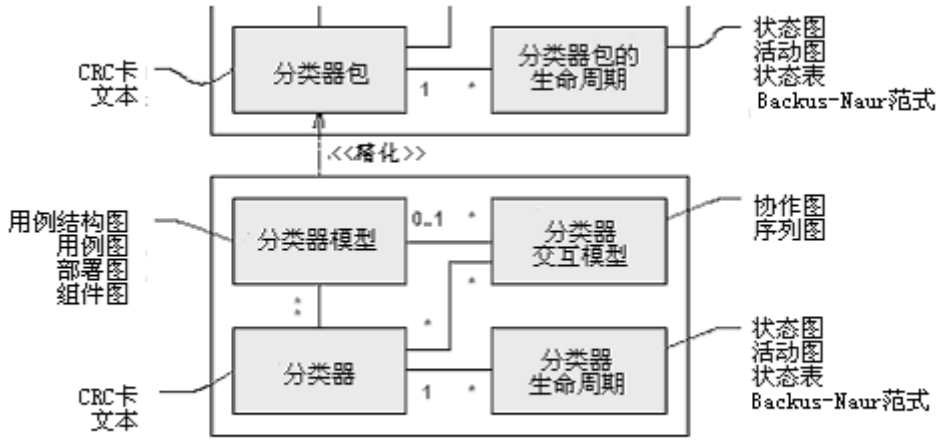


图 9　结构化项目知识库的设计工件模式

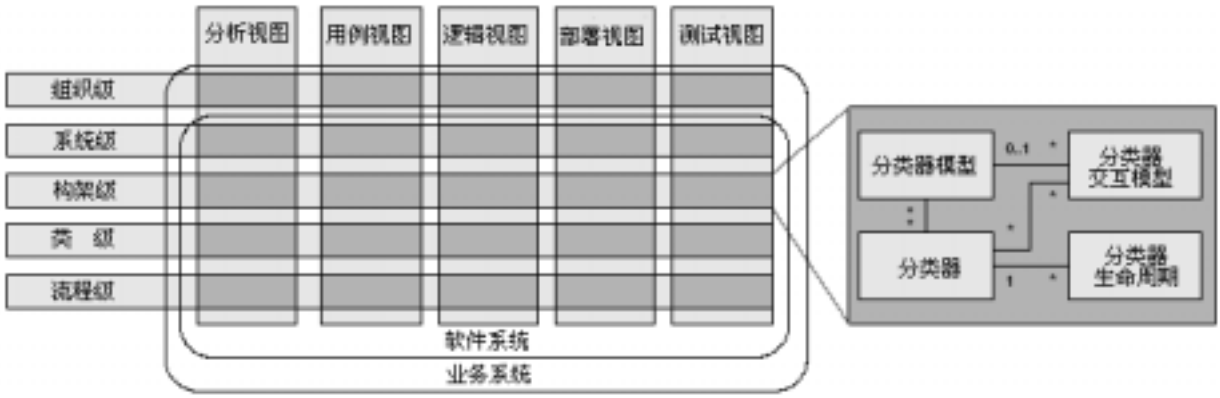


图 10 在每一抽象级的每一视图中，都可以用四种设计工件描述软件产品。
UML 分类器是：类、接口、用例、节点、子系统和组件。

分类器模型(classifier model)指定了分类器之间的静态关系。分类器模型可以是一组静态结构图（如果分类器是子系统，类或接口），可以是一组用例图（如果分类器是用例和执行者），可以是一组部署图（如果分类器是节点），还可以是一组组件图（如果分类器是组件）。

分类器交互模型（classifier interaction model）指定了分类器之间的交互。分类器交互模型可以用交互图表示：序列图或协作图。UML 表示指南（UML Notation Guide）仅仅描述了那些分类器是对象的交互图，并不描述那些分类器是用例、子系统、节点或组件的交互图。

设计工件叫做分类器（classifier），指定了分类器的职责、角色和分类器接口的静态属性（例如，分类器的操作表，有前置条件和后置条件）。分类器还可以用结构化的文字表示，如 CRC 卡。

分类器生命周期（classifier lifecycle）指定了分类器状态机和分类器接口的动态属性（例如所允许的有顺序的操作和事件）。分类器生命周期可以用状态图、活动图和状态转移表来表示。

