X-Programmer 非程序员

总第9期
2002
1

目 录

- 本期相关的参考资料
- 本期审稿：Think、jason chan　　版面设计：徐远正

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)

- XP On A Large Project
- Specification of Workflow Management Systems with UML

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

播种机
UMLChina.com

Kent Beck：只是一种正确做事的方法

2001年12月7日上午10:00-12:00，XP（eXtreme Programming）的创始人Kent Beck先生作客UMLCHINA讨论组的聊天室，畅谈XP。Kent Beck是软件开发方法学的泰斗，有17年面向对象的编程经验。他倡导软件开发的模式定义，CRC卡片在软件开发过程中的使用，HotDraw软件的体系结构，基于xUnit的测试框架，重新评估了在软件开发过程中测试优先的编程模式。Kent Beck是《The Smalltalk Best Practice Patterns》、《Extreme Programming Explained》和《Planning Extreme Programming（与Martin Fowler合著）》等书的作者。以下是交流实录。由Areca Chen和think翻译。因时间匆忙，错误难免，如有疑问，请对照原文。

tensile: 能否谈谈 XP 和 UML 的关系

Kent Beck: XP是一个软件开发的社会(social)系统，UML是一个描述软件的标记法(notation)。你可以在XP中使用UML，但团队良好的进展并不需要许多文档，除了代码及测试外。

yhufo: XP 中的文档重要吗？需要写什么样的文档？

Kent Beck: 你要写的文档就是代码及测试。

superlong: XP 如何用在一个可视化项目中？

Kent Beck: XP程序员并不是不做视觉上的思考(think visually)，他们只是一般不储存图片(picture)。

notyy: XP 可以解决什么问题？

Kent Beck: 当你有许多关于需求的不确定性，XP让你快速得到一个具体、运行中的系统，同时让你以一个相当稳定的速度(pace)修改。以这种方式你无须在开始之前知道所有的事。

yhufo: 但 CRC 卡及故事--这些都不是文档。

Kent Beck: 你无须永远保存。你从CRC卡及故事获得内容，然后把他们丢掉。

yhufo: 请问你认为最好的 XP 站点是哪一个？

Kent Beck: 目前xprogramming.com和extremeprogramming.org这两个地方都是学习XP一个不错的起点。

joe_lee: XP 是一种递归过程（recursive process）吗？

Kent Beck: 是的，或者我可能说『不规则碎片形(fractal)』。以一年为期，描述你所想要的（故事）然后去做。以一分钟为期，描述你想要的（单元测试）然后去做（编写代码）。分解的不规则碎片形是经过相当的深思熟虑的(deliberate)。

taoxie: 你提到了单元测试。我同意 rerun 所有的单元测试是合理的，那么 rerun 所有的集成测试也是合理的吗？

Kent Beck: 如果你有适当的准备而且够快的话，你可以随时执行测试。

yhufo: 如果我们没有文档，我怎么知道怎样能控制软件中的变更呢？我想我们必须建立文档！

Kent Beck: 当你们说『控制变更』，我则说『鼓励变更(encourage change)』。控制来自测试、双人编程、大家坐在一起讨论及持续集成。XP对于受过良好训练的程序员运作得很好，但是如果目前不觉得有良好的训练，别担心。你的搭挡可以帮你，不然你的集成工作无法达成，你必须把你的代码丢弃。

dggh: 能谈谈 xp practise games 吗？

Kent Beck: 我做过一个练习让人们描绘一个咖啡壶的图像。从这里显示有更好的game。在伦敦有人建造乐高(Lego)机器人。Joshua Kerievsky有一个团队写了一个剧本（电影的脚本）。

gigix: 我想我们必须先小心地设计。如果我们忽视了设计的重要性，XP（或重构）将不能对整个项目有改进。对此你如何看？

Kent Beck: 我就知道迟早要讨论这个。我，同样地，坚持我们需要小心地设计。我宁愿等待并且在我有一些经验后再做，而不愿依据猜测(speculation)做设计。前置设计（Up front design）是一种正向反馈的循环(positive feedback loop)。你越有经验，你越可以考虑前置设计，直到你完全无用(useless)。另一种使用设计经验的方式是等待你的好想法，直到这些想法显露出来，立即就可以用得上。

joe_lee: 如果没有文档，我们怎样维护软件？

Kent Beck: 你有两个非常有价值的文档-代码及测试。如果测试总是能够100%执行而且绝不会允许逆行(regress)，你还需要什么？

yhufo: 在双人编程中，两个人应该经常变换角色吗？一天或两天，我是说多久换一次，一天或两天？

Kent Beck: 在比较大的团队中，每天变换（switch）2-4次。在一天之中我无法维持一个双人组6个小时以上，因为专注的程度太过激烈。

xlp223: 既然 XP 没有专家，那 XP leader 在程序员当中扮演什么角色，客户，经理？

Kent Beck: XP是有专家，但他们的角色并不像以前一样是固定的。专家的出现是由团队中的每一个人依据经验挑选出来的。促使这种专家的出现是管理者扮演的角色。

yhufo: 在双人编程中，当一个人编写代码时，另一个人在干什么？只是看和说话吗？

Kent Beck: 双人编程并不是看着别人编写代码。编写代码就像是在沟通。键盘每几分钟便传来传去。双人编程时，你们讨论相关的想法，试试一个，建议另一个测试，把你下一个想法画个草图，写成代码，做些重构等等，持续进行下去。

dggh: 如何处理“程序员很少接触客户”的情况？

Kent Beck: 我想这些接触一般运作得不是很好。这就是为什么我说XP是软件开发的社会系统。频繁地和其生活受你写的软件影响的人接触，这能使你重视它。

tensile: 软件开发的社会系统？

Kent Beck: 首先，所有的团队成员聚集坐在一个大房间内：程序员、管理者、客户、分析师、测试员、用户文档、交互设计。其次，“客户团队”(customer team)提出编程团队必须通过的验收测试(acceptance tests)。学习如何通过这些测试需要沟通。

skyin: 每个人都有自己的任务，那如何双人编程时如何分配任务呢？

Kent Beck: 今天早上我将与你共同做你的任务，下午你可以帮我做我的。

yhufo: 能否介绍一些 XP 的工具？

Kent Beck: 我使用Eclipse作为Java工具，我真的很喜欢它。有时我也使用IntelliJ，还有，JUnit或它在其他语言的姊妹产品，在单元测试及建立验收测试架构时非常有帮助

xlp223: 每对双人编程都有自己独特的风格和水平，重构怎样才能使它变得一致和平衡？and person or group to do refactoring need be arranged in addition.

Kent Beck: 合唱团中的人，每人都有自己的声调。他们是如何让这些声音和谐？他们先决定最重要的和声然后“以他们的方式”唱出。在程序员团队中也是同样的道理。团队中所有人的成功，比个人的闪耀更重要。

notyy: 安排一个“在场客户”（on-site customer）在某些情况下实在困难。

Kent Beck: 想办法找到一个，否则准备接受失败。旧的Taylorist 软件开发社会结构没有用了，要有新的社会结构。如果让软件开发运作成功是重要的，坐在一起是绝对关键的。如果想做好却不认为那是重要的，或许这个项目应该取消。

yhufo: 为什么 XP 不需要文档！

Kent Beck: XP确切需要两种文档-代码及测试。如果你的团队觉得需要更多的文档，那么去做就是了。大多数的团队发现他们并不需要其他的内部(internal)文档。

taoxie: 你说客户注重合同型（Contract-style）的项目，我就碰到了这种情况。XP 怎样应用在套装软件产品开发中？

Kent Beck: 对于套装软件(shrink-wrap)开发公司而言，客户团队（行销、测试、可用性、客户服务）是比使用自制型(in-house)开发的公司要大。除此之外，大家都同意使用单一的需求序列（故事）可以让软件开发越来越平滑(smoothly)。

tooliu: 不介意解释一下 WinDNA 和 XP 的关系吧？

Kent Beck: XP是Extreme Programming，不是操作系统。

xlp223: 做重构的人或组需要额外安排吗？

Kent Beck: 设计是每一个人的责任。当我们坐在一起而我们看到需要重构的地方，我们就重构。如果你还需要去安排进度，那你就等得太久了。

notyy: 我们应该首先教育客户吗？

Kent Beck: 客户需要知道XP吗？绝对需要。他们是团队的一份子，因此他们需要知道团队如何运作。故事及验收测试是需要先教给他们的两个最重要的事情，然后是发行(release)（3-12个月）及迭代(iteration)（1-3周）的计划。

skyin: 看起来 XP 更适合客户导向的应用软件，那其他类型的软件呢？

Kent Beck: 还有其他类型吗？如果没有客户，为什么要写这些软件？

Charity_Zhou: 因此我想 XP 对客户有更高的需求。

Kent Beck: XP当然对于客户有更多的要求。他们是可以看到的而且可以控制，同时他们负责挑选系统的范围。

smilemac: 我觉得 XP 和演化（evolution）过程很象，你能解释一下区别吗？

Kent Beck: XP确实使用演化或成长的隐喻(metaphor)。然而XP指定许多实践来支持长期的持续性成长。Gilb的演化式的交付工作在概念上是类似的，但对于如何达成目标。他没有说明太多。XP说：先测试、双人编程、坐在一起、随时集成然后你就能使你的软件成长及演化。

fpeng: 发行计划（release planning）和迭代计划（iteration planning）的区别是什么？

Kent Beck: 发行与业务循环同步，所以它们是3-12个月的时间。迭代与工程师需要的里程碑同步，所以它们是1-3周的时间。

dggh: XP 需要团队中的每一个人都是系统分析专家吗？这也太难了！

Kent Beck: 团队需要系统分析专家，但不意味着每一个人都需要做系统分析。类似的，团队可能需要数据库专家，但只需3-4个人深入了解数据库的知识，而其他所有的人如果需要可以协助做数据库的任务。

Kent Beck: 我不认为推卸责任是在中国的特别问题，我认为这是任何地方的特别问题。我们从Fred Taylor那里获得工作的社会结构概念，一位上个世纪初的工业工程师。他假设工人是懒惰而且愚笨的，因此他们需要有人为工人计划并有人检查工人的工作（品质保证(QA)）。这个社会结构在软件中并不能良好运作，因为大部分的程序员既不懒惰也不愚笨。不管如何，Taylor的范例执行得非常深入以致多数人甚至不知道概念是来自何处。

dggh: 当团队速度变慢时，我们能做什么？

Kent Beck: 我今天刚写了一封长信到Yahoo小组的XP邮件列表中。简单的回答就是要让团队速度加快，唯一的方式就是鼓舞团队的士气。

zhujigang: 我们的团队确实使用了双人编程，但如果有一对完成了他们的故事，而其他人还没有完成时，这一对该做什么？他们需要等别的人吗？

Kent Beck: 在每一次迭代中每个程序员签认一些任务。如果一个任务完成了，开始另外一个，或帮助你的同伴开始做他的任务。

simplebest: 你是否使用设计模式？

Kent Beck: 我绝对使用模式。如果你检视 JUnit，到处都是设计模式。它是我曾经做过的设计最密集的部分之一，不管如何，设计模式被应用是因为我们需要它们，而不是因为我们认为它们可能是有用的。我们写一个测试，使之运行，然后注意到，如果我们引进设计模式，代码是否可以更清晰。我们通过重构来把设计模式放到正确的地方。4 年后我们仍将做相同的事情。熟练的设计师可以通过使用设计模式，把讨论设计的时间从几小时缩短到几秒钟之内。

skyin: 你认为 XP 象管理领域里的“学习型组织”吗？

Kent Beck: 是的。XP依赖整个团队持续学习如何更好地互动。

yhufo: 是不是团队中的每一个人都应该测试软件？

Kent Beck: 如果我们要协调权力与责任，那么每一个程序员，他们有权力产生缺陷，必须同时有责任去测试。我看不到其他的方式可以解决这个难题。

fpeng: 你认为我们在使用 XP 时，是否需要软件来管理整个 XP 的过程

Kent Beck: 当初我对XP所做的第一件事情，就是写一个项目管理工具，实际上这是浪费时间。XP是一种新的习惯。用最简单可能的设备（几张纸贴在墙上就不错了）养成这个习惯。只要养成这种习惯同时团队找到其精神，你就可以开始引进软件来管理，而不至于破坏过程。

adylee: 你如何看待 CMM 和 XP？

Kent Beck: CMM起源于制造业。那是从所谓的制造业成熟模式(Manufacturing Maturity Model)复制而来。软件并不像实体的制造业。每一个软件的开发都不一样，而有时候根本就是如此（and sometimes radically so.）。

simplebest: 你认为 RUP 和 XP 哪种更适合中国的程序员？

Kent Beck: 这一点我不清楚。XP提出来自每一种文化的程序员的问题，其不同的部分也有提到。XP在中国最困难的部分在哪里？

tensile: XP 是依赖软件还是依赖人？

Kent Beck: XP绝对是依赖人的，好的人倾向于做好的事情，功力差的人做得差一些。

dggh: 代理客户不知道客户的需要，那怎么办？

Kent Beck: 当我上次在日本，他们说他们无法诚实地面对客户，因为客户是上帝。我问是否一个即时(just-in-time)供应商会对他们的客户诚实。“喔，是的，当然。否则是不行的”。我想这就像XP。我们需要

