第 4 辑

荟 萃 全 球 软 件 智 慧

Dr. Dobb's JOURNAL CHINA
www.ddjchina.com

软件研发

项目管理 软件变更管理

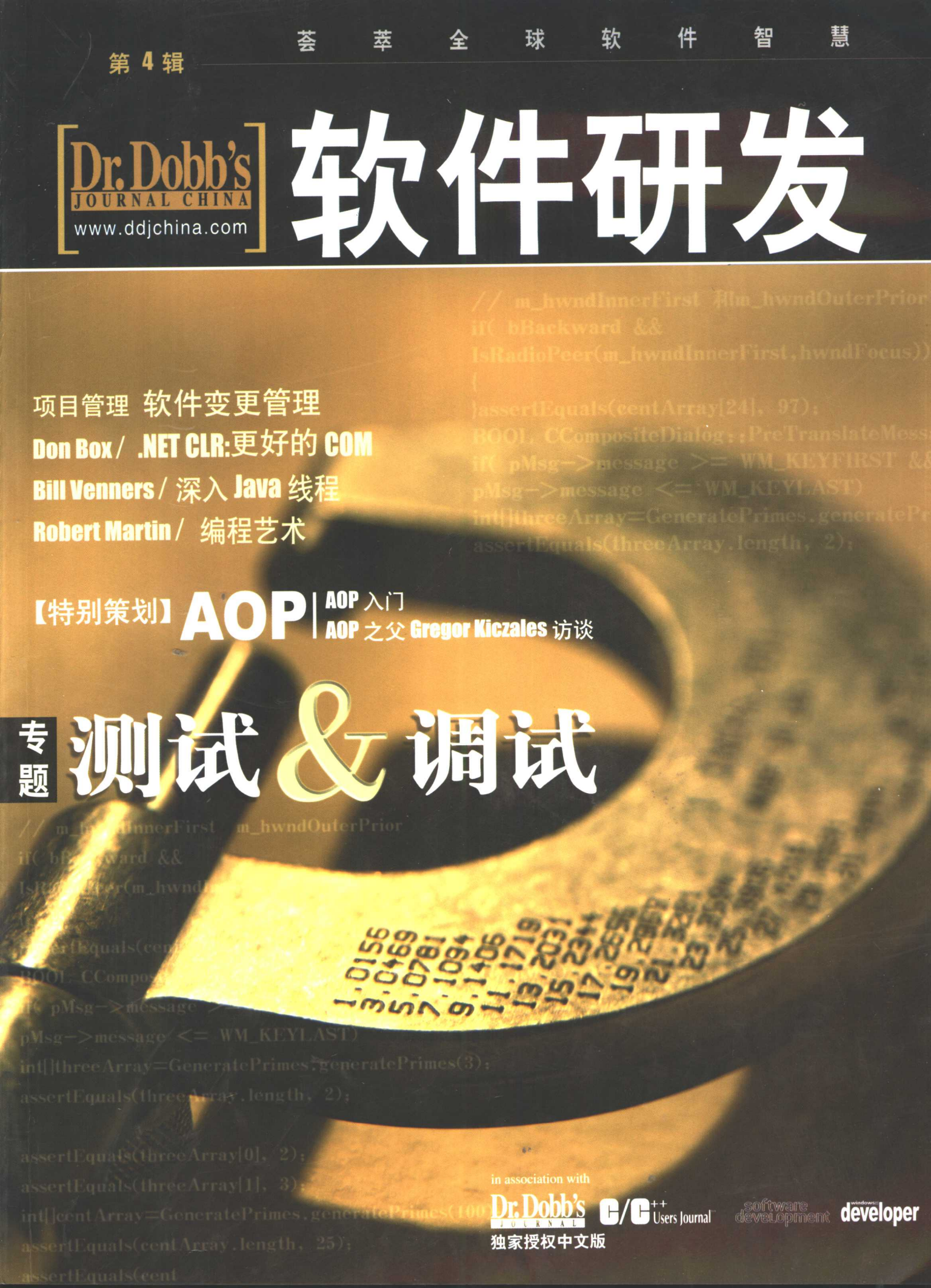
Don Box / .NET CLR:更好的 COM

Bill Venners / 深入 Java 线程

Robert Martin / 编程艺术

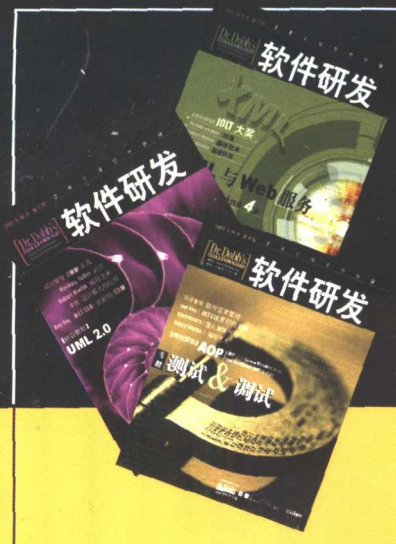
【特别策划】AOP | AOP 入门
AOP 之父 Gregor Kiczales 访谈

专题 测试 & 调试



in association with
Dr. Dobb's JOURNAL C/C++ Users Journal software development windows developer

独家授权中文版



Dr.Dobb's JOURNAL CHINA
www.ddjchina.com

软件研发

2004年征订工作全面启动，欢迎订阅！

订阅参考

2004年全年共12辑，每辑18.00元，订阅价216.00元 **免邮费**

优惠政策

订阅日期	优惠折扣	总 计
截止到2003年11月30日止	8折	172.80元
截止到2008年12月15日止	8.5折	183.60元
截止到2003年12月31日止	9折	194.40元
2004年1月1日以后	—	216.00元

特别说明：凡订阅了2003年全年的读者，订阅2004年全年始终享受7.5折优惠。

2004年编辑计划

1 JANUARY	2 FEBRUARY	3 MARCH
数据库编程，.NET	C/C++程序设计，项目管理	软件设计最佳实践
4 APRIL	5 MAY	6 JUNE
Linux与开源软件	Web服务，Agent	算法，软件工程教育
7 JULY	8 AUGUST	9 SEPTEMBER
嵌入式系统，性能优化	软件开发过程	JAVA程序设计
10 OCTOBER	11 NOVEMBER	12 DECEMBER
测试与调试	无线与通信与网络编程	信息安全

010-88379061
010-88379062
010-68995260
reader@ddjchina.com
北京市西城区百万庄南街1号 309华章公司
邮编：100037

Dr.Dobb's JOURNAL CHINA www.ddjchina.com

软件研发

目 录

软件技术

Dr.Dobb's 软件研发

www.ddjchina.com

Feature: Testing & Debugging

Special: AOP

Column

Management

Technology

执行主编：刘江
责任编辑：贾梅
编　　辑：吴怡 陈剑瓯
编辑信箱：editor@ddjchina.com
美术编辑：锡彬
排版制作：金浩
网　　站：周明涛

读者服务
电子邮件：reader@ddjchina.com
电　　话：88379061 88379062
网　　站：www.ddjchina.com

版权登记号 图字：01-2003-1691
图书在版编目(CIP)数据
Dr. Dobb's软件研发. 第4辑／（美）科斯（Koss, R）等著；荣耀等译. —北京：机械工业出版社，2003.11
ISBN 7-111-13437-0
Ⅰ. D… Ⅱ. ①科… ②荣… Ⅲ. 软件研发 Ⅳ. TP311.52
中国版本图书馆CIP数据核字（2003）第092560号

出版发行
机械工业出版社
（北京西城区百万庄大街22号 邮政编码 100037）
印刷：中国电影出版社印刷厂
版次：2003年9月第1版第1次印刷
889mm × 1194mm 1/16 · 7.5印张 16彩页
定价：18.00元