可以把分类器模型的一个实例链接到分类器交互模型的几个实例上，所有这些实例都链接到分类器的实例上，分类器的实例链接到分类器生命周期的实例上。

跟踪业务流程到软件设计

在结构良好的设计文档中，有关业务流程和软件产品的信息可以很容易定位，并且相关信息链接在一起，同时结构还体现了不同设计工件的完整性和一致性，它是业务专家、顾问和软件开发者沟通的基础。此结构还提供了明显的关注点，由业务人员而不是开发者定义所有的业务规则。这一节讨论业务流程和软件设计工件之间的关系，用它把项目知识库结构化，给出了把业务流程的以及它们与软件设计工件的关系的构造信息。参看[4]获得知识库结构化的具体实例。

结构是基于业务系统中业务对象、角色、工人协作的业务流程（在 UML 中用用例表示）。业务流程（用例）定义为协作类型，指明了协作职责、目标、前置条件、后置条件和协作中调用的系统操作。业务流程实例（用例实例）为协作实例，指明了在协作中的行为和事件、系统状态和状态转移的具体顺序。

图 11 指明了业务流程的设计工件和软件系统的逻辑设计之间的关系，工件是按照不同的抽象级别组织起来的：组织级、系统级、构架级和对象级。

组织级（organization level）指定了一个组织（如公司、学校和政府机关）的职责，以及该组织的业务环境。工件组织（organization）指定了组织的职责和相关的静态属性。工件组织模型（organization model）指定了组织与其他组织之间的关系。工件组织用例（organization use case）用流程目标、前置条件、后置条件和业务流程必须符合与其相关的静态属性的业务规则来指定组织范围的业务流程。这个业务流程是组织与其他组织之间的协作，这种协作是在工件组织用例模型（organization use model）中指定的，见图 11 中的依赖关系<<协作>>。组织业务流程的实例是由组织交互模型（organization interaction model）用组织与其他组织间的交互来指定的。组织业务流程可以精化到更具体的系统业务流程，见图 11 中的依赖关系<<精化>>。工件组织用例生命周期（organization use case life cycle）指定了所允许的系统业务流程。组织用例交互模型（organization use case interaction model）指定了典型的业务流程实例序列，见图 11 中的<<实例>>依赖关系。组织业务流程的实现用软件系统和它的用户（团队角色）之间的交互来指定，见图 11 和 12 中<<实现>>依赖关系。

系统级（system level）指定了软件系统的环境以及与它的角色之间的关系。工件系统（system）用职责、前置条件、后置条件、参数和返回值来指明系统的接口和操作。若角色职责和接口是相关的，并由工件角色（actor）指定。系统生命周期（system lifecycle）指定了所允许的系统操作和事件。系统模型（system model）指定了软件系统和角色（其他系统和用户）之间的关系，系统交互模型（system interaction model）指定了软件系统和角色之间的交互。这些交互是系统业务流程的实例，见图 11 中依赖关系<<实例>>。工件系统用例（system use case）用流程的目标、前置条件、后置条件、非功能性需求、业务规则和其他相关静态属性指定了在系统范围内的业务流程。这个业务流程是系统与其它系统或用户的协作。这些系统与它的角色之间的协作是在工件系统用例模型（system use case model）中描述的，见图 11 中的依赖关系<<协作>>。业务流程接口的动态属性，如在业务流程范围内所允许的系统操作顺序，是在系统用例生命周期（system use case life cycle）中指定的。系统用例交互模型（system use case interaction model）指定了典型的业务流程实例的序列。系统业务流程可以精化到子系统业务流程中，见图 11 中的依赖关系<<精化>>。系统业务流程的实现用构架级的子系统间的交互来指定，见图 11 中的依赖关系<<实现>>。

