

【新闻】

- 1 Telelogic收购Popkin...

【方法】

- 2 爱丽丝漫游用例奇境
21 控制软件设计的复杂性
38 领域驱动设计讨论

【工具】

- 45 UML相关工具一览(I-N)



中国机会

X-Programmer
非程序员
软件以用为本

联系: think@umlchina.com

<http://www.umlchina.com/>

本电子杂志免费下载, 仅供学习和交流之用
文中观点不代表电子杂志观点
转载需注明出处, 不得用于商业用途

Telelogic 收购 Popkin

[2005/5/18]

也许你已经忘记了，大概在三周前，Telelogic 收购了 Popkin 软件公司。

Popkin 是十年前老的 CASE 开发商中独立存活下来的最后一家了。在 Rational 被 IBM 收购，现在 Popkin 也已离去之后，几乎已经全军覆没，关注应用开发方面更是没有了。实际上，Popkin 自己，和 Mega 和 Casewise 等公司一样，都已经将重心转移到企业架构领域，在该领域 Popkin 是市场的领导者之一。这样，在纯粹的开发领域几乎就一家都没有了，Select Business Solution 是最著名的一家，另外还有 Excel 软件公司。



Useful links

- » [Telelogic Home](#)
- » [Telelogic About](#)
- » [Telelogic Management](#)

Popkin Software and Telelogic

Telelogic, the leading provider of solutions that automate and promote best practices across the development lifecycle has joined forces with Popkin Software, the market leader in enterprise architecture tools as ranked by industry analyst, Gartner. Telelogic's acquisition of Popkin Software is effective 18 April 2005.



Useful links

- » [Popkin Home](#)
- » [Popkin About](#)
- » [Popkin Management](#)
- » [Popkin Products](#)

Popkin 过去是企业架构领域的领军公司，尤其是在政府（美国）方面，以及航空和国防工业。而 Telelogic 是需求管理方面的老大，其 DOORS 产品占领了 40% 以上的市场份额。另外，Telelogic 还有其它两个产品：提供变更和配置管理的 Synergy product 以及基于 UML 的建模工具 TAU product。

这两家的产品中，TAU product 和 System Architect（Popkin 的产品）都是基于 UML 的，存在一定的重叠。但是，Popkin 更侧重于业务策略和 IT 架构，而 Telelogic 倾向于从 IT 架构方面开始，然后深入到开发层面，因此，虽然存在重叠但可能并没有想像的那么严重。

该公司目前的想法是两个产品都会保存为独立的形式，但会进行集成。当然两个产品的外观会进行统一，在 System Architect 和 DOORS 和 Synergy 的集成会更紧密一些，提供更完整的应用产品生命周期的套件。

尽管 Telelogic 仍在否认，我猜想一旦集成完成，很自然地，TAU 和 System Architect 会融合在一起，尽管可能是通过模块的形式来融合，这样用户可以选择自己需要的功能。但如果两个产品同时靠近的话，我想没有人会关心，我也不认为是否处理得到会是多大的问题。

（自 it-analysis，袁峰 摘译，不得转载用于商业用途）



THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

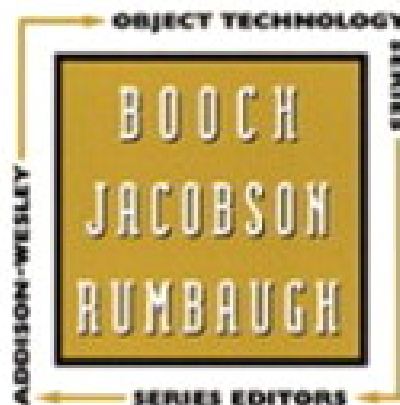


Covers UML 2.0

UMLChina 译
(王海鹏、李嘉兴)



CD-ROM
Included



《UML 参考手册》2.0 版中译本
即将由机械工业出版社华章分社出版

爱丽丝漫游用例奇境

——并非如你所想的超现实主义题材

Doug Rosenberg 著, hairui 译

讨论



[译者按: 本文作者用讽刺的手法批评了当前软件开发实践中存在的两个极端现象, 即分析瘫痪和忽略分析, 并提出折衷——经裁剪过的用例驱动软件开发方法——ICONIX 公司的 QuickStart RUP 插件。]

内容简介

本次演讲可能将会和你们之前听过的大多数演讲都不一样。在我开始之前, 我最好先让每个人都知道这次发言将包含一些讽刺。由于它基于一部伟大的讽刺著作——《爱丽丝漫游奇境》创作而成, 因此很自然地它也不得不这样。因此, 如果你很容易被讽刺性幽默激怒的话, 这次演讲也许不合适你。



我在演讲中使用讽刺手法, 是为了指出当前软件开发实践中存在的一些现象, 比如说, 一些人从一些好的点子开始着手, 甚至就一直就拥有一些很好的点子, 但事情的多数方面却朝着他们一些预期目标之外的方向发展。至少, 在我看来, 它们似乎是这样的。

实际上，虽然在过去的 30 年中更黄金的时期，我一直以这样或者那样的形式和软件开发打交道，并且用了这个时期最后 10 年中多数时间讲授《面向对象分析与设计》课程，但是这些日子以来我接触到的一些材料却令我感到完全陌生。因此，我把副标题定为“并非如你所想的超现实主义题材”。稍后，我们将要讨论“代码气味”以及“设计由代码产生”之类的话题，但我最好确认每个人都知道——我并未打算通篇都谈这些东西。不久之前，我在一个新闻组读到一篇文章，它上面说“少许预见性计划将导致大量的工作，因为预见使用假想的反馈方式来使自己保持正轨。”这是我平生第一次听人断言在软件开发过程中思考是有害的。

爱丽丝和我试图使用幽默来指出我们从这些实践中察觉到的风险，与此同时，我们也阐述一些很严肃的论点。

接下来……是三个部分的简要介绍

第一部分，爱丽丝受到用例驱动开发吸引，进入了用例奇境并遭遇了“分析瘫痪”的危险。

第二部分 在寻求避开“分析瘫痪”时，爱丽丝遇到了一些极度古怪的人物，并遭遇了忽略分析的危险。

最后，在第三部分，爱丽丝从梦中醒来，发现了一个短小却依然足够的开发方法，该方法在不忽略分析的前提下避免了“分析瘫痪”。

第一部分

作为软件工程的范例，用例驱动开发于 20 世纪 80 年代晚期由瑞典的爱立信公司首先倡导，并于 1991 年前后，由 Ivar Jacobson 在《面向对象软件工程》一书向世界各地介绍了它。从那时起，几乎每一个开发方法都开始声称支持“用例驱动”，毕竟在从易懂的用户需求中设计出软件方面，它的优势似乎非常明显。

随着“用例”这一专用术语的传播，各种不同意见出现了，多数争辩都围绕着如何最好地贴近用例驱动开发。有许多被称作用例驱动的方法，所完成的事情和 Jacobson 和他的团队之前的提议都相去不远，所不同的是他们只是不管做什么都将用例的招牌挂到的前面。

仍然有许多人只是将用例作为最终目的，而不是将用例当作完成从客户需求到驱动软件设计最终目标这一过程的手段。

如此一来，许多自称“用例驱动”的方法将一个个项目带入了“分析瘫痪”之中，以及比较常见的用例“颠簸”现象。

所以现在让我们随着爱丽丝一起进入用例奇境……

爱丽丝在读书时睡着了

坐在一张大椅子上看书时，爱丽丝慢慢地感觉睡意朦胧。她所看的是本很好的书，它上面提到的观点非常有趣……那上面说，可以通过细化使用情境描述的方法来驱动软件设计。

但是这本书页数超过了 500 页，而且没有任何图片，在上面……哦，她实在太困了。不一会儿，她脑袋一歪，睡着了……

用例驱动开发的诺言

入睡后，爱丽丝梦见了一个简单而别致的开发过程，首先是所有人尽全力让需求完整而易懂，然后构建一个设计方案来满足这些需求，最后系统建立起来了，并在非常一致的、成文的行为需求基础上进行测试，以此来保证客户能够拿到他们所要的东西。

“太好了！”爱丽丝感叹道，“这个简单、易懂且按部就班的开发过程简直就是非常有意义。真奇怪以前为什么没有人想到这个呢”

“它真是一个聪明的主意。”爱丽丝自言自语地说道。“这就好比是先写用户手册，进行稍许细化，然后设计符合用户需求的软件。”

一个分析模型将用例文本和对象连接起来

在用例驱动开发过程的正中心，是一种称为鲁棒性 分析的聪明的小技巧，看起来它将其他所有的东西连接在一起，并让整个用例驱动建模过程运作起来。但对于爱丽丝来说，虽然曾经花大量的时间阅读关于 UML 的书籍，但神奇的是她在此之前从来没有遇到过这种技术。

爱丽丝看到，软件结构和用户界面设计如何被连接到系统行为需求简单明了的表述之上。

“所有的一切看起来都那么直观和明显。”她说道。“但是我确信已经读过了《UML 用户指引》的 482 页的 32 个章节，但它却从来没有提到过这些，甚至在讲构造型的那一章都没有提到过。是什么原因让他们鬼迷心窍地将它排除在外呢？”她感到非常困惑。“没有了这种技术，它们还怎么能够被称为用例驱动呢？”

简单和明了

“让我们看看。”爱丽丝说道。“用例描述了系统行为的动态视图，而类和对象是关于静态结构的。”她说道。

“而这个健壮性图表则通过强迫分析师在用例文本用名称引用对象，从而将动态行为需求直接连接到软件结构。我猜这就是‘用例驱动’真正的含义了，因为软件结构可以直接从行为需求驱动。”聪明的小爱丽丝说道。

“将对象模型的静态和动态部分连接到一起，这看起来当然是非常有意义的事情，比将软件结构和行为描述孤立开来要有意义得多了。”她沉思道，“但是它看起来和代码的距离仍然不是非常接近。”她说，然后她继续沿着所走的小路向用例奇境深处走去。

<<includes>> 或 <<extends>>

爱丽丝沿着小路一直走着，突然她看到两个男人在用棍子敲打对方的脑袋，似乎在激烈地争论什么事情。爱丽丝停下来，看着那两个矮胖子。注意到爱丽丝之后他们停止了争吵。所有人安静了一阵子，然后爱丽丝问道：“你们在争吵什么呢？”但这两个半斤八两的家伙只是互相看了对方一眼然后咧嘴笑了起来。

然后，爱丽丝非常有礼貌地问道：“请问哪条是从用例到代码最好的路吗？现在天色已经很晚了，很快大家都要开始写代码了。”但是两个半斤八两的家伙突然又开始争吵起来了，很快他们都变得不知所措起来。

其中一个喊道：“扩展用例覆盖了被扩展用例的一部分，所以当被扩展的用例改变时，所有扩展用例不用修改就能继承这种改变。”另一个则叫道：“扩展告诉开发者用例之间的关系是一对一的，但包含告诉开发者关系是一对多的。”

他们就这样不停地跳着、喊着，敲打着对方，完全忽略了可怜的爱丽丝。过了一会儿，她决定放弃了，迷惑地摇了摇头，又开始继续沿着小路走了下去，再次开始寻觅从用例到代码之路。

我们已经迟了！我们要开始写代码！

爱丽丝走着走着，突然一只大白兔从身旁冲了过去，在她面前突然停了下来，转了三圈，嘴里喊道，“我们迟了！我们迟了！我们要马上开始写代码！公爵夫人要发怒了！”



“你到底在干什么呢？”爱丽丝问道。“迭代！”兔子说道，然后又开始转了圈。“停下来！”爱丽丝说道。“这样的迭代不会让你头晕吗？”她问道。“你能告诉我怎么样从用例到代码吗？”

“用例？”兔子喊道。“没有时间做用例了，太迟了，我们不得不开始写代码。”他说道，然后冲向了远方。“没有用例？”爱丽丝说道，“我真怀疑在没有任何系统行为需求的情况下，他们是否知道要写什么代码。”

“越来越古怪了。”爱丽丝说道，然后继续沿着小路走了下去。

爱丽丝非常想知道如何从用例前进到代码……

“唉，要能找道一条从用例到代码的快捷之路多好啊！”爱丽丝想，虽然她并不完全知道如何解释兔子的行为。“在分析模型中一切都非常有意义，但似乎将行为需求转化成软件设计之间还有一个窍门。”她想。

“标识一些对象和标识一些行为都没有问题，但如何找出哪些对象执行哪些行为呢？”她感到很疑惑。“也许答案就在路的尽头。”她充满希望地自言自语道。

抽象……本质

爱丽丝沿着小路又走了一阵子，很快发现在面前一棵树上坐着一只呲牙的猫。“对不起”爱丽丝向猫问道，“你能告诉如何从用例前进到代码吗？”猫咧嘴笑了笑，然后嘴里开始不停地重复几个词“本质、抽象、技术无关和独立实现”，但是突然，当它做这件事情的时候，它慢慢地变得越来越淡……

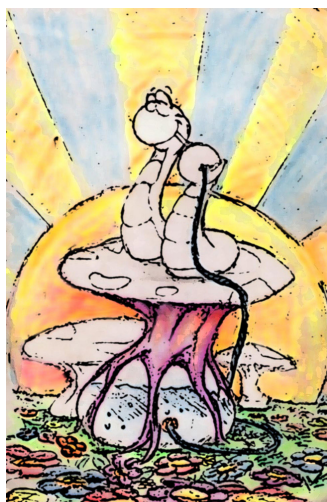
有点太抽象？

爱丽丝看着这只猫逐渐虚化直到只剩下了咧着嘴的笑容，但甚至当它消失之后，猫的嘴里都一直在重复着那几个词“基本、抽象、技术无关和独立实现”，

爱丽丝被猫消失吓呆了。“我看过不咧嘴笑的猫，但是却没有看过没有猫的咧嘴笑。”她轻轻地摇摇头说道。“但我看不出这对我从用例转到代码有什么帮助。这也太抽象了，似乎没有足够的细节来建立一个设计。”她说道，然后继续踏上了路途。

以目标为中心.....

