第7辑

荟 萃 全 球 软 件 智 慧

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

软件研发

【特别策划】项目管理

- 理解项目的文化
- 项目管理的经验与教训
- 如何与管理层合作

访谈/ 群贤纵论兵器谱

大师论道/ Ivar Jacobson: AOP 与用例（上）

安全实验室/ 红队程序安全测试

【专 题】

C++程序设计

C++ 程序员的丰盛大餐：

编译器标准兼容性测评，Boost，C++&.NET 委托与事件，元模板编程，大三律……

in association with
Dr. Dobb's JOURNAL
C/C++ Users Journal
software development
windows developer
独家授权中文版

建模 算法 安全 嵌入式 算法 C++ .NET XML

杂拌儿

北京旧时过大年，无论贫富，家家都要预备一种混合糖果，大体有：瓜条、青梅、蜜枣、麻片、寸金糖、小开口笑等等，这就是所谓“杂拌儿”。不但色彩鲜艳，吃起来亦又香、又甜、又脆。俞平伯先生曾出过两种杂文集，名字就叫《杂拌儿》。

——《旧京逸闻录》

今年过节期间，我在一份经济类报纸上读到关于微软向中国政府承诺62亿人民币合作计划落实情况的一篇报道。其中谈到，按承诺三年内微软应该提供1亿美元的软件外包合同，但是“和其他承诺的推进情况相比，软件外包方面的进度的确要缓慢得多。”文中把原因归咎于“中国软件产业的现状”，因为中国企业目前还没有赢得国际信任，“不敢相信中国外包商是美国企业的通常思路”。试想如果人家许诺给的钱我们还没有办法挣到手的话，又怎么去大江大河中与如狼似虎的强大对手竞争呢？如何改变现状，同志们还须努力。

※ ※ ※

这个月的主题是C/C++。最近一期的《C/C++ Users Journal》杂志上，P. J. Plauger的社论取名为“变换的季节”。他用轻松的笔调谈起C和C++标准委员会最近的一些工作。由于C99标准目前只有一个前端和一个库完全实现（均来自Edison Design Group），广大用户才刚刚开始得到遵守标准的产品，C标准委员会决定，并同时请求上级机构ISO的SC22委员会同意，再等待几年时间，现有标准将继续适用，不会很快修订。当然，这并不意味着C语言的发展已经停顿，委员会将按照当年C99开发中Numerical C Extensions Group的形式组织社区，开发非标准化的技术报告（TR），解决各种专门问题。委员会已经完成了Embedded TR和另一个处理Unicode问题的TR。刚刚开始的三个TR包括：微软提议的在标准C库函数之外，添加更加安全的版本；IBM提议的增加对IEEE 754R的支持；以及特殊的数学函数，包括贝塞尔函数，这同时也是C++新启动的TR目标。而更加有意思的是，Plauger曾经开玩笑说在C中加入贝塞尔函数是蠢到极致的典范。

C++标准委员会则更加雄心勃勃，在Bjarne Stroustrup的领导下，Evolution子委员会正在开足马力，全速前进。主要的工作当然是C++0X，但是从Bjarne最新的报告来看，X很可能等于7，甚至更大。委员会已经采纳了C99中所有预处理器方面的特性。Performance TR的工作也已完成，它包含了C语言的Embedded TR。

有一点肯定是令大家都感到高兴的，两个委员会已经达到共识，将更加通力合作，互通有无，坚持在兼容的道路上走下去。

※ ※ ※

春节后刚上班，ddjchina.com的新闻编辑就到我这里诉苦，说是自己熬夜抢到的关于Borland CTO跳槽的头条新闻（据说是国内媒体中首家报道），第二天就被某知名开发网站几乎不加修改地刊载了，而且未写出处。这使我联想到新出的国内某知名软件技术杂志上，赫然以某位国人老兄的名字刊登Ramnivas Laddad AOP大作的译文，而且还是一个堂而皇之的系列。继而联想到几年前居然有所谓国内专家把James Coplien的名著《Advanced C++ Programming Styles and Idioms》翻译出来当成自己的原创作品在某家顶级大学出版社出版，而且居然几年后又变成此书中文版的译者再来挣一次钱！还有更多使用盗版软件来开发软件的兄弟们……

《计算机世界》上谈到中国软件业为何做不大的时候，把文化环境看作重要的因素（http://www.ccw.com.cn/news2/zl/htm2004/20040212_09IPS.asp），我深以为然。如果我们这些依靠知识产权吃饭的人，自己都如此漠视甚至践踏我们赖以生存的规则，那么我们还能对整个产业的前途报多大的指望呢？

2004年2月

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

软件研发

目 录

软件技术

Dr.Dobb's JOURNAL CHINA www.ddjchina.com 软件研发

Feature: C++ Programming

Column

Special: Project Management

Management&Engineering

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7辑／（美）钱德瑞博斯（Chandrabose.s）等著; 赵振平等译. —北京: 机械工业出版社，2004.2
ISBN 7-111-14039-7
Ⅰ.D… Ⅱ.①钱… ②赵… Ⅲ.软件研发
Ⅳ.TP311.52
中国版本图书馆CIP数据核字（2004）第013689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：2004年2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

行，成立几个委员会——需求、架构和计划，来指导项目开发。

Eclipse是三年前IBM向开源社区捐献平台源代码而成立的，如今已经拥有50多个会员公司，主持着4个开源项目、19个子项目的开发。新成立的董事会由策略开发商、策略用户（此两类包括爱立信、HP、IBM、Intel、MontaVista、QNX、SAP和Serena等知名公司）、插件提供商（包括Borland、Novell、Oracle、WindRiver等）和开源项目负责人组成。

当然，Sun公司最终没有加入，未能实现两个并行的开源项目Eclipse和NetBeans的合并，的确令人遗憾。

同日到5日举行的EclipseCon大会盛况空前。开源社区的大碗Cygnus公司创始人，Red Hat公司的CTO Michael Tiemann在主旨发言中对Eclipse大加赞赏，称项目打破了Java与开源之间一直存在的“种族隔离”。Grady Booch的演讲题目是“From IDE to XDE to CDE”。Erich Gamma和亲密战友John Wiegand合作介绍了Eclipse的开发现状。AOP之父Gregor Kiczales当然也少不了前来助阵，毕竟AspectJ已经成为Eclipse的一个子项目。虽然从参加会议的人数来看，Eclipse本身还不能与Java群体相提并论，但是潜力着实不小。

使用最广泛的操作系统

世界上最流行的操作系统是什么？Windows？UNIX？还是Linux？你大概不会想到在日本市场处于统治地位的标准嵌入式操作系统ITRON。这个操作系统是1989年日本政府推动的一个自主产权项目的一部分，旨在主导学校计算机市场。但是项目随即被美国方面指责为不公平贸易壁垒，最后整个架构重新定位在消费类设备上。