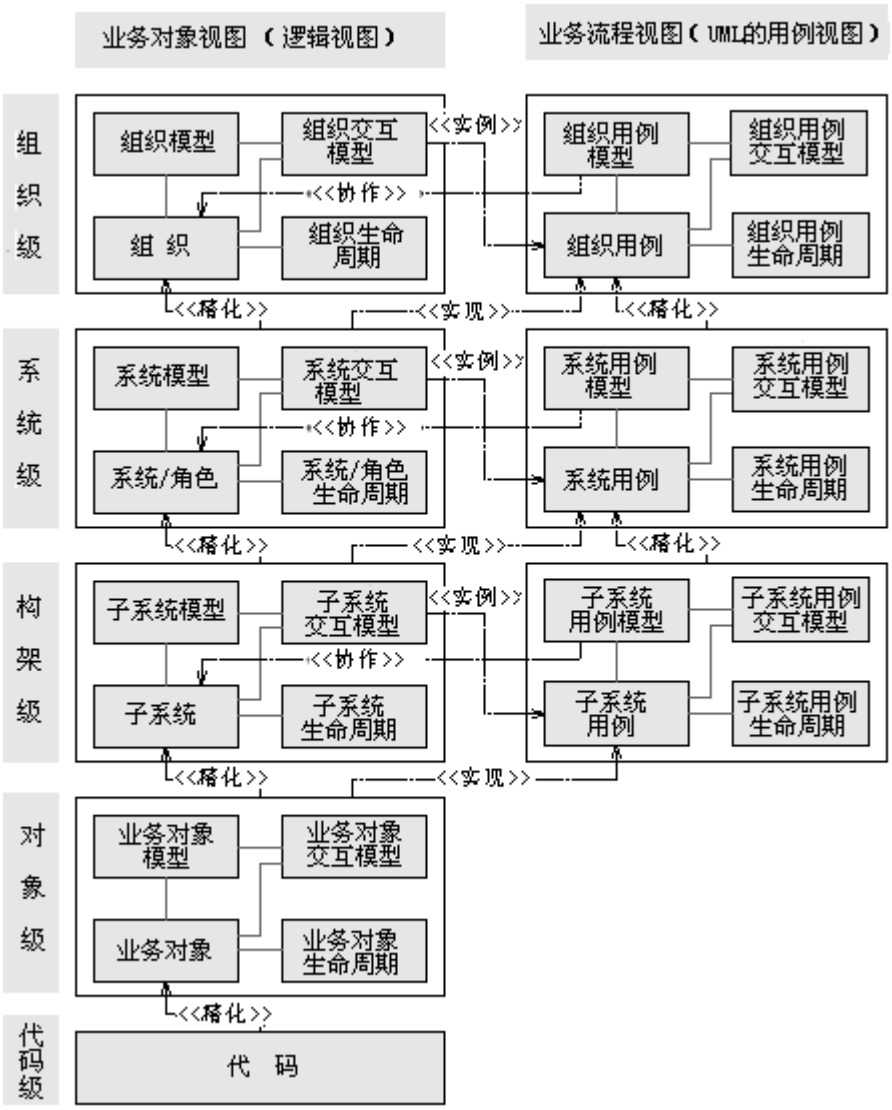


图 11　设计工件在逻辑和业务流程视图中描述了软件系统

构架级（architectual level）定义了子系统（组件）、子系统的职责、接口、关系和交互。工件子系统（subsystem）职责、前置条件、后置条件、参数和返回值指定了子系统接口和子系统操作。子系统生命周期（subsystem lifecycle）指定了所允许的子系统的操作和事件的顺序。子系统模型（subsystem model）指定了子系统和其他子系统之间的关系，子系统交互模型（subsystem interaction model）指定了子系统之间的交互，这些交互是子系统业务流程的实例，见图 11 中依赖关系<<实例>>。工件子系统用例（subsystem use case）指定了在子系统范围内的业务流程，这个业务流程是子系统与其它子系统、系统和用户之间的协作。子系统和它的角色之间的所有协作是在系统用例模型（system use case）中描述的，见图 11 中的依赖关系<<协作>>。子系统业务流程接口的动态特性，如在业务流程范围里所允许的子系统操作顺序是在子系统用例生命周期（subsystem use case life cycle）中指定的。子系统用例交互模型（subsystem use case interaction model）指定了业务流程实例的典型序列。子系统业务流程的实现用类级别上对象之间的交互来描述，见图 11 中的依赖关系<<实现>>。

对象级（object level）用业务对象、业务对象的职责、关系和交互来描述子系统的设计。业务对象模型（business object model）指定业务对象之间的静态关系。业务对象交互模型（business object interaction model）用业务对象之间的交互指定子系统操作的设计。工件业务对象（business object）指定业务对象的职责和业务对象接口的静态属性，如用前置条件和后置条件描述的接口操作。业务对象生命周期（business object lifecycle）指定了所允许的接口操作顺序。