欢迎与告别

新的《C/C++ Users Journal》刚拿到手，我吃了一惊：卷首语的作者居然是杂志元老级的前任资深编辑P.J. Plauger！怎么回事？仔细看看报头（masthead），果然，P.J. Plauger已经重新出山，接替离开的Chuck Allison担任资深特邀编辑。卷首语“回来真好”说明了一切：

“早在1987年我就开始为CUJ写文章了。当然我万万没有想到，自己竟然会连写13年，而且整个90年代几乎还一直担任资深编辑的职务。然而，如果不是因为要管理自己的小公司，我还真的不愿意选择离开。

“如今，我已经快60岁了，做职业程序员已经满40年，而C语言标准化20周年的庆祝晚宴也将在一周后举行。这么多年以后，许多地方已经物是人非，但是我欣慰地看到，C和C++依然保持着旺盛的生命力，CUJ依然茁壮，我们的行业依然保持着无限活力。未来充满期待，而CUJ也因此有更多的东西可以报道。一句话，回来真好。……”

Plauger以精力旺盛而著称，从1987年到2000年5月，他一期不拉地为CUJ写卷首语、写技术文章，与此同时，还为其他三种杂志撰写专栏，并长期服务于ISO C标准化委员会（任召集人）和C++标准化委员会，开发了Visual C++中使用的C++标准库，成功经营自己的公司，撰写了《The Standard C Library》、《C++ Standard Template Library》等名著……。

Plauger老将出马，又能读到他老辣的文字了，而他深厚的人缘，相信会有更多高手为《C/C++ Users Journal》贡献大作，这些我们当然求之不得。没啥说的，热烈鼓掌欢迎吧。

Chuck Allison的告别比较突然。按道理他和Bruce Eckel合写的《Thinking in C++》第二卷也应该交稿了，时间比原来应该更充裕才对。说老实话，对Chuck的离开我非常遗憾。他写的卷首语是我见过的技术杂志中最有特色的，差不多篇篇都是精品。他几乎从来不采取重述后文内容这样的老套路，往往是时事、杂感相得益彰，旁征博引、技术严谨之外，又不失幽默风趣，而文风的多变几乎要赶上《Dr. Dobb's Journal》中的长期固定专栏“Verity Stob's Adventure”（我们没有选择这个奇妙的专栏，因为它几乎无法翻译）了。就拿今年的几篇来说吧，有的套用（paraphrase）摇滚乐曲的歌词，有的完全就是一篇精巧的技术小品，有的包含大段代码和公式却又能做到活泼有趣。有机会的话我们也许可以拿出来几篇与大家“奇文共欣赏”。衷心希望Chuck的离去是暂时的，期待着他能经常回家看看。

如果我没有记错的话，Chuck Allison今年已经至少四次在CUJ卷首语中提到了软件的质量问题。这透露出目前整个社区对此问题的深切关注。近来微软公司由于软件漏洞被人频频利用引起病毒肆虐，波及甚广，不仅公司名誉受损而且还惹上诸多官司。这提醒我们：软件已经渗透到人类社会的方方面面，提高软件质量，既是公众对我们的要求，也是软件产业正常发展的根本所在。然而，目前的现实却是，软件开发作为一门学科本身历史尚短，还很不成熟，而如今的软件复杂性（非线性、人为因素、环境影响）又达到了空前的程度，要想根本解决软件质量问题，依然是“路漫漫而修远兮”。作为软件研发界的一分子，我们肩负的使命艰巨而重大。

解决软件质量问题，调试和测试显然是最基础的工作。这正是本辑的主题。顾名思义，测试的任务是发现问题，调试则是解决问题。历史上，调试和测试长期被人看作是一门艺术。在我国，测试开始受到重视还只是这几年的事情，整体水平亟待提高。本辑的专题文章均出自国际名家之手，希望能够对读者有所启示。同时，我们还会继续开辟有关测试的专栏，请大家继续关注。

《软件研发》明年的征订方式和编辑计划都已经发布，读者可以访问www.ddjchina.com了解更多情况。

2003年11月

Google：下一代霸主？

最近有没有注意到Google的变化？你可能没有想到，在不怎么变化的首页背后，Google却在日新月异地飞速发展。自5月以来Google已经增加的各项业务、功能、区域多达数十项，几乎每星期都有新东西出来，发展之快真是令人咋舌：计算器、度量单位转换。字典？新闻？股票行情？电话簿？等等。最新的进展是Google Deskbar，下载运行后，工具栏里就会常驻一个搜索框，可以不打开浏览器，随时用Google查询。这让我开始浮想联翩：如果Google这样发展下去，它完全可以成为一个独立的平台；如果还能在瘦终端上实现，还能集成一些功能，许多人可能真的不再需要PC机和Windows了。难道，未来真的属于Google为代表的Internetware？

Wintel联盟分裂？

在将近30年PC时代中，微软和Intel两大霸主结成的Wintel联盟一直牢不可破，从整个IT产业链中获得了丰厚的收益。相比之下，以创新而著称的Apple则因其特立独行，常常面临危机，业界一直猜测Apple什么时候会屈从Wintel帝国联盟的压力。然而，最近的事态似乎有了变化。IBM与微软近日签订了一份重要的协议，将在下一代Xbox中使用“IBM的微处理器系列”作为主芯片。虽然用语含混，但是明眼人早已心知肚明：谁不知道IBM的微处理器就是PowerPC？而最近PowerPC的表现也确实不凡，Apple 6月推出的基于Power PC G5的新一代Macintosh已经大放异彩。而Virginia Tech大学师生在一个月内花费500多万美元，用现成的1100台Macintosh G5（2200个IBM G5处理器）组装成功一台绰号“Big Mac”的超级计算机，居然速度能达到每秒可执行7.41万亿次运算（也有数据说后来达到了9.55万亿次每秒的水平），一举跻身全球超级计算机前15名，然而成本却低得多。也许这正是后PC时代的特点，什么事情都有可能发生。

软件工程盛会

10月20～24日，由ACM和IEEE主办的第六届UML国际会议“UML2003”在旧金山举行。会议的热点当然是UML2.0和MDA，方面技术也是大家关注的对象，大会总主席Grady Booch本人参加的就是UML面向方面建模研讨会。

使用对象技术的人可不能不知ACM主办的OOPSLA（Object-Oriented Programming, Systems, Languages and Applications）大会。要知道，许多重要技术的诞生与OOPSLA会议都有着密切的关系：比如模式，比如JUnit（由Kent Beck和Erich Gamma在去OOPSLA会议的飞机上合作开发）。今年的会议于10月在加州Anaheim召开，会期几乎接着UML2003，与PDC完全重合，因此参加会议的人比平日少了一些。会议主旨报告总的感觉比较平淡，但是各分会场技术内容极大丰富，而且绝对可以说是群贤毕集：我们熟知的软件研发界人士，几乎一个不拉地出现在各种场合，其盛况大概只能用海洋来形容。会上比较热的主题是泛型、敏捷开发方法、模式、AOP、MDA与生成式编程（generative programming）、垃圾回收、领域驱动的开发。从中可以看到面向对象技术的发展趋势。Eclipse在会场上相当抢眼，所有与会人士在注册时都得到了Kent Beck与Erich Gamma合著的《Contributing to Eclipse》一书。