现在，据项目首席架构师Ken Sakamura的估计，ITRON操作系统已经在30亿～40亿个手机、数码相机、传真机等设备中使用，而Windows装机量大约只有1500万台。

按照TRON协会的数字，ITRON控制着嵌入式开发39%的市场，而Linux是11%，VxWorks是7%。

趋势：软件项目成功率有所提高

在业内有很大影响的Standish集团最新版（第10版）CHAOS年度报告最近出炉。Standish集团从1994年开始研究美国IT项目失败的原因，10年中已经研究了超过4万个项目。历史上该报告提供的数据曾经成为众多教科书中软件工程实施必要性和背景的证据。

2004年度的CHAOS报告名为“CHAOS Chronicles（CHAOS大事记）”，研究表明美国项目总共浪费了550亿美元，其中直接损失380亿，成本增加170亿。总的项目花费达到了2550亿美元。

而在1994年，Standish集团估计美国IT项目的浪费金额是1400亿美元，总项目花费是2500亿。

最新的报告说明，10年来软件项目的成功率已经得到了100%左右的提高，所有项目的成功率达到34%（1994年只有16%）。而失败率则大幅降低，从1994年的31%下降到只占15%。

对于项目成功率提高的原因，Standish集团的主席Jim Johnson说：“主要原因是项目与以前相比要小得多。相对于传统的瀑布型方法要求所有项目需求预先定义，使用迭代式的过程来管理项目，实际上是分解了项目，这是一大进步。”

“人们对项目管理的认识要深入多了，”Jim Johnson说。“[相比10年之前]我们开始此项研究的时候，项目管理还是一种绝技。”

但是在今年的调查中，所谓“被质疑（challenged）”的项目（超期、超预算和/或没有满足关键的功能或者需求）所占比例是51%，依然很高，相比其他行业更是不可同日而语，说明要达到更高水平仍是任重而道远。今年的调查中，大多数此类项目超过预算在20%以内，比1994年有很大改进。2004年在所有成本超预算的项目（包括失败项目）中，平均超支为43%。

而在2001年CHAOS报告中，三类项目（成功、失败、被质疑）的比例分别为：28%，23%和49%。

2003年，世界软件业谁执牛耳

一年一度《Software》杂志评选的软件500强（含软件产品和服务）最近出炉，排名前15位的是：

公司	收入（百万美元）	收入变化
IBM	49 434.00	3.2%
Microsoft	28 365.00	12.1%
EDS	21 502.00	1.7%
艾森哲	11 574.00	1.1%
Computer Sciences	11 426.00	8.6%
HP	10 185.80	25.1%
Oracle	9673.00	-10.9%
日立	8207.50	2.5%
SAP	7687.30	19.1%
安永	7389.00	-16.3%
NTT Data	6040.70	-24.6%
Unisys	4285.00	-7.5%
Sun	3998.70	-0.4%
Atos Origin	3189.30	18.5%

从整体来看，2003年世界软件业呈下滑的趋势，500强总收入从2002年的3010.8亿美元跌至2003年的2890.7亿美元，相对比率达到了4%，为历年来少见。雇员总数更是减少到2173.941，比去年剧减了18%（因此应用开发工具厂商普遍不景气）。从领域来看，财务软件和安全软件、协作/项目管理软件增幅较大。

我们熟悉的公司中，冠群电脑排名是17位，EMC是21位，仁科是23位，Siebel是26位，VERITAS是29位，Compuware是30位，赛门铁克、VerySign、苹果、SAS、Adobe、Synopsys和Novell分列33～39位，BEA 42位，Sybase 47位，Autodesk 49位，CheckPoint 71位，趋势科技 77位，Macromedia 84位，Borland 105位，RSA 108位，SCO排名232位。

（更多新闻和评论，请访问www.ddjchina.com）

2002年图灵奖背后的故事（下）

上回说到，闲云野鹤般的Whitfield Diffie、独辟蹊径的Ralph Merkle在独具慧眼的Martin Hellman帮助下找到了绝世妙法，解决了困扰人类两千年多年的信息共享加密问题。但是，这只是公钥密码学传奇的开始。有人曾经开玩笑说，公钥密码学实际上是由一组密码似的字母发明的：DHMRSAECW。怎么讲？且听小史慢慢道来。

陷门单向函数之战

论文“New Directions in Cryptography”在《IEEE Transactions on Information Theory》上的发表，标志着公钥密码学大厦已然奠基。论文简洁地指出：“只要能找到一个合适的带陷门的单向算法，就能为通信双方都生成两套密钥，一套用于加密，一套用于解密。加密的那一套可以是共享的、公开的，解密的那一套则是保密的。由于单向算法的性质，从公开的密钥是很难逆向破解私钥的。数字签名也能通过这种方式由甲方用公钥加密，然后由乙方用私钥解密。这样就既解决了加密问题，又解决了数字签名问题。”Diffie在论文的开始豪迈地宣称：“今天，我们正处在密码学革命的边缘。”

然而，只有概念，没有实例证明是不行的。问题的关键，集中到寻找符合要求的陷门单向函数/算法上。其实，这一问题，计算机学界早已经有了不少求解思路：

- 同在斯坦福大学电机系的John Gill在Hellman的求援下，提出了一种离散取幂法，因为其逆运算——离散对数是很难求出的。
- Aho，Hopcroft和Ullman合著的《The Design and Analysis of Computer Algorithms》一书NP完全问题一章中提出了可用于此目的的背包算法。
- Donald Knuth的巨著中提到，素数乘法很容易，但是求素数因子却非常难。

很快人们就发现，这三种算法正好代表了三个主要的公钥加密方法流派。

首先取得突破的还是Diffie所在的研究小组。一年后，1976年5月一天清晨，Diffie正在撰写论文，Hellman突然打电话告诉他：“我取得了进展，就按Gill的办法，使用幂和对数就可以，形式非常简单！”这就是历史上发表的第一个公钥算法。

此时，Ralph Merkle也加入了他们的研究团队。原来，Diffie和Hellman 1972年底的论文在正式发表前就以预印本形式寄给了许多同行。多年碰壁的Merkle很快读到这篇论文，他意识到，自己苦苦等待的伯乐和知音终于出现了，他很快与Diffie和Hellman取得了联系。Hellman再次显示出自己的伯乐本色，他看出Merkle的潜力，很快就找到资金支持，邀请他到斯坦福大学做博士研究。

当年晚些时候，Merkle开始了采用背包算法构造陷门单向函数的研究。背包问题的思路，是对许多重量各一的物品，通过秘密选择其中一部分，放到背包中，背包总重量是公开的，所有物品也是公开的，但是背包中物品是保密的。他的论文于1978年9月发表，并在此前取得了专利（专利号为4,218,582）。因为导师是Hellman，所以这种算法又称为Merkel-Hellman背包算法。Merkle对自己的算法极有信心，他在办公室贴了一张纸条，声称如果有人攻破自己的背包系统（即使是单迭代），他将奖励100美元。这可是公然向天下高手下战书。世界之大，卧虎