爱丽丝很快来到一只庞大的毛虫面前，它坐在一个巨大的蘑菇上面，抽着水烟袋。“你好！”爱丽丝向毛虫问候道。“你看起来真的充满智慧，先生。你能够帮助我寻找从用例到代码之路吗？”



“你是谁？”毛虫用困乏的声音问道，“你为什么要编写代码？”“我的名字叫爱丽丝，多谢你的问题，先生。”爱丽丝说道。“我需要编写代码，是因为如果我不这样做的话，有人可能就要出面并取消我的项目，那样的话我该怎么办呢？你知道吗？”

“我不知道。”毛虫说道。“我们将基本用例定义为：对于某个特定的角色或者系统中角色的关系的一个简单、离散、完整、有意义并完整定义的任务，它在完成那个任务的过程中用户的意图和系统职能组成，使用抽象的、技术无关的、独立实现的应用域和扮演角色的外部用户语言术语来描述。”他深吸了一口水烟袋然后说道。

“在认识到这样一个以目标为中心的用例建模方法和在用例构建时向前抽象方面，我们并不孤单。”过了一会，他又补充到。

“噢，但这些 5 音节的词汇真的让我晕了。我想也许你能够说得更清楚一点”，爱丽丝说道。

毛虫只是坐在那里，抽着烟，过了一会，爱丽丝变得非常着急了。“难道用例不应该是易被理解的吗？”她毛虫问道。“难道说‘用户这样做而系统那样做’之类的话不比唠叨‘抽象、基本、目标中心’之类的话更好理解吗？”她问道。“所有这些专用术语让我自我感觉非常渺小，真的。”她不耐烦地说道。

“别上火。”毛虫说道，“你会习惯的，只是需要时间”。然后他把水烟袋放到嘴里又抽了起来。爱丽丝等了一会儿，感觉有点无所适从。一两分钟过后，毛虫从嘴边把水烟袋拿开，打了一两个呵欠，自己摇了摇头。然后它从蘑菇上爬了下来，爬入草丛中，回头说道：“如果你感觉渺小的话，尝尝这蘑菇。”

我们真的要为所有的用例指定这一切吗？

虽然对毛虫提出的品尝蘑菇建议是否真是个好主意有所怀疑，但当爱丽丝她考虑这个问题的时候，她真的感觉到了自己的渺小，她意识到原来从吃过早餐后自己就一直没有吃过东西，她真的非常饥饿了。“我只尝一小块。”她告诉自己，然后从上面剥下了一小块。“味道不错”，爱丽丝说道，“实际上它真的很好吃。”

吃了蘑菇之后，爱丽丝继续沿着小路前进，很快她看到一块路牌，上面写着“本路通向模板森林”。由于根本不知道该往哪里去，她决定继续前行，但很快她就被写满用例模板的大牌子包围了。

爱丽丝伸长脖子想看个究竟，突然她发现自己变得 10 英尺高！她开始阅读其中的一个模板。

用例名称、角色、优先级、状态、前提条件、事件流、基本路径、可选路径、事后条件……

“天哪！”爱丽丝叫了出来。“我们真的要为每个用例指明这所有的一切吗？”她不禁问道，并继续读了下去。

被包含用例、被扩展用例，“哦，不！”爱丽丝叹息道，“千万别有这些东西了！”她继续读了下去。

泛化用例、行为图、用户界面、数据库映射、情境、序列图……她继续读着。“我恐怕这比我叔叔的所得税表格还要糟糕。”她说道。“但我确信听他说过明年要用更短的表格——也许用例也有更加简短的方式。”

爱丽丝继续阅读巨大的用例模板……次要用例、参与类视图、其他工件（其他任何你想包括的东西，也许是一个分析模型、设计模型或者测试计划）。

“嗯，我不是很喜欢这些。”聪明的小爱丽丝说道。“将你想要包括的其他任何东西都添加到用例模板中看起来会让我们永远也没法开始写代码。”

“让我看看这一个是不是好一点。”爱丽丝说道，转身看着另一个模板，然后她开始阅读……特性信息、上下文目标、范围、层次、成功终止条件、失败终止条件、主角色触发、主成功情境、扩展、子变化、相关信息、优先级……

“噢，天哪。”爱丽丝说道，“到哪儿才是个尽头呢？”她继续读着……

“性能目标、频率、高级用例、低级用例、主角色泳道、次要角色、次要角色泳道。”

“次要角色泳道？”爱丽丝非常震惊地说道。“我甚至连它是什么都不知道，更不用说在我的用例中用到它。我想这就他们所说的‘分析瘫痪’。列出一堆的无用信息，永远也走不到编写代码那一步。我最好再看看。”她说道，随后继续沿着所走的小路走了下去

第二部分……

好了，现在我们讲到第一部分的末尾了，我希望我能够讲清楚了用例为什么逐渐被广受欢迎的，什么是用例驱动，以及造成今天行业领域内用例“颠簸”现象泛滥的在用例方面的一些传统智慧。

在下一个部分，爱丽丝遇到了一些非常古怪的人物，他们和她一样决定避免陷入“分析瘫痪”。但爱丽丝发现在这些人的方法和观点（我并不会通篇只谈这些东西），像口头文档、代码气味、“代码就是设计”等等好像有一些极端化。现在，让我们重新加入爱丽丝的行列，随着她深入越来越奇怪的用例奇境旅行。

爱丽丝口渴了

连续走了一段时间后，爱丽丝感觉非常口渴。又走了一会儿，她来到了一小块空旷地之前，她看见了一张桌子，上面放了一个玻璃瓶。

在玻璃瓶的瓶颈上，系着一张纸标签，上面用漂亮的大写字母写着“喝吧”，在后面还用小写字母写着，“保证避免‘分析瘫痪’”。

“‘喝吧’两个字真是太动听了。”她想，“而且我也很想避免‘分析瘫痪’——我的脖子被那些巨大的模板弄得现在还是僵的。”她说。但是聪明的小爱丽丝并不打算马上拿起奇怪的瓶子并急忙把它喝干。“我还是先看看吧。”她说，然后仔细查找是否标有“有毒”的字样。然而，这个瓶子并没有标着“有毒”的字样。爱丽丝决定冒险尝一尝。因为发现它非常好喝，很快她就把整瓶都喝干了。

爱丽丝感觉头晕……

爱丽丝变得非常眩晕，开始看到了许多一团团的颜色。“真是奇怪的感觉！”她说。“我怀疑喝掉它是否真的是个好主意。也许我应该坐一会儿。”坐下后，她听见有人在唱一首古怪的歌，但根本听不懂在唱些什么……

想象（向 John Lennon 致歉）

想象如果没有需求。你的尝试就会觉得容易。

只要几行代码，你就能到达天空

想象所有的人们，为今天而编写代码

想象没有时间进度，做起来也不会太难

没有该死的项目截止日期，也没有人监督你

想象所有的人们，手牵手一起编写代码

你也许会说我是个极端分子，但我并不是唯一的一个

我希望有一天你能加入我们，充满乐趣编写许许多多的代码

想象一下口头文档，我怀疑你是否能做到

不需要 UML 视图，只需要从一个人到另一个人的语言传递

想象只需要重构，如同在沙滩上玩耍

你也要说我是个极端分子，但我不是唯一的一个

我希望有一天你能加入我们，充满乐趣编写许许多多的代码

爱丽丝站了起来，精力充沛地摇了摇头。“我真想知道在我身上发生了什么事情。”她说。“我知道肯定要发生一些很有趣的事情了。每次我在这里吃了或者喝了些东西后都会这样。但是，当所有的东西都不是上下颠倒的时候，如果是在家里将会更加令人舒畅。也许我最好找到回家的路。”

[注意：你可以从 <http://www.SoftwareReality.com/lifecycle/xp/extremers.jsp> 找到更多的《极端者之歌》]

结对编程意味着从不记下需求

爱丽丝仍有一点头晕，踉跄地沿着小路走到一块开阔地前，她看见一群编程者，成双成对地一起编写代码，嘴里轻声唱着歌曲。空气中洋溢着平和、宁静的气氛。

爱丽丝在一块牌子前停了下来，上面写着“时间进度表的概念取决于进度，由于软件永远无法完成，它只是以匀速开发。”附近的另一块牌子上写着，“书面需求是懦夫的行为——不要惧怕口头文档。”

“天哪。”爱丽丝说道。“我在写需求的时候从来没有感到过害怕……我只是想确信我理解了客户的需要而已。所有的这些人看起来那么快乐，而且确信他们所做的是对的，所有人成双成对地在一起编程。但我怀疑他们是否对自己的失败是否真的视而不见。没有了需求，他们如何确认自己在建造正确的系统呢？为什么会说‘我写我想要的’和‘他们想要我写的’是一回事呢？”

爱丽丝现在感觉好多了，她继续走了下去。

没有时间记下需求

突然，爱丽丝曾经遇到过的那只大白兔冲了出来，对着爱丽丝发了疯似地反复喊叫着“没有时间记下需求！”“而且用户们永远都不知道他们想要什么！”它说道，继续在原地绕着圈子。

“他们只是告诉你一个故事，然后开始对它进行编码”它说。“每个早上他们都要数次改变自己的想法——总之，唯一跟上他们的办法就是在他们的主意改变之前重新构造代码。为什么，我们能够在典型的用户改变主意的时候经历 5 次迭代，你想想看如果我们做不到怎么办”他继续说道。

“重构？”爱丽丝问道。“我想我在什么地方听过。”“是最近新出现的事物。”兔子说道。“每个人都在使用它，设计完蛋了，你知道。有足够的重构的话，你根本不需要设计。设计视图就是从代码出来的。你可以问戴帽子的人。”他说道，随后又急匆匆地离开了。

你最好说“代码就是设计……”

不知道还有什么可以做，由于听了“设计从代码出来的话”爱丽丝又开始感到有些摇晃了，她决定朝着兔子一起走。在他们前进的时候，兔子每过几分钟都要走几步就停下了在原地打几个转。最后，兔子突然以最快的速度向前冲刺起来，把爱丽丝落下太多了，以致爱丽丝很快就看不到他的身影了。

爱丽丝继续往前走，过了一段时间她看到兔子在一块空地上，和他在一起的还有一只巨大的老鼠和一个戴着大帽子的小矮人，所有人都围着一张桌子和一些椅子在疯狂的工作。当爱丽丝走近后，发现所有桌子、椅子的腿都在地上堆成了一堆。爱丽丝看到兔子、戴帽子的矮人和冬眠鼠各自从那堆腿中抓了四条然后将它们拧到椅子上去。但问题是，那些腿的长度并不一样，爱丽丝非常惊讶地看到他们每个人都完成了椅子的拼装工作，然后把椅子翻了过来并坐到了上面。椅子经常会翻倒，因为它们的所有的腿长度都不一致。当这种事情发生的时候，所有人都异口同声地喊叫道“单元测试失败！”然后他们将椅子反过来，将腿拧下来并扔回一堆。

“你们在玩什么游戏呢？”爱丽丝问。“这不是游戏，我们在重构家具。”戴帽子的人回答说。“你们为什么不看看拼装家具的文档呢？”爱丽丝问道。“那上面肯定指明了哪条腿是哪张椅子的相关说明。”“它的文档是口头的。”戴帽子的人说。“书面文档是为懦夫准备的。我们可以非常迅速地重构，因此我们很快就可以勇敢地让设计自己完成自己。”他补充道。

“喔，是的。兔子说我可以问你关于重构的事情。”爱丽丝说道。“设计如何从代码自动完成呢？”她问道。“你这个可怜的无知小孩。”戴帽子的人充满优越感地说道，“代码就是设计，你难道不知道吗？也许你最好让公爵夫人向你解释一下。”他说道，然后继续重构那些椅子。

谁关心用例？

爱丽丝记起了一路上看到的一些东西。“我记得在网上我读到过关于重构、结对编程和口头文档的内容。”她一边走一边自言自语地说。“是哪个网站呢？——是的，好像叫做 Wiki Web”她终于回忆起来了。“我记得我读过关于工资系统项目的内容。它好像叫做 C3 之类的名字，让我想想，那个项目不是被取消了吗？”她说，“看起来他们对它言过其实了，它似乎并不是那么成功。”她沉思道。

[注意：请不要以爱丽丝的话来判断它，请自行阅读 <http://c2.com/cgi/wiki?CthreeProjectTerminated> 的内容]

这一次，爱丽丝很快来到了一个中间有栋小房子的地方。爱丽丝怯生生地走上前敲了敲门，但没有人响应。爱丽丝能够听到里面传出非常奇特的声音——持续的嚎叫声和喷嚏声，中间不时地夹杂着巨大的爆裂声，听起来就像是碟子或罐子列成碎片的声音。“嗯，看起来在外面等是没有用的。”爱丽丝说道，打开门走了进去。

正对着门一个巨大的厨房，里面浓烟滚滚：公爵夫人坐在当中的一把三条腿的凳子上，在照看一个婴儿，厨子靠在炉火旁，烧着一大锅看起来像是汤的东西。

“汤里面一定放了太多的胡椒粉！”爱丽丝对自己说，她感觉有点想打喷嚏。”公爵夫人看着爱丽丝说：“你能闻到代码的气味吗？”“代码？”爱丽丝说道。“我还以为那是汤。”公爵夫人说道：“那，是代码的气味，我们目标提供者正在重构代码并让它闻起来更香。”

“你的代码要实现什么呢？”爱丽丝问道。“你拥有需求或者用例吗？”她问道。“谁关心用例？我们不喜欢书面的需求。”公爵夫人说。我们让客户和我们同处一室，并让他进行验收测试。我们叫他目标提供者。用这种方式，我们可以自由地添加任何我们想要的功能，不需要那些市场里的蠢人们的干涉。他们什么都不懂。”她嘲笑道。

爱丽丝记得曾经在 Wiki 的网站上读到过“目标提供者”，在他们和管理层（被称作黄金拥有者）之间存在一些类型的争论。“但如果目标提供者和黄金拥有者之间有分歧怎么办？”爱丽丝向公爵夫人问道。“黄金拥有者也许会马上取消项目。此外，如果客户们可以编写验收测试，他们还需要程序员干什么？”她非常困惑地问道。

