敏捷开发方法及一个非典型应用实例

林 海 徐晓飞 潘金贵
(南京大学软件新技术国家重点实验室 南京210093)

摘 要 敏捷(agile)软件开发方法是近几年来新兴的一种软件开发方法,它的主要特征是允许对过程进行自主调整,并且强调软件开发中人的因素,和传统开发方法有着很多不同。本文对这一新的软件开发方法作一些简单的讨论,简要介绍了其中有代表性的几种方法,并给出了自己的一个应用实例。

关键词 Agile 方法,软件工程,适应性,面向人,XP

Agile Software Development and an Atipical Case

LIN Hai XU Xiao-Fei PAN Jin-Gui
(State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210093)

Abstract Agile methodology, which is adaptive and people-first oriented, is one of the new methodologies in software development. This paper will discuss why we need agile and its main characteristics, with some popular methods that fit under this agile banner being shown. And an atypical exercise in which agile method is used will be given at last.

Keywords Agile methodology, Software engineering, Adaptive, People-first orientation, XP

1 引言

讨论软件开发,首先需要明确软件开发的含义。目前有几种代表性的观点:①数学观,以 CAR Hoare 为代表;②工程观,以 Bertrand Meyer 为代表;③艺术/工艺观,几乎所有富有个性的程序员都这么认为。而本文陈述的软件思想认为:"编程本身是一种个体的,富有灵感的、逻辑性强的活动,但是现代的软件开发更是一种群体活动[1]。"

对软件的不同理解引出不同的软件开发方法,但目前最常见的开发方法不外乎两种方式:一种是"边写边改"(code and fix),另外一种是所谓的"正规方法"(methodology)[2]。对于第一种方法,设计过程往往没有统一的、完整的规划,在小系统开发时还可以应付,但是当系统变得越来越复杂的时候,就变得难以控制,无法保证软件开发的效率和质量。第二种方法和第一种则正好相反,它对开发中的各个步骤都有严格而详尽的规定。它借鉴了其他工程领域的实践理论并应用到软件开发上来。这种软件工程的方法取得了一些效果,但是过于官僚繁琐,影响软件开发的效率,因此有些人称此种方法为重型(monumental)方法(Jim Highsmith 语)。

和上面两种方法不同,最近几年出现了一类新的方法,英文叫做 Agile method,中文一般译为敏捷型,轻灵型,或者是轻载型方法,本文采用第一种译法,即敏捷型方法。"这类方法的诱人之处在于对繁文缛节的官僚过程的反叛。它们在无过程和过于繁琐的过程中达到了一种平衡,使得能以不多的步骤过程获取较满意的结果[2]。"

2 敏捷方法的特点

敏捷型方法主要有两个特点,这也是其区别于其他方法,尤其是重型方法的最主要的特征:

1、敏捷型方法是"适应性"(Adaptive)而非"预设性"(Predictive)的。重型方法试图对一个软件开发项目在很长的时间跨度内做出详细的计划,然后依计划进行开发。这类方法在计划制定完成后拒绝变化。而敏捷型方法则欢迎变化。其实,它的目的就是成为适应变化的过程,甚至能允许改变自身来适应变化。

2、敏捷型方法是"面向人"的(people-oriented)而非"面向过程"(process-oriented)的。它们试图使软件开发工作能够利用人的特点,充分发挥人的创造能力。它们强调软件开发应当是一项愉快的活动。

下面我们将对上面两点做详细的解释。

2.1 适应性和预设性

传统的软件开发方法的基本思路一般是从其他工程领域借鉴而来的,比如土木工程等。在这类工程实践中,通常非常强调施工前的设计规划。只要图纸设计得合理并考虑充分,施工队伍可以完全遵照图纸顺利建造,并且可以很方便地把图纸划分为许多更小的部分交给不同的施工人员分别完成。