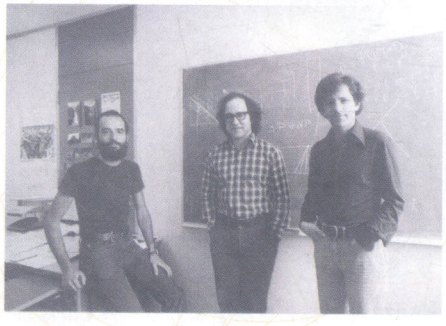
1977年的公钥密码学三剑客（从左至右）：Ralph Merkle、Martin Hellman和Whitfield Diffie

藏龙，当然有不信邪的。Crypto'82会议上，一位来自MIT的科学家破解了单迭代的背包算法，他就是当时世界一流的破解高手Len Adleman。Merkle的信心并没有为此动摇，他甚至在《时代》杂志上登出广告，以1000美元悬赏多迭代背包系统的破解方案。两年以后，来自俄亥俄州立大学的博士生Ernie Brickell不仅得到了这1000美元，还彻底毁灭了Merkle的信心：他在一台Cray-1超级计算机上证明，可以在一个小时内破解出40次迭代、100种重量的背包系统。

此后20多年中，各国科学家根据各种思路设计了数以百计的加密算法。然而，事实上的胜出者却是在论文"New Directions in Cryptography"发表后不久诞生的RSA。而RSA——Ronald L. Rivest、Adi Shamir和Leonard M. Adleman也正是依靠此项发明，不仅创办了颇为成功的安全技术公司，而且摘得了2002年度的计算机界最高荣誉阿兰·图灵奖。呵呵，经典的名利双收。

MIT三少年

三位图灵奖得主的名字都不算好记（注意到Adi Shamir中东味道十足没有？因为他是一位犹太人），我们就用他们姓氏的首字母来称呼好了（方便大家，亏了我一个——稿费显然要少了），三个字母加起来——RSA，正是他们借以扬名立万乃至如今功成名就的算法名称，也是他们创办的如今已经成为业界翘楚的RSA Security公司（最初创办的公司名为RSA Data Security，在被Security Dynamics并购后改为现名）名字的由来。



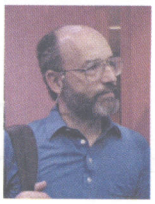
当年的翩翩三少年（从左至右：S，R，A）

1976年，斯坦福科学家石破天惊的论文"New Directions in Cryptography"发表后，关注此领域的科学家们纷纷开始转而沿着论文指出的"新方向"前进——寻找陷门函数的圣杯。在东部的另一个计算机研究重镇MIT计算机实验室，刚从斯坦福大学博士毕业不久的青年教员R凭借着自己深厚的数学功底，决心在此一展身手。他并不孤独，有一位来自以色列的数学专家S志同道合。S是以色列魏茨曼学院的教师，当时利用休假来MIT客座。他们把战斗室就设在技术广场545号8楼。

因为数学上证明一个加密算法的安全性还不可能，而证明其不安全就容易多了——找到破解方法。所以有人说，如果不懂得如何破解密码的话，就不可能设计加密系统。R和S的组合方式恰恰印证了这一点，他们工作的时候，往往是R提出一种加密算法，然后由S来思考是否有破解之法。当主攻方向转移到数论方法上时，他们的工作逐渐遇到了瓶颈：两人在关键技术——数论算法复杂度方面都涉猎有限，因此他们决定在校园里招募一个此道中高手。天赐贵人。与他们年纪相仿的A此时刚刚在伯克利算法大师图灵奖得主Manuel Blum门下完成了自己的博士论文"Number Theoretic Aspects of Computational Complexity"，来到MIT的数学系担任讲师，他的研究方向正是数论算法复杂度。更巧的是，A还是小圈子

Ronald L. Rivest, Adi Shamir, Leonard M. Adleman小档案

Ronald L. Rivest博士（以下简称"R"）是麻省理工学院电气工程与计算机系计算机科学Viterbi讲席教授。ACM会士，美国艺术与科学院院士，美国国家工程院院士，国际密码学研究协会理事。主要研究方向是密码学、计算机与网络安全、算法，还从事过机器学习和VLSI设计方面的研究。他从耶鲁大学获得数学学士学位，在斯坦福大学获得计算机科学博士学位。他是算法名著《Introduction to Algorithms（算法导论）》（影印版已由高等教育出版社出版）一书的作者之一，《Dr. Dobb's Journal》的撰稿人。1997年他与CP/M操作系统之父已故的Gary Kildall共同获得了"Dr. Dobb's程序设计杰出奖(Excellence in Programming Award)"。除了RSA Security公司外，他还创办了开发微电子支付技术的公司Peppercoin。



Adi Shamir（"S"）博士现在是以色列魏茨曼理学院应用数学系Borman讲席教授。他在特拉维夫大学取得数学学位，魏茨曼学院计算机科学博士。除了RSA之外，他还在差异密码分析（Differential Cryptanalysis）方面造诣颇深。

Leonard M. Adleman（"A"）是南加州大学计算机科学Henry Salvatori讲席杰出教授和分子生物学教授。美国国家工程院院士。他从加利福尼亚大学伯克利分校获得数学学士学位和计算机科学博士学位。除密码学之外，他还是"计算机病毒"一词现代意义的发明者。他将数学运用于艾滋病研究，取得了独特的成果。偶然的一次机会，他发现聚合酶"读取"DNA的方式与图灵机的原理极为相似，产生了构建DNA计算机的革命性想法。他目前仍然在为此奋斗。

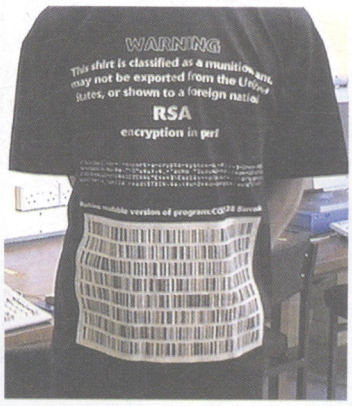
里闻名的密码破解高手。用A自己后来的话说："我是在正确的时间出现在正确的地点，而且还拥有正确的知识。"于是，S把主要的破解重任交给了A，自己除了继续承担一部分破解工作之外，还和R一起设计新算法。