对我们的客户诚实并且向他们保证。他们一开始并不喜欢这样，但结果是如此的棒，所以他们会学习去要求这种新层次的投入。

notyy: Kent，你应该知道这样一句话，“没有人因为使用 IBM 的产品而受到惩罚”，同样，那么多公司的经理想使用 RUP 和 CMM。

Kent Beck: 在“没有人因使用XP而被解雇”成为现实之前会如何？我们将努力达到这个目标。“喔，你的团队已经三个月没有交付任何软件了。怎么办？你被解雇了。”我将举一个技术问题为例--模式产生架构。原来的问题是“模式与架构之间的联系是什么？”，这发生在设计模式还是新概念的时候。因此Ralph【编注：指设计模式GOF的Ralph Johnson】和我跑进一家旅馆的房间去封闭思考，尝试回答这个问题。我们找到的答案就像定理(theorem)可以从第一个原理(principles)导出，因此一个架构可以从第一个原理（模式）被导出。这个论点--没有人能够证明是或不是—就是任何架构都可以从一小撮模式的后续应用推导出来。

dggh: QA 部门需要详细的需求，但我们的团队只有简单的文档，怎么办？

Kent Beck: 在XP中，QA有一个前置的角色（up-front role）。在每一次迭代中，在技术团队开始把故事分解成任务之前，最好的团队随着他们的故事交付验收测试。你需要测试员编写这些测试，同时有设备支持他们。对于一个程序员最重要的事是勇气--冒险的勇气—看起来很笨，勇于挑战困难和尝试新技术，勇于清楚地与别人沟通，甚至是很难支持他们时。（The most important thing for a programmer to have is courage--courage to risk looking stupid, courage to try difficult and new techniques, courage to communicate clearly with others even when that is hard support them.）

simplebest: 在中国，大多数优秀的程序员不喜欢和别人坐在一起，这怎么实践 XP？

Kent Beck: 在我的团队中，最优秀的程序员是最会沟通的人。或许我们对于“优秀”的定义不同。

simplebest: 关于优秀的定义：意味着你能解决别人不能解决的问题。

Kent Beck: 我对于优秀的定义是使别人解决他们曾认为他们不能解决的问题，也就是一个团队可以共同做的有多少。

阿楼: 噢，天哪，我想我的经理用不了 XP，因为他喜欢控制每一件我为他做的事。

Kent Beck: 想要为你作决定的管理者就不是管理者。你可能需要离开，或可能是他们应该离开。在XP你想要从管理者那里得到的是建立及维护社会网络的能力。这个网络断裂，他们便应该去修复。客户可以指定什么层次的工作量(load)需要支持，同时他们编写测试以决定这个层次是否已被支持。如果未通过测试，技术团队修复这个系统。

fdu_se01: 什么是 XP 项目的特征？例如涉及的领域，开发团队的规模，软件产品的规模。CMM 的数据（特别是在美国）是基于 DoD 的。

Kent Beck: 团队大小：3-40个人（包括整个团队）。领域：可想到的任何事。产品大小：30个生产人员可以在几年内生产的任何东西。

notyy: 很难发现一个隐喻，怎么办？

Kent Beck: 找寻好的隐喻需要相关的软件经验。真正强大的隐喻并不会立即浮现。我曾经有一个大的隐喻是在处理一个程序片段12年之后获得的。我希望我现在应该更聪明了，但我怀疑:-(

zer: 作为项目经理，如果有人离开（被解雇），那应该怎么办？

Kent Beck: 如果团队中有人离开，那一般不是大问题。速度可能下降（也可能提升）。计划的实践已经可以良好地控制速度改变。

xlp223: Linus 是一位优秀的程序员，但他有时候有些害羞。

Kent Beck: 双人编程的方式可以帮助人们变得更能社交。

skyin: XP 发展的方向是什么？

Kent Beck: XP的下一步是更加拥抱整个团队--如何把一个巨大的故事切割成合理的小故事？在团队中彼此如何沟通及协调。

adylee: 你认为 XP 现在是不是完美的？

Kent Beck: 你一定是在开玩笑。一般而言自然界的紧急系统使用3到5条规则（Emergent systems in nature use 3-5 rules）。很可悲的是XP似乎需要太多的规则。我一直在找寻简化XP的方法。你是否可以维持可视性及控制而无须先编写测试？我不认为如此，但我太积极投入了，以致不能了解。（but I'm too emotionally involved to know.）

notyy: 你是说 XP 将更简单吗？但我想轻快的过程会更复杂。

Kent Beck: 我想让XP变得更简单，但现在我看不到怎么做。

tooliu: 你是说擅长社交比刻苦编程更重要吗？

Kent Beck: 我比较喜欢有些人是可以包括这两方面。每一个团队中的成员应该持续学习新的社交及技术技能。

simplebest: 我们是否能剪裁 XP，使它适合我们的团队？特别是针对我们中国团队成员的特征。

Kent Beck: 你可以调整任何程序。你必须对你的结果承担责任，因此在你的实践上你必须有权力。不管如何，我希望人们依据经验调整，而不是丢掉一些重要的东西，只是因为他们害怕去尝试这些东西。

tooliu: 你对 XP 有什么不满意的地方吗？

Kent Beck: XP Explained这本书在1999年发行。（其中有许多是我现在想要改变的，但此时我在做别的项目。）

fdu_se01: 要多长时间 XP 才能在团队中生效（学习曲线？）

Kent Beck: 一般团队在几周或几个月后都能获得好的结果。一般约9-12个月后明显地变得更有效率。我在Iona辅导的一个团队花了将近一年的时间才真正觉得满意，他们仍然能变得更有效率。

xlp223: 现在有“轻快软件开发”的说法，它和 XP 有什么关系？

Kent Beck: “轻快(agile)”来自一个研讨会，其中有许多趣味相投的方法论者集中讨论他们的软件开发过程。我们之间的共通点多于差异点，因此我们决定使用“轻快”这个词来描述我们的共通点。

notyy: 你愿意来中国指导一些软件公司吗？

Kent Beck: 我非常希望能访问中国。我辅导的团队包括整个美国及欧洲许多地方。现在我开始在亚洲的工作。我发现在不同文化当中找寻什么比较困难或者什么比较容易是很有趣的。瑞士日耳曼人喜欢编写测试，墨西哥人喜欢双人编程，重构在美国中西部非常流行。

lovelybug28: 我是一个 XP 的初学者，请问 XP 是否有可操作性（maneuverability）？

Kent Beck: 假设你可以将你对一个大型软件项目的期望切割成一周可以完成的细部。我们每周可以会面，你告诉我哪一个细节是你认为在下一周是最有价值的部分。我不在乎你挑选哪一个部分，所以你可以彻底地改变对于软件的整个概念而从我这边听不到任何怨言。是否这就是可操作性？有一本不错的书“XP in Practice”，其中讨论一个实际的（如果是小的）项目。

taoxie: 关于 XP 是否有一些学术研究工作在进行，或者它只是面向工业界的？

Kent Beck: 我们正在取得进展。有一些杰出的教授像Ralph Johnson，开设有XP的课程。XP有许多教育上的优点。迭代式开发让学生有机会尝试许多不同的变化。

skyin: 能不能告诉我们你自己的 XP 编程工具箱？

Kent Beck: 我的工具箱？我用VisualWorks写Smalltalk程序，如果我自己编写代码时使用Refactoring

Browser，如果我使用Java编写代码时，使用Eclipse及JUnit。我只使用两种语言。

lovelybug28: 关于可操作性…XP 出现之前我们已经这样做了…

Kent Beck: 如果你已经知道如何做，XP可以从你身上学到一些东西。

notyy: OOAD 必须在 XP 过程中使用吗？

Kent Beck: 你必须做分析及设计决策。问题是何时及你如何用符号表示(notate)。详细分析决策在每一次迭代中决定，而符号表示就是自动测试。设计决策是在故事评估、迭代计划、及编写代码中决定而符号表示就是代码及单元测试。

tensile: 怎样在公司中执行 XP？

Kent Beck: 有关如何在一家200人的公司导入XP的问题。我发现你需要一个技术高手(champion)，他愿意深入学习并且教导这种技术，一个企业高手去说服企业组织尝试故事及迭代，及一个执行高手来保证每一个人不会因异常而导致第一次做事就出错。所有的改变都伴随着混乱。在XP中game的目的是在任何给定的日期尽可能地交付最有价值的软件。要做到这点，团队中的每一个人必须放弃完美的观念(notions of absolutes)，并且持续地学习。

notyy: 有没有执行 XP 的实际项目？效果如何？

Kent Beck: 在XP邮件列表中有超过100个实际项目。我猜，全世界可能不超过1000个项目使用XP。

charles_y，adylee: 为什么 XP 叫 XP，而不是别的名字？

Kent Beck: “终极(eXtreme)”就像终极运动员，这些人事前有周密的准备并且达到他们最大的极限。“程序设计(Programming)”是因为最后我们从执行的系统获得我们的报酬。起名时，名称必须是准确、好记和说得通的(defensible)。说得通是最重要的。我需要一个我的“敌人”绝不会说“他们正在做这件事”的名字。

xlp223: 谁是你的敌人？

Kent Beck: 我不是真想使用这个字眼，我只是尝试打字打得快一点。当我命名XP的时候我的心中有Grady Booch这个名字。他对于软件开发和我有不同的概念（不完全是错误，只是不同）。如果我要建立一个流行的事物，将会对Rational有一些压力说他们也在做这件事。他们不曾说过他们是“终极”，即使他们说他们是“轻快(agile)”。

zer: 在编码之前，我们必须设计软件工程里的每样东西…

Kent Beck: Zer，为什么你要设计所有的东西，如果你只设计前面一半会如何？

xlp223: RUP 和 XP 在理念上有很大区别吗？

Kent Beck: XP是一种开发过程，RUP是过程的框架（process framework），RUP有预设的举例说明(instantiation)看起来非常类似制造业(manufacturing)。

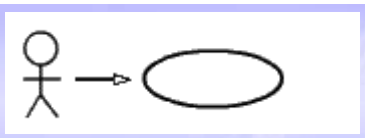
morenew: 计划 XP 的规则是什么？XP 的关键是什么？

Kent Beck: 协商的范围。时间、品质及成本是固定的。这是XP的方式。

fdu_se01: XP 是否推崇“每个人对组织都有不同的重要性”的理念？如果是这样，你如何处理人员的流动？

Kent Beck: XP计划中的规则是每一次迭代（1-3周）你计划要完成的与你实际在上个迭代当中完成的一样，如果你提早完成，你可以要求更多的工作。如果你快要来不及了，你要提出延迟要求。这个简单的规则优雅地控制休假、人力的成长、雇用、及项目间的转换。

Kent Beck: 这里有许多问题是关于“我们应如何在中国恰当地实施XP”。我建议你们由小的区域性的团体自行回答这个问题。你们的答案应该会比我的好。全世界有约20个这类团体，他们确实在帮助参与者。XP在长远的未来将变得脆弱及老旧，并且将有某些更好的东西来替换，在50年之内。不管如何，我期望许多XP实践被当做“只是以正确的方式做事情的方法”接受。先写测试、重构、快速具体的交付(quick concrete deliverables)、结合业务的团队(teams combining business)及技术才能(technical talent)。



软件工程的播种机

| 新手指南 | 行业动态 | 文档中心 | 书籍推荐 | 专家解疑 | 嘉宾交流 | 服务中心 | 人才热线 | 非程序员 | UML

UMLChina讨论组

昵 称：

密 码：

注册 忘记密码 登录

搜索UMLChina

搜索

最新>>

- 《非程序员》第四期提供HTML浏览 It's
- 站内搜索引擎更新，酷！ It's
- 交互设计栏目新增文章
- 更新论坛精华
- 用例栏目新增文章
- 12月15日与Alan Cooper交流记录

最新招聘

- 北京软通动力科技有限公司 It
- Bamboo Networks Ltd

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 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 Watts Humphrey Mark Paulk John Vlissides 高焕堂 钱叶云文.....

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 模式 It's 工具 过程 组件 交互设计 It's 测试 国内书籍章 物件导向杂志 PMT评论

论坛精华：UMLChina讨论精华

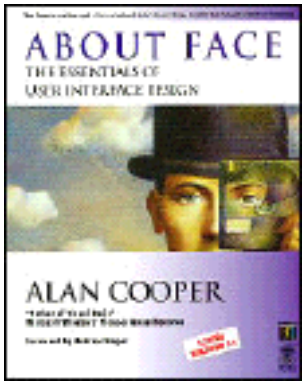
行业动态 OO和UML 模式 工具 过程 组件 交互设计 国内书籍

服务中心：

Architect组 翻译组 OOAD培训，培训资料下载

Alan Cooper：垃圾，都是垃圾！

2001年12月15日上午10:00-12:00，“交互设计之父”Alan Cooper作客UMLCHINA讨论组的聊天室，畅谈交互设计。Alan Cooper是软件交互设计的先知和传播者，客户咨询公司Cooper Interaction Design的创始人和总裁。他设计了第一个可视化开发工具Visual Basic，他的公司帮助IBM、3M、Ericsson、Sun、Visa...等客户开发了许多具有市场竞争力的产品。要更多了解Alan Cooper的思想，请看以下资料：（1）《非程序员》第七期的第一篇文章“交互设计之父Alan Cooper访谈”；（2）去书店买他的著作：《软件创新之路--冲破高技术营造的牢笼》。以下是交流实录。由dewen翻译。错误难免，如有疑问，请对照原文。



fly cat: 大家对您的《软件创新之路--冲破高技术营造的牢笼》这本书有很深的印象，在这本书中你想说的是什么呢？

alancooper: 我尽力想阐明另外一条关于软件如何设计和发展的思路。

extreme: 能告诉我在哪儿能下载这本书的英文版本吗？谢谢！

alancooper: 我想可能没有地方能够下载到该书完整的英文版本，你可以从Amazon.com在线购买一本。

umlchina: Cooper先生，你认为Windows的界面怎样？和Mac比较呢？从交互设计的角度出发！

alancooper: 我认为Windows和Mac的界面是相似的。

umlchina: 相似——那是好还是坏呢？

alancooper：在1984年是同样的好，但在2001年，就是同样的过时、灰暗和单调了。

yuminghui：绝大部分中国大学生仅仅知道简单的VB编程，但是却认为自己是顶尖高手。

alancooper：在美国也是如此。

yuminghui：我指的是那些年轻的家伙，包括那些刚开始编程的人。他们在开始时不应该用VB或UML等类型的工具。

alancooper:优秀的程序人员编程不依赖于任何具体的程序设计语言。

boss_ch：你认为Windows XP的界面怎样？

alancooper：我认为Windows XP和其他的Windows及Mac是一样的。

extreme：据说C#将很快取代VB，Cooper你对此怎么看呢？

alancooper: C#看起来象一个很好的语言。

guodd：顶尖的程序员是否象艺术家？

alancooper：我不认为编程是一种艺术，我觉得它像一种工艺，就象制陶或吹制玻璃。

fly cat：你强调了交互设计的重要性，不过在我看来，当一个程序需要人机交互的时候，它是有用的。不过对于有些程序来说.... 例如对于实时监控设备的内嵌系统，还有用吗？你认为呢？