再见Red Hat Linux

Linux作为玩家工具的时代看来真的要结束了。在Novell收购第二大Linux发行商SUSE的同一天，全球第一大Linux发行商Red Hat也宣布，将不再发行新版本的Red Hat Linux产品，对原有产品线的支持最晚截止到2004年4月。也就是说，Red Hat Linux 9将是多年来自由软件迷心目中至爱——红帽子Linux的绝唱。Red Hat公司的业务重点将放在Red Hat Enterprise Linux（RHEL）产品线上——又是面向企业级！好在红帽子并没有真正消亡，Red Hat Linux将并入社区支持的开源项目Fedora中。

会议：微软PDC 2003

10月份的天气当然最适合开会了。所以本月会议之多，让人眼花缭乱：UML2003，OOPSLA2003，Mac OS X大会，STAR West等等。中国开发人员最关心的，当然是微软一年一度的专业开发人员大会PDC，因为媒体早已经曝光，这次会议将提前展示新一代Windows操作系统Longhorn的庐山真面目。

被Windows95之后历次升级弄得已经有些麻木的人，对于本次升级变化之大恐怕都无法视而不见。从预览版来看，Longhorn将充分体现微软多年来在自然人机界面方面的研究成果。三维、动画、语音、音乐等等都可能融入其中（相信其中有不少亚洲研究院的功劳），随着文件系统和存储模式的改变（目前新的文件系统WinFS仅用于Documents and Settings文件夹），搜索功能也应该会有极大提高。

当然，对开发人员来说，影响最大的恐怕是编程模型和框架方面的更新。会议上展示了.NET框架的明天——WinFX，这是完全托管的Windows API，将完全取代Win32 API。其中最显著的是新的界面API集Avalon。其实，早在.NET发布的时候，就有不少人发现Windows Forms和Web Forms非常相似，甚至开始预测两者会在下一版本合并。这虽然没有成为现实（两者在WinFX中都保留下来，名字空间也没有变），但通过Avalon（名字空间System.Windows或MSAvalon.Windows），微软实现了桌面和Web编程模型的统一。而其中奥妙还是XML。微软专门为此开发了一种XML方言——取名XAML（XML Application Markup Language，读做“Zamel”），也不管IBM、HP、Oracle、Sun等厂商共推的标准早已使用了这个缩写（XAML，Transaction Authority Markup Language）。底层方面值得注意的是CLR层次上实现的泛型（generics）。

（更多新闻和评论，请访问WWW.ddjchina.com）

【会议报道】

2003'软件研发最佳实践大会

9月15~18日，本刊英文版《Software Development》主办的“软件研发最佳实践（SD Best Practices）大会”在美国波士顿召开。会议除主题发言之外，主要由各种技术报告会、讨论会、课程以及展览组成，涵盖了软件研发生命周期的所有阶段：“需求与分析”、“设计与架构”、“测试与质量”、“构建与部署”、“人、项目与团队”、“过程与方法”。

今年的主旨报告人为Watts Humphrey和Ed Yourdon，还有敏捷方法三杰：Martin Fowler，Bob Martin和Ken Schwaber。

各个分会场、研讨班中群贤毕至：

软件工程界的Scott Ambler（《敏捷建模》作者，本刊英文版特邀编辑），Larry Constantine, Lucy Lockwood（均为《Software for Use》作者，本刊英文版专栏作家），Jeff Deluca(FDD方法创建者)，Luke Hohmann（敏捷方法专家），Jim Heumann(IBM/Rational)，Linda Northrop (CMU SEI)，Mary Poppendieck（本刊英文版特邀编辑），Johanna Rothman（项目管理专家，本刊英文版专栏作家），Joshua Kerievsky（模式专家，本刊英文版专栏作家），Jennifer Stapleton（DSDM方法专家）；

C++界的Dan Saks，Stephen Dewhurst（《C++ Gotchas》作者），Andrei Alexandrescu（《Modern C++ Design》作者）；

开源社区的Brian Behlendorf（Apache），Nat Friedman（Ximian）；

Java界Elliotte Rusty Harold（《Java Network Programming》、《Effective XML》作者），Ted Farrell（Oracle），Sue Spielman（《Struts in Action》作者），Danese Cooper（Sun公司）；

以及测试界的Robin Goldsmith（本刊英文版专栏作家），James Whittaker（《How to Break Software》作者），Web服务专家David Chappell（Sonic公司副总裁）。

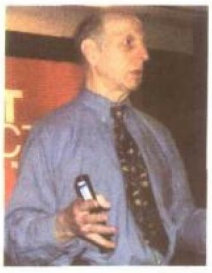
从会议各研讨班的主题来看，敏捷方法、建模（MDA和UML2.0）、模式、重构、安全、项目管理是大家的关注焦点。其中敏捷方法占据了一半左右，探讨和应用的深度和广度都达到了空前的程度。

颠峰教导

9月15日，Watts Humphrey的主旨报告——“做一个软件专业人士（Being a Software Professional）”揭开了四天会议的序幕。让我们听听来自《Software Development》杂志女主编Alxandra Weber Morales的现场报道。

报告是这样开始的：“有一次，小提琴家耶胡迪·梅纽因在完成了一次音乐会之后，有一位妇女走上前来说：‘我真想自己一辈子都能像这样演奏小提琴。’梅纽因回答说：‘是的，夫人，这正是我在做的。’作为开发人员，我们的工作往往是从一个理解很不充分的应用场景开始的，然后我们需要分而治之，将其简化为计算机也能理解的形式。这工作需要进行艰苦训练。”面对全场的听众，Humphrey说：“人们花钱雇佣我们，我们应该肩负起责任，按照计划好的进度表交付高质量的产品。如果做不到这些，老板不会满意，业务会受影响，我们的个人发展也会出问题。”

从梅纽因到美国的奥林匹克短跑选手莫里斯·格林，Humphrey的报告中旁征博引，举了不少例子说明个人的生产率能够通过严格的科学训练得到提高。因为有在IBM二十多年和卡内基-梅隆大学SEI十多年的工作经历，他自己就是一部活历史。多年积累的丰富经验、新奇的故事加上有趣的动作、幽默的言语，Humphrey俨然是一位十足现代的导师，为全场与会者带来了业界颠峰级的体验。虽然本次会议敏捷方法隐隐然已经有了主流的意思，但他报告的主题仍然是围绕着自己一手创造的开发过程：PSP（Personal Software Process）和TSP（Team Software Process）展开。



Watts Humphrey小档案

Watts Humphrey在1986年从IBM退休以后，加入了卡内基－梅隆大学的软件工程研究所（SEI）。在IBM效力的27年中，他担任过各种技术领导职位，包括所有IBM商业软件开发的管理，其中著名的有OS/360的前19个发布版本。退休前他是IBM程序设计质量与过程总监。在SEI，他创建了声誉卓著的过程项目，领导了软件能力成熟度（CMM）最初的开发，并且引入了软件过程评估和软件能力评估的概念。

没有计划，不要承诺

Humphrey最近的工作集中在重塑程序设计实践上，其成果就是PSP和TSP。在报告中他谈到了变化的世界中计划的重要性。

“我经常为自己编写程序，真的很有意思——需求可以随心所欲地改变。”Humphrey说。听众中发出了会心的笑声。“我们的产品实际上都免不了要改变需求。这又是一个承诺问题：我们需要对一个变化的世界做出承诺。如果对每一点小变化都漠然置之的话，到头来你其实是在一步步接近完蛋。我们必须接受训练，学会如何处理所有的小变化。”

“许多年前，我负责管理了704、709计算机的开发。还有人记得这两种计算机吗？这是第一台晶体管计算机，是为[美国]军方开发的。负责的长官说：‘我们没有足够的钱做测试了，因此需要把进度延后三个月。’‘那可能要花更多钱，’我说。‘为什么？’他吃惊地问。”Humphrey解释说让工作人员停留更长时间肯定需要花钱。“四个月之后，他又来了，说：‘Watts，我们有钱了，可以加速项目赶上原来的进度。’我回答说：‘那可能要花更多钱。’‘为什么？’他问。我回答：‘我们已经添加了新的功能，事情已经变化了呀。’这个故事说明，所有变化都会增加成本。”