就这样，R和S设计一个算法，A就破解一个。有的只需要几分钟，而有的可能要花费几天——R和S虽然沮丧，毕竟每个算法都牺牲了自己不少脑细胞！但同时也不得不心存佩服：高手就是高手。就这样，斗转星移，季节更替，转眼已经到了1977年的4月。春天的麻省校园，风光旖旎。连日奋战而且饱受身心摧残的设计者R却病倒了，也难怪，他和S在过去的一年中一共设计了差不多42种算法，可是都没有过A这道鬼门关。R得了严重的感冒，咳嗽，而且发烧，卧床不起。迷糊中，他突然想起大师Donald Knuth指引的道路——素数分解来。正向运算：将素数乘起来何其容易，而将一个大数分解为素数之积，却是超级数学难题呀！他从床上一跳而起，冲到战斗室，冲向A。A立即意识到这可能真的是一个正解。接下来更严谨的分析表明，破解这种算法，凭借当时的硬件能力，可能需要花费数年。

有趣的是，RSA的突破性进展最早并不是发表在学术刊物上的。1977年8月，《科学美国人》杂志的著名专栏作家马丁·加德纳在他极受欢迎的专栏"数学游戏"中刊登了他们的成果。其中，RSA还声明，可以为任何感兴趣的人提供完整的报告。美国国家安全局意识到了算法的力量，对其自由发布感到害怕，于是要求RSA停止自己的公布行动。可是，这是没有法律依据的。RSA按照自己科学家的法则继续我行我素，他们于1978年2月在《Communications of the ACM》上发表了一篇更详细的论文，于是算法在全世界都开始传播。

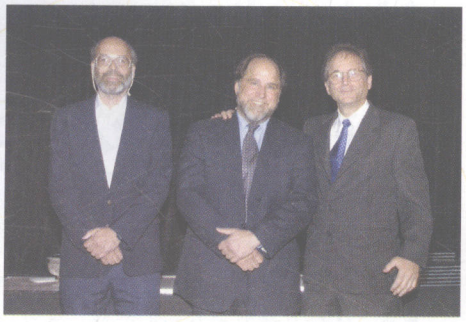
随后的故事就是大家熟知的了。三位都是30岁出头的年轻学者立即获得了空前的学术声誉。同时，MIT在技术转化方面也比斯坦福更胜一筹：学校很快为RSA的发明注册了美国专利（专利号为4405829），由于事先公开，许多国家的专利已经无法注册。1982年，三个人到加州Redwood注册了RSA Data Security公司。此专利虽然已经于2000年9月到期，但MIT和公司显然都已经收获颇丰。

关于RSA还有一些有趣的插曲，美国政府和民间围绕RSA算法公开与限制的斗争持续了很多年。这一点成了许多黑客的好题材。最有趣的是，Adam Back曾经只用一个5行的Perl程序就实现了这个算法，而美国政府也一如既往地发出了禁止出口的命令。网上网下开始热闹起来：为了抗议政府的越权行为，有人将这段代码作为电子邮件签名四处散发，有的则打印在T恤上、作为花纹刺成文身——这大概是程序最具流行色彩的经历了。



印有RSA算法程序的T恤

而故事的主角们并没有继续合作很长时间，S当然返回祖国继续教学，A于1980年去了南加州大学，开始更多的传奇故事，与RSA算法的开发过程中，他严格意义上讲只是一个受雇者，而不是提出问题者，相比他在艾滋病和DNA计算机方面的研究则完全是开拓性的。R是惟一一个仍然留在MIT的，而且在RSA公司中任职时间也最长。然而，在2003年接受图灵奖的演讲中，三个老友又显示了默契的合作，一个综述、一个回顾、一个展望，20多年前的三剑客仿佛又回来了。


在图灵奖颁奖典礼上（从左到右：S，R，A）

湮没的历史

随着Internet的兴起，公钥密码学的应用越来越广泛。这项发明足以让美国人感到自豪。然而到了1997年12月，英国人却拿出了证据，说明自己才是公钥密码学这项伟大发明的最初创造者，而且把历史向前推到了20世纪60年代。这到底是怎么回事呢？

原来，在布莱尔工党政府上台后，发起了"政府开放"行动，要求更多政府部门将政务公开，其中就包括历史文件。英国负责监听监视工作的GCHQ（政府通信总部，Government Communication Headquarters）在网站上贴出了前GCHQ雇员James Ellis于1987年撰写的回忆文章"The Story Of Non-Secret Encryption（非密加密术的故事）"。文中，Ellis说自己早在1960年代就受贝尔实验室的一篇报告（发表于1944年）启发，构思了称为非密加密术（NSE）的设计，并于1970年发表在内部报告上，具体的方案是采用查询表，与Diffie-Hellman最初的方案一样，这一方案只有理论价值。

【下转32页】

访　谈：

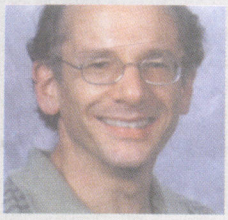
群贤纵论兵器谱

有人说工具最多只能与使用它们的手相提并论，但我们都知道它们可能决定项目的成败。从行为准则到基本的实践，从白板到UML，我们的业界名人就自己喜爱的工具畅所欲言。

Amit Asaravala

从简单的锄头、扫帚，到未来的编程语言，是工具使人类成为生态圈最高等的生物——我们依靠工具辅助我们的各种工作，从劳累的体力劳动到深邃的思辨。最近，我们对一些代表着业界最高智慧的名人进行了飞行采访（有的不超过一分钟），他们谈到了自己对开发工具的偏好和对软件开发未来的一些想法。其中的一些观点可能会令你大吃一惊。

Stan Lippman



Lippman 在贝尔实验室供职超过10年，曾与Bjarne Stroustrup一起工作，开发cfront（Stroustrup的C++编译器实现）的早期版本。Lippman担任过《C++ Report》杂志的主编，撰写了几本影响很大的C++图书，包括《C++ Primer》（与Josée Lajoie合著，Addison-Wesley, 1998），已经出版到第三版，以及《Inside the C++ Object Model》（Addison-Wesley, 1996）。在梦工厂和沃尔特·迪斯尼动画公司之后，他于2001年加入微软任Visual C++架构师，从事托管C++的开发工作。

在梦工厂和沃尔特·迪斯尼，你使用的是什么工具，和现在在微软有什么不同呢？

在差不多整个20世纪90年代，SGI是电影业和航空业高端图形的通用平台，但是SGI提供的工具最开始差得惊人。你能想像得到吗，SGI的C++编译器还是老版本的cfront，也就是Stroustrup的编译器，我后来曾担任这个产品的主程序员。

在Unix世界中，无论工作在什么应用领域，只要你已经有了一定岁数，主要的文本编辑器都是Emacs。但是更老一些的人，比如我自己，都是从使用Bill Joy的vi开始的。它目前仍然是我们这些老派人物编辑器的首选，虽然这总是会被那些Emacs用户所嘲笑，他们喜欢不知疲倦地表明Emacs的优越性。Emacs当然是一个很棒的程序，但是我用vi感觉更舒服，效率也更高。我认为，和芭蕾舞和网球一样，文本编辑器需要从比较早的时候就开始训练。现在在微软，我使用的是一种内部版本的vi，虽然Visual Studio内置的编辑器我也很喜欢。