alancooper：我不认为交互设计对于程序员做任何系统都是有效的，交互设计是针对交互设计师的。

fly cat：编程是工艺，但是设计是艺术？

alancooper：设计也是一种工艺。

extreme: Cooper先生，你认为作为一个交互设计师需要什么样的背景或知识基础。

alancooper：好的交互设计师需要接受交互设计方法的培训。

extreme：你是说，绘画或者艺术？

alancooper：艺术意味着在软件设计或规划方面没有规则。

shenqw：交互设计师需要懂得编程吗？

alancooper：交互设计师需要懂得什么对于程序设计人员是重要的，但是不需要知道怎样编程。

fly cat：什么样的工作背景对交互设计师来说是最重要的？技术或市场？

alancooper：“技术和市场”，两者都不需要，交互设计的字义本身就已经说明了它自己。

extreme：我猜想他可能需要懂得心理学方面的知识，你说呢？

alancooper：交互设计师必须知道哪些对于心理学家是重要的。

yuminghui：这样说来老板需要招聘心理学专家参加项目了？

alancooper：不，正好相反，因为有了交互设计师，所以他们再也不需要心理学专家参加项目了。心理学专家不能设计交互系统。

wrymy：现在流行的网页设计，例如HTML、XML等，我们发现想找一个界面友好的网站是很困难的，特别是企业网站。Web页面上的控制是非常困难的，特别是当界面很复杂时！

alancooper：“网页的控制非常困难”，千真万确！

shenqw：但是交互设计师怎样知道他们应当在屏幕上显示什么东西？

alancooper：这和程序员知道该在编辑器中输入什么样的代码一样！

wrymy：那么你对于网页的编写有什么建议吗？

alancooper:把所有的交互组件放在一个页面上。

extreme：如果某人想成为一个交互设计师，他应该怎样做呢？

alancooper：停止编程。

extreme：为什么要停止编程？

alancooper：因为程序员和设计师的目标是有冲突的，你不可能同时做好两件事情。

umlchina：“程序员和设计师之间有冲突”：但是他们在同一个团队中。

alancooper：不，程序员和设计师的“目标”是有冲突的。

extreme：看来我没有机会成为一个交互设计师了，因为我是如此地喜欢编程。

alancooper：如果你如此喜欢编程，为什么你想成为交互设计师？

fly cat: Cooper先生，你能简要描述一下交互设计的实质吗？

alancooper：让技术为用户服务，而不是让用户服务技术。

extreme：那么，你认为作为一个程序员或PM应当了解交互设计吗？

alancooper：好的交互设计师在你采用他们的计划时可以给出令人信服的原因。

umlchina：一个好的交互设计师一般使用哪些工具？你能介绍一个吗？

alancooper：我们用定性的现场调查技术，用户建模方法，设计原则，设计模板，许多纸和白色书写板。

extreme：但是如果作为一个PM对此没有任何了解，我怎样知道那小子将带领我们走向成功呢？

alancooper：对，你怎样才能知道呢？编程方面的知识能帮助你吗？我认为不能。

extreme：那么，什么能帮助我？

alancooper：一个交互设计师。

extreme：但是我认为现在在中国发现一个好的交互设计师不是很容易。

alancooper：寻找一个好的交互设计师和寻找一个好的程序员同样的困难，在任何地方都是如此。

wrymy：我认为extreme的意思说，怎样判定交互设计师设计的好坏呢？

alancooper：为什么程序员要越过交互设计师来参加判断呢？

hdw1978：灵感，在绝大多数的案例中，你从哪里得到灵感？

alancooper：程序员从哪里得到灵感？

umlchina：有什么好的参考书能告诉我们怎样一步一步来做交互设计？

alancooper：还没有，但是我们现在正在编一本。

fly cat：我有一个感觉，Cooper先生是想鼓励我们自己寻找答案，而不是直接告诉我们。

alancooper：交互设计不是工程类学科。

umlchina：对于新书的编写你有什么计划吗？

alancooper：在Cooper有两个高级设计师从事教材编写的工作。

extreme：我建议Cooper先生做一个中国的培训计划，以便能给我们一些实际的指导。

alancooper：这是一个好注意。

shenqw：什么样的人能够成为交互设计师？他们应当有哪些方面的能力？

alancooper：交互设计师需要具备凭空想象复杂行为的能力，交互设计应当在任何代码编写之前做。交互设计师必须能够在代码被写出来之前，想象它是做什么的。

fly cat：但是我认为如果交互设计师没有实际的经验的话，他们想建立威信是非常困难的。

alancooper：确实如此

extreme：这种凭空想象复杂行为的能力能够训练出来吗？

alancooper：只有当你生来就是个天才，才不用接受训练。就象如果你生来就是编程天才，就不需要编程能力训练一样。

fly cat：我感到很困惑。

alancooper：困惑--这是一个非常好的开始！

fly cat：然后呢？

alancooper：然后你得作好准备学习一种非常困难、非常不同的技术。

alancooper：我不懂一个工程师为什么想做交互设计师？工程师们厌恶交互设计师所做的工作。交互设计师要处理的是人。没有CPU、编程语言、操作系统！

extreme：我们只是想能更好地和交互设计师沟通。

alancooper：这真是太好了！

shenqw：按你的观点，交互设计师必须有一些编程经验吗？

alancooper：为什么你这样认为呢？

fly cat：因为如果他没有编程方面的经验，他想让程序员听他的将非常困难。

alancooper：你的意思是工程师不听从交互设计师的安排？为什么不呢？

hzmajw：艺术家能从事交互设计师的工作吗？或者办公室助理能从事交互设计工作吗？

alancooper: 交互设计不是艺术。办公助理能写代码吗？交互设计与写代码相比是一个复杂的、老练的、困难的工作。

fly cat: 在交互设计师建立他的威信前，形势是非常困难的。成为一个好的程序员也是建立他的威信和自信的一种方法。

alancooper: 为什么交互设计师要向程序员证明自己？为什么不是程序员向交互设计师证明自己？

fly cat: 但是在实际中，这是现实存在的。

alancooper: 我的工作就是要改变现实。

shenqw: 好，交互设计师不需要编程经验。那他怎样在代码被写出来之前想象它是做什么的呢？

alancooper: 那是交互设计师要做的，如果你不能做到这一点，你就不是一个交互设计师。

developerly: 在你成为一个设计师之前，你做过编码方面的工作吗？

alancooper: 仅仅做了大约15年。

extreme: Whao！仅仅15年！

fly cat: 在我看来，你也需要通过VB之父这种经历来建立你的权威。

alancooper: 我编写了VB的可视化部分的代码。

developerly: 那么你认为你的编码经历对你的设计工作曾有过帮助吗？

alancooper: 在我的公司，我们有40位交互设计师，他们没有一个人写过代码，他们只是将代码要实现的功能形象化。

extreme: 这样的话，你怎样让我相信一个好的设计师不需要了解任何编程方面的东西呢？

alancooper: 许多Cooper's的设计师从来没有写过一行代码。

umlchina: 有没有交互设计方面的工具？

alancooper: 有，但是没有一个能脱离软件。

alancooper: 你们刚才问我的问题，和六年前加州的程序员们问我的问题差不多。

extreme: 那也是为什么我们要来这里听一些意见的原因。

alancooper：我真的感激你们的参与和对此的兴趣。

fly cat：好象一个好的程序员很难成为一个好的交互设计师。

alancooper：为什么他们想成为交互设计师？

umlchina：既不是艺术家，也不是程序员，想开发这样一个新的职位是很难的。

alancooper：对，没错。我们现在已经做到了。

fly cat：在中国，程序员的发展之路是：编码-->设计-->管理。

alancooper：我认为这是不正确的。应当是初级程序员-->程序员-->高级程序员；初级设计师-->中级设计师-->高级设计师；初级管理者-->中级管理者-->高级管理者。程序员通常是一个差的设计师和差的管理者。所有的工作职称都含糊不清和令人困惑。

jazy：我想你的意思是交互设计师负责表达，程序员负责逻辑，缺少任何一部分都将做不成事情。

alancooper：你说得对。“缺少任何一部分都将一事无成”：对，对，对！！！！！

hzmajw：每个人都应当考虑哪种角色最适合他自己。当他有目标时，他将能把他的工作做得很好。

alancooper：真正喜欢编程，而且确实精于编程的人，应当继续编程，做别的都是浪费。

extreme：但是有时候老板需要你成为一个PM，那应该怎么办？

alancooper：那老板应当回去编程。

extreme：你现在已经成为一个著名的交互设计师了，这是否意味着你不再喜欢编程了？

alancooper：我喜欢，但是我对它不再有足够的兴趣了。

fly cat：但是我认为，当我们年轻时，我们喜欢做一些编程方面的工作，去了解一些事情，然后我们有了更多的经验，自然地我们就想做一些设计方面的工作。

alancooper：这仅仅对有些人来说是对的。

umlchina：根据你的交互设计观点来看哪一种手机更好？Motolora, Nokia, Ericsson?

alancooper：我认为他们都是垃圾。

umlchina：但是...他们都很笨吗？加上microsoft、IBM？他们都是一样的笨？

alancooper：手机不应该成为一团。为什么我需要将键盘举到我的耳边？为什么作为一个和人交流的工具要有号码？我认为手机应当是隐形的，一个带小型麦克风的听筒放在头部。

extreme：隐形是什么意思？藏在衣服里？

alancooper：“藏在衣服里？”，这是一个好的开始。

fly cat：蓝牙？！

alancooper：蓝牙是一个好的主意，但不是一个非常好的协议。手机最重要的部分是如何控制它，而不是如何通话。为什么手机需要用到我的双手、双眼、一只耳以及我全部的注意力？

品雪：有很多这样的例子：用户告诉你需要在很短的期限内完成项目，然后给你提出许多不断变化的需求来。这里还能有某种交互设计吗？

alancooper：这只能说明那里的管理非常差。

品雪：您对这样的坏环境有什么建议吗

alancooper：建议，在没有完成软件行为描述文档前，告诉管理者不要让你们开始编码。功能和表达不是一回事。特征和行为不是一回事。

extreme：你认为我们能怎样将行为文档化？

alancooper：画图形。

extreme：用什么工具？photoshop?

alancooper：用铅笔。我们经常用白板。PowerPoint也挺好的。

extreme：那么，我们到哪里和怎样发现这样的设计师呢？

alancooper：他们就在那里。可能不在编程人员中，可能在技术支持，测试，文档编写...等人员中。当交互设计师被压抑如此长的时间之后，寻找他们是非常困难的。

umlchina对品雪说：我认为交互设计是对产品的，而不是对项目的。

alancooper：产品和项目，其中的差别是不明显的。对于用户来说，没有一个比设计更重要。

fly cat：在美国，交互设计师是否已经普及了？

alancooper：正在不断发展。

umlchina: web将何去何从？HTML作为交互设计并不适合？

alancooper: HTML是垃圾。

fly cat: 在美国推广交互设计遇到的最大困难是什么？

alancooper: 让高级主管相信不需要花费很多金钱和时间，他们的软件可以有重大的变化。

umlchina: 但是它简单。

extreme: 也流行。

alancooper: 所以是犯罪。

extreme: 但是对于在互联网上的信息交流，它做得非常好。

alancooper: 不，完全不是这样。

umlchina: 流行<-->犯罪？

alancooper: HTML<--->犯罪！许多HTML的限制能通过好的设计来弥补，但是很少有程序员能听从设计师的意见。越过HTML的限制也需要大量复杂的代码。

fly cat: 你考虑过在中国推广你的观念吗？仅仅依靠书是不够的。

alancooper: 我现在正在推广它们。回去也请告诉你们的老板！

extreme: 我非常希望能在中国看到你的书的英文版本。

alancooper: 我也想看到我的书在中国销售。

kenxia: 多少钱？我希望它不要太昂贵。

alancooper: 对书我没有办法控制，它由出版商决定。

fly cat: 你能列举几个不是你们公司的好的交互设计师吗？可能他们和你做同样的事，但是不叫交互设计。

alancooper: Rob Haitani设计了Palm Pilot。Ben Schneiderman设计了SmartMoney.com marketmap。

kenxia: 你认为C++怎样？

alancooper: 学习C++和学习弹钢琴一样容易。

umlchina: WEB将走向哪里？既然HTML是垃圾。

alancooper: 更聪明的浏览器。它应当能记住信息，将相关资料写到磁盘上，和服务器交互通信。它应当有更好的交流方式，比如象拖拉、卷动、刷新。

extreme，umlchina: “往磁盘上写信息”安全吗？“聪明的浏览器记录资料”与隐私权相冲突吧？

alancooper: 对，那是需要解决的问题。但是解决起来并不困难。

fly cat: 如果交互设计师做错了一些事情呢？他们在项目中的地位是如此重要，所以很容易毁了所有的事情。

alancooper: 如果程序员做错了一些事情呢？

extreme: 解雇他！

alancooper: 对，专业人员不能犯错误。

smilemac: Alan你好，对于一个项目中有限的时间及好的交互设计之间的冲突你是怎样看的？

alancooper: 为什么有时间限制？谁将时间定得这么短？为什么定得这么短？他缩短时间是为了什么？

joy_wind: 时间就是金钱。

alancooper: 浪费了的时间是金钱。

joy_wind: 老板关心他的钱，我们关心项目。

alancooper: 管理者将时间期限定得短是因为他们不知道程序员在做什么。交互设计师能告诉管理者程序员在做什么。交互设计师能给程序员争取更多的时间。

extreme: 按你书中的观点，管理者必须布置一些事情。

alancooper: 仅仅是因为在程序被做出来之前，他不知道它是什么样子的。

smilemac: 例如市场压力、预算、管理水平等等，这些都可能造成项目时间限制，许多产品是它们之间相互妥协的结果。

alancooper: “...市场压力，预算...”，所有的都是管理者掩饰他对程序不了解的借口。

fly cat: 我认为交互设计关注的焦点是用户的需求而不是设计。

alancooper：对！！！就象我在前面所说的，交互设计和用户界面设计不是一回事！

extreme：那么谁负责确定软件将要做什么？交互设计师吗？

alancooper：交互设计师就是具有能非常好地解答上述问题，以便别人顺利完成任务的人。

extreme：谁控制过程？交互设计师吗？

alancooper：过程应该由管理者来控制。

joy_wind：交互设计师的工作职责是什么？

alancooper：程序的外观和行为应该由交互设计师控制。技术和编码应当由程序员控制。

kenxia：控制？通过权力还是通过技巧？

alancooper：今天程序员通过什么来控制呢？他们好象不需要特定的技能来做设计。

extreme：这样的话，交互设计师看起来象分析员。

alancooper：交互设计师的工作之一就是分析。但是“分析”不是设计。

joy_wind：交互设计师是否更像建筑师？

alancooper：对，对，对！！！

extreme：但是一个毫无编程知识的家伙怎样来做分析呢？

alancooper：那程序员怎样能决定用户想要什么呢？

extreme：我不是说程序员应当了解需求，我仅仅是想说，一个交互设计师应当了解一些编程知识以便做一些分析工作。

alancooper：为什么？我觉得你的真正意思是你担心交互设计师将会让你去干一些愚蠢的事情！

extreme：对，我不认为一个对编程毫无了解的人能做好分析工作。

alancooper：但是一个好的专业交互设计师不会设计出不能实现的、难以运行的东西。

品雪：你说的分析主要是关于用户知识领域。

alancooper：对！

tipsyy：那么请告诉我，交互设计师在一个项目中负责做什么？他担负什么样的职责？

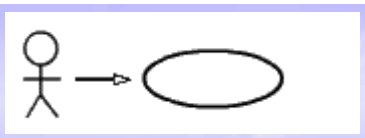
alancooper：交互设计师应当负责程序做什么和怎样表达。我们设计项目的第一部分是针对问题域详细研究项目。