但是,软件开发与上面的土木工程有着显著的不同。软件的设计是难以实现的,并且需要昂贵的有创造性的人员。土木工程师在设计时所使用的模型是基于多年的工程实践的,而且一些设计上的关键部分都是建立于坚实的数学分析之上。而在软件设计中,完全没有类似的基础。我们对开发计划所能做的只是请专家审阅。这就使得我们无法将设计和实施分离开来,一些设计错误只能在编码和测试时才能发现。根本无法做出一个交给程序员就能直接编码的软件设计。

所以,软件过程不可能照搬其他工程领域原有的方法,需要有适应其特点的新的开发方法。

软件的设计之所以难以实现,问题在于软件需求的不稳定,从而导致软件过程的不可预测。但是传统的控制项目的模式都是针对于可预测的环境的,在不可预测的环境下,就无法使用这些方法。

但是我们必须对这样的过程进行控制,以使得整个过程能向我们期望的目标前进。于是 Agile 方法引入"适应性"方

法，该方法使用反馈机制对不可预测过程进行控制。

下图分别是 XP 方法(Extreme Programming，Agile 方法的一种)(图1)和"进化式"开发方法的模型(图2)[2,6]，从图中可以看到，从需求分析开始，这两种方法省略了常规的系统和架构的设计步骤，从简单的计划直接进入编码阶段，并进行迭代循环。整个过程中编码和设计不存在显著的先后关系，很多时候是同时进行，互为因果。本文后面介绍的实例中也是采用的这种类似的迭代方法。

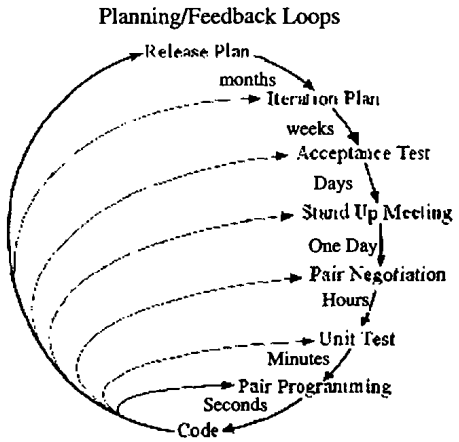

图1 XP 方法模型

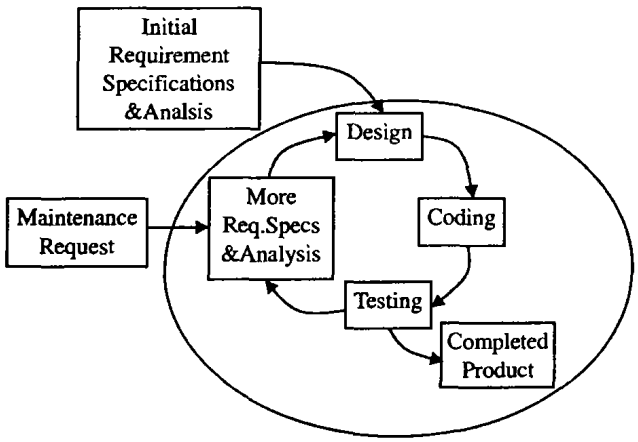

图2 "进化式"方法模型

2.2 面向人而非面向过程

"实施一个适配性过程并不容易，它要求一组高效的开发人员。高效既体现在高素质的个体，也体现在有能让团队协调一致的工作方式。"[2]

传统正规方法的目标之一是发展出这样一种过程，使得一个项目的参与人员成为可替代的部件。这样的一种过程将人看成是一种资源，他们具有不同的角色，如分析员，程序员，测试员及管理人员。个体是不重要的，只有角色才是重要的。这样考虑的一个重要的出发点就是：尽量减少人的因素对开发过程的影响。但是敏捷型方法则正好相反。

著名软件开发专家 Alistair Cockburn 在 1994 年曾经提出了人是"非线性"的观点。这种非线性表现在，给一个人双倍的输入(如双倍的时间，或双倍的报酬诱惑)，你不能期待着他就能做到双倍的输出(双倍的思考质量，或双倍的代码行)[1]。