按照Humphrey的说法，要成为软件专业人士必须采用五项原则：跟踪和报告、计划、过程度量、需求和设计、质量。对于第一个原则，他说，业务总是要按进度和承诺运营的，如果开发人员不能跟上进度的话，就无法成为整个业务的有效部分。“我们的客户和管理人员必须了解我们工作的状况。为了精确地跟踪状况，就需要进行度量。而要度量，就必须有可以度量的计划。”最后，Humphrey建议：“每个项目都要做好计划，永远不要在没有计划的情况下对老板指定的日子做出承诺——当然，老板的话通常只是虚张声势而已（笑声）。”

“最近，我指导了一个由九位软件工程师、四位硬件工程师组成的开发团队。他们要开发一个全新的设备——一种电话线测试仪，对市场极为重要。在有经理和营销人员举行的会议上，经理在对产品进行了概要描述之后，下了死命令——必须在9个月内开发出来。‘先不要争论’，我告诉他们。‘我们先回去制定计划。’他们上一个项目花了两年时间，完全是一场灾难，所以他们提出了一个历时18个月的计划。经理一听就爆了：‘什么？你们是要把公司搞垮！我们撑不过18个月——竞争对手们可能今天就会有更好的产品出来。’‘今天？’我发问了。‘那你想他们的产品是从什么时候开始开发的呢？预测市场难道不是你的工作吗？’最终，项目成功了，计划如愿完成。他们没有丧失一个客户，产品交付还提前了6

个星期。”

Humphrey自豪地说，在他指导下使用TSP开发的50个小组，都取得了成功。这可不全靠运气。其中的奥秘何在呢？他解释说，制定计划的时候，同时也在定义过程，而定义了过程之后，计划也就更容易制定了。

“如果能够在有度量、跟踪和指导的情况下进行，人的表现，当然也包括软件的性能，实际上是没有止境的。”

“就拿世界百米飞人莫里斯·格林来说吧，他是怎么打破世界纪录的呢？”Humphrey说。“将自己百米10秒钟的整个奔跑过程用视频记录下来，分成11段，逐一分析，然后想办法提高每一段。历史就是这样创造的。”

质量第一

在简单谈到构建系统前要进行设计的主题之后，Humphrey强调了质量的重要性。他开玩笑说：“如果产品不起作用也行的话，你的开发真的可以很快。”他说，在使用PSP开发了3万个程序之后，SEI已经收集了大量程序员工作方式方面的数据。从中能得到什么教训呢？要想发挥最大潜力，开发者个人实践必须进行定义、量度、跟踪和管理。

“最好的项目都是巧妙地平衡了互相冲突的各种力量：业务需求、技术能力和客户要求，”Humphrey说。“如果任何方面稍有偏差，那么产品质量就堪忧了。为什么有些大软件公司，拥有许多研发机构，但开发出来的产品却非常差劲呢？原因之一就是他们离实际太远了。最好的程序都是在现场开发出来的。”同样，最好的团队都会自己定义过程，自己制定计划。TSP是指导这类团队的一种方式，事实上已经被ABB、AIS、Bechtel-Bettis、波音、Comnet、EDS、Honeywell、Kaiser、Litton、洛克希德、Microsoft、NASA和美国海军等机构所采用。

Humphrey在报告中演示有关缺陷密度和组织CMM级别相关度的一项研究。在CMM 1级中——也就是混乱的开发组织，平均密度是每千行代码（KLOC）7.5个缺陷。而2级达到了每千行代码6.24个缺陷，3级是每千行代码4.73个，4级显著下降到了2.28个。（Humphrey对此解释说：“这一级别开始进行度量了。”）5级也就是CMM的最高级，每千行代码缺陷只有1.05个。——再往上，采用TSP，就脱胎换骨了：每千行代码只有0.06个缺陷。实际上Humphrey希望CMM 4级和5级能够更精确地定义，吸收TSP实践。

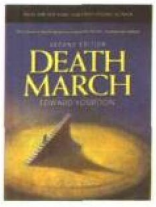
Humphrey报告的最后是听众提问。对于最常见的问题——极限编程是否与TSP能够配合呢？Humphrey给予了肯定的回答，他解释说极限编程的许多实践都是经典的软件工程技术。另一个与会者希望推荐一些收集度量数据的工具。Humphrey回答说：“工具其实非常简单。有些人抱怨度量太花时间了，但关键在于如果你能实施的话，这些数据是极有价值的。记录缺陷数据的平均时间是每缺陷20~30秒，当有几千个缺陷时总时间加起来确实不少。但每个人开始跟踪缺陷经常是从半途开始的。此外，我们都总是会犯同样的错误，因此有前车之鉴是非常重要的。”Humphrey声称，跟踪缺陷能够帮助开发人员在编译之前找到80%的缺陷，在系统测试之前找到99%的缺陷——波音公司的一个项目曾经通过缺陷跟踪将测试时间从8个月减少到两个星期。

最后一个问题是：哪个TSP实践最难采纳？“人们比较反对的是程序评审。无论编译和测试与否，评审程序的时间都是一样的。如果不评审的话，就要花总项目10%~12%的时间在编译上，20%的时间在测试上。但如果先评审程序，那么编译上只需要花2%的时间，测试上也只需要花5%~10%的时间。但是人们必须实践才会相信这一点。”

死亡之旅2.0

大家有多少人读过《死亡之旅》（Death March）？这本1997年出版的书至今仍然是最重要的软件项目管理著作之一。作为最后一位主旨发言人，9月18日Ed Yourdon的报告“死亡之旅项目新论（A New Look at Death March Projects）”吸引了大部分与会者，Hynes会议中心的大会议厅爆棚了，组织者不得不另外又开放了一个有大屏幕的会议室，以容纳更多听众。然而，报告本身对于听众似乎也成了一次“死亡之旅”——在一小时之内Ed Yourdon旋风般地切换了60张幻灯片，从传统的项目管理技术（80/20法则、关键链技术、系统动力学等等）到环境因

素（项目里的中断问题、工作空间问题等等），其间还有丰富的成功和生存建议。



《Death March》2.0一书的封面

“历史上的所有软件项目都有一定的风险和压力，但是人们通常把重压之下的项目戏称为‘死亡之旅’项目。从我的书出版以来，经济大环境已经发生了变化，但是今天的商业环境更加充满了变化和不可预测性，项目进度压缩、预算和资源限制，更成了家常便饭，因此‘死亡之旅’项目显得更加普遍了。似乎惟一的成功之道就只剩下整个团队“每天16个小时、每周七天、项目完成之前没有假日”的搏命行动。虽然此类项目对于公司而言，目标是实现不可能的任务，创造奇迹。而对于项目经理和团队成员，他们的个人目标就可怜地缩减成一点点生存权利：“保住自己的饭碗，维持与夫妻、孩子的关系，不要得心脏病、胃溃疡。”Ed Yourdon的报告是这样开始的。

随着强调变化的敏捷方法日趋流行，Ed Yourdon一年多来一直在修订自己的名著，以反映前一版出版6年以来的新情况，包括敏捷方法怎样在死亡之旅项目中应用。9月份，Yourdon的手稿已经接近完成，这次报告的内容正是以此为基础的。

“对于正在管理死亡之旅项目的诸位，”他说。“我的报告不会给你建议，教你怎样成为专制的老板，也不会教你怎样成为受人爱戴的经理。我们的重点将是死亡之旅项目中至关重要的因素：时间。因为时间是项目中最宝贵的资源。我们往往一分钟都浪费不起。”