smilemac：交互设计师和项目管理者可以是同一个人吗？

alancooper：为什么？为什么让一个领域的专家工作于另一个领域？

fly cat：时间快到了，你们认为结束聊天好吗？

alancooper：我想我必须走了。很高兴有机会和大家谈话。非常感谢大家！！！！！



软件工程的播种机

| 新手指南 | 行业动态 | 文档中心 | 书籍推荐 | 专家解疑 | 嘉宾交流 | 服务中心 | 人才热线 | 非程序员 | UML

UMLChina讨论组

昵 称：

密 码：

注册 忘记密码 登录

搜索UMLChina

搜索

最新>>

- 《非程序员》第四期提供HTML浏览 It's
- 站内搜索引擎更新，酷！ It's
- 交互设计栏目新增文章
- 更新论坛精华
- 用例栏目新增文章
- 12月15日与Alan Cooper交流记录

最新招聘

- 北京软通动力科技有限公司 It
- Bamboo Networks Ltd

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 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 Watts Humphrey Mark Paulk John Vlissides 高焕堂 钱叶云文.....

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 模式 It's 工具 过程 组件 交互设计 It's 测试 国内书籍章 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

服务中心：

Architect组 翻译组 OOAD培训，培训资料下载

极端编程中“坏气味”的发现与响应

Amr Elssamadisy，Dr. Gregory Schalliol 著，herman 译

[摘要]极端编程（XP）这种灵活的软件开发方法，与传统方法相比较它能更快速地交付软件。在本文中，我们描述一个使用极端编程方法的大型软件开发项目，讨论几种低生产率的实践经验，在这两年中，它们威胁到快速开发项目的按时完成。我们已经把问题存在的范围扩展到整个开发周期，包括分析、设计、开发和测试。对于每一种实践经验，我们将讨论为纠正它们所采取的解决方案，更为重要的是如何检查它们的早期征兆（坏气味（bad smell））。为了保证 XP 项目遵循它快捷的路径，这些征兆是项目经理、分析人员以及开发人员都必须警惕的东西。

[分类和主题说明]

D.3.3 [软件流程]：分析，演进（evolution），流程改进，受益实例(demonstration of benifit)

[普通类别]

设计

[关键词]

灵活方法，XP，重构，演进式设计，分析，坏气味（bad smell）

1 简介

Martin Fowler 在他的书“*Refactoring*”中引用 Kent Beck 对“坏气味”隐喻，描述如何识别一种早期的警示信号，它们指示程序代码的某一部分必须重写。在本文中，我们希望把这一隐喻扩展到必须重新构造整个软件开发流程的早期征兆。对于大型项目，把重写所包括的所有流程与重构单一的过程相比较，我们觉得错误流程的征兆（坏气味））应该引起更多的关注。

我们要讨论的特殊案例是一个 J2EE 开发项目，由于使用传统方法的效率低下，它转而使用极端编程方式。项目团队由 50 人组成，约有 30 个左右的开发人员，都有超过三年的开发经验。应用程序是为一个综合企业制作的工业设备租赁资源解决方案，包括应收帐目、资产管理及合同终止等所有方面。现有 500,000 行执行代码，只有极少数是从采用不同过程的早期开发工作中保留下来的。我们有两年多的时间是这个团队的成员（Amr 是开发人员，Gregory 是分析人员，二人是本文作者），在本文中用到的例子都是

我们在那两年中执行的日常工作的切身体会。尽管在这个项目的执行中，XP（极端编程）被证明是更有效率并且更成功的软件开发方法，然而，这种方法容易产生错误的转折（wrong turn），从而威胁团队快速开发高质量软件的能力。本文将向读者展示这些。

2 分鸡问题

2.1 气味

把整个开发分成“情节卡片（story card)”，这种方法取得过成功，现在却行不通了。

2.2 解决方案

每一次新的迭代之前，重新考虑上一次迭代中使用的过程是否仍然有效。

2.3 讨论

在功能被分为“情节卡片”时，XP 的一个特性就是迭代开发周期。这些情节成了迭代开发的基本元素，而且开发的总预算来自各情节所分配的预算之和。有一些指南教人怎样把应用程序分解成适当的情节，什么可以在迭代中使用，以及什么是用户要求看到的功能，等等。在我们的工程中，我们在不同的阶段遵照这些指南把整体分成细小的部分。可是随着迭代的进行，按照指南在两到三次迭代之前所分解的情节，比起早期的类似项目，产生了很差的预算估计。人类的本性总是趋近于懒惰，我们倾向于重用不考虑使用条件的优先策略。当应用程序变得越来越大、越来越复杂时，在早期迭代中很容易建立的功能模块在每一次迭代中都变得更复杂而且更难完成。所以如果我们坚持在第六次迭代中的 X 卡片中使用第二次迭代中的 Y 卡片（也就是说，坚持在一次迭代中完成用户可用的功能），在第六次迭代结束时，你将得到一个不完全的 X 卡片。理由很简单，为了使这一功能能够在第六次迭代中完整运行，我们必须考虑各种依赖和互连，这些在第二次迭代时都不曾出现。当我们承认这一现实时，我们认识到不要在第六次迭代中期待完整的测试与用户可用功能。如果我们期待迭代保持早期那种令人满意的完成率,情节卡片的分割粒度必须做得更小。

对于这一基本原则的一个比喻就是把整只鸡切分成不同的鸡块。初学者并不容易分辨最容易的分割点（关节），但有经验的人却能很轻易地发现它们。如果某人坚持用同样的方法分割（如切成正方形），那么有些部位不会有问题（如鸡腿肉），另一些部位可能变得很困难（例如从胸骨正中切开鸡脯）。如果我们不坚持把鸡脯（第六次迭代中的卡片）像鸡腿肉（第 2 迭代的卡片）一样切成正方形，我们可以在更合理的预测下更好地区分情节。如果期望开发成功，软件开发流程中的“支持变化(Embrace change)”必须在其自身的特殊实施方式上进行。

3 什么时候顾客应该高兴

3.1 气味

在产品按早期迭代交付时，顾客没有抱怨，但是随着迭代次数的增加，顾客抱怨所有迭代中的很多东西。

3.2 解决方案

用户必须被“训练”成在开发流程的一开始就诚实、真实地反馈问题。

3.3 讨论

在顾客导向的社会里，以下的想法已成为习惯：出钱买东西的人只需做很少一部分工作去避免自己生产的麻烦与头痛。不管如何，在销售完成之后，顾客愿意在哪些方面付出努力以确认其满意度，可以在消费者至上中找到不少例证。打个比方，为了得到期待质量的衣服，传统制衣方式的购买人做的仅仅是花一点时间与裁缝待在一起。

为了使他们所定购的软件满足质量要求，定制软件的客户需要以从裁缝处买衣服的人的思考方式考虑更多的东西。如果可能的话，他们必须完整地投入到开发流程之中。对于像 XP 一样的迭代流程，这是指整个（全部的）开发流程。随着每一次迭代后的新功能交付，以及随着 XP 在每一次迭代后允许客户重新计算其优先级，一个 XP 的客户就再也不能做一时的 IT“购物”，在提交测量数据（=传统的需求采集）后就等待着，期待在几个月或几年后接收一套完整的合乎当时要求的系统。没有在每一次迭代之后持续、诚实的反馈，最后得到令人失望的产品的风险将大得惊人。再回想一下裁缝的例子，如果生产衣服的时间持续数月到数年，如果顾客不定期将新的尺寸送到裁缝店里，衣服不合身的风险会大大提高。体形会随时间变化，力求在高速变化的竞争中站稳脚跟的公司的软件业务需求更是一样。

如果一个 XP 的顾客在早期迭代之后找不到任何可抱怨的问题，他（或她）很有可能是对这一项目参与得不够。这种问题已经在我们的项目中发生过。开发小组鼓励我们的顾客派遣他们的人以全日工作方式参与开发过程，而他们也照做了。来自于顾客的“合作伙伴”尽管与这一系统的最终用户只有一步之遥，他们却不愿遵照我们的建议在每一次迭代之后亲自写功能测试报告以确定系统是不是正在做它应该做的事，而是坚持相信我们所写的功能测试报告。当项目临近结束，开始系统最终用户培训时，用户们开始抱怨在前期迭代中早就交付而且通过检查的功能。造成这种局面的部分原因是客户的“合作伙伴”与最终用户之间沟通不利，而另一部分原因是客户代表在开发流程应该更多地参与验收测试的“艰苦工作”。不亲自测试并撰写测试报告，他们产生一种错觉，认为我们所设想的他们的需要就是他们真正的需要。但是显然，这一点并不是总是正确，因为他们的事务随时间不断变化，而沟通却并非总是完美的。

XP 几乎可以说是软件开发的一种新途径，不管如何，使客户投入整个开发流程是使用 XP 方式的 IT 团

队肩上的责任。在促使客户成为一个真正的“合作伙伴”、参与开发过程的计划与验收的方面我们做得很失败。我们让客户保持舒适，这样我们自己也感到更舒适，但是到了最后，我们为此付出了巨大的代价。如果要客户成长为 XP 所要求的真正的“合作伙伴”，这种新开发方式的从业者必须认识到他们的附加责任：成功地教导客户考虑更多的东西，就像购买高级套装比起购买普通衣服所应该做到的那样。如果客户在一开始就有机会认识到开发的严酷性，在项目结束时，他们将会更加高兴。

4 功能工作，但是不一起工作

4.1 气味

当情节卡片写好并经过分析之后，负责卡片的团体觉得他或者她不能确定在功能测试中计算的这张卡片上的所有功能是不是都已经开发。

4.2 解决方案

对于复杂的应用程序，为开发团队提供一些概要图来提高开发人员对各个情节在整个应用程序中位置的了解。

4.3 讨论

XP 过程最初是为比较小的开发项目所设计的，开发小组不超过 10 个人，而且产品并不是太复杂[2]。对于项目大小的限制有利于实现 XP 的另一基本优势，也就是说，直接免去了写第一行代码之前（或者更早）繁重的设计与需求收集阶段。可是随着 XP 的多种好处变得越来越明显，一些开发团体（就像我们）敢于把修改后的 XP 方法用于更大，更复杂的系统。做这一点的人必须很清楚，他已经超出了使用这一方法的推荐范围，并且准备好适应变化。

我们的项目处理租赁帐目的复杂细节，包括政府规定、税务以及专业会计标准，并不涉及管理所有其他与资产相关的事情。为了完成这一艰巨的任务，我们聘请了几位租赁业务方面的资深专家，他们在培训用户的同时指导实际的开发工作。但是在几次迭代之后，随着应用程序变得越来越复杂，即使是那些专家们也发现他们自己也不能确定正在执行中的新的情节卡片是不是包括了系统的其余所有内容。依照 XP 方法，我们没有进行指示各部分如何结合到一起的概要设计和建模，就直接开始对系统的基本构造块进行编码。因此，每一次新的情节卡片投入使用时，我们都要依赖租赁专家细致分析有哪些其他的卡片依赖于这一卡片的结果，在卡片开始执行之前又要注意哪些东西。随着使用中的情节卡片累积到四位数，我们的专家开始漏掉应该放到卡片中的这个或那个依存关系，而我们自己也发现少数新的功能单独运行得很好，却无法与和它相关的 35 个或更多功能在系统中结合起来一起工作。

所缺少的是某种“图示”，或是一种总体观念，它们将提醒我们在一个飞快变得很复杂的系统中，内

部联络几乎完全失去。光有成堆的情节卡片是不够的，因为情节卡片本身在编写之后也不是一成不变的。XP 不需要前期设计的承诺使我们误认为，即使我们超出了推荐使用这一方法的项目大小，仍旧可以不要前期设计。没有任何可用的“比喻”能帮助我们从这一状况中解脱，因为租赁过于复杂，以至于无法通过更熟悉或更容易理解的图像来使它变得更明晰。

如果我们再一次在同等复杂的项目中应用 XP，我们将拥护在开发与数据更新中包括更传统的图示与图表来显示整个应用程序，作为每一次迭代任务的一部分。在正在开发的应用程序的部件变得越来越复杂和需要更多的内部联系时，这些努力的目标是为编写情节卡片和相关联的功能测试提供指南。为了与 XP 所提供的灵活性相一致，这种图片，与它所表示的功能一样，在开发的阶段中应该在某种程度上易于修改。这种总体图片将提供成堆的情节卡片所不能做到的工作，换句话说，一份保证新的情节卡片被完整地编写和在与所有相关的其它情节中测试的指南。

5 完成 VS.“完成”