请问，所有项目都能从图形化开发工具中获益吗？

一般而言，软件的优势既体现在能够使原来需要繁重手工劳动的过程自动化上，也体现在减少进行工作和评估结果之间的不连续上（比如所见即所得就能很好地提高生产率）。图形化更容易解决后一个问题。因此，像调试器或者分析器这样的开发工具会因为图形化得到极大地改进，而将编译器或者文本编辑器图形化，所得就有限了，至少对于现在而言是这样。

除了C++，你还使用其他什么语言，或者说个人比较感兴趣？

现在我更感兴趣的不是某种语言，而是微软.NET项目分布式、动态的本质。我怀疑在编程语言有了长足发展，能够更好地体现其表达能力之前，我们是否能完全认识到它的真正潜力。我认为C#和Java都没有达到这样的水平，很不幸，C++也没有。现在我正在与其他开发人员通力合作，力争使C++能够与.NET下的新环境俱进。到那时，我们就让自然选择来发挥优胜劣态的作用好了。

Ted Farrell

在加入Oracle公司担任应用程序开发工具的策略总监和架构师之前，**Ted Farrell** 曾经是WebGain公司的CTO。在办软件公司、设计和构建软件产品的经历中，Farrell与人合办了好几家针对小型企业的软件公司，包括Night Owl Enterprises, Novasoft Systems Inc.和Tendril Software。在Tendril，Farrell领导开发团队构建了StructureBuilder。

在Oracle的工作中你一般使用什么语言和工具？

我使用Oracle JDeveloper和Java, J2EE与XML。我原来用的是vi，很多时候都在编写脚本，后来发现JDeveloper提供的环境正好是我所需要的，因此转向了。

现在什么语言最令你感兴趣？为什么呢？

我是Java和J2xE标准的超级信徒。这里也包含用于数据传输的XML和互操作性的Web服务。由于J2EE架构的健壮性和可伸缩性已经得到了检验，我想没有什么理由还需要考虑其他平台。

Steve McConnell

McConnell 是多本名著《Code Complete: A Practical Handbook of Software Construction》(Microsoft, 1993)、《Rapid Development: Taming Wild Software Schedules》(Microsoft, 1996)和《Professional Software Development》(Addison Wesley, 2003)的作者。1999～2002年期间，他担任《IEEE Software》的主编，1996～1998年是该杂志“Best Practices（最佳实践）”专栏的编辑。目前，他担任着Contrux软件公司的CEO和首席软件工程师。

软件项目中，工具发挥着什么样的作用？

好的工具当然有用，但是我认为有关为什么、什么时候和怎样使用工具的知识更加重要。否则，就会落入“进入是垃圾，出来的还是垃圾（garbage in, garbage out）”这样的怪圈。如果世界冠军老虎伍兹和我一起打高尔夫的话，他的分数一定比我高，就算我使用他的球棍，而他用我的。

对于软件项目而言，成功=95%的技术+5%工具支持。35年前，在著名的“IBM Chief Programmer Team（首席程序员小组）”项目中，Harlan Mills[①]在一年内编写了大约8万行高质量的产品代码。与今天的程序员使用的工具相比，那个项目的工具支持水平显然是非常原始的，但是这位软件工程的先驱达到的个人生产率是今天最好的程序员所能达到的四五倍。

当然了，即使如此，老虎伍兹还是会喜欢用自己的球棍，而不是我的，而软件专业人士同样也会发现好的工具所能带来的好处。

在计划、编程、编译和调试等等工作中你一般使用什么工具？

我经常要做一些非常特殊的工作，因此使用的工具往往是通用的Word、Excel和Access，它们能适合我的特殊需求。我认为人们对工具能做什么过于关心了，其实更重要的是我们想用工具来干什么。当我使用Excel这样的工具来计划一个项目时，我可以完全把注意力都集中在要做的事情上。

我们运用工具的成功范例是版本控制。我们把所有项目文档和所有公司的文档都放进了版本控制系统中。我们的经理人员、销售和市场人员、管理人员和所有其他雇员都知道如何使用这个系统。

在过去的《IEEE Software》专栏文章中，作为主编，你曾经写道，缺乏培训是软件创新的一大障碍。那么有什么工具能缩短学习曲线吗？

我认为与其花费太多时间寻求缩短学习曲线的工具，还不如寻找办法减少学习中时间和精力上的浪费现象。

软件开发是一个年轻的领域，开发人员还没有非常确定的职业发展通道。大多数开发人员的职业生涯都由从事一系列的项目和学习一系列技术组成。我认为缺乏职业发展通道是缺乏培训的部分原因——很多开发人员都认为培训主要是为了完成当前项目的需要，而不是为了持之以恒地提高自己专业能力。因此，培训所带来的好处是非常短暂的，无法形成积淀。

Eve L. Maler

作为《Developing SGML DTDs: From Text to Model to Markup》(Prentice Hall PTR, 1996)一书的合著者（另一作者是Jeanne El Andaloussi），**Maler**是W3C（万维网联盟）XML工作组的创始成员，并一直负责Sun公司在W3C的所有活动。作为Sun公司的XML标准架构师，她长期专门研究安全和B2B方面的标准与词汇。Maler拥有Brandeis大学语言学学位。

许多人发现DTD（文档类型定义）很难阅读和编写。有什么工具能够用来处理DTD吗？

过去我在设计工作中使用纸和笔，配合“榆树图(elm tree diagram)”[②]和在《Developing SGML DTDs: From Text to Model to Markup》书中介绍的方法。具体实现是用普通的文本编辑器。但是我现在已经逐渐改用XML Spy[③]了，这样将结果图形化和生成引用文档都比较方便。

创建文档的时候，我经常使用的是Arbortext公司出品的Epic。我自己大多数的文档写作都是用XML完成的，一般是XHTML、DocBook或者XMLspec DTD。W3C标准的许多编辑都使用XMLspec。

译注1：Harlan Mills（1919～1996）是杰出的计算机科学家，其贡献遍及计算机学术界和工业界。他是IBM名士，以创造了净室软件工程和首席程序员组而闻名于世。

译注2：“榆树图”是XML系列标准中常用的图形记号。

译注3：XMLSpy是Altova公司出品的XML IDE，屡获好评。

Guido van Rossum

20世纪90年代在荷兰国家数学和计算机科学研究院期间，**van Rossum**创造了Python，直至今天仍然负责着语言的开发和维护。1995年他来到美国，在CNRI进行研究工作。此后加入了Zope公司，担任Python核心开发组PythonLabs的负责人。2003年7月，他加盟由因开发Satan、Titan等安全工具而蜚声开发社区的Dan Farmer新近创建的始创公司Elemental Security。他是《Dr.Dobb's Journal》英文版特邀编辑，1998年度"Dr.Dobb's程序设计杰出奖"得主。2002年他被自由软件基金会授予"自由软件促进奖"。