传统方法是让开发人员"服从"一个过程而非"接受"一个过程。但是一个常见的情况是：软件的开发过程是由管理人员决定的，但是管理人员往往已经脱离实际开发活动相当长的时间了，如此设计出来的开发过程是难以为开发人员所接受的。

敏捷型过程还要求开发人员必须有权作技术方面的所有决定。IT 行业和其他行业不同，其技术变化速度非常之快。今天的新技术可能几年后就过时了。只有在第一线的开发人员才真正掌握和理解开发过程中的技术细节。所以技术方面的决定必须由他们来作出，这样一来，就使得开发人员和管理人员在一个软件项目的领导方面有同等的地位，他们共同对整个软件开发过程负责。

敏捷方法特别强调开发中相关人员之间的信息交流。Alistair Cockburn 在对数十个项目的案例调查分析后得出一个结论，"项目失败的原因最终都可追溯到信息没有及时准确地传递到应该接受它的人"。在开发过程中，项目的需求是在不断变化的，管理人员之间、开发人员之间以及管理人员和开发人员之间，都必须不断地了解这些变化，对这些变化作出反应，并实施在随后的开发过程中。敏捷方法还特别提倡直接的面对面的交流。Alistair Cockburn 认为面对面交流的成本要远远低于文档交流的成本，因此，敏捷方法一般都按照高内聚、松耦合的原则将项目组划分为若干小组，以增加沟通，提高敏捷性及应变能力。

3 主要敏捷方法简介

目前已经出现很多敏捷型方法，它们有许多的共同特征，但也有一些重要的不同之处。这里就其中影响比较大的几种敏捷方法作一些简单的介绍。

1. XP(Extreme Programming——极限编程) 在所有的敏捷型方法中，XP 是最为引人瞩目的。它源于 Smalltalk 圈子，特别是 Kent Beck 和 Ward Cunningham 在20世纪80年代末的密切合作。通过在一些对费用控制严格的公司中的使用，已经被证明是非常有效的。

XP 有四条基本价值原则：交流，简易，反馈和勇气。在此基础上建立了十多条 XP 项目应遵循的实践准则。XP 还有一个最突出的特点，就是它对测试极端重视。XP 的设计过程是"纪律性"与"适配性"的高度统一，这也使得 XP 在适配性方法中成为发展得最好的一种方法。

2. Cockburn 的水晶系列方法 水晶系列方法是由 Alistair Cockburn 提出的。它与 XP 方法一样，都有以人为中心的理念，但在实践上有所不同。Alistair 考虑到人们一般很难严格遵循一个纪律约束很强的过程，因此，与 XP 的高度纪律性不同，Alistair 探索了用最少纪律约束而仍能成功的方法，从而在产出效率与易于运作上达到一种平衡。也就是说，虽然水晶系列不如 XP 那样的产出效率，但会有更多的人能够接受并遵循它。

3. 开放式源码 这里提到的开放式源码指的是开放源码界所用的一种运作方式。开放式源码项目有一个特别之处，就是程序开发人员在地域上分布很广。这使得它和其他敏捷方法不同，因为一般的敏捷方法都强调项目组成员在同一地点工作。开放源码的一个突出特点就是查错排障(debug)的高度并行性，任何人发现了错误都可将改正源码的"补丁"文件发给维护者。然后由维护者将这些"补丁"或是新增的代码并入源码库。

4. SCRUM SCRUM 在 OO 界里已经出现很久了，像前面所论及的方法一样，该方法强调这样一个事实，即明确定义了的可重复的方法过程只限于在明确定义了的可重复的环境中，为明确定义了的可重复的人员所用，去解决明确定义了

的可重复的问题。

SCRUM 强调迭代阶段计划与进度跟踪。它与其他敏捷型方法在许多方面都很相似，特别是它可以与 XP 的编程准则很好地结合起来。

5. Coad 的功用驱动开发方法（FDD——Feature Driven Development） FDD 是由 Jeff De Luca 和 OO 大师 Peter Coad 提出来的。像其他方法一样，它致力于短时的迭代阶段和可见可用的功能。在 FDD 中，一个迭代周期一般是两周。