C3 项目终止了

公爵夫人对爱丽丝感到非常愤怒。“你怎么敢这样胡说八道！”她怒吼道。但爱丽丝并不退缩。她记起自己带着能够无线上网的新掌上电脑，而且她已经把 Wiki 网站加到了收藏夹中。“这不是胡说八道！”爱丽丝抗议道。“看看它是怎么说的，就在……在这个题为《C3 项目被终止》的页面上。据说目标提供者 和黄金拥有者 的目标不一致。程序员们不断地向代码中添加新的功能，而管理层只想关闭大型机因为 C3 是一个千年替代项目。为什么？因为那个项目中除了完美的、过时的功能没有其他任何东西，另外它有太多的糟糕的案例。”她说道。“你想过吗？如果工资程序在 2000 年 1 月崩溃会发生什么样的事情？”爱丽丝问道。



公爵夫人脸色变得青紫。“你这个白痴。”她开始反击了，“你根本不懂这些，对吗？你最近写了多少代码？”她很生气地问道。但勇敢的小爱丽丝仍然拒绝退缩，她平静地回答道：“无用功能的存在，就是为书面形式记录需求的重要性所在。只有这样，程序员们才不会写上一堆看上去很酷但客户并不为之付款的东西。”“滚出去！”公爵夫人尖声叫道。“你将必须接受王后的质询！”

[注意：请不要以爱丽丝的话来对它做出判断！请自己阅读 <http://c2.com/cgi/wiki?CthreeProjectTerminated> 的内容]

一次且只有一次？

爱丽丝并不非常想离开，相反到了这个时候，她也对公爵夫人的态度感到非常烦恼。她继续在掌上电脑中阅读那个网页。

“你真值得同情。”爱丽丝说道。“你将自己高贵的主张建立在一个没有成功的项目之上。”她说道。“这里说，C3 项目是一个始于 1996 年的千年替代项目。当它于 2000 年 2 月终止时，最初有意为它付款的人中只有 1/3 进行了支付。看看这说了什么……不仅客户永远不会再尝试这个方法，它甚至导致任何想让管理层都慎重对待面向对象这个术语……”“记住！”爱丽丝说道。“这个 Wiki 主页不是我的网站，它是你们的！这让我想起了皇帝的新装的故事。你还有什么大型项目的成功案例要吹嘘的吗？”爱丽丝向公爵夫人问道。

公爵夫人无言以对。“滚出去！”她气急败坏地说。“王后会砍了你的脑袋。”这种无理的行为终于让爱丽丝无法忍受了，她非常厌恶转身离开。

[注意：请不要以爱丽丝的话来对它做出判断！请自己阅读 <http://c2.com/cgi/wiki?CthreeProjectTerminated> 的内容]

爱丽丝拒绝不写需求就开始编码

爱丽丝继续走了一会儿，当她正在考虑是否可能使用掌上电脑找到一份地图可以让她离开这个令她迷失的古怪地方时，两个看起来像索引卡的士兵向她走了过来。“对不起，女士。”第一个士兵说道，“心之王后要看看你写了多少代码。”

由于依然在为和公爵夫人之间不快的经历而生气，爱丽丝回答道：“请告诉陛下还没有任何人给我任何需求，所以我没有任何代码。”两个士兵互相看了一眼，“没有代码？”第一个士兵说道，摇了摇头。“王后不喜欢这样。”“不！”第二个士兵低身向第一士兵说道，“这是肯定要砍头的”。而爱丽丝并没有听到这句。“你最好和我们走”士兵大声向爱丽丝说道。

由于根本没有地方要去，爱丽丝跟着士兵们来到了一块草坪前，有一群人正在玩着门球。爱丽丝看到远处有一座宫殿。一阵喇叭声之后，两位士兵大声宣布“王后驾到！”并迅速低下了头。

爱丽丝抬头看见面前站着一位王后，双手抱在胸前，面上乌云密布。“好吧”王后说道。“在哪儿？代码在哪儿？”“请原谅，陛下”爱丽丝一边单膝跪下，一边非常谦逊地说道。“我还没有写任何代码，因为没有任何人告诉过需求是什么。”

王后的双脸因愤怒而涨得通红，像野兽一般盯着爱丽丝看了一会儿，然后尖声叫道“砍了她的头！砍了她的……”

“可笑”爱丽丝非常大声且坚定地说道，王后倒沉默起来。

国王将手拉住王后的胳膊，小心地说道“亲爱的，再考虑一下，她还只是个孩子！”

王后非常生气地转过来，向爱丽丝说：“我命令你马上开始写代码。”“我决不会这样做！”爱丽丝说道，那份勇气连她自己都感到奇怪。“没有书面需求，就没有计划性。我的项目就会变得毫无用处。没有需求，我不打算开始编码。”

王后再次开始吼叫起来，但这时国王开口了，“在我们砍掉这个小姑娘的头之前。”他向王后说道，“我们必须进行一次审判。”随之他高声宣布，“对爱丽丝的审判马上进行！”，然后，士兵们站起来，抓住爱丽丝的胳膊，整队人马走向审判室。

你犯有 BDUF 罪……

国王和王后坐在他们的宝座之上，周围簇拥着一大群人。在国王身边是那只白兔，一手拿着一个喇叭，另一手拿着一卷羊皮纸。在法庭的正中间摆着一张桌子，上面放着一大碟的小甜饼：它们看上去非常可口，以至于让爱丽丝连看它们一眼都会感到非常饿——“我真希望他们在审判结束后分发这些点心。”她心里想。但似乎没有这样的机会；所以她开始观察身边的每件事情来打发时间。

爱丽丝之前从来没有去过法庭，但她在书中读到过，她非常高兴地发现自己能叫出那里几乎所有东西的名称。“那是法官。”她对自己说，“因为他戴着巨大的假发。”顺便提一句，法官也就是那个国王，在假发上面戴着皇冠，看起来一点也不舒服，因为那确实不会让人舒服。

“传令官，宣读指控！”国王说道。

于是，白兔吹了三声喇叭，然后展开羊皮纸卷轴，宣读了以下内容：

“与我们对质的孩子名叫爱丽丝

在今天的这些诉讼程序中

因为坚决要求预先进行大量设计

并拒绝立刻进行编码！”

“请做出你们的裁决。”国王对陪审团说道。

“还没有！还没有到时候！”兔子赶忙阻止。“在那之前还要有大量程序。”

“传第一证人。”国王说道；然后白兔吹了三声喇叭，喊道：“传出了第一证人。！”

爱丽丝非常紧张地就位。“出示你的证据。”国王说道。“请您原谅，陛下，我只是想在我开始编写代码之前想写下需求。”爱丽丝说。“因为我想如果我记下了它们，我就能够更好地理解它们，而如果它们只是口头上的，我就无法一直遵循它们。”她说道。

“需求”，国王说道，“并非软件的四大要素之一。软件的四大要素是代码、测试、聆听和设计。对软件来说有这些就够了。凡是告诉你其他不同事情的人都只是在卖弄。设计只需要记录在索引卡上，早晨结构变化时它们将变得陈旧而应该马上丢弃。”

“让陪审团做出裁决。”国王说，这大概也是一天内他第二十次说这句话了。“不，不！”王后说道。“先宣判，然后才裁决。”

“鬼话连篇，胡说八道！”爱丽丝高声地说道。“哪有先宣判的道理！”

“住口！”王后说道，脸变成青紫色。“我不！”爱丽丝说。“砍掉她的头！”王后用最大声音喊道。

CMM 该死！砍掉她的头！

人群立刻嚷嚷起来，“太多 BDUF 了……CMM 该死……砍掉她的头……砍掉她的头！”爱丽丝立刻被几十个敲着鼓的索引卡士兵包围了。

虽然曾经听说过生产能力成熟度模型，但她个人从未涉入过追求 CMM 第五级别的工作，她需要的只是一个简便易行方法，这个方法能够包括合理数量的预先计划并且提供合理数量的文档，她并没有完全理解对 CMM 的宣传。“为什么试图寻找可重复的开发过程会导致这么严重的后果呢？”她对此非常困惑。

鼓声和叫喊声把爱丽丝吓坏了。她确信自己将很快被行刑，尽管还不明白为什么。她捂住自己的耳朵，并因恐惧而抖缩着。

若干设计的彻底重构……

当那组士兵试图袭击爱丽丝时，她尖叫起来，一半是因为恐惧，另一半则是因为愤怒，她试着反击他们。“谁怕你们？”爱丽丝喊道。“你们只不过是一堆卡片！”

突然一阵狂风吹来，吹散了那堆涂写了设计的卡片。爱丽丝醒来，并且发觉自己还坐在起居室的那张大椅子上面。窗户开着，强风不断吹了进来。

“哦，我做这么奇怪的一个梦！”她说道。“我无法断定哪种情况更加糟糕：陷于‘分析瘫痪’当中，还是在理解需求之前直接跳到编码部分。我多希望有一种介乎两者之间的简单方法啊。”她说道。

第三部分

因为第一本自称用例驱动对象建模的书已经让爱丽丝确信用例是个好办法，她又拿起了第二本，这本要比第一本薄得多。

[注意：参阅 <http://www.iconixsw.com/>]

“薄得多的书。”她说。“比厚厚的第一本的要吸引力得多。老天爷也许知道我并不想再睡着做一个那样的梦。谁知道下次我会梦到什么，说不定是运行时编译结构框架或者其他什么的。”她说道。

爱丽丝醒来……

“嗯。”爱丽丝说道。“这本书的封底上说上书中有一些帮助人们避开通用建模陷阱的‘分析瘫痪’的警告，关于需求及如何以可跟踪的方式进行设计的广泛讨论。

“同时，它并没有试图讲解《UML 用户指南》中所有的东西，而只是将重点放在那些视图的一个子集之上。”她说道。“看啊，他们只使用关键子集中的 4 种不同视图。”她说。“我猜这是他们在《UML 用户指南》中提到的‘你能用 20% 的 UML 完成 80% 的建模。’”爱丽丝说道。“只有这本书告诉你那 20% 包括哪些。”

[注意：请参阅 <http://www.iconixsw.com/>]

缩短“做什么”和“怎么做”之间的间隙

“啊！”爱丽丝说道。“又说到那些失踪的连接视图了。现在我知道它是多么重要了。从‘做什么’到‘怎么做’似乎是软件开发过程中最难跨越的一步。”

“毛虫和咧嘴笑的猫从未能够完成对系统应该做什么的讨论”，她说道。“而兔子、戴帽子的人和那个可怕的公爵夫人和王后除了代码之外什么事情也不考虑。”

“他们所有的人都惧怕书面需求，这真的很滑稽，不是吗？恐怕他们只是不想有任何人告诉他们任何事情，这样他们就可以为所欲为地构造自己想要的了。”聪明的小爱丽丝说道。

将静态和动态模型链接到一起……

“这真的很有趣。”爱丽丝说。“在初步设计期间有那么多事情要做。在鲁棒性设计阶段用例模型和对象模型都得到了更新。”她说道。“在新对象被发现的同时，用例文本的模糊性减小而严谨性增强。

“模型的静态和动态部分被链接到一起，而且被迫彼此保持一致性，行为描述得到了双倍检查以保证其正确性、完整性和构造可行性。”

“真是项不可思议的有用技术！”爱丽丝大喊道。“它一定能够设法从需求层用例描述转到设计层的顺序图。”她补充道。

顺序图中发生的行为配置……

“这看起来像个大结局。”爱丽丝说道。“这是一个简单明了的过程，建立顺序图的头三个步骤都可以用脚本自动完成，第四步只是简单地将重点放在行为配置的决定上。为什么这还会让面向对象设计变得那么神秘呢？”她说。

“这样的话肯定不会发生‘分析瘫痪’。”爱丽丝评论道。“所有事物都从用例开始；与客户间的进行需求评审，然后在初步设计期间提炼出行为描述，随后是客户也可以参加的初步设计评审。详细设计出来后，进行严格的设计审查，但客户不再参加，而仅仅是一种开发团队的内部检查。然后由负责质量保证的家伙们测试检查开发者们对用例文本的构造情况。

“因此问题简化成了：‘这是我们认为你想要的，如果我们错了请纠正。’然后是纠正之后，‘好了，我们按照你们所建议的作了修改，并进行了一些细化，如果我们没有正确理解请告诉我们’；随后是，‘现在，我们要继续编写你们所告诉我们的东西了，你可以检验我们是否按要求构造了我们已经达成了共识的东西。’同时，对于那些计划要实现的用例，可以选择进行任意规模的迭代。”她说。“这不仅能够为小项目服务也能为大项目所用，也能为迭代中的大小功能模块所用。需求不会变得混乱，因为所有事情都被记录在案，设计可以直接回到行为需求，需求也可以直接用来制定验收试验计划。”

“发现一种简捷而易行的方法真让人感到了解脱。”爱丽丝说道。“既不略过分析而又避免分析瘫痪毕竟是可能的。有些时候，你可以在梦中学到很多。”她细细回味道。

“现在如果有人为 RUP 构造一个插件就能让我的整个计划遵循这个过程了……”爱丽丝沉思着。

[注意：参阅 <http://www.iconixsw.com/quickstart.html>]

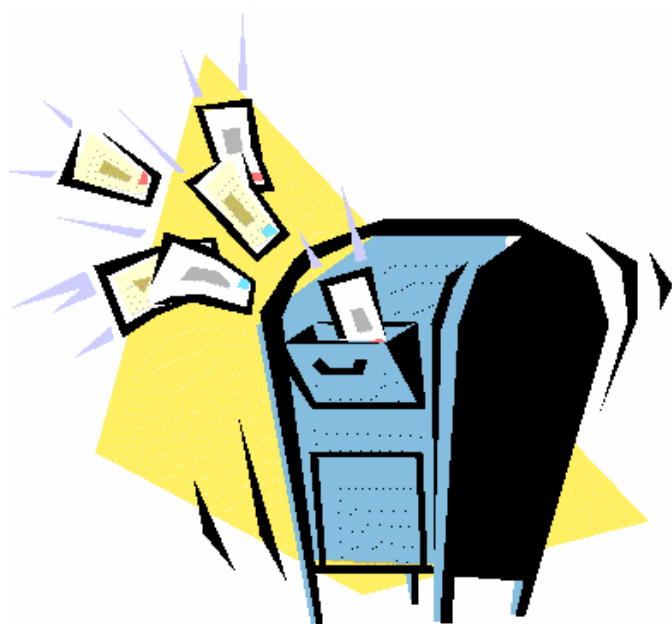
行为准则是……

“我非常同意你的观点。”公爵夫人说。“行为准则是——你看起来是什么你就是什么，或者你可以更简单地说，‘决不要假设你自己你和别人不一样，你是什么样或你也许是什么样和你已经是什么样似乎没有不同，这一点你和别人看起来没有什么不同。’”