你一般使用什么Python开发工具？

编写Python程序的时候，我只使用几种工具：XEmacs，Barry Warsaw的python-mode.el用来编辑源代码，Neal Norwitz的PyChecker用来在不运行的情况下检查Python代码中的错误。

在Windows上，我用IDLE编辑和运行Python程序。这是一个Python发行版中附带的简单IDE。现在SourceForge上还有另一个IDLEfork项目，目的是给IDLE添加许多激动人心的新功能。

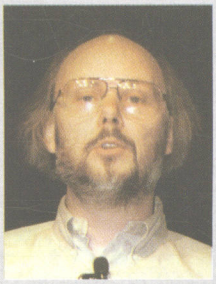
如果一个开发人员找不到满足自己项目需求的语言，你是否鼓励他自己创造一个呢？

创造一个完整的语言，其复杂性是难以置信的——比大多数人开始时能够想像的要复杂得多。从我自己的亲身经验来看，这是千真万确的。

但是，创造能把一件事情干好而并不想包治百病的小语言（minilanguage）就容易多了。尤其是声明性的语言非常有用。不同意？配置文件通常就是声明性语言编写的小程序。

如果找不到一种语言能够编写所有代码，应该考虑用两三种不同的现有语言编写各个构件，然后使用一种构件机制或者其他更简单的方案将各部分粘起来。像Python这样的语言就能满足这一目的，其作用就是"胶水语言"。许多成功的Python项目中，Python语言的作用都是胶水。但是我必须强调，还有很多项目Python是主要的实现语言，比如Zope就是。

Bjarne Stroustrup



在1979年获得了剑桥大学的计算机科学博士学位后，**Stroustrup**创造了C++语言，并担任着AT&T实验室大规模编程研究部门的负责人。除了撰写了5本书之外，Stroustrup还是在效率要求极高的应用领域使用面向对象和泛型技术的先驱。他目前是Texas A&M大学工程院的计算机科学教授。

对于应用程序的设计，建模工具的重要性如何？

如果你指的"建模工具"是运行在计算机上的东西，那我几乎从来没有用过。我使用分析器生成程序（parser generator）来检查语法二义性，但是这很专，相对而言比较少用。我大多数的设计都是与同事讨论的过程中在黑板上完成的，然后图形以纸面形式经过多次迭代，最后变成代码。

我发现许多高层的设计决策不容易用图形的方式表达（比如"内存资源不能泄漏"），而许多底层的决策用代码表达最直接。

平常你使用哪些编译器和调试器？

因为大多数工作都与C++有关，而且对可移植性非常重视，我喜欢使用几种C++编译器，比如Borland C++ Builder，GNU C++ [gcc]，Microsoft [Visual] C++，Metrowerks C++和SGI C++。

我愿意根据对标准的遵守程度、标准库的质量来选择编译器。如果对标准的遵守都能够接受的话，我喜欢有优秀优化器的编译器。我强调对标准的遵守——即对正式和公开开发的各种标准绅如ISO C++标准）的遵守，因为它们是保护用户不受厂商控制的关键之一。

我很少使用调试器，我更喜欢直接去看栈轨迹（stack trace）。

如果一个开发人员找不到满足自己项目需求的语言，你是否鼓励他自己创造一个呢？

除非是非常专门的特定领域语言，我不会鼓励他自己再创造一个。通用语言的成功率微乎其微，而那些成功的先例也都花费了10年甚至更长的时间。很少有项目值得承担这么大的风险，付出这么大的努力。

今天有许多优秀的语言，很少有项目不能通过现有的几种语言，或者语言的结合清晰而且低成本地完成。

Richard Stallman

Richard Stallman 开发了第一个Emacs文本编辑器，并在1984年发起了GNU计划，其宗旨是“给计算机用户他们大多数人已经能够丧失的自由”。Stallman还创建了自由软件基金会(FSF)，发起了一场全球性的运动。目前，他依然执著地守护着地道黑客纯洁的精神家园，为自由软件和开源软件之间的区别进行不懈地辩护。

你怎样选择自己使用的工具？

评价开发工具或者任何其他软件的时候，我主要不是根据技术质量来判断。我当然知道程序的技术有优劣，但是我最关心的是程序是否尊重用户的自由。如果程序不是自由软件，如果我作为用户没有研究和改变源代码并进行重新发布的自由，那么这种程序就是一种毒药，我会毫不犹豫地拒绝它。如果我不能和你共享程序，我就不会接受任何副本，这是一种关乎大义的原则。

如果将用户自由作为选择工具的主要标准的话，那你曾经遇到过这样的情况吗，所选的工具技术上不具备你所需要的所有功能？

在这种情况下，我有两种坚守自由的方式：要么使用现有的自由软件，要么开发一个新的自由程序来完成任务。

事实上我遇到过这种情况。1983年，当时几乎还没有什么自由软件，也没有用于现代计算机之上的自由操作系统。因此，只要你使用计算机，就会陷入这种尴尬境地。为了解决这一问题，我开始开发GNU操作系统。因为没有自由的编辑器可以使用，我编写了GNU Emacs；因为没有好的自由C语言编译器，我写了GCC；因为没有自由的调试器，我写了GDB。

有没有什么情况下，你认为开发者使用不遵守自由软件哲学的工具是可以接受的呢？

只有一种情况可以使用非自由软件：为这个软件开发自由替代品。例如，我们需要使用Unix来启动GNU计划，这是合情合理的，因为GNU是Unix的一种自由替代品。

你自己日常使用哪些工具？

我主要使用的工具是GCC和GDB。版本控制我使用CVS。在Emacs的开发中这些程序都非常有用，现在我进行编程基本上都是在Emacs里。

Blake Stone

作为原来JBuilder产品线的架构师和Borland的首席科学家，**Stone**在大规模Java开发方面有丰富的经验，而且直接接触过许多使用Java技术的公司的各种项目。在采访后不久，就传出了他辞职离开Borland的消息。

你一般都使用什么设计和开发产品？

我仍然执著地相信在设计过程中使用白板、3x5卡片和其他不太技术化的工具。在项目的计划中过早地痴迷于形式化地收集各种想法，是一种常见的陷阱。只有想法开始固化，变更过程减缓下来的时候，我们才能用更结构化的图形来表示各种事物。即使在这个阶段，我仍然愿意将UML看成是交流思想而非表达实现的一种方式。

有几个基本工具我每天都用。CVS是可靠解决方案的优秀范例，虽然它已经渐露老态。我愿意原谅它的一些局限，因为大多数时间这并不影响我们的工作。[Kent Beck和Erich Gamma的]JUnit测试框架则是另一种有益的经历。它并不试图解决所有能够想像得到的问题，它只解决能解决的问题，而且把这一点做好。