围绕着时间管理，Yourdon讲述了如何处理项目中无处不在的“中断”问题。各种“中断”，比如分散精力的杂事以及在几件事情之间切换。都是非常耗时的，很容易使项目偏离进度，必须认真对待。另外，采用关键链安排进度，找出并消除团队中的低效环节，也是时间管理的重要方面。

“对于一个项目经理，首先要搞清楚的，是项目成功的标准。我们达到什么的目标，才算成功了？成功与否又是谁说了算？”Yourdon说。“如果连这都搞不清楚，你趁早走人好了。要知道，主要的参与者和相关人也是成功的关键。如何与人协商，如何做出妥协，都是成功的要素。”他再一次提出了在书中阐述过的triage（“事情的轻重缓急”）概念。他打趣地说：“这次会议上讨论的许多最佳实践就不符合triage的概念。而对于死亡之旅项目，triage非常重要，弄不好会带来高昂的成本。所以必须尽早为项目各项事宜确定优先级。”

一个小时的时间，对于有着40年从业经验的Ed Yourdon而言显然是太少了。虽然Yourdon的报告平易近人、通俗易懂，可是大师有太多的东西需要慢慢咀嚼，很多概念因为没有时间展开，许多听众都抱怨会议时间安排得不合理。当然，Yourdon名著的新版年内即将问世，其中肯定有完整生动的阐述，我们就拭目以待吧。

作为世界知名的计算机顾问，Yourdon是软件工程界的大宗师之一，20世纪70年代，他是结构化分析与设计方法的主要推动者，20世纪80年代他提出了Yourdon/Whitehead面向对象分析与设计技术，20世纪90年代提出Coad/Yourdon方法，在各个时代都起到了重要作用。1997年他与Charles Babbage等一同被选入了计算机名人堂。

Yourdon小档案

巴西：世界软件界的新力量

8月份《软件研发》第一辑首发仪式上，CMP媒体集团的代表就曾介绍，从全球软件研发外包的市场来看，除了印度、中国和爱尔兰、以色列以外，巴西近年来发展速度很快。9月15日晚上MIT举行的一场研讨会上，卡内基-梅隆大学的教授Francisco Veloso，巴西驻美大使，还有“软件研发最佳实践大会”的巴西参展商一起讨论了巴西在外包方面能够迅速迎头赶上的原因。会上，成立于1970年、拥有5500名雇员、1.3亿美元收入的巴西软件企业Politec，巴西第一个获得ISO 9001认证的构件软件提供商Ci&T以及巴西软件促进会等纷纷表示了希望获得美国更多外包合同的愿望。

从产业规模上来看，巴西与印度和中国已经形成鼎足之势，市场总值分别为77亿（只有1.5%出口，其余全部是国内产品和服务），82亿（70%以上是出口，主要是服务）和74亿（60%属于服务，5.5%出口）美元。虽然印度在劳动力价格、产业成熟度和语言上对其他两个竞争对手有明显优势，但是巴西由于货币贬值，劳动力价格方面其实更有优势。而且，巴西有更多跨国企业。而印度的2800家企业几乎全部是国内企业。此外巴西在许多垂直市场拥有很强实力：巴西在电子政府方面处于全球领先地位，已经开发了世界最大的电子投票系统，并且几乎实现了100%的电子报税。按Veloso教授的说法，这些经验与印度的优势——底层代码移植、编码和调试，和中国的强项——系统集成和产品可以媲美。看来，我们真的应该重视和关注一下这个新对手了。

好了，主编给我的版面用完了，报道还将继续，我们下月再见吧。

the 15% SOLUTION

访 谈：15%解决方案

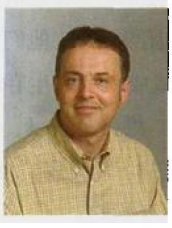
出自传奇般的研究机构PARC的一种全新语言，将“方面”的概念注入到Java对象中，这种概念可以解决面向对象编程中不易解决的安全、调试和其他问题。首席科学家Gregor Kiczales和他的小组并不只是局限在学术领域，他们雄心勃勃，要抓住其中巨大的商业机会。

■ *Alexandra Weber Morales & John Reitano*

虽然Gregor Kiczales没有赶上施乐Palo Alto研究中心（Palo Alto Research Center，简称PARC）20世纪70年代的创新盛期——图形界面、激光打印机、以太网、鼠标和客户/服务器架构，但他对PARC不从这些才华横溢的概念中赚钱的传统是知之甚深的。在这个著名的思想库担任科学家达18年之久[1]的Kiczales，认为自己的贡献中最有价值的是CLOS语言（Common Lisp Object System）。

近年来，Kiczales一直在思考“方面（aspect）”的概念——一种可以轻松跨越类边界的代码片断。有很多人认为，方面可能与对象一样，建立起一种有深远影响的新的程序设计范型。Kiczales虽然没有大学学位[2]，但是他却以计算机教授身份领导了位于加拿大温哥华的不列颠-哥伦比亚大学（UBC）的一个研究项目。这个最新项目——呃，应该说是产品，已经成为软件研发界正在讨论的热门话题。他自豪地说，从概念雏形（aspect-oriented programming，面向方面程序设计，简称AOP）到2001年AspectJ 1.0版本（一种对Java的扩展，能够从概念上将以往分散在源代码各处的各种任务合一）发布，项目在DARPA的大力支持下只花了7年时间就完成了。AspectJ 1.1版本……。他目前在不列颠-哥伦比亚大学带领一些学生进行各种AOP方面的研究，包括AOP在操作系统、重构、模式和各种新的语言构造上的应用。最近，在他刚刚为25名Java精英开发人员讲了一天的AspectJ课程后，我们采访了他，谈起了AOP的发展历程、现状以及他对AOP的远景设想。访谈分为两部分，上半部分以技术内容为主，相信对正在摸索AOP技术的读者将是宝贵的参考。下半部分偏重于如何使技术能够更好地得到采用，以及开发过程中采取的“15%解决方案”，对软件企业乃至高新技术企业如何推广新技术和新产品也有很好的借鉴意义。

Gregor Kiczales小档案



Gregor Kiczales，1978～1983年，麻省理工学院。其中1980～1983年担任该校计算机科学实验室研究员，1982～1983年还担任过Atari公司研究院的顾问。1983～1984年Symbolics公司剑桥实验室研究员。1984～1992，施乐PARC研究中心研究员，1992～1996，施乐PARC研究中心软件设计经理，1996～2002，施乐PARC研究中心首席科学家。2000年至今，不列颠-哥伦比亚大学计算机科学教授。2002年9月至今兼任Intentional公司顾问。ANSI CLOS设计组成员，CLOS参考实现的实现者，CLOS元对象协议的主设计者，《The Art of the Metaobject Protocol》（MIT Press，1991）一书的作者。AOP思想的创造者，AspectJ项目的负责人。发起召开了面向方面编程的国际会议AOSD。

Software Development: 给我们讲一讲AOP的起源好吗?