“我想我也许应该更好地理解你说的话。”爱丽丝很有礼貌的说。“如果我曾把它记下来的话。但在你说话的时候我没有能够跟上你”

“如果我做了选择的话，我说什么都没有关系了。”公爵夫人高兴地回答道。

“但愿你不会说比那更长的句子来给自己添麻烦。”爱丽丝说道。



征稿

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



Domain-Driven

DESIGN

Tackling Complexity in the Heart of Software



众多问题根源在于
领域模型没有捕获到
业务的深层内涵

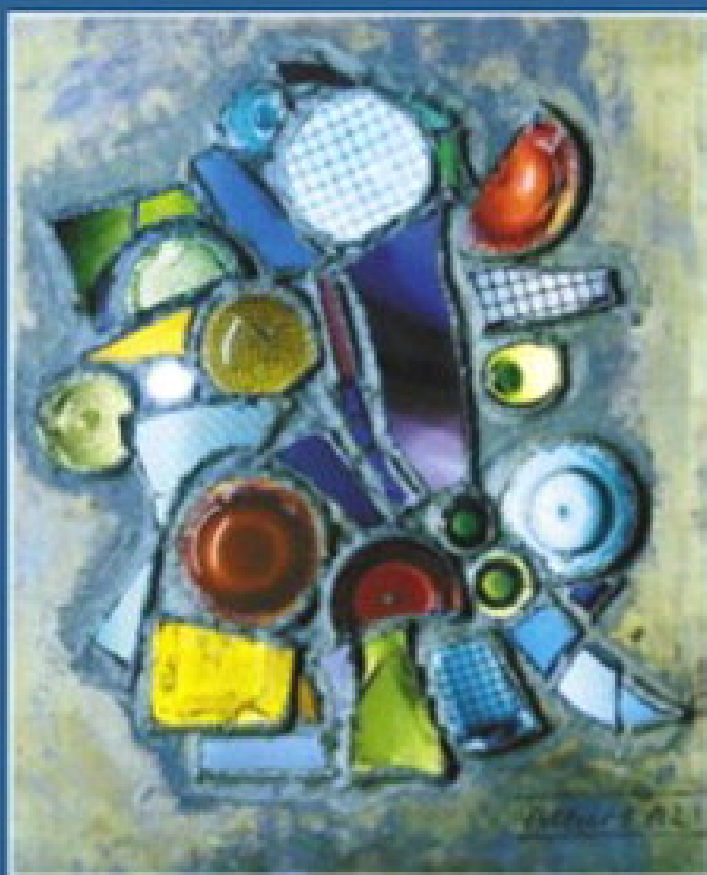
Eric Evans
Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

Object Design

Roles, Responsibilities, and Collaborations

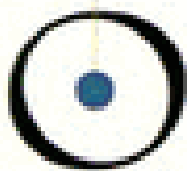


一谈起“面向对象”就是“设计模式”？——对象的回归！

Rebecca Wirfs-Brock and Alan McKean
Forewords by Ivar Jacobson and John Vlissides

《对象设计》中译本即将由人民邮电出版社出版

UMLChina 辅助教材



OBJECT-ORIENTED

SOFTWARE CONSTRUCTION

SECOND EDITION



The Most Comprehensive, Definitive O-O Reference Ever Published

An O-O *Tour de Force* by a Pioneer in the Field

CD-ROM Includes Complete Hypertext Version of Book and Object-Oriented Development Environment



BERTRAND MEYER

《面向对象软件构造》中译本，UMLChina 辅助教材

即将由 机械工业出版社 出版

控制软件设计的复杂性

Karl J. Lieberherr 著, 李静 译

讨论 

摘要

我们的研究聚焦于辨识出软件开发过程中采用的技术,应用这些技术开发的软件能够经受住代码重构的考验。在这方面,“Demeter法则”(LoD)作出了贡献。但是它也导致了其他的问题。当前的技术发展主要集中在“面向方面的软件开发”(AOSD),伴随着技术的发展,重新审视“Demeter法则”(LoD)的基本目标,使它适应技术发展的新观点是很有益的。因此我们引入了“关于关注点的Demeter法则”(LoDC),并且讨论了这些方法在横向上的重要交叉点。我们探究“Demeter法则”(LoD和LoDC)的分支以便更好地使用改良的软件开发方法实现基于关注点的分离。“自动机理论”的新应用使语言机制得以实现,我们的研究得到了这些语言机制的支持。本论文的修订版和相关幻灯片可以从Demeter的网站获得: http://www.ccs.neu.edu/research/demeter_in_the_directory/papers/icse-04-keynote/



1 引言

我们的研究聚焦于辨识出软件开发过程中采用的技术，应用这些技术开发的软件能够经受住代码重构的考验。在这方面，“Demeter 法则”（LoD）作出了贡献。“LoD”，通俗的表述就是只与你的朋友们通信，在严格面向对象的开发环境下，遵循“LoD”引起了一些问题，正是这些问题促使我们在“面向方面的软件开发”（AOSD）这个领域进行研究。

在“LoD”含义的表述中，对于“friends”和“talk”有多种不同的解释。“朋友”可以是对象，类，组件。“谈话”可以理解为“访问”或“调用”。最常使用的通式中，我们有一组模块（方法，对象），每一个模块只与和最紧密相关的模块（朋友）通信。因为一个模块如果不涉及太多其他的模块，而仅仅和恰当的模块通信，恰当的模块是完成目标必须涉及的模块，那么模块会变得更易于理解。“LoD”有助于控制软件开发者的信息超载。通过最小限度地利用接口，“LoD”超越了封装。“LoD”的一些工具明确了不良耦合的含义，使得不良耦合可以检查，“LoD”是低耦合原则的应用。

我们的研究焦点限定在包含类和方法的面向对象系统中，在这样的系统环境下来解释“LoD”，这也是首次从面向对象的角度对“LoD”进行解释。这么做不仅仅是因为面向对象系统的普及性，还因为它本身固有的属性，面向对象的开发把每一个实体作为一个模块，整个系统由这些相互关联的模块构成。然后，对这些关联建模，形成一个有趣的图表结构，在我们的讨论中会用到这个图表结构。“LoD”最早表述为：所有的方法仅向“首选的供应者对象”传递消息。一个方法的“首选的供应者对象”是它的朋友，包括直接的“子对象”，以参量的形式传入到当前对象方法中的对象，也包括当前对象和在这个方法中直接定义的对象。进而，我们归纳“首选的供应者对象”的概念如下：直接的子对象（包含存储角色的对象也包括承担计算角色的对象），这个概念基于 Walter H. Ruesch 的建议。“计算角色对象”是这样一种对象，它由传递给当前对象的信息返回。

2 关于关注点的 Demeter 法则

在适应性程序设计时代的早期，我们就开始使用了一个较强的形式，这里我们称这个形式为“关于关注点的法则”（LoDC）。在“LoDC”中，我们进一步的限制“朋友”：朋友是那些引起一组关注点的朋友。因此“LoDC”的要义是：只和那些引起一组关注点的朋友通信或者只和那些有共同关注点的朋友通信。软件开发中有很多关注点，通信应该严格限制给那些引起关注点的“首选的供应者对象”。Colyer 等人也持有与此类似的观点。

我们不能消除与那些没有引起最直接关注点的朋友通信，但是我们想要使用一个独立的模块来影响这种通信。这样做的优点是使那些与最直接关注点相关的代码保持独立性，而且使代码不会因为与外部关注点通信而变得混乱。这样做使代码更易重载也更有灵活性。

以一个可以实现一组关注点的数据库应用软件为例，现在我们想要记录在这个软件中的一些方法调用。换句话说，我们想要实现在“面向方面的软件开发”(AOSD)中经常使用的“日志”。这个例子没有很好的反映出“AOSD”的功能，但是它足以说明我们所寻找的那种内在关系。“日志关注点”与“软件关注点”是不同的；实际上，“日志”是一个插入的关注点，通过“日志”来监视这个数据库软件的运行。由“日志”把数据库中的信息传递给一个“日志文件”。

我们需要从外部增强这个数据库软件，而不是在这个软件中增加日志通信的代码，增加代码的方式违反了“LoDC”。我们需要找到一个便利的方式来详细说明程序执行的位置，在那些位置，程序与“日志文件对象”发生附加的通信。“日志”的例子是一个由关注点横切数据库程序的例子。

依照“LoDC”，遵循创建抽象类的指导原则，我们创建一个有趣的抽象类：这个抽象类不是通过复制或修改一些程序来形成的。这里我们不想复制上面的日志。有趣的是这个抽象类横切了一些方法和类。

“LoDC”意味着“LoD”。“LoDC”使我们认真思考那些值得单独考虑的“关注点”。“关注点”意味着设计师和程序员关注的任何问题，例如，“系统同步”，“用例实施”，“异常处理”。“LoDC”提倡分层的软件开发，每一层执行一个或多个“关注点”。每一层有一组“扩展点”，下一组“关注点”可能与这些“扩展点”相关。使用“坐标系统”在当前层的外部定义“扩展点”，这里的“坐标系统”存在于当前关注点中。“关注点”的固有属性要求它们以复杂的和系统的方式相互影响，分层软件开发机制使得“关注点”可以在充分指定的和公开获得的点上互相影响。在层的外部定义的这些点包括了那些由新的关注点解说和合并而形成的非扩散性的“系统扩展点”。另外，当“关注点”是横切的，它们的执行将会是分散的，这导致了这些关注点与其它关注点的定义发生冲突。“LoDC”对于我们如何构建软件有比较敏感的影响。如果需要从一组“外部扩展点”中增加一个新的关注点，我们应表达清楚它在扩展点中的坐标。

我们考虑一个程序，这个程序执行一组关注点“Cs”。现在我们新增加一个关注点“Cnew”。假设“Cs”，“Cnew”和这个程序完成。我们希望从一个独立的抽象类中得到从“Cs”到“Cnew”的一组调用。调用之后代码开始执行，直到返回到“Cs”中最初发生调用的部分时，我们把这段代码作为对“Cs”代码的单独的增加。“Cnew”通过一个增添到程序中的“外部抽象类”实现，这个类引起了一组关注点。“只与你直接的朋友们通信”由“Cs”中的代码组织。

在很多情形下，我们可能能够区分基于不同关注点的对象。如果是那样的话，“LoDC”不允许调用那些不是朋友的对象的方法，也不允许调用那些没有引起直接关注点的对象的方法。在某些情形下，“Cnew”适合一个对象，但是在更多情形下，它横切很多对象。解决“横切关注点”的非扩散性扩展问题正是“AOSD”的主题。有关“AOP”的历史，参见[40]。“Composition filters”是早期的技术，提出了应用“composable filters”的层来扩展一个类，使其具有非扩散性。Ivar Jacobson 也提出了面向方面的想法，但是他没有用可以表达的符号描绘出一组扩展点。

2.1. 扩展语言

分解上述“日志”的例子，我们可以得到什么样的启示呢？我们可以随意插入或取出日志。但是如果我们不得不一一列举出调用发生的位置，我们的收获是很有限的。以 Filman 等人的术语来说，我们忘却了量化。可能记录下调用的坐标比记录调用本身更繁杂。然而，实际中，我们常常发现调用坐标往往跟随一个易于表达的模式。例如，我们可能想要以 `C.withdraw(..)` (`..`代表参数)记录所有的方法调用。在这个例子中，对于日志关注点进行分解有助理解。我们保证记录所有的方法调用，但是手工的方法可能会产生遗漏。AspectJ 是这样解决这个问题的：

```
aspect Logging {
    pointcut find_Cwithdraw():
        call(* C.withdraw(..));
    after find_Cwithdraw() {
        System.out.println(thisJoinPoint);}}}
```

这段程序假设我们不会突然调用一个与 `withdrawal` 无关的 `withdraw()` 方法。

在语义上描述坐标需要采用一些技术。当你同意和你的朋友在火车站见面后，你可以会说：让我们在那个标有箭头记号的地方见面吧。当你的朋友问详细地点的时候，你可能无法马上回答，因为你并不确定具体的地点在哪里。当你告诉你的朋友在瑞士航空办事处向左转的时候，你可能会感到比较麻烦，因为瑞士航空办事处根本不存在了。有时见面的地点已经搬走了，而我们还在沿用那些我们常用的地名。

再者，手工增加日志会增加原始模块中信息的数量。新增的信息与模块的操作信息混杂会增加模块的代码库，因而会降低模块的功能能力。对于其他关注点迭代手工增加更会加重这个问题。

然而，我们需要谨记当我们修改程序或者以 `C.withdrawal(..)` 的形式增加新的方法调用的时候，这些调用会自动被添加到日志中。我们称这种情况为“改写困境”。在 3.1 部分我们提出了这种困境的解决办法。

我们应该使用何种扩展语言呢？通常，我们需要的语言本身要具有内部结构还要能够反映出大量的坐标（扩展点）对于我们所采用的每一个扩展点，我们需要描述它的扩展部分。（参考[42]有关程序语言更好的参考机制）。从一个极端看，我们使用一个目的性很强的方法，在这个方法中使用的语言专门针对关注点。例如，如果是“同步关注点”，则我们使用的语言专门用于同步。如果“协作关注点”，我们使用的一种专门的语言来遍历程序，来表达协作过程中发生的遍历。Duzan 等人讨论了其他有关关注点专用语言的例子。专门针对关注点的方法有很多的优点，也由此产生了很多小的，协作性的语言。它们通常有相似的实现机制。因此，使用一种带有通用目的的方法成效更为显著，AspectJ [23] and HyperJ 是最突出的两个。“谓词调度语言”也是最接近通用目的的语言。“环境类”也是通用目的方面（general purpose aspects）的早期形式。

在本文的第三部分我们将考察“LoD”和“LoDC”的共性。第四部分我们探讨复杂的需求接口，解释由于遵循“LoD”产生的更宽泛的接口。第五部分我们讨论“LoD”在域的应用而非在方法上的应用。第六部分我们讨论遵循“LoD”使用的工具。第七部分略谈整合的方面和组件。第八部分谈挑战和开放性的问题。第九部分作论文总结。