我一直感觉对于复杂问题，太完美的解决方案往往存在软肋。我认为NeXTstep的Distributed Object和[Recursion的]Voyager看起来都是分布式计算模型的真正答案。它们都关注设计的相对透明性，但是最终它们却会被更加完整但是不那么优雅的标准所淘汰。

在Java之外，现在你还对其他哪些编程语言感兴趣？

编程语言最近使我产生了浓厚兴趣，这部分是因为我相信“按契约编程”不为广大开发人员熟悉时间太长了。还有UnrealScript，虽然它还说不上是一种通用语言，但在分布式编程模型方面却有一些真正聪明的想法。是的，我自己仍然在寻找“下一个大家伙”(the next big thing)。我衷心希望会有新的语言产生，解决今天最有影响力的语言中最明显的问题：

- 比C++的“友员”更好的可视性解决方案，避免临时的API定义。
- 多层次的语言方言，可以向不同层次的编程人员公开系统编程、构件构造和脚本编写功能。
- 语言能具有运行时规范，明确地避免阻碍动态编译器的发展。

大师论道：AOP与用例（上）

Ivar Jacobson

面向方面程序设计可能最终会超越代码层次的构造，实现模型真正的横切可视性。我们做到这一点花了25年——但是现在，我们处在变革的前沿，一种强大的新范型将把方面和用例联系在一起

Ivar Jacobson 是Objectory方法（后来发展为RUP）的创造者，UML的创造者（与Grady Booch和James Rumbaugh一起）。在瑞典爱立信公司，他在大规模重用方面工作的顶峰是建立在一种类UML语言之上的基于构件的方法。他撰写的书包括《Object-Oriented Software Engineering: A Use Case Driven Approach》（与Magnus Christerson、Patrik Jonsson和Gunnar Overgaard合著，Addison-Wesley, 1992），《The Object Advantage: Business Process Reengineering with Object Technology》（与Maria Ericcson和Agneta Jacobson合著，Addison-Wesley, 1995）和《Software Reuse: Architecture, Process and Organization for Business Success》（与Martin Griss和Patrik Jonsson合著，Addison-Wesley, 1997）。他最知名的发明则是用例和用例驱动的开发。

我第一次听说面向方面软件开发（aspect-oriented software development，AOSD）是在OOPSLA'97上，PARC的科学家Gregor Kiczales就此发表了一个主旨演说，当时这还只是研究人员关心的话题，对一线开发人员没有产生太大影响。但自那以后，对这种开发方法的兴趣有了实质性的增长。大多数支持者都在称道“方面”在基础系统中添加横切功能（安全性、日志功能、持久性、调试、跟踪、分布、性能监视）上的价值。这些特性的确都值得赞许。但是，方面这一概念更伟大的用途还在我们前方。

方面将被证明在应用程序的开发中极为强大，我们能够借此在应用程序中，为未来许多年添加新的用户功能或者说用例。我相信，面向方面软件开发的概念将使我们在系统整个生命期的每次迭代、每次发布中优雅地扩展原有软件系统。这将实质性地提高软件的质量，也将对软件的所有传统度量标准：成本、质量和上市时间都产生巨大的影响。

早在1978年，当我还是一位软件架构师的时候，我就说过现在称为面向方面的技术将降低20%以上的生命期总成本。这是一个巨大的改变，说老实话，我认为这个数字其实还有些保守。

因此，我将给出一个多年前所做的案例研究，这可不仅仅是为了子孙后代着想。我当时做出的结论就暗示说明，我们需要编程语言来对现在支撑着面向方面的各种概念提供支持。在本文的第二部分，我将讨论这“缺失的环节”[现在已经在Kiczales的PARC小组（AspectJ）和Harold Oscher的IBM小组（Hyper/J）的工具中实现了] 和用例驱动的开发有着怎样密切的联系，它又是怎样为我们提供一种成熟的AOSD方法的。

内，在所有构件的整体之上。我曾一直在寻找功能模块性，但那是在用例之前——今天，我更愿意称之为用例模块性。为了实现这个梦想，我需要两种机制：

1. ***用例分离机制***。用例的设计目的是将系统分割成多个与使用相关的部分。这对于同级用例（peer use case）来说非常有效：没有哪个用例比其他用例更基本、更必需或者更可选。这样的一个例子是电话通话，这是电信系统中一个基本的用例。它的同级用例——与使用相关的不同部分，包括本地通话、到/来自另一个地区或者另一个国家的通话。

但是，有些用例要依赖其他更基本的用例来工作。为了分离这些用例，我们需要一种扩展机制，能够从一个基础用例开始，然后用更多行为不断地扩展它，以开发（分析、设计、实现、集成和测试）复杂系统——其中的这些行为就称为非干扰（nonintrusive）扩展。我们的目的是通过从一个基础开始组织设计和代码，在与基础无关的语句（即使对于其他行为非常重要）不添加到基础中的情况下，使系统生长，从而获得容易理解的设计和代码。

这种扩展机制应用于用例时，能给我们带来“扩展用例（extension use case）”。在一个包含用例的基础之上，我们添加扩展用例，当组合两个基础时，我们将得到一个新的基础，带有新的扩展用例。通过这种方式，我们可以保证大多数用例在转变为代码，甚至是可执行文件的过程中始终保持分离。

2. ***用例组合机制***。为了使系统能工作，我们需要将多个分片（也就是分离的用例）组合或者集成为一个一致的整体，以获得二进制代码。我们有几种选择：例如在编译前、编译时或者执行时将它们编织起来。将扩展用例与其基础组合起来，如果给定当时我使用的扩展机制，这是相对比较容易做到的。但是，我们还需要将不是通过扩展分离的同级用例组合起来。这里我们必须将有重叠行为的用例组合起来，比如，两个有操作类似但是不相同的同级用例。这代表着一个更复杂也更一般性的问题，因为组合扩展用例只是组合同级用例的一种特殊情况。

在两种机制中，我更重视通过扩展机制分离用例的能力。因为通过它，我们能向下直到构件代码层次都保持用例的分离。构件的开发人员就能将分离的用例集成起来。组合机制通过在开发环境添加一个预编译器就能非常简单地引入。而且，从我的经验来看，即使是这么小的技术改变就能获得巨大的回报：许多新的功能只需要设计上简单的扩展，这些扩展可以用极为简单的方式测试——我们因此能节约更多成本。

所以，我的工作集中在用例分离机制和用例组合机制上，而复杂得多的同级用例组合的研究工作就得留待将来了。

最初的扩展

为了解释这一概念，我使用了下面的例子。斜体显示的文字（除了少量修改之外），直接引用自我1979年发表的论文“Use Case Modularity”（用例模块性）。这里我用术语“用例”代替了“功能”，而“参考点（reference point）”就是现在的“扩展点（extension point）”。括号中是一些简单的解释，而不相关的文字就用省略号代替了。