5.1 气味

每个人都说所有的情节卡片都已经在规定时间内完成了，可是现在离应用程序交付还有整整 12 周开发时间。

5.2 解决方案

创建一张在认为“完成”的时候必须完成的所有工作的精确清单，然后严厉并且诚实地执行它。

5.3 讨论

通过使用测试第一的编码方式和强迫在周期性的开发迭代后交付，XP 试图避免一种众所周知、影响了众多软件开发项目的，关于“完成”时间的恐惧，具体来说是指充斥着臭虫（bugs）的编码需要大量的时间来修正。此外，借助于从多次早期情节的预估中获得经验，对新的情节的估计将会更加准确，更高频率的最后期限将使得计划编制更加可靠。可是就像我们在项目中发现的那样，使用 XP（如同我们所做的）对这一方面基本没有帮助。

在我们最后一次正式迭代之后，我们的代码充满了臭虫（bugs）和不一致，在向客户交付可以工作的版本之前，不得不花几个月的时间来“清理”。有很多原因导致了这一结果，包括缺少一套自动检测机制（见后面的“气味”），但是一个最根本的原因是我们普遍不愿承认事实。在项目的前一半迭代中，很多团队成员（开发人员和分析人员都一样）发现他们自己急切地冲向每一个迭代时间期限，而不是承认对卡片开发的“速度”估计错误。绝大多数的开发团队成员，包括项目经理，都在寻找修正速度的方法，而他们唯一找到的方法是减少一些功能（cut corner）。项目经理免去了开发人员编写自动测试的工作，开发人

员在迭代完成之后直接投入编码，避开了小的重构。此外，某些应用的重要因素，如应用程序的图形用户界面标准、写入文件等，被轻易地忽略了。这些措施使得开发可以保持着相似的速度，但是当然，这也意味着编码质量不断下降，直到最后“惊讶”地发现编出来的东西无法工作。所有的界面没有一致的标准，而且因为缺少测试而使得臭虫（bugs）丛生。最后的结果是放慢整个项目的速度。由于进行设计是要付费的[3]，很明显这是为脆弱而且难以理解的拙劣编码支付金钱。最后，企图维持速度和加速多半延长了整个项目的完成时间。

我们相信造成这一错误的最主要原因是，很多专业人员不愿承认他们自己对完成情节卡片的任务估计是不准确的。XP 给予项目成员估计他们自身工作的权力，但是同样需要再次明确，这些成员学习预估卡片工作需要时间，而且在最初，他们的估计很少正确。没有足够的鼓励去学习准确的估计，开发团队成员一般习惯于先选择用“减少部分功能”的办法隐瞒错误的估计（就像我们的成员所做的一样）。当他们这么做了几次之后，就变成了一种习惯，最后拖累了整个项目。因为这种习惯在发展起来后很难消除，所以我们劝告其他面对类似项目而且使用 XP 的团队，应该从设立严格并且全面的预测指南（关于情节卡片包含的内容与真正的完成时间）开始。

6 工厂(factory) VS. 实例和前瞻设计

6.1 气味

尽管我们应该是在能够工作的情况下做最少的事，却有一些类对象实例变得越来越多，尽管我们知道在今后的迭代中有一种可以产生所有的实例的灵活工具替代这些东西。

6.2 解决方案

建立一个工厂来代替构造不同的实例。向前看——发现哪些地方你将使用公共方式。即使你不是那么需要附加的灵活性，附加的花费对于整个项目来说完全可以忽略不计。

6.3 举例

想像一下……我们为应用程序设计一个开票功能——所以，第一件我们要做的事是求最小值。让我们从一种特殊的发票格式开始。我们知道——或者至少我们以为自己知道——我们在程序的最初版本中需要一些发票格式，然后我们需要使这种功能变成完全的用户定制，这样，应用程序的使用者将能够任意创建和修改发票格式，只要他们愿意。

所以，我们就像是一个极端编程者（XPer）所做的一样，我们做了保证工作可以运行的最简单的事，我们开发了一个单独的发票格式。太好了，它在迭代中运行得很好，而且我们开始开票。现在我们的顾客在 3~4 次迭代中向我们提供了其他的发票格式。再一次，像一个极端开发成员一样，我们只做了最少的工

作。我们发现自己在不懈地重写代码——直到发票格式间的公共点都不重复。这样我们可以高兴了，因为我们只做了需要做的工作，完成得很快，并且不懈地重写代码，所以我们有了一个好的编码基础。所有的事情都很完美，不是吗？

错！

错？？？你为什么说错呢？如果真是那样，我们当然觉得很好。我们可以为防弹包（package）开防弹发票。而错误是，我们知道将在长时间的运行中综合这些东西。前面所做的工作中没有任何方法可以使我们的客户开一张其他的他们所需要的发票。所以，尽管我们在最初版本中出票功能运行良好，但我们所没有做的事是，本来这部分代码可以很容易编成适应于所有版本的灵活出票功能，而我们却对出票的所有参数进行硬编码（hard-code）。那么我们为什么忽略了这些，而做了这么愚蠢的工作呢？也许在某种程度上希望自己遵从 XP 的前提，尽管我们潜意识地知道我们将需要重做很多工作。

所以最后发生的是更改普通结构（这也是另一个坏气味-下一节 大量的重构），把几种发票实例并入一个发票工厂，它基于一个有大量的参数资料的数据库。

这个努力仅仅只耗费了实际中把所有发票收集到一起的时间。那些数量无节制的硬编码（hard-code）被扔到一边——我的意思是说参数应该放到数据库中。

我们所应该做的事情是认识到向前看和相信自己的直觉是对的，我们走错了路。我们是应该做更多的工作，但是在第一个实例之后我们应该取出这一件工作，并用一个工厂（factory）来代替它。

这将解决哪一类问题呢？无论何时，当你发现自己在为一个解决方案做几个实例工作，或是发现自己要决定一个分级解决方案应该采用写在类级别中的硬编码参数还是工厂解决方案。第一个解决方案一般比较简单，因此你趋向于向它靠拢。不过当你知道，或是怀疑这将给用户带来方便时，那么建立一个工厂只不过是在开始时需要一点开支。这并不会带来大的设计/维护费用[3]——仅仅是大一点的启动花费。总是使用常识，不要忽视它，想想流程知道什么是最好的。

7 大型重构的臭味

7.1 气味

基本上，这儿的前提是如果你发现自己不得不做一个大型重构，那是因为你太懒了，以前没有做你应该做的小的重构。

7.2 解决方案

你还是得做这个大的重构，但是下一次，你可以借助更多的小的重构来避免这一情况，并且不要急于

拼凑你的代码。

7.3 举例

修改租约的一个功能是解除和重订租约。更改正在执行中的租约条款将更改其基本财务原则。在这个部分，我们一开始执行的是基本上修改每一个需要更新的细节部分，但是随着越来越多的功能加入，开发人员（一般每一次迭代都是不同的人员）不断地更新编码使程序能够工作。渐渐我们发现整个方法全都错了——我们需要重做整个解除/重订功能。

真不幸（也许这是种幸运，他们从这个错误中学到的和写在这篇文章中的东西将警告其他的人），所有的编码人员都积极修改这一错误。而且其中有一个喜欢埋怨的人不断抱怨说：“我们自始至终都知道这件事情——对于迭代来说有些事情的确不对头，我们正积极地修改它。”而令人伤心的事实是，他是对的。所有的开发者在最初的运行之后了解到编码需要重构，但是为了这个或那个的原因（见上面的“完成”气味），他们从没有做过重构。

这种气味的确没有容易的解决方案。因为懒惰而造成的设计决策错误累积得过高，所以不得不进行大型重构。在开始时选择容易的道路，并且完全忽视错误信号，我们的编码把自己逼到了角落。所以这里的教训是：当你发现自己要做一个大型重构时，停下来看一看是不是自己忽略了太多小的重构。再也不要做这样的事情！另外，察看整个项目，几乎可以肯定你将发现类似的情况，这一次你可以更早地抓住它。

8 自动功能测试

8.1 气味

所有的单元测试都通过了，可是系统仍旧无法运行。

8.2 解决方案

自动功能测试。就像单元测试一样，当你修正了一个 bug，你需要测试来确定这再也不会发生。很多这样的测试可以被编码，而且随着单元测试自动进行。

8.3 举例

让我们回到上一个气味中的解除/重订合约的例子。整个的解除/重订功能对租赁程序的用户来说是非常有用的。很不幸，这是个非常复杂的事务过程，它取消了所有的财务事务，允许用户按他们的需要编辑，并且按需求重建财务事务。一个特殊的例子是修改租约起始日期，这将使得租约解除并且重订，然后重新计算所有的东西。即使已经通过单元测试，这一更改运行后系统还是崩溃了。也许你会说测试不完善，或者说更改租约起始日期在这一功能内并未完成，但是这不是问题所在，单元测试已经完成。这个问题是因

为其他的业务对象完全没有考虑到这一点。也就是说其他对象的测试不完整，它们没有测试租约起始日期更改的情况。

所以我们设计了另一套测试，自动进行的功能测试。这些测试是针对用户的合格测试。如果这些测试投入使用的话，当更改涉及到多个对象时产生的 bug 被抓住的机率就更高，并且可以在单元测试中就发现这些问题。

9 对象源（MOTHER）和作为测试工具的工厂（factory）特殊实例

9.1 气味

大量单元测试中的设定和分拆（setup and teardown）功能和在复杂对象生命周期的不同部分建立它的困难。

9.2 解决方案

像所描述的那样，创建可以在不同状态下返回不同对象的工具[5]。这也是“General Fixture”[6]中描述的另一方式，这一解决方案对于更大、更复杂的业务对象更为有用。

9.3 讨论

测试，测试，测试是 XP 的主心骨。它可以让你以极快的速度编码，因为你可以依赖于测试来检测错误。当编写与一个大得多的场景（scenario）在一起的小场景的单元测试时，你会发现有很多次你需要一个对象放在场景生存时间中的一个特殊点。举个例子，在租赁系统中，要测试租约起始时间改变的影响，你首先要有一件执行中的租约，租约至少有一项资产，一张票据清单，并且其中要做出一些改变。创建一件达到这些要求的租约是一件很烦重的工作。当然这里有模拟对象解决方案[6]，但是当对象变得非常复杂时而你又想建立自己的测试库时，它很快失去了灵活性。

现在回到问题所在：很多时候在设置合适的测试时，你需要一个或是一些业务对象来编写正确的单元测试，它们处在其生存周期的某个特定时间点和某个特定状态。这是在所有与复杂业务对象相关的开发中都重复出现的需求。这种气味是你发现自己在做测试之前写了大量的设置代码。或者更糟的是，没有写测试，因为设定实在是太多而且不易理解。你发现自己不懈地编写设定代码，所以重新编码，用一种工具产生那些对象。

[7] 一文处理了这个问题，该文出自于 2000 夏季在 Raleigh, Nc 的 XP 全球会议上。在介绍中，一位听众举手问了两个问题，介绍者在回答提问者，他说了大意如此的话：“祝贺你，你刚刚重新发现了工厂（factory）模式。”我们的意见倾向与提问者不同。我们不认为重新发现了工厂模式，但是，实际上，我们认为这是一个很有意义的关于工厂模式的实例，它应该被放到独立的地位，因为它对所有的应用程序有

效测试都是有用的。

的确，这一解决方案没有什么新东西。这些工具是工厂(factory)的一个特殊部分也是一个事实。但是，与普通的工厂（factory）方法不一样，它们一般从最初状态建立对象，而这种工厂（factory)方式用另外的方法建立同样的不同状态的对象。最主要的输入是当整个系统处在正确状态下时——包括所有你容易忘记的添加与设置，因为这些方法实际上创建了初态下的对象，并且遵照执行对象，直到达到你所需要的状态。

10 结语

XP 是一种软件开发方法，它促进了几个多样化项目合理的实现。但就像它所承诺的在时间限度内交付高质量软件巨大成功一样，它，也是依靠于尽责的、仔细考虑怎样才能取得成功的使用人员。对于所有的承诺改进了旧方法的新方法，新用户可能会以为方法可以纠正或者克服问题。以我们的经验来说，我们认为 XP 是一个有价值并且有效的软件开发方法，只要认识到两点：

1）没有尽责的成员它不可能取得成功；

2）如果项目超出了它的创建者所推荐的“小团队”的限制，就必须调整 XP 方式。

11 参考书目

[1] Fowler, M. Refactoring (Reading, MA, 1999), Addison-Wesley, Chap. 3.

[2] Beck, K. Extreme Programming Explained (Reading, MA,2000), Addison-Wesley, p. 157.

[3] Wiki Discussion, Cost Of Design Carry, http://www.c2.com/cgi/wiki? CostOfDesignCarry

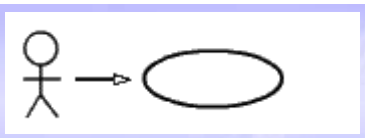
[4] Wiki Discussion, Do The Simplest Thing That Could Possibly Work, http://www.c2.com/cgi/wiki?DoTheSimplestThingThatCouldPossiblyWork

[5] Deursen, A., Moonen, L., Bergh, L., and Kok, G.Refactoring Test Code,

http://www.xp2001.org/xp2001/conference/papers/Chapter22-vanDeursen+alii.pdf

[6] Mackinnon, T. Freeman, S., and Craig P. Endo-Testing: UnitTesting With Mock Objects http://www.connextra.com/about/mockobjects.pdf.

[7] Schuh, P., and Punke, S. ObjectMother: Easing Test Object Creation In XP, http://www.xpuniverse.com/Testing03.pdf.



软件工程的播种机

| 新手指南 | 行业动态 | 文档中心 | 书籍推荐 | 专家解疑 | 嘉宾交流 | 服务中心 | 人才热线 | 非程序员 | UML

UMLChina讨论组

昵 称：

密 码：

注册 忘记密码 登录

搜索UMLChina

搜索

最新>>

- 《非程序员》第四期提供HTML浏览 It's
- 站内搜索引擎更新，酷！ It's
- 交互设计栏目新增文章
- 更新论坛精华
- 用例栏目新增文章
- 12月15日与Alan Cooper交流记录

最新招聘

- 北京软通动力科技有限公司 It
- Bamboo Networks Ltd

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 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 Watts Humphrey Mark Paulk John Vlissides 高焕堂 钱叶云文.....