3 遵循 LoD 和 LoDC

“LoD”是在一组关注点内组织软件的规则。“LoDC”提倡将新的关注点同原有的分离，如果这个新的关注点可以单独考虑的话。

“LoD”引起了一个困境（LoD 困境）：如果我们遵循它，我们不得不写很多包含重复信息的小方法，如果我们不遵循它，我们的代码很难维护。借助“遍历战略注释器”（例如，DJ）或者“代码生成器”（例如，DAJ[31, 10], DemeterJ[35]），“遍历战略”提供了摆脱这种困境的方法。

3.1. 改写困境

我们继续沿用“日志”的例子。详述如下：我们认为 Cwithdraw = “find all calls of the form C.withdraw(..)”是可以改写的，因为它可以被很多数据库程序使用。但是改写是要付出代价的：当数据库程序改变的时候，我们需要重新测试以证明正确的元素仍然发挥作用。我们这样定义“改写困境”：

将一个带有参数的程序抽象为 $P(Q)$ ，为 $P(Q)$ 提供一个很宽的定义域，向 $P(Q)$ 传递实参。当我们修改实参的时候，我们需要重新测试 $P(Q)$ ，如果需要的话，还要对 P 进行修改。例如将 $P(Q_1)$ 改为 $P(Q_2)$ 。

在上面的例子中，P 是对 `find-Cwithdraw` 的说明，Q 是数据库程序。在存在“改写困境”的环境中，P 要根据实际的参数改变它的含义。它要适应参数，但是在 P 的定义中不用涉及参数的细节。在下面的讨论中我们会看到，“改写困境”也会出现在一个“适应性程序设计”的过程中，“改写困境”这个名字就是源于这个过程。最近，“改写困境”在其他方面也受到关注。

因为在“AOP”中使用的“高层程序抽象类”（称为AOP抽象类）可以产生相似的适应性，所以它们和传统意义上的抽象类不同。使用“AOP”的开发人员使用诸如AspectJ, AspectWerkz, JBoss-AOP, JAC, DemeterJ 等开发工具。（参见<http://www.aosd.net>），好的开发工具可以更好地与“AOP抽象类”合作。因为“AOP抽象类”会产生很多冗长的代码，不使用工具容易出错，而且这些代码将分散在很多方法中，不利于插入。为了避免给“AOP抽象类”打上使用错误或者破坏模块完整性的标签，最好找到更好的方法与“AOP抽象类”合作。

Jonathan Aldrich 定义了一个数据库程序中的模块接口，如果程序变化不影响接口，那么“Open Modules”可以担保当数据库程序变化的时候不需要重新测试日志说明。这个解决方案忽视了一些容易被忽略的问题，参见参考文献[13]。寻找一个新的方式来定义数据库程序和方面的接口是一个非常重要的问题。

3.2. 汇总

现在假设有一个“公司对象”，它包括不同的部门，工作组，每一个工作组由一些雇员组成。每一个雇员拥有一个“工资对象”。假设有这么一项任务，通过汇总所有的“工资对象”来计算公司支出的全部工资数额。注意这个任务其实和“日志”的例子颇为相似。在“日志”的例子中我们需要找出所有的 `C.withdraw(..)` 形式的调用，而在这里我们需要找到所有的“工资对象”。在这两个例子中我们都拥有一个图，我们需要在此图中找到特定的节点，然后对这些节点应用特定的方法。唯一的不同是在一种情况下我们使用的是“动态调用图（树）”，而在另一种情况下我们使用的是“对象图”。在这里我们也要应用“改写困境”，当类图变化的时候我们需要重新测试“找到所有的工资对象”是依然正确还是需要进一步改进。

我们怎么表达“工资对象”的坐标呢？我们可能会违反“LoD”。这个后果就是我们会违反“DRY 原则”（DRY 原则的意思是不要复制自己），因为从方法的类图中我们复制了完全相同的信息。

通过模拟“日志”的例子我们可以得到解决这个问题的好方法。“工资对象”不需要了解部门，工作组的信息。毕竟，我们可能需要用这个“汇总”方法来计算一个只由一个工作组组成的公司。所以我们可以这么表述：“从公司对象中访问所有‘工资对象’”。与“日志”一样，“汇总”总存在一个小问题：当我们修改“公司对象”时，我们需要注意所有的“工资对象”都要加总到总工资中。（“改写困境”）

借助 Java 和动态程序设计 (DJ) 库, 我们可以使用图 1 来解决“汇总”问题。“classGraph”是一个公司类图 (可能是一个新公司也可能是一个大公司)。“whereToGo”是从“公司到工资”, “whenAndWhatToDo”是“SummingVisitor”类的一个实例。“whereToGo”是一个含有节点和与本例边界相关的图。在这个例子中, “whereToGo”是一个只有一个边界的图。“visitor”方法描述了“When”, 程序体描述了“What”。

```
classGraph = new ClassGraph();
objectGraph = Source.
    parse("object description");

classGraph.traverse(
    objectGraph,
    whereToGo,
    whenAndWhatToDo);
```

图 1. Java 使用 DJ 的 Demeter 法则

```
class SummingVisitor extends
    edu.neu.ccs.demeter.dj.Visitor {
    int total;
    public void start() { total=0; }
    public void before(Salary o) {
        total += o.get_value(); }
    public Object getReturnValue() {
        return new Integer(total); } }
```

图 2. 使用 DJ(whenAndWhatToDo)的 Summing Visitor

3.3 共性

日志和加总的例子背后的抽象结构是什么? 首先我们指出它们的不同。在“日志”中程序的完成要遍历“动态调用图”。在“汇总”的例子中, 我们能够选择如何遍历类图, 找到所有“工资类”。我们把带有“语言选择器”的包含两层的图, 视为这两个例子共有的结构[36, 37]。从“日志”图反应出的目录结构中我们可以获得既定节点和边界的信息。从“汇总”图中可以获得目录结构的元信息。我们用这个图的结构明确阐明“语言选择器”中的语句, 这些语句从“日志”图中选择节点和边界。要义是静态的思考, 动态的行动。“节点选择器”的传送条件在静态元信息中形成, 在动态图中操作。

“汇总”例子中的元信息是一个非常出名的实体：它是一个类图（UML 中称为类框图）。类图用来确定对象图的哪个边界最易发生遍历。例如：“公司对象”可能有一个属性用来描述公司内大仓库的结构，但是这个与“工资对象”无关。所以不需要遍历仓库，我们可以通过类图中的元信息来静态的确定遍历的范围。注意这里的遍历是用来高效地汇总所有的“工资对象”的[52]。

“日志”中的元信息是一段程序或者是程序的抽象类。如果我们想要查询 `C.withdraw(..)` 调用的所有方法，我们没有必要动态地检查所有的方法调用，而是可以在程序中标记出这些调用。在其他一些情况下，我们不得不动态检查，但是元信息有助于减少动态检查[45, 60]。在这个例子中，我们没有必要通过遍历来汇总坐标，因为我们的设计提供了一个默认的遍历[57]。我们需要的只是在动态执行时弄清楚何时我们处在正确的点上。

“汇总”的例子展示了“LoDC”的应用。“汇总”由三个关注点组成：一个“遍历关注点”（`whereToGo`），一个“对象结构关注点”（`classGraph` and `objectGraph`），一个“协作关注点”（`whenAndWhatToDo`）。我们可能喜欢把这三个关注点分开。“对象结构关注点”由一个静态属性（`classGraph`）和一个动态操作（`objectGraph`）组成。“协作关注点”指出紧随“遍历关注点”后应该做的是做什么。“协作关注点”通过“`visitor`”对象来自然执行。“`visitor`”方法的名称体现出遍历需要在什么位置增强，“`visitor`”方法的程序体则指出是由哪些代码来执行增强的。使用 Java 和动态程序设计库，三个关注点有机联系起来，图 1 表达了这种联系。注意保持改变类图代码的活力。“协作关注点”可能不需要更新，“遍历关注点”可能仅需要较小的更新。“Demeter 工程”的重要目标就是追求这种“结构的松散”和“关注点松散”。

“日志”和“汇总”的另一个共性是我们追求元信息（类图中的元信息和程序执行的元信息）的松散。参考文献[3]中提出的规则限制“外部方面”对程序提出调用内部模块的要求，这个规则确保了“外部方面”相对于模块内部的“执行松散”，因此改变内部执行不会影响到“外部方面”。“开放模块”规定了一个严格的“执行松散”的形式，这种形式也存在局限。

此外，还有另外一个共性，仍以“日志”为例。我们关注所有的 `C.withdraw(..)` 调用。我们可以把结果看作一个子树，它的根节点调用 `main()`，在树里面调用 `C.withdraw(..)`。这和“汇总”的例子非常相似。在“汇总”中我们关注的是“公司树”的子树，“工资对象”是子树的叶。参考文献[60]很好地阐述了这种相似，而且解释了为什么 AspectJ 切入点可以被视为是遍历战略。

从 1992 年“LoDC”就在“适应性程序设计”上发挥着作用。“遍历”代码被视为与类图不同的独立的元素，“遍历者”也被单独的编写。在实际程序开发中，“遍历战略”究竟发挥了多大作用，这是一个经验问题。Wu 和 Wand[61]最近收集了一个系统的统计表（Demeter[35]），证明“遍历战略”有助于创建一个更易实现的系统。

4 “复杂的需求接口”战略

“遍历”的实现比它的提出要复杂的多[37, 51, 52]。人们普遍采用的商业上成功的遍历技术是 XPath。它可以在遍历 XML 文档和 Java 对象 (JXPath) 时使用。

除了 XPath 的原因, 还有一些其他的原因促使我们使用“遍历战略”, 参考文献[37]描述了这些原因。“遍历战略”和它的执行经历了一些反复。我们从简单的战略形式“从 A 通过 X 到 B”讲起。然后我们考虑“混联战略”, 它由简单战略和两个操作者构成: “加入”和“合并”[52]。“混联战略”的编辑运算法则受到限制, 因为它仅对一组类图和遍历战略的联合体发挥作用。然后我们形成了战略及战略的实现法则[36, 37], 使得战略的运算法则更通用更迅速。一份技术报告[38]从简单的语义学角度展示了运算法则的系统语源, 这个角度更易于程序员理解。技术手册中, “从 x1 到 x2 的”的“遍历战略”遵循下面的规定:

$$POSS(x1, x2) = \{e \mid x1 (\leq e \geq)(\leq C \geq)^* \leq x2\}$$

公式可以通俗的归纳为: 只遍历允许有到达目标类对象可能的静态类型的路径。详情见参考文献 [38, 37]。

假设“公司对象”的“工资汇总”问题的变体, 最早的版本由 David Bock [18, 7]提出。送报童来到你的门前, 索要 20 美元作为送报的酬劳。你的钱夹里有一些钱, 但是不够。厨房的橱柜里还有一些钱, 但是还是不够, 为了防备意外, 你有更多的钱在卧室。你可以让这个报童知道所有以上细节, 但是这看起来似乎不合适而且会引起误会。

在 (PaperBoy 类中) 我们没有使用如下程序:

```
customer.wallet.money;
customer.apartment.kitchen.
    kitchenCabinet.money;
customer.apartment.bedroom.
    mattress.money;
```

我们加宽了“客户类”的接口, 增加了方法“customer.getPayment(..)”。

“getPayment”方法通过一个采用“遍历战略”的方式的复杂的请求来实现, 这个“遍历战略”的形式是: “从客户到酬劳”(类似于从公司到工资的战略)。详细代码, 参见参考文献[7]。

遵循“LoD”会引起一个一般性的问题，那就是导致宽泛的类的接口，因为为了达到不同目的，遍历一个复杂的对象结构可以有很多的方式。因此我们使用了一个公开的方法来明确我们要去哪里。（例如从客户到酬劳）。我们传递一个复杂的请求而不是很多单独的请求。

Mitch Wand 指出，客户和服务端之间的这样的复杂请求存在很多优点：

- ◆ 因为服务器得到一个结构化的请求，所以有机会优化。例如，如果类图显示浴室中没有钱，那么“从客户到酬劳”的遍历就不需要访问浴室。
- ◆ 从客户的角度来看，关于请求的知识是以公开的方式表达的，而不是内含在客户的控制结构中，这有助于提高知识的位置。以送报童为例，我们注意到由于遵循了“LoD”的要义，在一个复杂的请求中，关于类图的信息被最小化的复制了。
- ◆ 在一个分布式环境下，复杂请求使那些需要在客户与服务端之间传递的信息的数量最小化。

复杂请求也可以是方面的。“SummingVisitor”就是一个方面的复杂请求，用它修改遍历的执行。

5. “LoD”的其他应用

“LoDC”除了应用于书写方法，还可以应用于其他领域。

5.1. 类图的成长计划

“LoD”不仅能构建方法体，还可以用于构建数据结构（类图）。以一个尚未入住的公寓为例：它包括厨房，客厅，卧室，浴室。当一个房客搬入以后，公寓有了家具：床，桌子等。床可能放在卧室，但是家具的信息应该与基本的公寓的结构分开，单独保存[15]。

这里的模块是类。它们之间如果存在关系则要发生通信。每一个类与一个关注点联合。

在“Demeter”方法中，我们使用“成长计划”的概念[27, 39]来渐进地开发一个应用程序。一个“成长计划”就是不断扩大类图的各个阶段组成的序列，这些类图最终成为应用程序类图。这个序列经过精心的构建，所以每一个阶段都可以有效的得到测试。

成长的阶段可以作为“方面”被封装起来。但是“扩展点”可能需要非常精心的标记，从而实现封装。例如，在成长阶段 i ，我们可能有一个“指定陈述”，在成长阶段 $i+1$ ，这个“指定陈述”会依一定条件被封装。这就要求我们能选择一个“指定陈述”作为“扩展点”。这也要求我们精心的标记模型来表达“扩展点”，到目前为止 AspectJ 还没能做到支持这种表达。

5.2 创建对象

我们已经暗示了如何使用适应性方法编写符合“LoD”和“DRY”原则（不要重复自己）的松散的代码。下面我们着力于创建对象。假设一个“构造器调用”是一个对于类对象的调用。我们的目标就是依照“LoD”，在给定的方法中使用最少量的类对象，因为每一个都揭示了类的一些信息。