图 12 中，设计工件描述了组织团队结构。事实上，它与描述软件子系统结构的工件没有什么太大的区别。团队的角色可以表示成 UML 构造型的类，工件角色（role）指定工人角色的职责及其它相关的静态属性。工件角色生命周期（role lifecycle）指定了角色的动态属性、它们的状态以及它们所响应的事件。工件角色模型（role model）指定角色之间的静态关系。成员级业务流程（member level business process）指定团队成员角色与其它团队成员角色之间的协作，见图 12 中的依赖关系<<协作>>。这些业务流程的实例是在工件角色交互模型（role interaction model）用角色实例之间的交互指定的，见图 12 中的依赖关系<<实例>>。

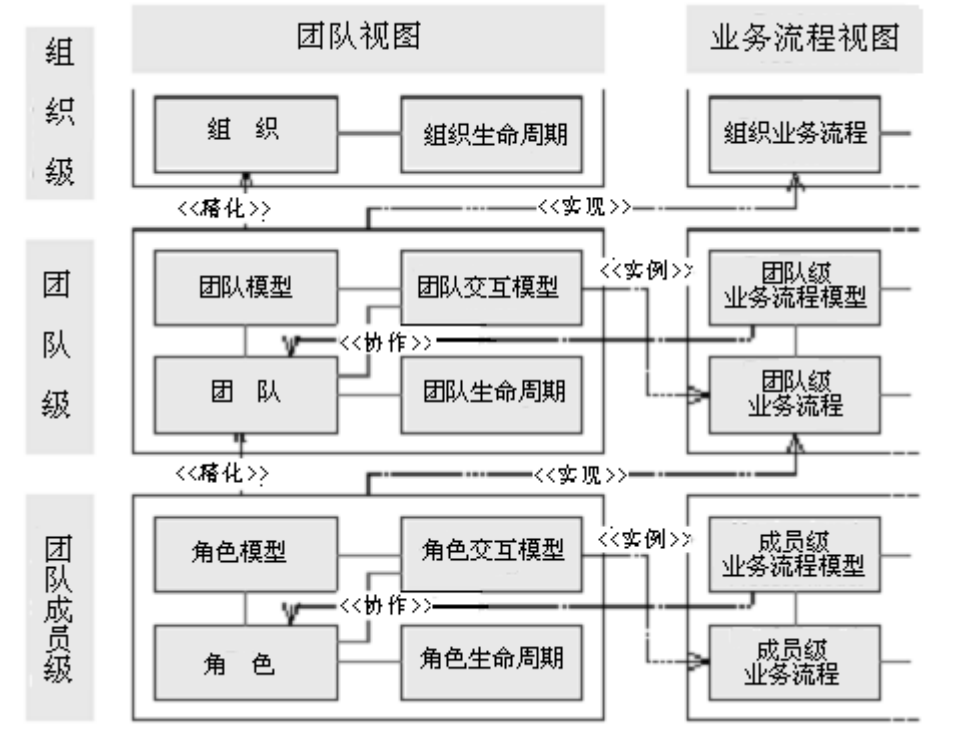


图 12 在团队和业务流程视图中，设计工件描述了软件系统

称为团队（team）的设计工件是一个角色包，指明了团队的职责以及相关的静态属性。团队的动态属性是在工件团队生命周期（team life cycle）中指定的。工件团队模型（team model）指定了团队之间的静态关系。团队级业务流程（team level business process）指定了团队和其它团队之间的协作，见图 12 中的依赖关系<<协作>>。团队级业务流程的实例是在团队交互模型（team interaction level）中指定的，见图 12 中的依赖关系<<实例>>。团队级业务流程的实现用团队成员之间的交互以及团队成员与软件系统之间的交互来指定，如图 12 中的依赖关系<<实现>>。设计工件的模式可以用相似的方法应用在更高的抽象级 [4] 上。

注：[4] 具有复杂组织结构的典型组织，例如，在德国有很多级别，团队成员级、部门级、处级、公司级和企业级。在斯堪的纳维亚典型的有小组织或者具有简单结构的组织可被很好地表示为如图 10 中的三级：团队成员级、团队级和组织级别。