在 FDD 中，编程开发人员分成两类：首席程序员和“类”程序员（class owner）。首席程序员是最富有经验的开发人员，他们是项目的协调者、设计者和指导者，而“类”程序员则主要作源码编写。

6. ASD 方法及其他敏捷方法 ASD（Adaptive Software Development）方法由 Jim Highsmith 提出，其核心是三个非线性的、重迭的开发阶段：猜测，合作与学习。

其他的敏捷方法还有 Lean Development，Pragmatic Programming 等。

4 敏捷方法的一个非典型应用

4.1 敏捷方法的局限及应用范围

虽然敏捷方法在对不确定过程的控制上做得非常出色，但是它也有自己的局限：

1. 缺乏对分布式环境的支持；
2. 缺乏对转包的支持；
3. 缺乏对产生可重用积累的支持；
4. 缺乏对大 group 开发的支持；
5. 缺乏对 safety-critical 软件的支持；
6. 缺乏对大而复杂的软件开发的支持。

基于上面这些局限，一般说来，敏捷方法适用于小规模过程。对于一些需求不够明确、时间要求紧的项目，使用敏捷方法也是不错的选择。

4.2 应用实例

该应用是日本富士通（FUJITSU）公司将其产品 Collaboration Ring 从 SOLARIS 平台移植到 Linux 平台。

Collaboration Ring 是富士通公司的电子商务中间件产品，该项目需求明确，规模大（源代码共三百多兆），原则上并不适用敏捷方法。但是考虑到该项目为移植项目，需要的人员（一共五个开发人员）及实际的代码量并不多，而且需要进行频繁的编译调试过程。这些却非常适合用敏捷方法来解决，所以最终我们采用在传统方法中结合敏捷过程来进行开发。本例并非一个典型的敏捷方法案例，但是有着一定的普遍性和指导意义。

考虑到移植项目的特点，将整个项目分成两个主要阶段，采用三环迭代的方式（参考图3的工作流程图）。

第一个阶段主要是解决代码的编译问题。其目标是要在 Linux 平台上编译通过 Solaris 上开发的源代码。首先需要对源代码进行一些处理，包括代码的编码修改，编译选项的修改，格式转换，环境变量等等。然后就是对代码进行编译，并根据结果排除编译中的错误，重新进行编译或者进行代码的处理，直到没有错误产生。

第二个阶段是为了保证移植后代码能够正确运行，并得到和 Solaris 版本同样的结果。首先，需要根据产品结构安装上一阶段中编译出来的目标程序，配置相应的运行环境。然后对软件进行测试，设计构造通过测试案例，并对其中发现的问题进行解决。如果是安装或者测试中出现的问题，则需要再次运行调试，或者重新安装产品。

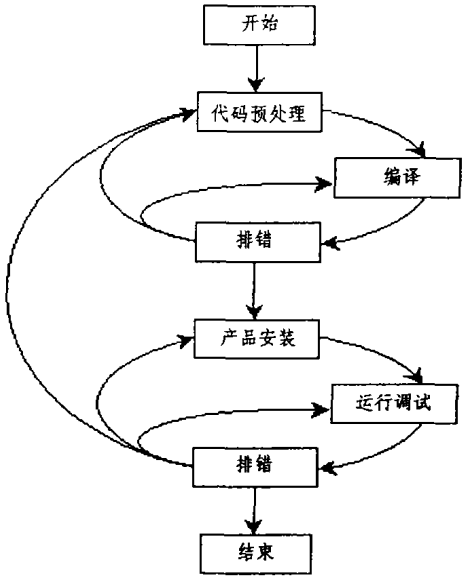

图3 项目工作流程

上面两个阶段都分别形成一个迭代的环状结构。同时，在第二阶段测试中如果发现问题是由第一阶段的移植过程中产生的，则需要重新回到第一阶段开始迭代。这样就形成第三个迭代环。当三个迭代环都停止迭代的时候，也就是项目的最终结束。