Kiczales: PARC有好几个研究小组一直在寻找改进软件架构的方式。20世纪90年代早期，我们已经有了10多年用OOP(面向对象程序设计)构造系统的经验，实实在在地看到OOP应用在大型系统、分布式系统和灵活性要求较高的系统时存在的局限。我们想寻求改进的办法。我很幸运地领导了PARC研究反射（reflection）和元对象协议（metaobject protocol）的小组。这些方式都曾经因为过于强大，人们以至于无法有效运用它们——当然，现在它们已经被认为是在面向对象程序中提供灵活性时至关重要的技术。此后我们开始研究一种称为“开放实现（open implementation）”的概念，这与现在所谓的“白盒抽象（white-box abstraction）”很类似。在多年之后，我们最终认识到“横切（crosscutting）”可能正是我们要找的东西。大约在1992年，John Lamping和我开始尝试如何获取横切。我们感到，自己已经抓住了这些技术的要害问题——让程序员能够在自己的代码中将“横切结构”模块化。这就是AOP的起源。1997年，我们开始开发AspectJ。首次对外公开是1998年，1.0版发布是在2001年。这期间许多优秀的研究人员都贡献了自己的力量。

Software Development: 你的小组里还有谁?

Kiczales: 开始时我是惟一的成员。从一点小小的直觉发展到“这里可能值得研究一下”是一个人。发展到原型又有一个人。用Java实现依靠的是第三个人。每一阶段，我们都要求每个人追求极限，不断挑战自我，这意味着要承受先驱者的寂寞和长期的辛勤劳动。

Software Development: “追求极限、不断挑战自我”，能给我们解释一下吗?

Kiczales: 我们的出发点，是一个正常情况下不容易接受而且还没有清晰成形的概念，我们需要使这一概念付诸工程实践，这也绝不是令人轻松的任务。我们从研究性的概念发展到1.0的产品，仅用了8年，极短的时间，而这通常要花15～20年。我认为我们之所以能做到这一点，就是因为每个人都将自己发挥到了极致。这个小组中大部分人都不会将自己归为学术研究者，我们需要的是用户，而不是论文。

Software Development: “横切结构”是什么意思呢?

Kiczales: 就是说有两个或者更多结构无法清晰地互相配合。我们可以看一个经典的AspectJ例子。以下图为例，其中黑色的结构包括FigureElement、Point和Line，它表示的是图形元素的状态和绘制操作。另二个结构是红色的，包括DisplayUpdating“方面”，它考虑的是改变图形元素的状态会如何触发显示刷新。

黑色的图形元素结构中说明了setX、setY、setP1和setP2方法的一个方面——设置方法如何改变图形元素的状态。这些方法另一个方面在红色的显示更新结构中——它们都要在调用后刷新屏幕。

这个例子看上去有点简单，但是它完全能说明横切的概念。图中，DisplayUpdating是独立的单元。但是在黑色的结构中类似的行为却不能用独立的单元集中起来。如果那样做的话，实现这一行为的代码肯定在改变显示状态的所有方法中。我们说，图形元素状态这一关注点的结构与显示刷新这一关注点的结构是互相横切的。

横切结构与层次结构不同。横切意味着要将系统以不同方式切开，而不只是转移细节、获得更抽象的视图。

横切是一种比分散层次更深的属性。分散（Scattering）指的是在给定的实现中，某个关注点的代码遍布各处。而横切指的是内在结构，比如图形显示行为和刷新行为内在就是互相横切的。AOP的目标就是以模块化(而不是分散的)方式实现横切关注点。

Software Development: 很多人谈到AOP时，都只是提到一些系统关注点，比如安全、日志、调试、事务划分等等，能举几个用于业务的横切关注点的例子吗?

Kiczales: 在一个具体的系统中，除这些显而易见的关注点之外，还会有很多其他包含横切结构的关注点。方面可以用来保持类中几个方法的内部一致性。它们很适合“按约定设计（Design by Contract）”风

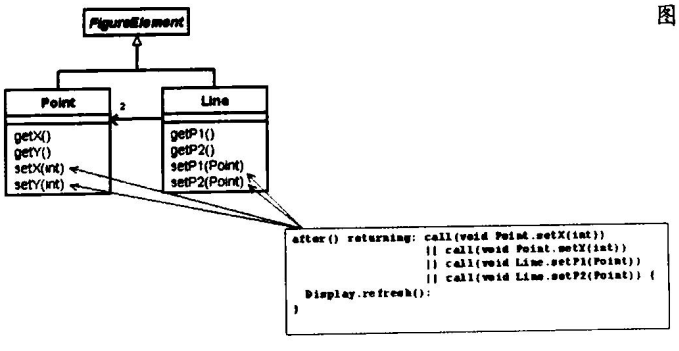


图1

格的程序设计，可以用来保证遵守共同的编码规范。许多面向对象设计模式也有横切结构，也可以用AspectJ以模块化和可重用的方式编程。当然，许多AspectJ的书中也有很多好例子。

用AOP编程要同时涉及类和方面，它们都是很自然的功能单元，方面和类一样都是很一般性的概念，可以无所不在。在今年的AOSD会议上，Ivar Jacobsen发表了一个演讲，他认为用例也有横切结构。AspectJ项目现在的负责人Adrian Colyer对AOP也有极好的阐释。(详情请关注以后各辑的相关专栏。)

Software Development: 你认为未来会有工具统一运行时编织和编译时编织吗?

Kiczales: 编织（weaving）是AOP的一个术语，指横切的方面与其他代码的协作。因此通过预处理器、编译器、后编译的连接器、载入程序、JIT编译器或者虚拟机都可以进行。现在我们在不同时候进行编织有不同的工具。一些比较新的框架，比如AspectkWerkz和JBoss AOP，都是在运行时使用拦截器（interceptor）编织。而AspectJ是在编译或者编译后进行编织。

设计AspectJ的过程中，我们花了很大力气将编织时出现的问题与语义本身尽可能分隔开。因此AspectJ可以在从编译时到虚拟机支持等很大的范围内进行编织。我认为AOP的这种语义将非常易用。我希望未来人们使用不同工具只是因为它们提供的语义不同，而不是编织时间不同。也就是说，编织与方法分派（method dispatch）类似，只是一种实现技术，应该在幕后进行。

Software Development: 你对AOP的标准化怎么看?

Kiczales: AOP已经不是什么太新的概念了，从想法诞生开始算起已经8年，人们使用AspectJ也有5年了。核心要素也多年保持着稳定。因此我认为确实应该进行一些标准化工作，减少各种AOP工具之间不必要的差异，同时又能为继续进行实验留有余地。

我认为标准化的途径至少有两种。一种就是从现在已经成为事实标准的AspectJ着手，可以定出AspectJ的核心，然后AspectJ的开源开发者们保持这一核心的稳定。现在AspectJ已经成为Eclipse的一部分，因此对它的支持力度会越来越大，这应该是一种比较好的途径。当然，有些人会对AspectJ的若干问题存在疑虑，尤其是动态部署、对Java语法的修改以及占用空间较大等等。因此另一种途径是，我们可以开发一个AOP内核，支持各种系统包括AspectJ，JBoss，AspectWerkz乃至一些尚在研究的系统。这个内核并不定义语法，只是一些核心的语义。首先可以以标准草案之类的形式出现。这样在几年之内，进行底层实现的专家们可以集中精力在效率、安全等方面，而其他人可以主要考虑怎样更好地将编程接口用这些技术封装。

Software Development: 你认为元数据（JSR-175）和AOP之间的关系如何?我们能根据方法的元数据定义切入点吗?