传统的创建对象的方法是传递信息给很多类对象（或者，使用很多调用）。分析“对象描述”对此很有益（对象描述是用明确的语言写出的语句）：`Source.parse`（对象描述）。假设要创建一个空的“文件系统对象”。我们需要创建一个“文件系统对象”，一个“目录对象”，一个用于记录空的文件清单的“集合对象”，还需要创建一个“目录名称对象”等。我们需要这样写这个“对象描述”：“`directory ( ) root`”，这表示我们需要一个叫做“root”的目录，这个目录没有任何文件。文件名写在对象描述的括号里。（`directory ( ) root`）。

注意“对象描述”比冗长的“构造器调用”要简洁很多，后者使用过程中会复制类图中大块的信息。前者只是需要写下一个计划，在计划中定义一些类和自顶向下分析条件的语法，使用“解析器生成器”实现一些分析功能。利用 JAXB（Java 数据绑定工具）把这种方法应用于 XML 中。但是不幸的是，标记语法（如 XML）无法提供和 LL（1）语法一样的结构的松散性。一个 XML 文档不得不重复很多语法结构，而 LL（1）则不需要。我们的工具，诸如 Demeter/Flavors, Demeter/C++, DemeterJ, DAJ 和 XAspects 已经成功使用了这种技术很多年。

创建结构松散的对象的要义是：有形的句法比抽象的句法更抽象。这和“编译器类”的情况正好相反。有形的句法更抽象是因为它使用一个句子定义了一个完整的“抽象语法树”的家族，而一个抽象的句法仅仅定义了一个对象。这个家族上的一个特定的元素通过选择一个句法被选中。例如：“`directory ( ) root`”，当我们为文件结构选定一个特定的计划后，它就代表一个根的目录对象。

以传统的方法创建的原始对象是不完整的[54]，Richard Rasala 提出了创建成熟对象的建议。使用 Demeter 数据绑定技术，我们可以分两步来创建成熟类：第一步先创建一个“成熟树对象”，在接下来的第二阶段，采用遍历创建一个完整的链接，处理成熟对象。

我们精心收集了一些设计模式[28]，它们概要的说明了遵循“LoD”的基本技术：类图模式，成长计划，访问者模式，结构松散对象,结构松散遍历。这些模式是使用 Demeter 方法进行迭代开发的基础[27]。

6. 支持“LoD”的工具

我们可以获得一些支持“LoD”的工具。

6.1 LoD

AspectJ 是一个非常理想的语言,它可以对使用 Java 语言开发的程序编写很好的“LoD”检查程序[7,34]。AspectJ 检查程序和“LoDC”无关，只涉及“LoD”，尽管 AspectJ 有很强的对于关注点（横切关注点）进行分离的能力。

在面向对象的环境中，“LoD”，会导致分散和很多小的方法的产生。为了说明分散，我们引入了“遍历战略”，开发了包含处理“遍历战略”的核心运算规则的“适应性程序库”[9]。在“适应性程序库”上我们构建了 DJ[49]，DAJ[31, 10]（除了早期的 Demeter 系统[35]）。那些系统很大程度上推动了与遍历相关的协作：DJ 主要针对 Java，DAJ 针对 AspectJ。

DJ 采用的方法是使那些与遍历相关的协作的概念象 Java 类一样能够获得。最重要的就是类图，战略，遍历，访问的使用。DJ 有运行时开销，所以我们开发了 DAJ。DAJ 引入了 AspectJ 的“声明”扩展机制，声明战略和与访问者相关的遍历。这样便于 AspectJ 的开发者遵循 LoD。DAJ 通过静态的生成代码，避免了 DJ 的运行时开销。更多的关于 DJ 的信息，参见参考文献[31]。

JXPath 是另一个使得 LoD 易于遵循的工具。JXPath 使用 XPath 符号来操纵 Java 对象。如参考文献[37]中描述的，就结构松散战略而言，XPath 不如战略有表现力。

6.2 LoDC

AOSD团体已经开发出来很多支持“横切关注点”的工具。相关内容可以参见<http://www.aosd.net>。Crista Lopes 的论文[41]中指出，从 1996 年开始，我们使用DemeterJ来支持两类专门针对关注点的面向方面的语言。我们开发了一个简单的weaver帮助我们编译专门针对关注点的语言：类词典，那些与遍历相关的“协作关注点”的行为文件，关于“同步关注点”的COOL文件，关于“数据交换关注点”的RIDL文件。最近，我们开发了XAspects，它把AspectJ作为weaver使用，提供了一个框架，在这个框架中很多方面可以使用插入来聚合[56]，这样XAspects提升了DemeterJ。

WebJinn [25]提供了在 web 域内应用“关注点分离”的框架。WebJinn 组织代码,所以那些“表达关注点”,“功能关注点”,“控制关注点”,“计划关注点”是分离的。在以往的 web 域应用模型中,“计划关注点”是分散的:这使得涉及计划的时候,不可避免地需要访问“表达关注点”和“功能关注点”。这种做法违背 LoDC,因为“计划关注点”并没有引发其他的关注点。WebJinn 使用了一个叫做“计划扩展点”的特殊参考机制复制“计划关注点”,从而使 WebJinn 符合 LoDC。这使得 WebJinn 成为了面向 web 域的特殊 AOP 框架。

7 方面和组件

我们的目标是把一个以组件的形式新增加到系统中的关注点的信息封装起来。一个组件就是一个黑箱,它不能单独展开,它提供了一个或更多的特定的服务功能[59]。在以“组件—连接器”形式存在的软件体系结构中[14],组件的展开通过组件连接器实现。组件连接器提供了一个间接的层面,使得更多的组件可以复用。我们可能也希望方面具有同样的复用性,因此仿照“适应性插入播放组件”的概念[47],我们建议提出“方面组件”这个概念[32]。

“适应性插入播放组件”由一个观念上的类图组成(类似于一组合同中的参与者[16]),这个类图代表了能够表达组件接口的最小限度的结构,(考虑到期望的接口)。连接器用来勾勒基于组件所使用的类图而形成的观念上的类图。“方面组件”也是类似:除了一般的方法,“方面的方法”也可以修改其他方法。“方面方法”提倡“结构扩展点”的概念:我们有一组扩展关注点,通过它们实现相应的扩展行为,同时我们还拥有一个扩展点的图,通常我们还有一些扩展行为,这些扩展行为和从节点到节点的扩展行为不同。组件和方面可以共存[46, 50, 58],但是还有很多开放性的问题。

8 开放性问题

围绕 LoD 产生的论题是很确定的,但是关于如何遵循 LoDC 还存在很多开放性的问题。

控制方面的权力。按照 AspectJ 的风格,“方面”的权力很大,运用低层机制,“方面”可以在程序内相当多的位置进入代码的内部。研究“方面”的限定形式,用高层机制详细说明增强的坐标是很有用的。

D*J 工具(DemeterJ, DJ, DAJ)使用方面的有限形式(尽管更一般意义上说潜在的方面运算法则更有用)。第一步引入一组节点和边界遍历方法,使相关的遍历协作方面扩展类表,然后当方面遍历那些既定的节点和边界的时候,方面扩展了那些方法。第一步所作的,在 Demeter 中将方面很好的局部化了,如我们在图 1 中以 Java 方法调用为例论证的一样。“whereToGo”方面定义了遍历,“whenAndWhatToDo”方面在遍历过程中发挥作用,然

后又由“whereToGo”方面操作类图。在同样的程序中我们可能会使用到多层类图。再次提醒注意类图变化的灵活性（结构松散）：类图的重要改变不会引起对象描述：“whereToGo”“whenAndWhatToDo”的任何变化。

这只是限制方面权力的一种方法，但是我们从中获得了很大的收益。在参考文献[6, 33, 50]中介绍了其他的方法。我们也需要一些机制保护代码不受方面影响，同时也保证代码不会妨碍方面的作用的发挥。使用“方面合约”是一个很有效的方法。目标就是找到一个解决“改写困境”的办法。

方面的交互。当多个方面影响同一个扩展点的时候，我们就需要考虑方面与方面之间的组合和冲突问题。有关这个问题的近期研究参见参考文献[8]。

评价资源消耗。通常需要分析运算法则引起的资源消耗，但是分析方面，例如“方面组件”引起的资源消耗是相当困难的。这是因为以下的两个原因。第一，在程序中，方面是参数化的。对复杂的参数作参数化的分析很困难。第二，“改写困境”意味着方面是一个适合多种实参的灵活抽象。

9 结论

对于 Demeter 法则（LoD 和 LoDC）和它的分支我们进行了研究。LoD 带来了一些问题，我们可能会违背它（写一些难以修改的代码）或者我们写的代码分散在很多类中，与很多关注点发生冲突，横切了类图。为了控制这种分散和冲突，我们采用了面向方面的技术。

LoD 自然的过渡到 LoDC。例如，“whereToGo”这个关注点不需要被“whenAndWhatToDo”这个关注点调用。同样基本的代码也不需要被同步代码或者数据传递代码调用。

基于遍历，我们暗示着这样的一个通用结构，这个结构是“面向方面”的基础也是适应性程序设计的基础。这种关联使我们更好的编辑运算法则，更好的理解高层扩展语言，有助于提升 AOSD。

恰当的遵循 LoDC，换言之，找到一个合适的方法分解成不同的方面，仍然存在很多尚待解决的问题（例如，改写困境）。但是 AOSD 团体不停地开发新的解决办法，最终一定会成功。

致谢

感谢我目前所在的研究组（Doug Orleans, Johan Ovlinger, Jeffrey Palm, Therapon Skotiniotis, Pengcheng Wu, Sergei Kojarski）。感谢我的 CCIS 的同事：Will Clinger, Matthias Felleisen, David Lorenz, Richard Rasala and Mitchell

Wand, 感谢他们给我的灵感和支持。感谢Jonathan Aldrich, Ian Holland, Gregor Kiczales, Crista Lopes, Joe Loyall, Mira Mezini, Macneil Shonle, Andr' e van der Hoek and Wim Vanderperren.的反馈信息。我们的研究得到的资助有：国家科学基金会（NSF 编号为0098643）、美国国防高级研究计划局（DARPA）资助（编号F33615-00-C-1694，来自BBN的转包合同）、瑞士ABBD资助、IBM的Eclipse资助和来自Neeraj Sangal的资助。

参考文献

- [1] Companion of the 18__Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, Anaheim, California, 2003. ACM Press.
- [2] M. Aksit, K. Wakita, J. Bosch, L. Bergmans, and A. Yonezawa. Abstracting Object Interactions Using CompositionFilters. In R. Guerraoui, O. Nierstrasz, and M. Riveill, editors, Proceedings of the ECOOP' 93 Workshop on Object-Based Distributed Programming, volume 791, pages 152 – 184. Springer-Verlag, 1994.
- [3] J. Aldrich. Open Modules: A Proposal for Modular Reasoning in Aspect-Oriented Programming. In Proceedings of the FOAL workshop of the 3rd international conference on Aspect-Oriented Software Development, 2004.
- [4] D. Batory and B. Geraci. Composition validation and subjectivity in genvoca generators. IEEE Transactions on Software Engineering, 23(2):67 – 82, February 1997.
- [5] A. Colyer, A. Rashid, and G. Blair. On the separation of concerns in program families. Technical Report COMP-001-2004, Computing Department, Lancaster University, UK, 2004.
- [6] D. Dantas and D. Walker. Aspects, information hiding and modularity. Technical report, Princeton University, November 2003.
- [7] Demeter Research Group. Law of Demeter Homepage. <http://www.ccs.neu.edu/home/lieber/LoD.html>.
- [8] R. Douence, P. Fradet, and M. S' udholt. Composition, Reuse and Interaction Analysis of Stateful Aspects. In K. Lieberherr, editor, Proceedings of the 3rd international conference on Aspect-oriented software development, pages 141 – 150. ACM Press, 2004.
- [9] Doug Orleans and Karl J. Lieberherr. AP Library: The Core Algorithms of AP: Home page. <http://www.ccs.neu.edu/research/demeter/AP-Library/>.

[10] Doug Orleans and Karl J. Lieberherr. DAJ: Demeter in AspectJ home page. <http://www.ccs.neu.edu/research/demeter/DAJ/>.

[11] G. Duzan, J. Loyall, R. Schantz, R. Shapiro, and J. Zinky. Building Adaptive Distributed Applications with Middleware and Aspects. In K. Lieberherr, editor, Proceedings of the 3rd international conference on Aspect-Oriented Software Development, pages 66 – 73. ACM Press, 2004.

[12] T. Elrad, R. Filman, and A. Bader. Aspect-Oriented Programming. Communications of the ACM, 44(10):28 – 97, 2001.

[13] R. E. Filman and D. P. Friedman. Aspect-Oriented Programming is Quantification and Obliviousness. In Workshop on Advanced Separation of Concerns, OOPSLA, Minneapolis, USA, 2000. <http://icwww.arc.nasa.gov/ic/darwin/oif/leo/filman/text/oif/aopis.pdf>.

[14] D. Garlan and M. Shaw. An introduction to software architecture. In Advances in Software Engineering and Knowledge Engineering, volume I. World Scientific Publishing Company, 1993.

[15] W. Harrison and H. Ossher. Subject-oriented programming (A critique of pure objects). In Proceedings OOPSLA '93, pages 411 – 428, Oct. 1993. ACM SIGPLAN Notices, volume 28, number 10.

[16] I. M. Holland. The Design and Representation of Object-Oriented Components. PhD thesis, Northeastern University, 1993.

[17] A. Hunt and D. Thomas. The Pragmatic Programmer. Addison-Wesley, 2000.

[18] A. Hunt and D. Thomas. The art of enbugging. IEEE Software, pages 10 – 11, January/February 2003.

[19] W. L. Hürsch and C. V. Lopes. Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA, February 1995.

[20] IBM Research Team. Hyper/J home page. <http://www.research.ibm.com/hyperspace/HyperJ/HyperJ.htm>.

[21] M. A. Jackson. Software Requirements and Specification. ACM Press, 1995.

[22] I. Jacobson. Language support for changeable large real time systems. In Conference proceedings on object-oriented programming systems, languages and applications, pages 377 – 384. ACM Press, 1986.