我们的……语言构造必须补充明确改变“控制流程”的能力。我们将用一个例子说明这一点；首先，是我们今天[指1979年]的做法，然后是未来可能的进展：

例子：假设我们有两个用例，“通话处理”和“流量记录”[“通话处理”用例实例在电话用户将电话从支钩上拿起来的时候调用：将有一个离钩信号从电话传给用例实例。用例实例检查是否请求了流量记录。如果是，就递增计数器。这个计数器在通话进行时递增，当通话终止时随即停止递增。下一个操作是“连接接收器”，“发送拨号音”给正在拨打的用户，对于本例而言，无需考虑更多了。

另一个用例是“流量记录”，其目的是测量一段时间内（比如说15分钟）来自用户的平均流量。为此，它有两个流程，图1中只显示了一个。首先，每10微秒它都要访问系统中的通话计数器集合。计算递增了的数目，然后用计数器的总数去除它——结果就是在10微秒的时间内流量的情况。记住这个数字，在15分钟的时间内反复操作，类似地得出其他数字。另一个流程将计算周期内的平均流量。]

你可以认为前一个用例是独立于后一个用例的，但是反过来就不正确了。不过，要构造“通话处理”（用例），还必须了解“流量记录”用例的知识。

[图1是说明上述思考结果的一个非常简化的图。]

在“通话处理”用例中，我们必须包含与流量记录相关的用例部分。这是很糟糕的，用例应该独立。

通过使用简单的技术……这种情况是可以避免的。(参见图2)

这样，我们就引入了一种可能性，能够没有二义性地引用另一个用例描述，并改变一个用例的控制流程。"通话处理"和"流量记录"现在可以用用例模块性来描述了。

以下来自同一篇论文:

在用例描述中的一条语句执行之前（已经编译成目标代码），我们假设微程序会（同时）检查外层中的另一个用例是否引用了当前的指令地址。如果是这样的话，当前用例描述的执行将被中断，而执行引用方用例描述所插入的序列。

我为这种概念的微可编程实现申请了专利（它没有被批准，原因等一下解释）。因为一个大的扩展类可以安全地引入，无需干扰基础，对于这个类而言，回归测试在有正确的工具支持下就不再需要了。这将有望大大节约测试所需要的人力和成本。

为了实现这种机制，我引入了几个简单的构造:

* 可以用来无二义性地引用另一个用例描述（在描述中使用之前（*before*）或者之后（*after*）声明）的*扩展点*（*extension point*）。术语"描述（*description*）"的意思是一张图或者一段代码。

* 两个语句*insert at* <扩展点> 和 *continue at*，可以无需在基础中说明，就能扩展一张图或者一段代码。

在论文"Use Case Modularity"中，我写道："用来实现这种概念的技术，从原理上将令人吃惊地简单：在运行时编辑。"我认识到，这一提议正是一种新技术的发端，而此提议一旦得到接受，将会按自己的轨迹继续发展。

几年后的1986年（"Language Support for Changeable Large Real Time Systems." Proceedings of OOPSLA'86，1986年8月），我将扩展的概念一般化，引入了一个新词"*存在*（*existion*）"作为对象的原有集合（即基础）。以下文字直接引用自这篇演讲词（有一些不重要的修改——术语"探针（*probe*）"被术语"扩展点"取代了）。

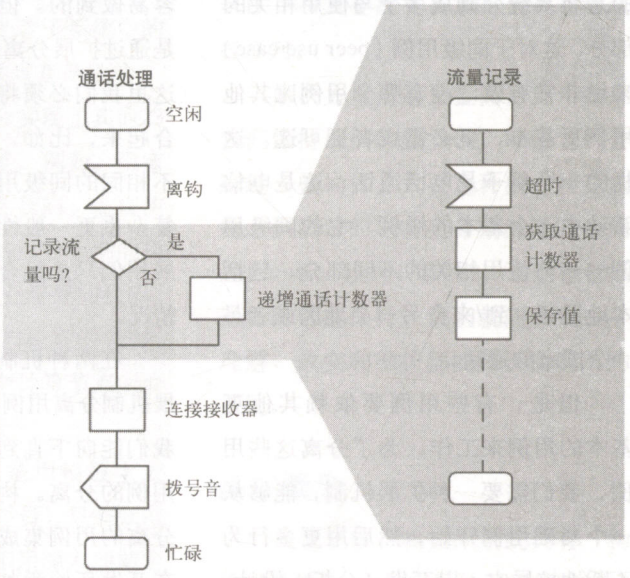
在存在和扩展之间只有两种关系:

** 扩展要么无需访问存在的权限，要么只有读访问权限。*

** 扩展需要有来自存在的控制权限（也就是说，扩展将使附加指令执行），但不能改变存在。*

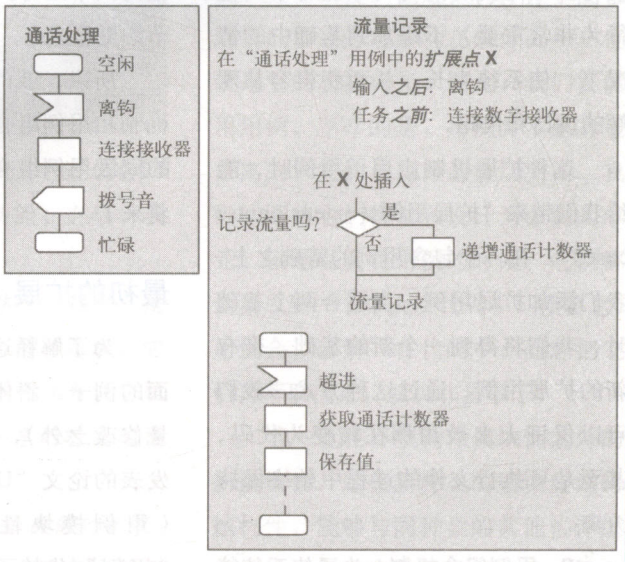
前一种情况意味着扩展可以使用对象实例现有的操作，只要这些操作不改变对象实例的状态。第二种情况意味着，虽然存在的执行是不可分割的，但是扩展可以在某些指定的点进行干预。当扩展执行后，控制将返回给存在让它继续执行。在存在执行当中，可以有一个以

图1 用例的混合



"通话处理"用例无法忽视"流量记录"用例所请求的活动（比如"递增通话计数器"）。图中所用符号是开发AXE产品时采用的。1976年这些符号融入了SDL，现在则成为UML 2.0中的元素。

图2 关注点的分离



通过一种简单的技术——用扩展和扩展点分离关注点，"通话处理"用例能够不再理会"流量记录"用例了。