Kiczales: 显然，AOP中的性质（attribute）会大有用场。只要JSR175标准制定完成，我估计AspectJ会相应扩展，至少切入点（pointcut）可以根据性质来匹配。经过一段时间以后，我们就能掌握什么时候根据性质使用切入点最合适，什么时候又应该使用其他种类的切入点。当然，这里面还有许多问题要解决。

很多人只把性质当成“宏”或“声明性程序设计（declarative programming）”来使用。这本身当然不是什么坏事情，但是它没有体现我们需要的模块化的优势。而且这样使用为AOP最强大的功能——切入点组合造成了很大困难。我认为使用性质的时候，应该首先小心地命名，能够描述POJD（plain old Java declaration，普通Java声明）的属性，然后加上一个类似于通知（adbice）的构造，说明哪个方面可以应用于带有此性质的POJD。这与目前用性质直接给方面命名的做法不太一样。

我们下面看一个常见的例子。我想代码应该写成这样（其中方法是放在类中，通知放在方面里）：

```
@ChangesDisplayState
 void setX(int x) { this.x = x; }

after(): call(@ChangesDisplayState
* *(..)) { Display.update(); }
```

而不是：

```
@UpdateDisplay
void setX(int x) { this.x = x; }

after(): call(@UpdateDisplay * *
(..)) { Display.update(); }
```

原因当然是为了更好地将横切关注点模块化。这段代码中的横切关注点是“改变显示状态的方法应该更新显示”。而第一段代码中这是模块化地封装在通知里的。另一个关注点“哪个方法改变显示状态”是发散分布的。而第二段代码中这两个关注点是混合在一起的，都是发散分布的。

我相信，遵循这样的风格，我们将获得可复用性更好的性质名——更可能以有益的方式在新的切入点中使用已有的性质。我要说明的是，上面的例子如果不使用性质，就像这样：

```
pointcut changesDisplayState(): call
(void Point.setX(int)) || ...;

after(): changesDisplayState() {
Display.update(); }
```

两个关注点在代码中都能够很好地局部化。这给了我们很好的启示，需要进一步研究，什么时候该用性质，什么时候不又应该使用。

Software Development：方面这个概念是否顺应了提升抽象层次的趋势呢？

Kiczales：那当然了。AspectJ中，你可以以模块化的方式表达横切结构。我们可以用简洁而且很高层的方式表达“移动屏幕上东西的任何事物”这样的概念。

一些提升抽象层次的工作本质上就是生成性（generative）或转换性（transformational）的。在我们的方法中，可以在实际代码里获取高层的结构。根本没有“我们先转换一下”的感觉。对于转换性的工具我有一点技术上的担心，它们会碰到软件的演化问题，因为人们不只是画画模型，生成代码就完了，就什么也不改了。他们总是要不断地修改代码。

AOP概念还可以应用于代码层次之上，比如代码的设计或者视图中。我们可以用AOP技术暂时集中将一些方法拖到一起——保持原有的模块性，然后扩展，这样就可以以不同方式观察它了，放开后，代码还可以很快地恢复。我称之为“流动的AOP”。可以想像在代码中或者UML中使用流动的AOP技术。

现在已经有一大批人在讨论这种多视角（multifaceted）的世界观了。Grady Booch曾在“Through the Looking Glass（透过镜子）”一文中写到这种新的远景。这篇文章是一次讨论会的总结，会上有他本人，John Vlissides、Charles Simonyi、Ralph Johnson和我参加。美国政府正在寻找有巨大前景的软件研究项目给予资金上的支持。从商业上讲，从现在开始五年之内，很难说谁会成为大热门。但是，方面的时代将会到来，因此不用过分陷入建模者和编码人员之间的无谓纷争中。

AspectJ今天只处在代码层次，但是我认为方面和横切的概念适用于任何层次。我们以前主要致力于代码层，但是理智和环境都告诉我们，不能仅仅局限于此。

Software Development：AOP和OOP之间是什么关系？

Kiczales：AOP对OOP是一种补充。AOP还可以与其他范型比如过程型程序设计共存。但是因为OOP现在已经成为主流，因此大多数AOP工具都是OO工具的扩展。我说“AOP对OOP是一种补充”，并不仅仅指许多AOP工具现在都是Java或者.NET的扩展，我的意思是AOP的实践与OOP的实践将是相辅相成的。优秀的面向对象开发者能很快地学会AOP，因为他们对于系统中哪些部分是横切的有敏锐的直觉。这一点也反映在他们的工作成果中。例如，一个好的AspectJ程序应该首先在85%或者更大程度上是普通的优秀Java程序。

Software Development：有些人说AOP违反了OO模块化的原则，你怎么看待这种说法？

Kiczales：模块化原则包括文本局部化（textual locality）、精确的声明性接口（narrow declarative interface）以及更少耦合、更多内聚等等。如果使用正确，AOP将有助于我们实现这些原则，而不是违反。应该依据AOP适用的例子，而不是那些普通OO已经能够保证有较好的模块性的场合进行评判。

在一个能够从AOP中获益的OO程序中，比如上面的屏幕刷新例子，普通OO解决方案的模块性确实会受到影响。会有用来触发屏幕刷新的代码同时出现在Point和Line类中。这一关注点的代码没有局部化，没有清晰的接口，存在文本性的耦合（be textually coupled），并且将降低所涉及的代码的内聚性（cohesion）。

如果用AOP写同一个程序，模块性将更好。触发屏幕刷新的那些代码可以局部化，它们为其他代码提供了声明性的接口（切入点），耦合更少，所有的代码内聚性更好。

当然，这并不意味着用AOP就不可能写出非模块化的代码，当然也会。我们

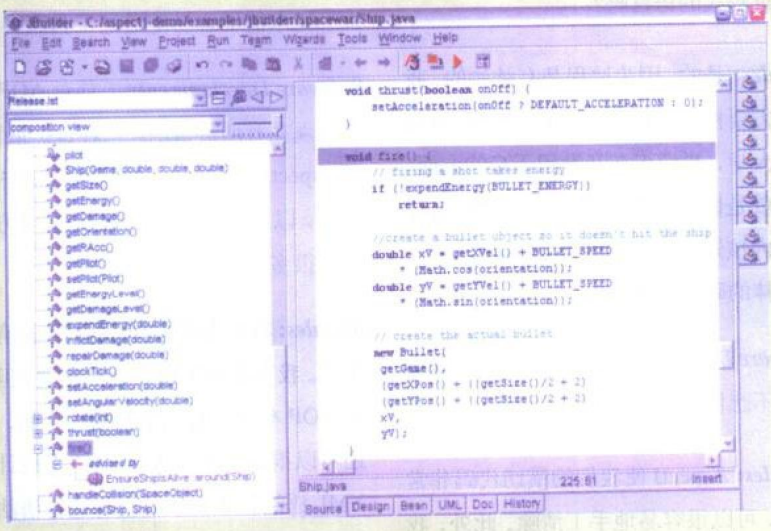


图2为AspectJ清晰地获取了代码中的横切结构，开发环境对AspectJ的支持能够帮助我们对此结构进行导航。这里，JBuilde6中对AspectJ的支持为程序员显示了一个spaceship类中fire()方法的建议。

还有很多工作需要做，比如给开发人员一些简单的实践规则，使他们能够编写优雅的AOP代码的，就像OO中使用私有字段和访问函数这样的规则。但是我相信更应该给程序员提供了能够编写优美程序的工具，而不是设置许多安全机制使他们不能写差劲的程序。事实已经充分证明，大量的程序员可以使用AOP提高模块性。

Software Development: 其他的程序设计范型会消亡吗？

Kiczales: 嗯……Lisp已经有些走下坡路了。逻辑程序设计和基于规则的程序设计，曾经似乎要担当拯救世界的重任，但是并没有成为现实。AspectJ或者其他方面语言中重要的一点在于，我们85%的工作还是在用对象编程。我们解决了对象技术没有解决的问题。当我们从过程走向对象时，我们并没有抛弃过程。来自其他单位的AOP计划也都很小心地没有抛弃以前的东西。我要强调的是，我们并没有也不想把一切推倒重来，我们提供的是一种15%的解决方案。