- [23] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An Overview of AspectJ. In J. Knudsen, editor, European Conference on Object-Oriented Programming, Budapest, 2001. Springer Verlag.
- [24] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In European Conference on Object-Oriented Programming, pages 220 – 242. Springer Verlag, 1997.
- [25] S. Kojarski and D. H. Lorenz. Domain driven web development with WebJinn. In Companion of the 18__Annual Conference on Object-Oriented Programming Systems, Languages, and Applications [1], pages 53 – 65. Special Track on Domain-Driven Development.
- [26] K. J. Lieberherr. Component enhancement: An adaptive reusability mechanism for groups of collaborating classes. In J. van Leeuwen, editor, Information Processing '92, 12th World Computer Congress, pages 179 – 185, Madrid, Spain, 1992. Elsevier.
- [27] K. J. Lieberherr. Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns. PWS Publishing Company, Boston, 1996. 616 pages, web book at www.ccs.neu.edu/research/demeter.
- [28] K. J. Lieberherr. Pattern Language for Adaptive Programming. <http://www.ccs.neu.edu/research/demeter/adaptivepatterns/pattern-lang-conv>, Feb. 1997.
- [29] K. J. Lieberherr and I. Holland. Assuring good style for object-oriented programs. IEEE Software, pages 38 – 48, September 1989.
- [30] K. J. Lieberherr, I. Holland, and A. J. Riel. Object-oriented programming: An objective sense of style. In Object-Oriented Programming Systems, Languages and Applications Conference, in Special Issue of SIGPLAN Notices, number 11, pages 323 – 334, San Diego, CA, September 1988. A short version of this paper appears in IEEE Computer Magazine, June 1988, Open Channel section, pages 78-79.
- [31] K. J. Lieberherr and D. Lorenz. Coupling Aspect-Oriented and Adaptive Programming. In R. Filman, T. Elrad, S. Clarke, and M. Aksit, editors, Aspect-Oriented Software Development. Addison-Wesley, 2004. In press.
- [32] K. J. Lieberherr, D. Lorenz, and M. Mezini. Programming with Aspectual Components. Technical Report NU-CCS-99-01, College of Computer Science, Northeastern University, Boston, MA, March 1999.

 **WILEY**

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective
Strategies for
the Agile
Software
Developer

Scott Ambler

《敏捷数据》

UMLChina 李巍 译

机械工业出版社即将出版



软件工程技术丛书

设计系列

企业应用 架构模式

Patterns of Enterprise Application Architecture

(英) Martin Fowler 著

王怀民 周建 译

UMLChina 审校

CHINA-PUB.COM

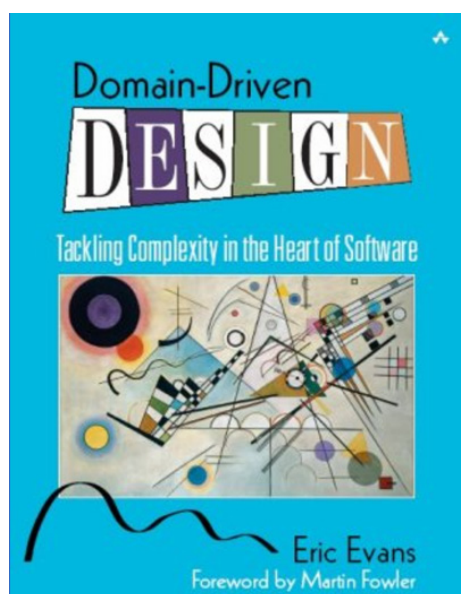
UMLChina 训练辅助教材

领域驱动设计讨论

潘加宇、孙向晖 译



从 2003 年 11 月 18 日到 2004 年 2 月 3 日期间，硅谷模式小组（Silicon Valley Patterns Group）把他们的每周例会花在学习 DDD（领域驱动设计）上。每次大约 20 个人参加，《领域驱动设计》作者 Eric Evans 也参加了他们的讨论。译者摘译讨论实录中的部分内容，供大家学习 DDD 作参考。



歌一首 领域驱动

（采用"Homeward Bound"的曲调）（译注：Simon & Garfunkel 的名曲）

Tracy Bialik, Russ Rufer

我坐在工作站前

面对两吋厚的文档（唉...）

每一页看起来都那么相同

不是 Visitors 就是 Factories

每个我看到的数据类

让我想起，我要——

合唱：

领域驱动

希望我们是领域驱动

领域，代码不再破碎

领域，我的头不再痛

领域，系统不断对我说话

.....

Russ: 最开始的谈话——也就是印刷电路板那个例子，有没有人有问题？

Robert: 谈话感觉很有代表性，表达了软件人员和领域专家之间所用术语之间的不一致。

Eric: 首先我想传达的意思是，这是我一无所知的领域，所以这个例子描述的是不懂领域的软件人员如何谈论这个领域。在对象的角度看来，这也有令人感兴趣的行为。我有另一个不同场景的完整版本，但书的草稿的许多读者说作为第一个例子，不用太过强求完美。

（众人讨论用图形来表达领域的话题...）

Eric: 你需要某些能表达对象模型的东西，我使用 UML。领域专家通常可以没有困难地理解它。决定哪些应该成为一个类或一个方法，更多的是软件开发人员的责任，而不是领域专家，但整个交互序列应该能让领域专家理解。刚开始沟通时，在尝试定义共同术语的过程中，会有很多挫败。所以当我浏览所有用例时，它们都提及了“网络”（nets）。什么是“网络”？它是连接若干芯片（chip）的一些东西。所以芯片和 nets 在模型中成为对象。文字、图形和解释都合上拍了，你就有信心画一些简单的、领域专家能理解的 UML 模型，这就是起点。我尽量避免使用更复杂的 UML，最多画一个菱形表示聚合，还有两侧的数量来描述可以关联多少个实例。这时模型不足以构造一个系统，但它正向建立一种通用语言迈进。

Tim: 我见过有人对同一个东西起了多个名字，隐式建立连接，还在不同的前提下描述不同的连接....使得从 UML 图中提取信息相当困难。

Eric: 听起来你说的是更复杂的建模问题，而不是表示法的问题。

Russ: 当在白板上画画时，是你往白板上加东西还是领域专家？

Eric: 我通常听他们说，然后画一些框框。领域专家很少向图中添加东西。但即使在第一张图之前，也有很多谈话。你不会知道最开始的谈话会给你带来什么，但至少这是一个开始。在电路版的例子里，最早我就学到了一块芯片可以和多个网络（nets）连接，一个网络可以连接到多块芯片。在另一个方向上，也有领域专家不熟悉术语的情况，这样也会深入到更加通用的通用语言（ubiquitous language）。例如，领域专家们说“参考编号”（Ref-des）而不是“元件”（component），最终使得“参考编号”成为共用语言。

Russ: 模型要走多远？

Eric: “管用”是判断的标准，如果它使得困难的事情变得容易了，就是“管用”了。通常对一个东西会有很多方法来建模，而且有时两个模型一样管用。所以不需要在意从什么好地方开始建模，不管从何处开始，你都要集中思路，遵循迭代的路线来推敲它。例如，即使我不是从“网络”开始建模，而是从信号（Signal）开始，我也能得到“管用”的模型。

Russ: 那如果从线路（Wire）和端点（endpoint）开始呢？

Eric: 也可以的，但我想我会很快发现这在印刷电路设计师在创建设计时的操作抽象级别之下。在后面的“深度建模”章节的运输模型中，有货物，船，集装箱，装卸的场地等。但最后更管用的模型中，除了考虑船只之外，更要考虑“航次”（voyage），这是计划和分配货物的正确思考路线。也许客户还是谈论船只，但船只不是他们做计划时关心的。他们可以在最后一刻更换船只。在书的后面我花了时间来讲述如何聆听领域专家的话。如果领域专家绊倒在模型的语言上，可能这个模型需要精化了。我不认为应该很抽象地构造领域专家不能理解的模型，但这常常发生，因为软件开发人员比领域专家要擅长抽象。

Russ: 你有没有一些早期阶段的经验法则，来记住内容丰富的小片信息，用来变成最终的领域模型？在这个过程中，你如何使自己免于无关细节的干扰？

Eric: 这类问题和用例相关。在印刷电路板例子中，我们从一个用例开始，并不是所有从中得到的对象和概念都在领域模型中保留，但用例对开发对模型的深度理解有价值。

Carlos: 你会推迟把东西往纸上或白板上摆吗？为了避免把谈话搅浑，模型过早变得具体，僵化。

Eric: 我没有觉得这是多大的问题。你不需要太早这样做，你需要足够的背景来发现一个开始的地方。印刷电路的例子中，在进入商讨（模型）之前，我们已经花了几天来讨论用例。

Carlos: 什么时候你开始写一些东西让它变成模型？

Eric: 任何你觉得应该的时候。但如果觉得模型没用，你要停下来，回到用例上。在这个过程中，可能有无数的困难和错误，可能坏答案比好答案还多。你必须保持丢弃更多的模型、更多的变化和可能性。如果你要重新脚踏实地，请回到用例。

Charlie: 对于用例，你喜欢它们是一种文字描述，还是喜欢一些特别的格式？

Eric: 我不想太过死板。只要是说系统所做的什么对用户有用，而不只是描述用户和 UI 交互，任何形式都可以的。我更喜欢 Cockburn 的格式。

Russ: 我喜欢这句话，“强大的新特性作为已有的特性的推导而出现”，这是好的领域模型的好处。

Eric: 如果你认为软件是改变的和进化的。典型地，一开始会变得很快，然后慢下来，慢到被认为是“遗留”相同，然后继续运行，基本上不再改变。

John B: 那么重构是为了防止这个进展缓慢下来，而深度建模会加速进展。是吗？

Eric: 是的。

Shiva: 如何避免分析瘫痪？

Eric: 不要把分析阶段和设计分离。编出一个能展示和运行的原型来，以帮助讨论需要什么。如果你得到了错误的答案，它能够帮助领域专家看出问题，和你讨论来澄清它。直到你走完一个完整的周期，用例、设计、原型。分析和设计相互依赖，结合到一起，这就是为什么要先有一个基于模型的运行的原型。

Tim: 我有一些例子和以前的失败，管理层对“产品是产品线的一员”这一点不作思考。“别管它，以后再修改”，“怎么，这个不行了？从头再来”，却不去捕捉或识别领域，理解问题的关联。最后结果是不得不添加更多的特殊情况，更多意外，更多条件。

John B: Mozilla 就是一个例子。Netscape 的代码在 1994 到 1997 年运行得很快，然后代码就变得破碎。你可以看看它的 CVS 树。

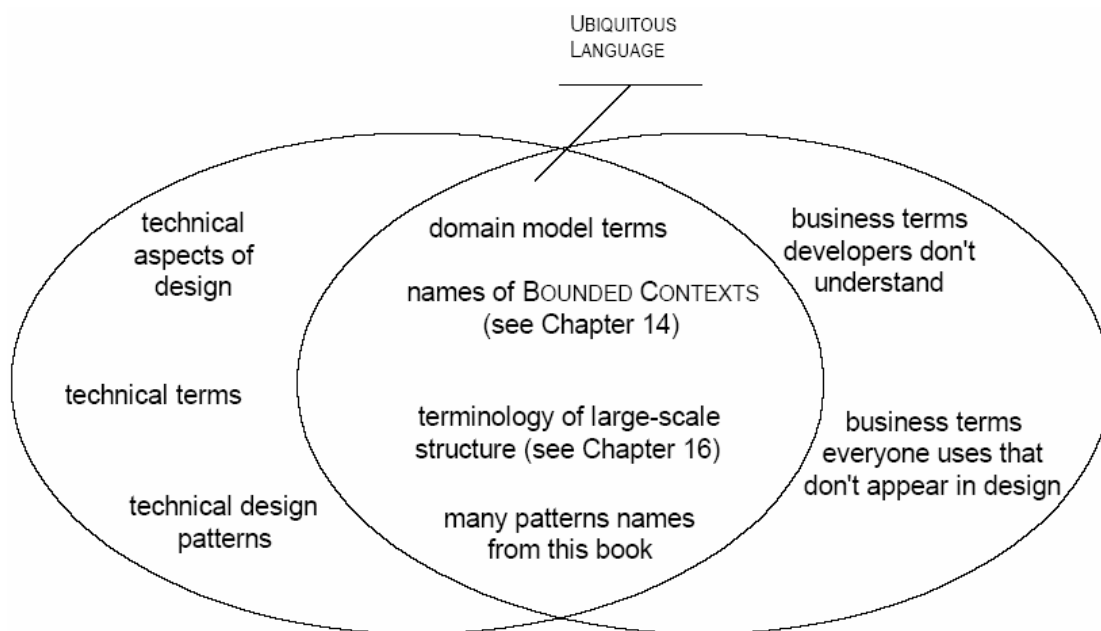
Charlie: 我所开发的软件是由小的团队快速开发，常会出现同样的字段有三套不同的名字，数据库中、领域层中和界面中。

Eric: 在机制不良的小组中，没有通用语言（ubiquitous language），每个小组使用自己的术语，然后由一两个最好的开发人员来在各个小组的语言间作翻译。我有没有足够强调说话的重要性？我觉得我们大脑的语言中心是很好的建模工具。

Carlos: 说话是暴露有毛病的术语的一种方法，而写作则不会。

Russ: 你提到“混杂语”（Pidgin），难道通用语言不是领域专家语言的一个子集吗？

Eric: 不是！通用语言的术语要在设计软件系统时有用，所以领域专家最喜欢用来描述问题的语言不是你可以用来编码一个解决方案的语言。这就是为什么我引入“混杂语”作为对领域和应用建模的通用语言的隐喻。



Russ: “混杂语”是在那儿吗？（指向中间）

Eric: 是的。你可以从右边学到一些业务术语和可能会在模型中出现的上下文。

Russ: 技术这一边的哪一部分在重叠部位？

Eric: 对象。领域专家的世界分解为对象的方式可能并不和软件模型一样.....如果你问一个领域专家“超额预订策略是可以分离的吗”，他们会回答“是的”，但他们并没有用 OO 的视角来看“船只”、“航线”、“货物”。

Mark: 人们会想到发生了什么, 但没有必要把它分解到对象和职责的术语。

Eric: 可能这本书的一半内容都可以用在非 OO 环境, 虽然我不熟悉非 OO 的东西。从领域专家那里获得知识, 消化它们, 直到你有了模型需要的东西。手法是一样的。

Russ: 我来看看这句话, “大多数有才华的开发人员对学习他们正在工作于其中的特定领域不感兴趣”, 这句话有多真实?