这一节描述的设计工件的结构可以通过两种方式简化：

- 只使用设计工件的子集；
- 把紧密相关的一些设计工件结合起来，构成更大的设计工件单元。

使用设计工件的子集不会导致信息的丢失，因为 UML 图系统不是正交的。意思是同样的信息可以用两个或更多不同的 UML 图来描述。例如，两个静态结构图和对象协作图描述对象之间的关系。两个状态图和交互图描述对象之间的消息。因为同样的信息可以从几个方位来描述，开发者所提供的仅仅是设计工件的特定子集，否则此工件必须接受关于一致性的检查。

由于实际的原因，设计者可以创建一个更大的可交互工件来包含几个紧密相关的工件。例如，分类器职责和生命周期总是与某一文档相关和结合。也可以把组织、系统、子系统和类用例图结合成一个大用例图，以清晰地区别用例级。同样的，系统、子系统和业务对象静态结构和相对应的所有级别里的交互图可以结合成一个文档，也可用 UML 语义把静态结构图、组件、节点和每一级的用例图结合起来表示。用这种方法，设计者可以在一个大的静态结构图中表达用例、角色、子系统、业务对象、团队成员和业务流程之间的关系，然而“ UML 表示法指南 ”没有清楚地提到这样结合的静态结构图。

参看[4]在这一章描述的如何简化设计工件。

小　　结

这一章讨论了用UML表示面向对象工作流管理系统，给出了如何把典型工作流概念映射到UML概念的方法，并建议结构化设计工件，从而跟踪业务流程的定义和面向对象软件设计之间的信息。此结构展示了业务流程可表示为业务对象、团队角色和业务系统中其它实例的协作，此结构基于四种相关的设计工件的模式，这四种设计工件为分类器关系、交互、职责和生命周期。此模式可以应用于一个业务系统的不同抽象级和不同的视图，也可应用于业务和软件系统设计的其它方面。

参 考

[1] Alexander, I, A.: A Co-Operative Task Modeling Approach to Business Process Understanding, workshop on Object-Oriented Business Process Modeling, ECOOP 98, Brussels, Belgium, July 20-24, 1998.

[2] Cockburn, A.: Using Goal-Based Use Cases, Journal of Object Oriented Programming, November 1997, also available at http://members.aol.com/acockburn/papers/usecases.htm

[3] Henderson-Sellers, B, Graham, I., The OPEN Modeling Language (OML) Reference Manual, SIGS Books, NY, 1997, available at http://www.csse.swin.edu.au/cotar/OPEN/OPEN.html

[4] Hruby, P.: Structuring Design Deliverables with UML, <<UML>>’98, Mulhouse, France, June 3-4 1998. also available at http://www.navision.com/default.asp?url=services/methodology/default.asp

[5] Martin, J., Odell, J. J.: Object-Oriented Methods: a Foundation. Prentice Hall, Inc. 1998

[6] Martin, R. C.: RE: RE: (OTUG) use cases: abstract and concrete (<<precedes>>), Mailing list OTUG, available at http://www.rational.com/HyperMail/otug/hypermail/9807/0195.html

[7] UML Notation Guide, version 1.1, Rational, 1 September 1997, also at ttp://www.rational.com/uml

[8] Stark, H., Lachal, L.: Ovum Evaluates Workflow, Ovum Ltd. 1995.

部分中英对照：

Actor	角色
BNF	Backus-Naur 范式
Business Object	业务对象
Business Process	业务流程
Classifier	分类器
Classifier interaction model	分类器交互模型
Classifier lifecycle	分类器生命周期
Classifier model	分类器模型
Collaboration	协作
Composition	组装
Design artifacts	设计工件
Organization level	组织级
Refine	精化
Realize	实现
Team Role	团队角色
Stereotype	构造类
UML	统一建模语言
UML Notation Guide	UML 表示法指南
Workflow	工作流

欢迎访问

www.umlchina.com

关于需求，设计，构造，测试，维护，配置管理，管理，过程，工具，质量...翻译或原创均可。（详细）

投稿请 EMAIL: editor@umlchina.com

注：本杂志资料文章仅供学习交流之用，如需转载请与《非程序员》联系。