这样的三环迭代结构就使得任何一个过程中发现的问题，都可以很快地进行反馈，在下一轮迭代中产生影响。虽然敏捷方法中使用反馈的主要目的是为了适应需求的变化，但是在本项目中，使用该方法适应编译和调试过程中的各种错误，也起到了相当好的效果。原本代码编译部分计划至少需要两个月完成，最终只用了一个半月的时间，效率提高了25%。

上面谈到的都是有关项目的结构，其中主要是应用了敏捷方法中加强反馈的思想。而敏捷方法中的另一个主要思想，以人为本，在项目的人员安排，相互协作上，也起到了非常好的效果。

前面提到，项目一共由五个人完成。但是其中第一个阶段，完全是由三位在日本开发的组员完成的。由于代码多，人员少，我们没有采用很多敏捷方法中使用的“结对编程”的方式。但是充分调动了组员之间，开发人员和发注方之间的积极的充分的交流和沟通。由于使用迭代结构，每次迭代循环中，组员都会将各自发现的问题以最简单的 EXCEL 表格形式通知其他组员，使得这些发现不仅能反馈到自己的迭代过程中，同时也反馈到其他组员的工作中。小组还制定了频繁的小结制度。每天上午和下午项目组成员都要聚在一起进行一个简短的小结，将各自负责部分的问题，方法，难点等进行沟通。在任何时候如果发现严重的，或者可能对组员产生影响的问题，都可以立即召集所有组员进行交流。但是每次交流都要求简短扼要，时间最多不超过半个小时，这样可以在保证交流充分的条件下尽量减少因小结带来的工作停顿。同时，由于该项目代码复杂，其中的很多模块也是由发注方的不同部门完成的，因此在编译过程中经常发现有模块的缺失或者是不配合。这就需要和发注方进行沟通，项目第一阶段在日本进行，也给这样的沟通提供了可能。虽然有着语言障碍，但是基本上每次问题的反馈，都可以在一到三天内完成，这也是整个项目效率提高的一个重要保证。

在项目控制方面，我们使用了一些简单有效的工具。对于代码的管理和 BUG 跟踪，使用了 CVS 和 bugzilla。组员从 CVS 获取待移植的代码，提交修改过的代码，并且可以对不同组员之间的修改进行合并。同时对发现了的问题登记到 bugzilla 中，跟踪该问题的发现过程，影响范围，解决方法及最终状态。为了保证修改的完整性，我们要求每个组员每天最终提交的代码都必须是可以正确编译的，不能含有正在修改中的代码。对于其中缺失或者确实无法编译的部分，我们编写了替代(dummy)程序来保证它的编译通过。这样就可以保证任何一个时刻从 CVS 中取出的代码都是正确的、完整的而且是可以编译的代码。

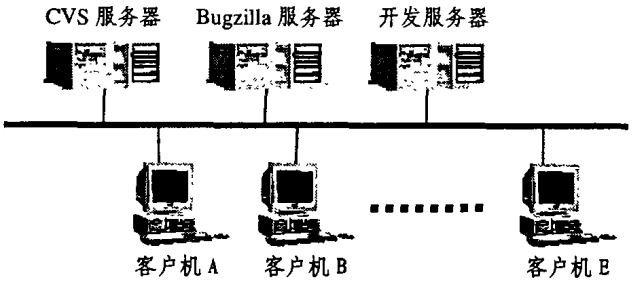

图4　开发使用的网络环境

图4是该项目使用的网络环境，CVS 服务器中包含所有源代码、编译环境及各种脚本。开发服务器是项目开发的平台。组员通过客户机连接到开发服务器上，从 CVS 服务器上 check out 需要修改的代码进行移植，并通过脚本进行编译。同时对编译中发现的问题记录到 bugzilla 服务器。由于使用三台独立的服务器，这样在任何一台服务器出现问题时，都可以根据另外两台服务器上的内容重建损坏的环境。在项目中我们曾经就因为硬盘损坏导致 CVS 服务器中所有最新代码丢失，损失了约30天的工作。在这样的情况下，我们利用原始代码，并根据 bugzilla 服务器上的记录，只用了3天的时间就将代码恢复到最新状态，并在开发服务器上重新部署。