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 模式 It's 工具 过程 组件 交互设计 It's 测试 国内书籍章 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

服务中心：

Architect组 翻译组 OOAD培训，培训资料下载

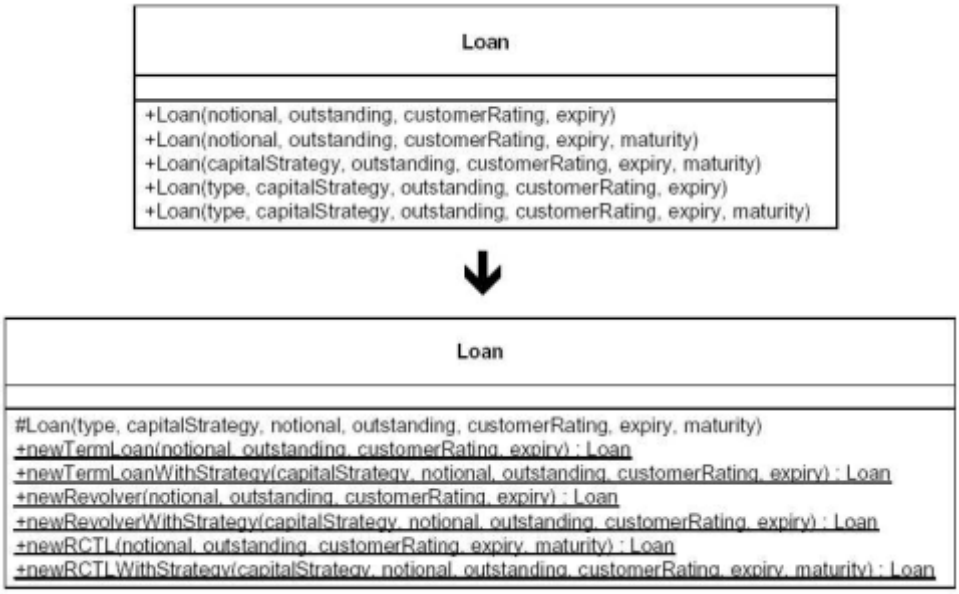
Gigix 译文专栏

用创建方法取代多个构造子

Joshua Kerievsky 著，透明 译

如果一个类中有多个构造子，在开发过程中将难以决定究竟选择哪一个。

用能表现意图的创建方法（Creation Method）返回对象实例，从而取代构造子的作用。



动机

某些语言允许你用自己喜欢的任何方式为自己的构造子命名，而不用管类的名字。另一些语言（例如C++和Java）则不允许这样做：每个构造子都必须按照所属的类的名字来命名。如果只有一个构造子，不成问题；但是如果拥有多个构造子，程序员就必须去了解构造子期望的参数、观察构造子的代码，这样才能正确选择自己要使用的构造子。这有什么毛病？毛病太多了。多个构造子根本无法有效描述意图。拥有的构造子越多，程序员就越容易选择错误。而且，程序员不得不去选择使用哪个构造子，这降低了开发的速度，而且调用构造子的代码经常不能有效地与构造出来的对象交流。

如果你觉得这听起来很糟糕，还有更糟糕的呢。随着系统日趋成熟，程序员们经常会加入越来越多的构造子，而不去检查以前的构造子是否仍在使用。不再使用的构造子仍然留在类中，这增加了类的静态负载，只会让类变得愈加臃肿而复杂。成熟的软件系统中往往充斥着这种“死构造子”，因为程序员没有快速而简单的途径来识别某个构造子是否仍被调用：IDE帮不了他们，判断某个方法确切的调用者所需的搜索语句又实在太麻烦。另一方面，如果对象创建调用主要通过一个特定名称的方法来进行，例如createTermLoad()或者createRevolver()，找到这些方法所有的调用者是轻而易举的。

那么，我们的同行们通常把创建对象的方法叫做什么？许多人都会回答“工厂方法（Factory Method）”，这是因为那本经典著作《设计模式》[GoF]这样称呼一个创建型模式。但是，所有创建对象的方法真的都是工厂方法吗？如果给“工厂方法”这个术语一个更宽的定义：只要创建对象的方法都叫工厂方法，那么答案肯定是“Yes”。但是按照这个创建型模式（Factory Method模式）的作者撰写它的方式来看，很明显并非所有创建对象的方法都能提供真正的工厂方法所提供的那种松耦合。因此，为了在讨论与对象创建相关的设计和重构时保证大家的清醒，我用“创建方法（Creation Method）”这个术语来表示“一个创建对象的方法”。也就是说：工厂方法都是创建方法，但相反则未必。这还意味着：在Martin Fowler或Joshua Bloch使用“工厂方法”这个术语（分别在他们精彩的书“Refactoring”[Fowler]和“Effective Java”[Bloch]中）时，你都可以代之以“创建方法”。

交流	重复	简化
多个构造子不是一种好的交流形式——很明显，通过能揭示意图的创建方法提供对实例的访问，这是一种好的交流形式。	没有直接的重复，只是有许多看上去几乎完全一样的构造子。	指出应该调用哪个构造子并不是一件容易的事。通过能揭示意图的创建方法来构造不同类型的对象，这个问题就简单多了。

技巧

1. 识别出拥有多个构造子的类。这些构造子通常有一大堆参数，而这就让开发者需要获得一个实例的时候更加迷惑了。

2. 识别出catch-all构造子，或者用Chain Constructor（见第8期《非程序员》）创建一个catch-all构造子。如果catch-all构造子的可见性是public，将它改为private或protected。

3. 针对每个构造子所能构造的那种对象，创建一个能够揭示意图的创建方法。测试，确保每个创建方法都返回正确的对象，确定每个创建方法都被客户代码调用到了（如果某个创建方法没有使用者，将它删掉；到需要它的时候再放回来）。

4. 把所有对构造子的调用都换成对相应创建方法的调用。这需要费些力气，但是可以

使客户代码的易读性大大提高。

范例

1、在下面的示例代码段中，我们拥有一个Loan类，其中有多个构造子，分别表示定期贷款（Term Loan）、活期贷款（Revolver）和定活两便贷款（RCTL）。[1]

```
public class Loan {
private static String TERM_LOAN = “TL”;
private static String REVOLVER = “RC”;
private static String RCTL = “RCTL”;
private String type;
private CapitalStrategy strategy;
private float notional;
private float outstanding;
private int customerRating;
private Date maturity;
private Date expiry;

public Loan(float notional, float outstanding, int customerRating, Date expiry) {
    this(TERM_LOAN, new TermROC(), notional, outstanding,
    customerRating, expiry, null);
}
public Loan(float notional, float outstanding, int customerRating, Date expiry,
            Date maturity) {
    this(RCTL, new RevolvingTermROC(), notional, outstanding, customerRating,
        expiry, maturity);
}
public Loan(CapitalStrategy strategy, float notional, float outstanding,
            int customerRating, Date expiry, Date maturity) {
    this(RCTL, strategy, notional, outstanding, customerRating,
    expiry, maturity);
}
public Loan(String type, CapitalStrategy strategy, float notional,
            float outstanding, int customerRating, Date expiry) {
    this(type, strategy, notional, outstanding, customerRating, expiry, null);
}
```

[1] 译注：这里的 Term Loan, Revolver, RCTL 这三个词我都不太理解，请大家将就着吧。

```
public Loan(String type, CapitalStrategy strategy, float notional,
            float outstanding, int customerRating, Date expiry, Date maturity) {
    this.type = type;
    this.strategy = strategy;
    this.notional = notional;
    this.outstanding = outstanding;
    this.customerRating = customerRating;
    this.expiry = expiry;
    if (RCTL.equals(type))
        this.maturity = maturity;
}
}
```

这个类有5个构造子，最后的一个是catch-all构造子。如果只是用肉眼看，你很难知道究竟哪一个构造子创建哪种对象。我碰巧知道RCTL既需要终止日期也需要到期日期，所以我知道要创建RCTL对象必须调用让我传进这两个日期的构造子。但是使用这个类的其他程序员是否也知道这一点呢？如果他们不知道，构造子能与他们充分交流吗？当然了，费点力气，他们也能找出这些规律。但是他们本不应该费这些力气来获取所需的Loan对象的。

在继续进行重构之前，我需要知道上面这些构造子还做出了什么其他的假设。这里有一个比较重要的：如果调用第一个构造子，你会得到一个定期贷款对象而不是活期贷款对象。如果你需要的是活期贷款对象，请调用最后两个构造子，它们会要求你传递贷款类型参数进去。唔……这个类所有的用户都知道这一点吗？我很怀疑。或者他们是否一定要遇到一些非常讨厌的bug，然后才能学到这些？

2、下一步的任务是要识别出Loan类的catch-all构造子。很简单——接收参数最多的那一个就是：

```
public Loan(String type, CapitalStrategy strategy, float notional, float outstanding,
       int customerRating, Date expiry, Date maturity) {
  this.type = type;
  this.strategy = strategy;
  this notional = notional;
  this.outstanding = outstanding;
  this.customerRating = customerRating;
  this.expiry = expiry;
  if (RCTL.equals(type)
      this.maturity = maturity;
}
```

把这个构造子声明为protected：

```
protected Loan(String type, CapitalStrategy strategy, float notional, float outstanding,
        int customerRating, Date expiry, Date maturity)
```

3、然后，我们必须判断出Loan类的每个构造子创建的对象种类。在这个例子中，有下列的种类：

- 使用默认首选策略的定期贷款对象。
- 使用定制首选策略的定期贷款对象。
- 使用默认首选策略的活期贷款对象。
- 使用定制首选策略的活期贷款对象。
- 使用默认首选策略的RTCL对象。
- 使用定制首选策略的RTCL对象。

首先我要为新的创建方法编写测试，让它用默认定期贷款首选策略返回一个定期贷款对象。

```
public void testTermLoan() {
  String custRating = 2;
  Date expiry = createDate(2001, Calendar.NOVEMBER, 20);
  Loan loan = Loan.newTermLoan(1000f, 250f, CUSTOMER_RATING, expiry);
  assertNotNull(loan);
  assertEquals(Loan.TERM_LOAN, loan.getType());
}
```

这个测试无法编译运行，直到我为Loan类加入下面这个静态方法：

```
public class Loan...
static Loan newTermLoan(float notional, float outstanding, int customerRating,
                        Date expiry) {
    return new Loan(TERM_LOAN, new TermROC(), notional, outstanding,
                    customerRating, expiry, null);
}
```

请注意看这个方法如何代理步骤1中识别出的protected的catch-all构造子。我还创建5个类似的测试

和5个附加的能够揭示意图的创建方法，对应于剩下的5种对象。在这项工作完成之后，Loan类已经没有public的构造子了。重构完成后的Loan类看起来象这样：

```
public class Loan {
private static String TERM_LOAN = “TL”;
private static String REVOLVER = “RC”;
private static String RCTL = “RCTL”;
private String type;
private CapitalStrategy strategy;
private float notional;
private float outstanding;
private int customerRating;
private Date maturity;
private Date expiry;

protected Loan(String type, CapitalStrategy strategy, float notional,
               float outstanding, int customerRating, Date expiry, Date maturity) {
    this.type = type;
    this.strategy = strategy;
    this notional = notional;
    this.outstanding = outstanding;
    this.customerRating = customerRating;
    this.expiry = expiry;
    if (RCTL.equals(type)
         this.maturity = maturity;
}

static Loan newTermLoan(float notional, float outstanding, int customerRating,
                        Date expiry) {
    return new Loan(TERM_LOAN, new TermROC(), notional, outstanding, customerRating,
                    expiry, null);
}
static Loan newTermWithStrategy(CapitalStrategy strategy, float notional,
                                float outstanding, int customerRating, Date expiry) {
    return new Loan(TERM_LOAN, strategy, new TermROC(), notional, outstanding,
                    customerRating, expiry, null);
}
static Loan newRevolver(float notional, float outstanding, int customerRating,
```

```
                        Date expiry) {
    return new Loan(REVOLVER, new RevolverROC(), notional, outstanding,
                    customerRating, expiry, null);
}
static Loan newRevolverWithStrategy(CapitalStrategy strategy, float notional,
                            float outstanding, int customerRating, Date expiry) {
    return new Loan(REVOLVER, strategy, new RevolverROC(), notional, outstanding,
                    customerRating, expiry, null);
}
static Loan newRCTL(float notional, float outstanding, int customerRating,
                    Date expiry, Date maturity) {
    return new Loan(RCTL, new RCTLROC(), notional, outstanding,
                    customerRating, expiry, maturity);
}
static Loan newRCTLWithStrategy(CapitalStrategy strategy, float notional,
        float outstanding, int customerRating, Date expiry, Date maturity) {
    return new Loan(RCTL, strategy, new RevolverROC(), notional, outstanding,
                    customerRating, expiry, maturity);
}
}
```

现在，如何获取需要的Loan实例就很清楚了——你只需要看看自己需要哪种对象，然后调用合适的方法就行了。新的创建方法仍然有相当多的参数。Introduce Parameter Object[Fowler]是一个可以帮助你减少传递给方法的参数的重构。

参数化创建方法

在考虑实现这个重构的时候，你可能会在脑子里算一笔帐：为了支持你的类提供的各种对象配置，需要大约50个创建方法。编写50个方法，这听起来可不是件有趣的事，所以你可能会决定不做这个重构。但是，也有办法对付这种情形。首先，你不需要为每种对象配置都生成一个创建方法：你可以为几种最常见的配置编写创建方法，并留下一些public的构造子来处理剩下的情况。另外，经常可以用参数来减少创建方法的数量——我们把它们叫做*参数化*创建方法。比如说，一个简单的Apple类可以根据以下条件实例化：

- 根据苹果的品种。
- 根据苹果的产地。
- 根据苹果的颜色。
- 有籽还是无籽。

- 剥好皮的还是没剥皮的。

这些选项表现出了几种不同类型的苹果，尽管它们没有被显式定义为Apple类的子类。为了获取你所需要的Apple实例，你必须调用恰当的构造子。但是对应于很多类型的苹果，Apple类可能有很多构造子：

```
public Apple(AppleFamily family, Color color) {
    this(family, color, Country.USA, true, false);
}
public Apple(AppleFamily family, Color color, Country country) {
    this(family, color, country, true, false);
}
public Apple(AppleFamily family, Color color, boolean hasSeeds) {
    this(family, color, Country.USA, hasSeeds, false);
}
public Apple(AppleFamily family, Color color, Country country, boolean hasSeeds) {
    this(family, color, country, hasSeeds, false);
}
public Apple(AppleFamily family, Color color, Country country,
         boolean hasSeeds, boolean isPeeled) {
  this.family = family;
  this.color = color;
  this.country = country;
  this.hasSeeds = hasSeeds;
  this.isPeeled = isPeeled;
}
```

正如我们前面提到过的，这么多的构造子只会让Apple类的使用更加困难。为了改善Apple类的可用性，而又不编写大量的创建方法，我们可以识别出最常见的苹果种类，并为它们准备创建方法：

```
public static Apple createSeedlessAmericanMacintosh();
public static Apple createSeedlessGrannySmith();
public static Apple createSeededAsianGoldenDelicious();
```

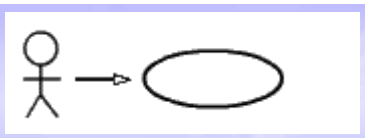
这些创建方法不能完全替代public的构造子，但是可以起到补充的作用，并且有可能减少构造子的数量。但是，因为上面的创建方法不是参数化的，随着时间的流逝，它们的数量很可能翻番，变成

许多许多的创建方法，这也同样会让Apple类的使用者难以选择。因此，在面临如此多的可能性时，编写参数化创建方法通常都是有意义的：

```
public static Apple createSeedlessMacintosh(Country c);
public static Apple createGoldenDelicious(Country c);
```

参考文献

- [Bloch] Bloch, Joshua. *Effective Java*. Addison-Wesley, 2001.
- [Fowler] Fowler, Martin. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.
- [GOF] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software.* Reading, Mass.: Addison-Wesley, 1995.