Robert: 可能吧。在 Sun, 你的领域是技术的东西, 为别人而做的。

Eric: 这个定律有例外, 应该是“如果领域是技术性的, 开发人员会对领域感兴趣”.....我喜欢有才华的开发人员对领域感兴趣, 这样他们能把代码编得更好。我希望领域驱动设计的一部分作用是为有才华的开发人员提供一种思考领域的更好方法。

Chris: 技术知识通常比领域知识更容易转移。

Eric: 领域相关开发(注意, 我没有说领域驱动开发)的做事方法往往是非常烦人的。当你把这些东西添加到简历上...你掌握了港口货物装卸的知识.....可能你都不愿意把这写上去。如果你能做很酷的技术诀窍, 其他开发人员会认为你真是聪明, 但你说你发现了一种更好的对帐方法时, 他们还会这样想吗? 不。

Tim: 有些领域的复杂性我就有兴趣, 例如跨国公司的多国税法, 如果我把它当做一个复杂问题来处理, 它是有趣的。现在我在做一个音乐播放列表, 我对这个领域更感兴趣。

Eric: 很多领域还在领域建模的阶段, 只有少数领域已经做得很好, 象会计, 已经有很多强有力的抽象, 总帐, 贷方, 借方等。但大多数其他领域还是处女地。

Russ: 我想你对用 C 来做模型驱动设计有点过分严厉了。如果你不需要继承, 也可以做出类似面向对象的东西, 可以传递“this”指针, 就像用面向对象语言编程一样。

Eric: 我不愿说得太夸张了, 但我想选择正确的语言确实是强有力的。错误的语言难写又难读。

Mark: 这更多和程序员有关吧。我见过糟糕的 C++ 代码, 也见过真正整洁的 C 代码。

Eric: 但如果你想要进行模型驱动设计, 现在最多的模型风格是以对象的交互存在的, 所以选择一种面向对象语言能够使得你的代码更贴切地表达模型。你需要一种能够支持你的模型范型的编程语言, 不管它是面向对象的, 还是数学等式, 还是逻辑规则链?

某人: 为什么不让资历较浅的人建模? 这样他们可以得到指导, 吸收技巧。

Eric: 我所说的超越了这一点。我说每个人都要建模, 每个人都要编程, 如果不是这样, 就不算是模型驱动设计。在编程时你也必须在建模。

Azad: 你怎么看“智能数据库”, 用存储过程来进行控制...

Eric: 你可以让它用模型驱动的方式工作, 但你不得不用实体、关系、触发器等术语来表达你的模型范型。

Mark: 我想问的事情书中好象没有说得太明白...可以想法让开发人员和团队接受(领域驱动设计)...但管理层怎么办? 你如何改变管理层的文化以便让开发团队做这些事情?

Eric: 管理层是时髦词汇(buzzword)驱动的, 如果他们对某个时髦词汇感兴趣(今天可能是 XP, 明天可能是 DDD), 就更有可能做这件事情。在我们网站上, 确实有“领域驱动设计经理指南”的内容。

Darlene: 如果领域专家告诉你的是错的怎么办?

Eric: 这本书的大多数东西假定领域专家所说的基本正确。如果在设计中有明显的冲突, 你可以通过阅读书籍来获得另一种对事物的观点或视角, 也可以从两位领域专家那里听取意见。当你找领域专家澄清时, 他可能会说, “噢, 我犯了点小错误, 应该是这样的...” 这样就会听到一些以前漏掉的上下文。

Russ: 我们在开发第一个版本时已经有了经验, 所以我们被团队的其他人当作领域专家, 这样也挺愉快的。

Eric: 要做领域驱动设计, 确实需要领域专家。我不指明说到底领域专家是什么样的人, 但不是所有领域都如此通俗易懂, 以至于软件开发人员可以自己充当领域专家。在印刷电路版的设计案例中, 领域专家是用户, 但很多用户对领域的理解其实很肤浅, 我找了一个相对有更开阔见识的人和他一起工作。

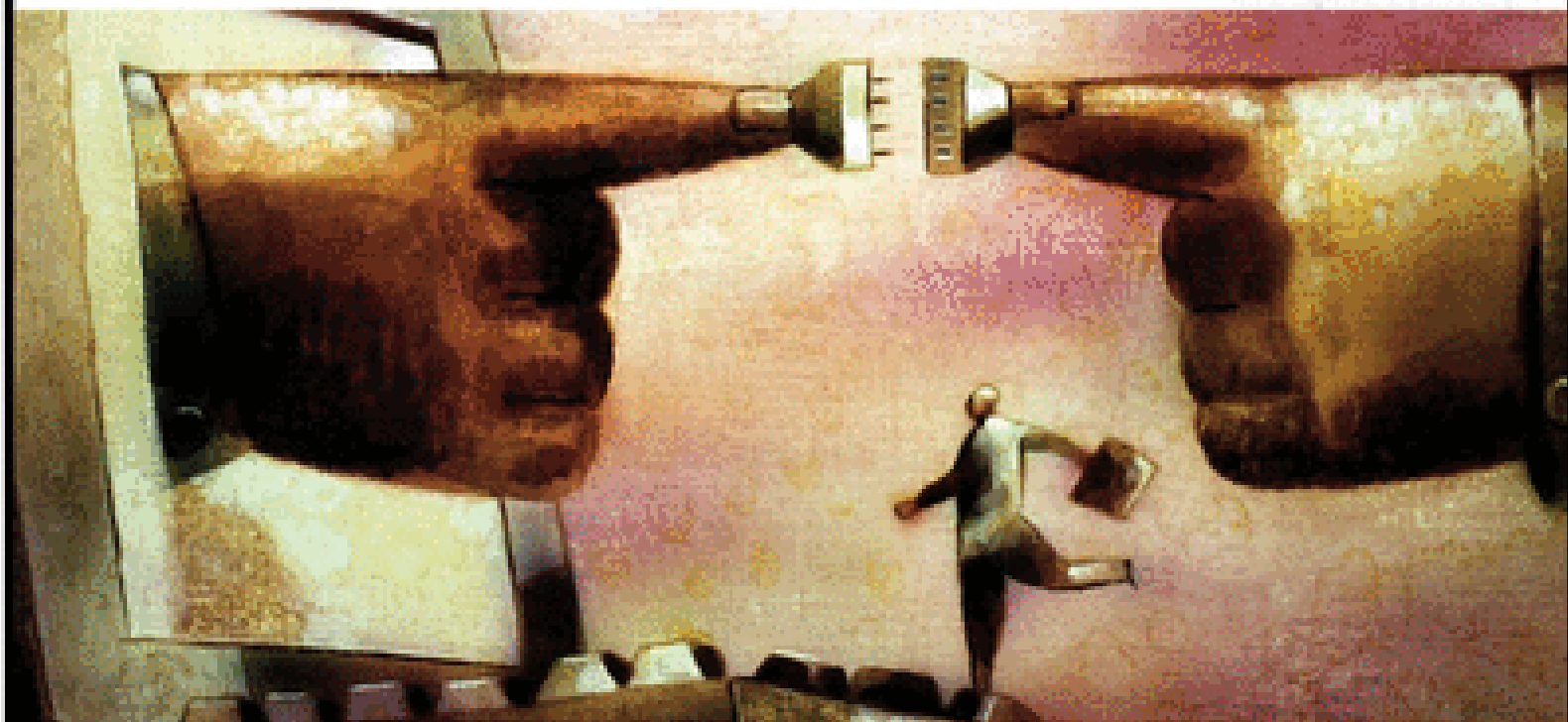
(首发于《程序员》2005 年第 5 期)

"About Face 2.0 could completely redefine how software applications are created!"

— Peter McIlreath, author, *Software Craftsmanship*

ABOUT FACE 2.0

THE ESSENTIALS OF INTERACTION DESIGN



ALAN COOPER
& ROBERT REIMANN

当木匠看到锤子时，他不想和锤子讨论钉子的问题。他会直接用锤子钉钉子。在车里，如果司机想改变方向，他转动方向盘。司机喜欢通过合适的设备从车子和外部环境直接获得反馈：挡风玻璃外面的视野；仪表板的读数；疾驰而过的风声；轮胎压在道路上的声音；对侧向重力的感觉以及路面传来的振动。木匠也希望有类似的反馈：钉子下沉的感觉，铁互相击打的声音以及举起锤子的感觉。

International
Bestseller

Completely revised
and updated

UMLChina 辅助教材。中译本即将由电子工业出版社出版，詹剑峰 译

审稿：国内首个交互设计网站 De Dream'

UML 相关工具一览(I-N) (截止 2005 年 5 月)



以下总结了全世界的各种 UML 相关工具，按工具名称字母排序。

工具 (最新版本)	厂商&地址	试用允许	UML 版本	支持代码环境	XMI	平台	备注
iUML 2.2 	Kennedy Carter http://www.kc.com/products/iu/ml/index.html	有试用版				Windows	xUML 建模工具。开发、执行和模拟 xUML 模型。自动禁止“无效”模型。支持 MDA。可以和 DOORS 集成。
lss-UML  Halstenbach <i>to be a little bit better</i>	Halstenbach (德国) http://www.halstenbach.com/home.php3			Eiffel			Rose 插件, 提供 Eiffel 双向工程支持。
J2U 1.3.3 	NASRA (法国) http://www.nasra.fr/flash/NASRA.html	有试用版		Java	✓	Java	顺序图双向工程, 从可执行 Java 代码到 XMI, 可以直接输出到 Together, 也可以间接输出到任何兼容 XMI 的 UML 工具。
Java Studio Enterprise 7.0 	Sun http://gceclub.sun.com.cn/ise/		2	Java		Java	支持 UML2.0, 可以通过模型生成源代码, 又可以在模型中编辑代码。
Javelin 6.5.8	Step Ahead http://www.stepaheadsoftware.com/javelin.htm	30 天试用		Java		Windows	基于 UML 的图形编程环境, 自动保持类图和 Java 代码同步。

										
JDeveloper 10g 	Oracle http://www.oracle.com/tools/jdev_home.html		Java					Java	结合了 UML 的 J2EE 工具	
JSequence 1.0A	Objective Ideas (瑞典) http://www.aptrio.com/Development/Java/sequence-5652.html	有试用版	Java					Java	自动从 Java 代码中产生顺序图。	
Jude 1.5.2 	Objectclub http://objectclub.esm.co.jp/Jude/jude-e.html	免费	Java					Java	Java 双向工程, 支持 UML 1.4	
JUMLi 1.4 	Kreditwerk (德国) http://www.jumli.de/		C++, Java, C#					Java		
KUML 	KUML http://sourceforge.net/projects/kuml/	开源						Linux	自 2000 年 10 月后没有更新。	

MacA&D 7.4 	Excel Software http://www.excelsoftware.com/macsoxproducts.html		2	C++, Java、Delphi	✓	Mac	支持 UML 建模, 结构化建模, 数据建模。
MagicDraw 9.5 	No Magic, Inc. http://www.magicdraw.com/	有试用版	2	C++、Java、C#、IDL	✓	Java	和 Eclipse 集成, 支持多国语言。支持 RUP 和 WAE 设计模式, 支持 DoDAF 框架。
MasterCraft Component Modeler 	MasterCraft (印度) http://www.tata-mastercraft.com/overview.asp						基于 UML 的构件建模工具
MEGA 6.1 	MEGA International (法国) http://www.mega.com/us/product/overview/	有试用版		Java、VB、C++	✓	Windows	系列套件, 从业务流程分析到构件设计, 基于 UML。
MetaEdit+ 4.0 	MetaCase Consulting (芬兰) http://www.metacase.com/	有试用版		Smalltalk、C++、Java、Delphi (Object Pascal)、SQL、CORBA IDL		Linux , Windows	领域驱动建模工具。

Metamill 4.1 	Metamill (卢森堡) http://www.metamill.com/	30 天试用	2	C++, Java, C#	✓	Linux , Windows	多用户建模支持。
Model Driven Generator Link for Eclipse 	Sparx Systems http://www.sparxsystems.com.au/products/mdg.html	有试用版		Java		Java	EA 的 Eclipse 插件。
Model Driven Generator Link for Visual Studio.NET 	Sparx Systems http://www.sparxsystems.com.au/products/mdg.html	有试用版		C#, Visual C++, VB.Net, J#			EA 的 Visual Studio.NET 插件。
Modelmaker 7.05 	Model Maker (荷兰) http://www.modelmakertools.com/	有 demo 版		Delphi, C#		Windows	支持构件和模式，支持某些“自适应”的方法学。和 Delphi 2005、C# Builder、VS.NET 2003 自然结合。
Mia-Generation 4.5 	Mia-Software (法国) http://www.mia-software.com/index.php?lang=en&theme=prod-miagen	有试用版	2				模型到代码的转换器。专注可裁减的代码生成。覆盖现在使用的各种主流语言。 
Mia-Transformation 3.5 	Mia-Software (法国) http://www.mia-software.com/index.php?lang=en&theme=prod-miatrans	有试用版	2				模型到模型的转换器 

Modelistic 1.1 	Modelistic（英国） http://www.modelistic.com/	有 demo 版		Java		Java	在 Eclipse 遵循 XP 原则来进行 Java 开发。
Nucleus BridgePoint 	http://www.acceleratedtechnology.com/			C/C++			XtUML 语言使得系统模型可以在设计时进行执行和检验，然后被翻译为 C 或者 C++ 的代码，直接编译到某个嵌入式系统上。
Novosoft UML Library 	Novosoft（俄罗斯） http://gemini.novosoft.ru/NS2B.nsf/w1/UML_Library	开源		Java	✓	Java	元数据框架。

*有一些上一次总结曾经列出的软件因没有更新而被废弃。

ISFxiAR	 Projexion Netsoft (法国) http://www.projexion.com/index.php?lang=fr&ID=10			Java			
JVISION 2.1	object-insight http://www.object-insight.com/product/ 	有试用版		Java		Linux , Solaris , Windows	
MIUML 0.98 	SoftwareFarm http://www.swfm.com/miuml.htm	免费			✓	Java	此项目目前已停止
Model Prototyper	Objexion (法国) http://www.objexion.com/				✓		从UML模型产生SQL和GUI原型