Software Development: 方面的概念能应用于过程式的语言吗？

Kiczales: 是的。因为过程是有结构的，我们也需要横切这种结构。在不列颠-哥伦比亚大学，我们正在开发用于C的方面，主要就是因为我们能够以此说明在操作系统内核这一性能至关重要的领域中一些很棒的研究成果。

Software Development: 如果我们出于某种原因不想使用方面了，能取消吗？

Kiczales: AspectJ使我们的横切代码非常清晰，可以很容易地手工清除。此外，我们还支持这样的模式："拿类和方面来，可以先把方面插入类，然后还可以把方面去掉。"这可是黄金降落伞，绝对安全。

Software Development: 有些人说AOP使程序更难理解了，你怎么看待这种说法？

Kiczales: AOP肯定能使程序更容易理解，而不是更难。同样，我们必须考虑适用AOP的例子，而不是那些OOP本身已经能够保证良好模块性的例子。还是那句老话，我们不能只见树木，不见森林。

在非AOP程序中，某个关注点比如缓存失效（cache invalidation）[3]的实现代码会分散在各处，很容易看到某一行代码是不是取消了缓存——其中肯定会有对某个取消方法的调用。这里的方法调用就是树木。比较难看到的是森林——缓存失效的全局结构，它能告诉你事情到底是怎么发生的，能够用来解决真正的难题。

在一个AspectJ程序中，对缓存取消的调用不会在执行处的代码中出现，会有简单的IDE支持告诉你，这里有调用。这样我们用简单的工具就实现了"看到森林"的目的。我们还可以循着通知通过查看切入点，清晰地看到全局结构。因此AspectJ程序使复杂的部分——森林变得直观、便于声明和清晰了。而简单的部分——树木能够容易地通过工具获得。

Software Development: 工具对AOP提供支持现在有两种做法：一种是扩展语法，以AspectJ为代表；一种是使用一些XML语法，以AspectWerkz为代表。双方争议还是很多的，对此你怎么看？

Kiczales: 我认为对这个问题的争论有些过头了。我不是说问题本身不重要，但是它对AOP本身没有太多影响。比如，我们还可以很容易地为AspectJ语义设计一套XML语法。AOP本质上与这种语法问题是无关的。

Software Development: 为什么大家会认为语法这么重要呢？

Kiczales: 许多人都不愿意在自己熟悉的语言中加入新东西，这很自然，就像当年面向对象发展的早期，人们宁愿用C来进行面向对象编程。这种倾向从某种意义上来说有好的一面。语言的演进应该持保守谨慎的态度。但我前面说过AOP并不是什么太新的概念，AspectJ的设计就是多年经验和用户反馈的结果。我认为终究有一天语言本身而不是语言之外的框架会支持AOP，但是让大家都接受这一点，需要更多时间。

关于AOP实现方式上的争议，还有一点我想澄清一下。有些人说AspectJ更难学，因为需要学习新语言，而AOP框架则不然。我认为这是错误的看法。两种工具其实都需要使用者学习新的子语言。只不过在AspectJ这样的工具中，子语言是紧密嵌入在Java中的。而在AspectWerkz这样的工具中，子语言是用XML编写的。对于框架，开发者则必须学习框架的API。

同样的问题其实在20世纪80年代也出现过，当时有些人说OOP不应该加到语言里，应该用框架实现。今年春天在TheServerSide网站上对此也有大量的讨论。我认为语言扩展的方案是OOP的成功之道，对于AOP也一样。但是我也相信目前围绕着框架的一系列创新是有价值的，有助于我们进一步探索如何做出设计选择。语言扩展的方案还有一个优点，听上去有些奇怪——它比JBoss等框架方案的功能要弱一些，因为它继承了语言中的一些限制，比如Java语言中的静态类型化。功能稍弱，意味着能够得到更加广泛地使用，因为IDE能够更容易地理解代码的意义，更容易提供支持。

Software Development: AOP和AspectJ现在的进展怎么样？

Kiczales: AOP技术目前正在从发明（invention）阶段向革新（innovation）阶

段发展。发明阶段的特点更多地关注实际的应用，能够比编程技术更快地商业化。我们正在寻找第一个杀手级的AOP应用。许多人认为可能是分布式企业级应用系统。

AspectJ现在是Eclipse项目的一部分了，IBM的Adrian Colyer已经接替我成为领导开发者社区的项目负责人。这将给AspectJ的长远发展提供更稳定的支持。1.1版在今年早些时候发布，比较突出的新功能有增量式编译（incremental compilation），这有助于大型系统的开发；字节码编织，这样就可以使用其他Java编译器了；还有对方面库的支持。IDE支持也有显著改善。Eclipse现在支持方面调试、内联通知注释，和高层的也称"森林式"的视图展示方面对系统的横切状况。面向方面的重构等功能也在开发之中。

除了IBM和Eclipse之外，其他厂商也在支持AspectJ。IntelliJ正在提供早期支持，BEA支持开发者在WebLogic中使用AspectJ。

Software Development: 目前AOP发展面临的主要挑战是什么？

Kiczales: 回答这个问题，OOP的历史，尤其是20世纪80年代的历史是我们的前车之鉴。主要的挑战之一是不要对技术过分宣传。我们要反对AOP能包治百病的说法。AOP是一种模块化技术，能够帮助对领域知识有透彻理解的优秀程序员更快、更清晰地开发程序，更有效地重用代码。当然，并不只限于这些，就像OOP一样，好的模块化技术能够帮助我们逐渐理解涉及的领域。但是AOP决不是万灵丹药，它只是解决一类棘手的软件结构问题的工具而已。

另一个挑战是抵御住在进一步的发展中全部推倒重来的诱惑。后来者很容易为了特殊的技术或者市场因素而偏离原来的道路，重新进行设计。整个社区有效地沟通，是非常重要的，当然也很困难。OO社区在这方面做得不错。因此我仿效OOPSLA发起了每年一次的AOSD会议（Aspect-Oriented Software Development，http://aosd.net/conference）。明年春天召开的会议将有IBM负责开发和软件技术的副总裁Danny Sabbah的主题发言。因为IBM在AOP方面已经投入了大量资金，这次发言非常值得期待。

对AOP社区还有一点非常关键，大家现在更应该做的是一致对外，而不是互相竞争。这是OOP的成功经验。每次OOPSLA会议上，来自不同语言阵营的人齐心协力，共同探讨所有OO语言的共性问题，如何在总体上改善OOP。

Software Development:开发人员是否应该现在就开始了解或者使用AOP了呢？

Kiczales: 我认为任何想要熟悉最新技术的开发者，都毫无疑问应该开始研究AOP了。我们从面向对象技术的历史可以知道，行动越早，获益越大。AOP和OOP一样，其真正价值就是在整个产品架构和开发过程中运用方面的思想。抢先行动的公司将领先同行。同时，我并不鼓励冒失地全盘转换到AOP。面向对象技术的发展历史也告诉我们，正确地控制采用新技术的进度对于成功至关重要。

工程篇

Software Development: 你准备怎样推广这项技术呢？

Kiczales: 我们对技术的接受规律颇有研究，我们努力工作的任务之一就是将AOP这种革命性的概念放入一种渐进发展的包装中去。通常，当人们有了一种新的程序设计范型时，他们还需要创造一种全新的语言。但是我们用Java实现了AOP，这样一个Java程序员能够在15分钟之内成为一名面向方面的程序员。我们采取的是一种循序渐进的策略。

Software Development: 你认为AOP会以一种循序渐进的方式进入主流吗？

Kiczales: 循序渐进是非常关键的。开发者

图3 2000年初，PARC的AOP小组，从左至右：Gregor Kiczales、Jim Hugunin、Crista Lopes、Bill Griswold和Mik Kersten。