软件工程的播种机

| 新手指南 | 行业动态 | 文档中心 | 书籍推荐 | 专家解疑 | 嘉宾交流 | 服务中心 | 人才热线 | 非程序员 | UML

UMLChina讨论组

昵 称：

密 码：

注册 忘记密码 登录

搜索UMLChina

搜索

最新>>

- 《非程序员》第四期提供HTML浏览 It's
- 站内搜索引擎更新，酷！ It's
- 交互设计栏目新增文章
- 更新论坛精华
- 用例栏目新增文章
- 12月15日与Alan Cooper交流记录

最新招聘

- 北京软通动力科技有限公司 It
- Bamboo Networks Ltd

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 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 Watts Humphrey Mark Paulk John Vlissides 高焕堂 钱叶云文.....

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 模式 It's 工具 过程 组件 交互设计 It's 测试 国内书籍章 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

服务中心：

Architect组 翻译组 OOAD培训，培训资料下载

通过 CMM 评估的战略

Victor Stachura 著，zhoufang 译

你们的过程改进工作将要被一组专家检查。他们将冷静仔细地研究你们的组织结构，发现其中的缺点。这是一个噩梦吗？不，这仅仅是在评估你们组织的能力成熟度等级。

你和你的团队为改善软件过程已经努力工作了一段时间，并希望你们开发和配置的过程和程序能够达到 SEI 的能力成熟度（Capability Maturity Model (CMM)）的要求。你们所有的培训，制作的过程文件，执行的质保检查都将被一个 CMM 的主评估师仔细审查。基于 CMM 的内部过程改进评估，或简称为 CBA IPI，是一个令人神经紧张的经历。就我所在的组织通过 CBA IPI 的经历而言是这样的。你，你的团队，你所在的组织正在做的所有事情都会摆在一个评估组面前，他们会仔细研究，发现你所在组织在软件开发和项目管理过程中的缺点。然而，如果你们的准备充分而且仔细，并且有正确的管理方针，那么在 CBA IPI 过程中就不会有什么问题。

我现在负责我所在组织的过程改进工作，并经过了几次 CMM 二级包括的关键实践在内的 CBA IPI。当然，我是主评估师，根据我的经验，我有些体会也许会帮助你通过你们自己的 CBA IPI。

评估过程

CBA IPI 是评估一个组织软件过程能力的一种方法。在 7 天的时间内（或更长的时间，这主要取决于评估的范围），通过发现组织的优点和缺点，便能准确地描绘出一个组织的软件能力。一个评估组会检查文档，与项目经理，软件开发者甚至中层经理面谈。CBA IPI 方法需要一个主评估师和一个评估小组。主评估师来自被评估组织之外，评估小组则是由来自组织内部和外部的人员组成的。评估小组的规模为 4 到 10 人，并且 SEI 要求小组中至少有 1 人来自通过评估的组织。主评估师负责在评估之前对小组成员进行培训。有一个现场协调员负责制订围绕 CBA IPI 的所有计划。

一次评估可以有如下一些目标：

- 确认你们的软件过程成熟度符合 CMM。
- 识别需要过程改进小组努力的薄弱领域。
- 识别可以移植到组织中其它部分的最好的实践活动。

在评估活动开始之前需要拟订一个评估计划。包括目标、范围、时间表，编制可能发生的评估过程的步骤。在评估范围中值得一提的是：你可以不选择那些不适用你们业务的那些关键过程域。如果一个组织从不发生第三方分包合同，那么软件分包管理可以声明为“不在范围之内”，排除在评估之外。软件分包管理是最经常不在评估范围之内的 KPA。

当评估开始时会发生什么？通常需要做一些团队建设方面的工作。在工作开始前，评估组成员间的相互了解是十分重要的。要开始的第一项实际工作是开始文档的审查：仔细审阅你和你的小组准备的成堆的文档。评估小组审查三类文档：第一种是组织级文档，是指那些组织成员都应该了解和执行的文档；第二种是项目级文档，是指具体项目程序文件；第三种文档是执行文档，这是用来记录工作是如何遵循组织级和项目级规定的过程执行的。SEI 建议应花 80%的文档审核时间来审核执行级文档；这些记录能表明活动是否遵循 CMM 的要求。

文档审核完毕后，真正有意思的事情——与参与者面谈开始了。大约会抽 45 人面谈，包括项目经理，职能部门的代表，中层经理和初级技术人员。通过面谈，评估组开始全面地了解该组织对于过程的理解并且将观察到的该组织的过程制度化情况——也就是该组织成员是如何理解和执行过程的情况，详细地记录下来。评估员想听到雇员是如何从事与 CMM 相关的工作的。面谈人员的问题十分广泛，目的在于发现一些特别的回答。由于问题是精心编制的，所以它们的答案是不可预见的。问一些能用“是”或“不是”回答的问题几乎没有什么用途。问题是与运用的过程和环境相关的。

在制作调查报告时，评估组必须遵守一些确保正确的准则。调查报告依据的信息是通过两次独立的面谈过程收集的，或者至少一次面谈加一次文档审查。调查报告的资料最好有多种来源。评估组讨论每个调查报告并通过表决决定它的有效性。一个有效的调查报告应具有准确性，确定性，并且和其他的调查报告一致。在保证有效性的前提下，评估应尽可能充分地覆盖组织的开发生命周期和 CMM 的关键实践域。当制作调查报告和进行有效性投票表决时，数据收集过程中的漏洞会表现出来。你将发现只有一个来源的调查报告是不完整的。（也就是说，该调查报告不能说明一个实践活动中的所有成分。）如果这样，要重新修改面谈问题，把重点放在那些涉及不多的领域。要象律师一样，为得到特定的信息而设计问题。这时，缺点就会暴露出来。但这时还不是讨论这些缺点的时候，要把它们留在等级评定阶段。

所有的面谈结束后，调查报告也已制作完毕并有了初步的结论，该结论提交给受评估的组织。这个过程也是一个信息收集阶段，这时被评估组织可以对某个特殊的意见表示赞同或反对。

通过评估的小窍门

以下是我总结出的顺利通过评估的 7 个小窍门。

1. **把评估作为一个小项目来运作。**准备一次评估活动需要做许多工作，有组织的评估活动能够使整

个活动的进展顺利得多。面谈时间安排，会议室的准备，用品购买，配置计算机设备，制作出入证等等，这些事情都必须事先准备。另外，被面谈人应知道面谈时间表并且届时要提示他。如果一个组织不能顺利地完成一次评估活动，那么怎么可能管理比评估复杂得多的软件开发项目呢？

2. **文档的准备。**不要将一大堆文档堆在评估组面前让他们自己决定那些是用于评估 CMM 活动的。详细地审查文档不是一件愉快的任务，并且紧凑的时间表使评估组难于发现所需的信息。如果将文档与 CMM 活动进行对应，评估活动会进行得很顺利。一些组织甚至用不同颜色的编码来表示不同的项目，以方便评估组判断某个具体的文档是出自哪个项目的。

3. **环境的准备。**一个舒适的工作环境会使评估活动变成一件愉快的事（实际上，我并不认为评估活动是愉快的）。每个评估组成员需要充足的工作空间。因为文档，文件夹，CMM 的相关资料将可能同时被使用。而一个炎热和不通风的房间会使评估组成员变得急躁。

4. **要放松。**我所在的组织进行 CBA IPI 时，我会经常做噩梦，但我建议你要放松。如果你已经作了评估过程中的所有工作，那么你就没有什么可担心的了。如果你是评估组的成员，记住要区分缺点的重要性和有效性。组织内部的评估员可能会感情用事和不可避免地紧张。当第一个缺点暴露出来时，就该缺点的重要性可能会有一个长时间但并不必要的讨论。尽管某个缺点不会影响评估结果，但人们仍会担心。当发现一个缺点后，评估组会准备一些面谈的问题来发现原因。继续进行面谈，并且允许对文档进行再次审查。

5. **要了解调查结果和推断的区别。**评估组成员必须懂得调查结果是从文档和面谈中得到的，而推断是虽然未被证明，但他们认为是正确的东西。制作调查结果时往往很容易将推断加进去。要不断地问自己调查结果中的内容是否真的是看到的或听到的。在我参加的评估活动中，我总是同来自被评估组织内部的某个人一起工作，他知道该组织存在某个特殊的实践，但是，如果没有证据，我们就不能将它包括在调查结果报告中。下一步，你可以修改面谈问题来发现需要的信息。

6. **对人员进行有关 CMM 术语的培训。**组织经常采用 CMM 上的有关概念、过程和程序的标准术语。要让被面谈者了解术语的另外一些说法，并且和评估组讨论现场可能提到的术语。最理想的情况是，面谈中的问题采用现场所用的术语，但不总是这样。

如果你询问组织中的某人他的项目业务手册是否受到管理和控制，他可能会迷惑地看着你。但是问他谁有权更改项目业务手册中的文档时，你会得到完全不同的答案——理想情况下，得到的答案是项目业务手册是有管理和受控的。

7. **在面谈间要留有充足的时间。**每次面谈都是很紧张的，因此不应该一个接一个地安排。必须给评估组时间让每个成员发表关于评估过程的意见和看法，并让那些作记录的人的手放松一下。把面谈过程中

的每一件事都记录下来并不容易。同时，尽可能在面谈过程一结束就检查笔记并作相应的标记。要记住，评估组要努力做到准确，不能在面谈结束后才发现遗漏或忘记了某些事。

美梦

总之，你们的 CBA IPI 不一定会遭到挫折。事先计划充分，你可能会睡得很好，不会做噩梦。如果真是这样，那就比我第一次经历 CMM 评估时强多了。

我正忙于我所在组织的 CMM 三级的工作并计划在我们准备好后进行再一次的评估。我坚信到那时我会睡得更好些。

资源

当你们准备好进行 CBA IPI 时，你可以与软件工程研究所或其他一些提供 CMM 评估的公司联系。以下是部分提供该项服务的公司。

Abelia Corporation
Fairfax, VA 22033-2819
Tel: (703) 591-5247
Fax: (703) 591-5005
E-mail: info@abelia.com
http://www.abelia.com/

Association of Independent
Lead Assessors
AILA@consultant.com

Bloodworth Integrated
Technology Inc.
12007 Sunrise Valley Dr.
Suite 105
Reston, VA 20191
Tel: (703) 295-0700
Fax: (703) 295-0999
E-mail: mailto:bitinc@bitspi.com
http://www.bitspi.com/

Chainbridge Technologies Inc.
http://www.changebridge.com/

Davis Systems
238 Northwood Dr.
Harvest, AL 35749
Tel: (256) 837-0058
Fax: (256) 895-9178
http://www.davissys.com/

European Software Institute
Parque Tecnológico, #204
E-48170 Zamudio
Bizkaia, Spain
Tel.: +34 (94) 420 9519
Fax: +34 (94) 420 9420
http://www.esi.es/

Global Systems Technology
5811 Amala Dr.
La Mesa, CA 91942-4156
Tel: (619) 697-9947
Fax: (619) 697-9948
http://www.g-s-t.com/

Process Enhancement Partners Inc.
711 Nob Hill Trail
Franktown, CO 80116-8717
Tel: (303) 660-9400
Fax: (303) 660-9217
http://www.pep-inc.com/

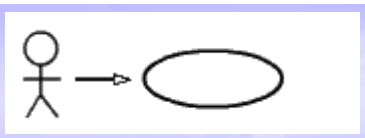
Process Inc.
Tel: (613) 722-8707
Fax: (613) 722-6296
E-mail: David.Constant@ProcessInc.com
http://www.processinc.com/

Process Transition
International Inc.
P.O. Box 1988
Annapolis, MD 21404 USA
Tel: (301) 261 9921
Fax: (410) 295 5037
http://www.processincus.com/

The Process Group
P.O. Box 700012
Dallas, TX 75370
Tel: (972) 418-9541
Fax: (972) 618-6283
E-mail: mailto:mollystevens@mindspring.com

TeraQuest Metrics, Inc.
P.O. Box 200490
Austin, TX 78720-0490
Tel: (512) 219-9152
Fax: (512) 219-0587
E-mail: info@teraquest.com
http://www.teraquest.com/

Theta Information Systems Inc.
4923 Bayshore Blvd.
Tampa, FL 33611
Tel: (813) 902-0402
Fax: (813) 902-0258
E-mail: mailbox@thetainfo.com
http://www.thetainfo.com/



软件工程的播种机

| 新手指南 | 行业动态 | 文档中心 | 书籍推荐 | 专家解疑 | 嘉宾交流 | 服务中心 | 人才热线 | 非程序员 | UML

UMLChina讨论组

昵 称：

密 码：

注册 忘记密码 登录

搜索UMLChina

搜索

最新>>

- 《非程序员》第四期提供HTML浏览 It's
- 站内搜索引擎更新，酷！ It's
- 交互设计栏目新增文章
- 更新论坛精华
- 用例栏目新增文章
- 12月15日与Alan Cooper交流记录

最新招聘

- 北京软通动力科技有限公司 It
- Bamboo Networks Ltd

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 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 Watts Humphrey Mark Paulk John Vlissides 高焕堂 钱叶云文.....

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 模式 It's 工具 过程 组件 交互设计 It's 测试 国内书籍章 物件导向杂志 PMT评论

论坛精华：UMLChina讨论精华

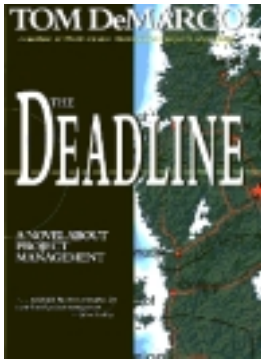
行业动态 OO和UML 模式 工具 过程 组件 交互设计 国内书籍

服务中心：

Architect组 翻译组 OOAD培训，培训资料下载

软件项目管理小说《最后期限》（草稿）节选

Tom DeMarco 著，透明 译



“The Deadline: A Novel About Project Management”是 Tom DeMarco 所著的经典著作之一，以小说的形式，阐述了软件项目管理的哲理。它的中文译本《最后期限》即将发行，译者为 UMLChina 翻译组的透明，这是翻译草稿的节选。

第 03 章 “硅谷”

汤普金斯先生在自己的床上慢慢醒来。他穿着自己习惯的苏格兰绒睡衣，盖着用了多年的蓝白花纹旧被单，脑袋下面是最爱的老式枕头。所有的东西都有家的味道，但是很明显，他不是在自己的家里。就在床的左边，有一扇大大的窗子，而在他的家里，那儿是没有窗子的。而且窗外有一棵棕榈树。想想吧，在新泽西出现了一棵棕榈树！当然，唯一的解释就是：他不在新泽西。

在床的另一边，稍远的那面墙上，还有一扇巨大的窗子。在那扇窗的旁边，他祖母的老摇椅前后摇晃着。坐在摇椅上的是莱克莎．胡利安。她从手上的书中抬起视线。

他的嘴里还有一些怪味，舌头就象一张干透了的厚毛巾。没费太大力气，他就从床上坐了起来。天哪，太口渴了。

莱克莎没有说话，只是指指床边的小桌。他转过头，发现一大杯加了冰块的水。他拿起杯子，大口大口的喝了个底朝天。

桌上还有一个水罐。他又给自己倒了满满一杯，喝到自己感觉不那么渴了才放下。长长的寂静，他不知道下面该怎么办。“那么，”最后他还是开口了，“你真的干了。”

“嗯。”

他诧异的摇着头：“你这种人啊。难道你就不觉得可耻吗？你竟然会破坏一个人的生活，逼他离开他心爱的……”

她给了他一个微笑：“噢，韦伯斯特，没有你说的那么糟糕。你心爱的什么？工作？居住的城市？当然，那儿有你的朋友，但是你被辞退了，如果在别的地方找到新工作，你还是得离开他们。现在，你在这儿，你找到了新的工作——很多的工作。我们破坏了你的什么？”

这倒也不假。谁会真的想念他呢？谁又不是随时可能离去呢？“我有一只猫！”他突然悲伤的说道，“一只可怜的小灰猫！在这个世界上，它只能依赖我一个人！我的猫的名字叫……”

“希福。”她接过话头，“是的，可爱的小希福。我们已经是好朋友了。”她挠挠椅子腿，一只白色脚爪的灰色小猫立刻蹦蹦跳跳的跑到了她的身边。

“希福！”汤普金斯大喊起来，“离开那个女的！”

希福根本不理会他。它爬上莱克莎的膝盖，蜷成了一团。莱克莎挠挠它的头顶，那可爱的小家伙发出快乐的咕噜声。

“叛徒！”汤普金斯气急败坏的叫道。

他的衣服摆在梳妆台上，一条牛仔裤、一件棉布衬衣、内衣和内裤。他直盯着胡利安女士，给她最明显的暗示：他想要一点隐私。她调皮的笑了起来。汤普金斯抓起衣服，走进浴室，关了门，插上锁。

浴室很大。厚厚的墙上打开着的窗子至少有六英尺高。他把头伸出窗外，看见这座建筑物石制的外墙。他所处的房间是在二楼，楼下是一个精巧的花园。浴室里的装置都是老式的，优雅的白瓷洁具和黄铜水管。所有的东西都是那么干净而典雅。他就象是在一家高档的老式瑞士宾馆里一样。

“还需要什么东西吗？”莱克莎的声音从紧闭的门外传来。

“走开！让我自己呆着！”

“我们可以隔着门说话嘛。”

“我们没什么可说的。”

“噢，可是我们有很多要说的。我们必须谈谈你的新工作。我恐怕你已经大大落在进度后面了。”

“我只是被迫来这里而已。”

“计划在执行中。难道不总是这样吗？我毫不怀疑你也遇到过这种情况。一旦落后，你就无法完成了。”

这让他有点生气。他一边扣着衬衣上的扣子，一边走出了浴室。“如果真是这样，如果我接到实际上根本无法完成的一份工作，唯一的可能就是计划从一开始就是错的。那么，又是谁定下的计划？毫无疑问，一定是哪个傻瓜！你应该让这个傻瓜滚得远远的！老是接到“不可能完成的任务”，我都烦死了！”

“你生气的样子真可爱。”

又是那个让人气恼的微笑，她看上去如此美丽。“我一点都不觉得这有多好玩，小姑娘，一点都不好玩。别跟我这样说话。”

“好的，先生。”她做出一个懊悔的表情。一个魔鬼般的女子，在装出懊悔的表情时更是如此。

汤普金斯在祖母那张垫着厚厚坐垫的软椅上坐下，面对着莱克莎。“别再跟我绕圈子了。如果我完全拒绝为你和你那愚蠢的工作做任何事，你会把我怎么办？如果我坚决的对你说”不”，你会把我怎么办？你会把我活埋了吗？”

“拜托，韦伯斯特，我们不会干那种事。如果你不认为这份工作是一个好机会，如果你不喜欢它，我们会把你、希福和你的整个世界完好无缺的送回新泽西，然后祝你好运。我们会先送你到罗马，让你好好渡个周末，休息一下。宾馆、航班都将是第一流的，全由我们付帐。还有比这更公平的吗？”

“我能相信你吗？”

“你能相信我吗？为什么你不试着相信我呢？我从来没有对你说过哪怕一个小谎，不是吗？你想想，难道我对你说的不都是实话吗？”

他轻蔑的摆摆手：“谁知道呢？……如果接受这份工作，我又能得到什么？”

“按惯例，”她说，“钱。当然，还有工作的兴奋、成就感、友情、意义重大的成果，所有这些。”

“是的。说说我的钱吧，有多少？”

她从旁边的文件夹中抽出一些文件："我们考虑一份为期两年的合同。"她递给他一封信。一封来自摩罗维亚某个部门的信，考究的信头写着他的名字。他看了看第二页上的"雇用条款"。他们打算付给他两倍的薪水，并且是税后收入，用美元支付。"呵呵……"他有点惊讶。

"另外，你还能得到一定的股份和期权。"莱克莎告诉他。

他夸张的耸了耸肩。作为一个国家，摩罗维亚能拿出什么股票？他无法想象。

莱克莎递给他另一份文件。这是属于他的忠诚投资公司[1]的帐户存单，在"存款总额"一栏中写着这份合同的总金额，两年的薪水。

"我怎么知道合同到期的时候你们会不会真的付钱？"

她又递给他一张单子，纽约银行的支票，正好是合同的总金额。"提前付款。你接受这份工作，我们立刻把所有的报酬汇进你的帐户。你可以通知你的律师，让他确定存款之后再告诉你。另外忠诚投资公司会寄给你一份书面确认。我们可以让你在一个星期之内拿到所有这些东西。在那之前，你都是我们的客人，你可以把这当成海滩的一次假期。"

"我甚至还不知道摩罗维亚到底在哪儿。"

"在海上。摩罗维亚在爱奥尼亚海上，意大利的东南方。天气好的时候，你可以从阳台上看到希腊中部的山脉。"

汤普金斯先生考虑了一下。"我的工作是什么？"他最后问道。

她俏皮的向他眨眨眼："我还以为你永远不会问呢。"

"我不想兜圈子，"汤普金斯看着面前的简报说，"实际上你们有一千五百名资格相当老的软件工程师。"

莱克莎点点头："这是最近的数字。他们都会在你的手下工作。"

"而且据你所说，他们都很优秀。"

1 译注：忠诚投资公司（Fidelity Investments）是美国，也是全球最大的共同基金公司。所谓"共同基金"，是代理储户投资的一种金融机构。

“他们都通过了摩罗维亚软件工程学院的 CMM 2 级以上的认证。”

“太厉害了！你们怎么做到的？你们都从哪儿找到这么多高级工程师的？我是说，在摩罗维亚这种弹丸小国，谁能想到……”

“亲爱的，你的偏见又开始作怪了。你其实是想说”你们这样一个第三世界国家怎么会有这么强的技术实力”吧？”

“就算是吧。”

“我们以前的政府的确干过一些很差劲的事情，但他们也干了不少好事。他们把市场搞得一团糟，但他们很重视教育。”

他只在书上读到过这些。“我最近在一本书上看到过相似的观点，而且我认为作者怀有一些尊敬的情愫。”

“是的，那是莱斯特.特洛夫的新书，我们发现你的床头边就摆着这本书。他讲述了前苏联的一些优缺点，而摩罗维亚的情况也跟这差不多。”

“那么，我手下所有的人都能用英语读写吗？”

“都可以。在这里英语能力非常重要。摩罗维亚语可不是全球化经济的首选语言。”

“也就是说，你们是想用这些训练有素的天才来打造世界级的软件工业。”

“是的。我们从六个关键性的项目开始，目标是制造出六个精心挑选的软件产品。我们的最高领袖——元首——亲自挑选出这些产品。而你，你的工作就是让这六个项目和整个机构正常运转。”

“这只是工作的一部分。我希望你已经准备好了足够的票子。我是说，人员培训和设备都需要巨大的投资。”

“韦伯斯特，无论如何我们都不希望拖你的后腿。两年以后，当你回想这段日子时，你绝不会说有谁不努力工作，绝不会说人手不够用，绝不会说你没有得到足够的支持。”

“那我们就先来谈谈支持的问题。”

“我会为你挑选一个经验丰富、足智多谋的的个人助理，你还将拥有由两百名顶尖的开发经理组成的核心团队、数十名关键领域的专家……”

“我要带几个我自己的关键人物过来。我会自己选择，然后你们必须把他们请来——注意，我不希望他们象我这样被绑架过来，我要他们自愿来这儿。”

“当然了。”她躲闪着他的视线。

“别想骗我，莱克莎。”

“噢，好吧。你真是个无趣的家伙。”

“另外我可能还需要一些顾问，一些世界知名的顾问。”

“一切都会如你所愿。你只要列出一个名单，我们会把他们都请来的。”

“太对了，你们会的。”他低头看看自己刚才写下的记录，“我还要求所有人都集中在一起。千万不要干傻事，不要让这些人在不同的地方工作。如果你的人分散在不同的地方，你就什么都干不成。把他们集中起来。”

“我们已经这样做了。我们把整个开发团队一起搬到了弗罗泽克盆地，元首把这个地区改名为”硅谷”。”

他们坐在汤普金斯的套间的起居室里。这一侧的房间远离大海，朝向内陆。莱克莎站起身来，示意他走到宽大的阳台上。远方有一个美丽的小山谷。“硅谷。”她说道，一只手划过这个山谷。在他们的脚下有一群新建的办公楼，她指着楼群说：“韦伯斯特，那就是爱德里沃利大学，你的新领地。从这儿步行过去只需要十分钟。”

“非常漂亮。如果漂亮的山谷是获得成功唯一的条件，我猜摩罗维亚早在几个世纪之前就已经站在世界的顶端了。”他又低头看看笔记本：“噢，对了，最后我还要谈谈每个项目的任务，你一定要满足这个要求，没有商量的。”

她做出百无聊赖的样子：“好吧。”

“还有良好的网络支持。也就是说，每个人的办公桌上都要有最新的计算机工作站，都必须用以太网

或更快的网络连接起来。我还要全职的网络支持人员，以及全套的集线器、路由器、T1 或 ISDN 出口。”

莱克莎打了个呵欠：“没问题。”

“还有什么？”他知道，他最好现在把需要的东西考虑周全。现在正是他提要求的时候。“我还忘了什么吗？”

“只忘了一件最重要的事。韦伯斯特，我们用了那么多的力量，只为完成元首拟订的六个项目，你对此有什么看法？”

汤普金斯低头看看她给他的项目列表。的确，工作的总量不会那么庞大。他们需要完成六个项目，制造出六个中等大小的软件产品。他还不知道这些产品具体是什么，但是没有哪一个项目会需要超过二十个人的团队。“我知道你的意思了。看起来我们只需要一百个人就够了。”

“正确。那么，你想怎么处置剩下的人呢？”

“我被你考倒了。这是我应该关心的问题吗？让他们度假去吧。”

“这不是你应该关心的问题，韦伯斯特，不过是你的一个机会。在你的职业生涯中，难道你从来没想过进行一次受控的管理实验吗？难道你从来没想过：如果你不只进行一个项目完成一些任务，而是同时进行三个或四个项目……”

汤普金斯出神的望着远方：“一个受控的实验……一个项目组承受巨大的压力，另一个的压力少些，第三个则几乎没有压力，三个组的任务完全相同。我们可以看看哪个组先完成。是啊，我一直都希望有机会做这样的实验。我们还可以设置一个人员过量的项目组，另一个组则人员不足，第三个组的人数则是我理想中最合适的，然后来观察……”

莱克莎接过话头：“一个团队全由资深人员组成，另一个则由一些资深人员和一些新手组成……”

他完全投入其中了：“一个团队由一直在一起工作的人组成，相对的另一个团队由陌生人组成。天啊，莱克莎，如果完成这个实验，我们就可以开始研究管理中的秘密了。我们可以真正理解项目成功的原因。”

“全在你手中了。整个摩罗维亚全在你手中了，你可以尽情的享受。”她冲着硅谷点点头。“它就在你的脚下，世界上第一个项目管理实验室。”