测试也是敏捷方法非常重视的一个方面。很多敏捷方法都建议甚至要求有快速的、自动化的测试方法。在我们的项目中，这样的测试尤其重要。因为项目庞大，涉及的功能多且复杂，手工测试非常耗时而且容易出错。为此，我们编写了相当多的脚本，实现测试的自动化。甚至在第一阶段，我们也使用了大量的脚本来修改一些代码中的公共移植点。这些脚本使得我们可以简单地通过一条指令就可以从编译，安装开始直到测试产品中的每个功能，大大加快了迭代的速度。可以在每个小的修改后就进行一次彻底的测试，从而尽量避免同时进行多处修改时修改之间的交叉影响。另外，还有一些脚本可以完成对代码信息的抽取。可以根据需要从代码中抽取出某日某位组员的所有修改，便于对修改的确认和讨论。

下面是名为 install.sh 的脚本的部分内容，该脚本完成对整个产品的重新安装和配置：

```
……
    RPM_FILE_NAME_FJSVcrpa  = "/FJSVcrpa/FJSVcrpa-5.0-
      1.i586.rpm"
    RPM_FILE_NAME_FJSVcrakb  = "/FJSVcrakb/FJSVcrakb-
      5.0-1.i586.rpm"
    RPM_FILE_NAME_FJSVcrakc  = "/FJSVcrakb/FJSVcrakc-
      5.0-1.i586.rpm"
    if[ $TARGET = $NAME_CRF ]
    then
     if [ "$SE_INST" = "NO" ]
     then
      PKG_DIR="$MOUNT_CDROM/CRF/EBF"
      PKG_RPM_FILE="$PKG_DIR""$RPM_FILE_NAME_
        FJSVebf"
      echo BIN_BASEDIR="$BIN_BASEDIR" > $INST_DIR/
        $PKG/work/crpm.resp
      echo ETC_BASEDIR="$ETC_BASEDIR" >> $INST_
        DIR/$PKG/work/crpm.resp
      echo $PKG_DIR
      RPM_INSTALL_CMD "$PKG_RPM_FILE"
……
```

而下面是一个可以完成代码信息抽取的脚本 recExtrator.sh 的部分内容：

```
……
exRec()
{
for i in `find . /\( -name '*.h' -o -name '*.H' -o -name '*.c' -
    o -name '*.cpp' -o -name '*.idl' -o -name '*.inl' -o -name
    '*.cph' -o -name '*.java' \) -print`; do
  if[-f $i ]; then
    if [ "$EXDATE" != "" ]; then
      grep "$EXDATE" $i >/dev/null
      retofgrep=$?
      if [ $retofgrep -ne 0 ]; then
        continue
      fi
    fi
    ret =`awk 'BEGIN{flag=0}/\/\/\\$/{if(flag==0)
      {flag=1;print flag;}}' $i`
    if[ -n "$ret" ]; then
      echo $i
      awk 'BEGIN{isprint=0;count=0;print"##########
           ##########################
           ##########";}\
           $1~/\/\/\$/{isprint=1}\
           {if(isprint==1){print NR,$0;}}\
           $1~/^ifdef/{if(isprint==1){\
                 count++;}\
               }\
           $1~/^ifndef/{if(isprint==1){\
                 count++;}\
               }\
           $1~/\#endif/{if(isprint==1){\
                 count--;\
                 if(count<=0){ isprint=0;print
                                 "----------";}\
               }}\
           /\/\/\@\#/{isprint=0;count=0;print
                                 "----------";}' $i
    fi
  fi
  done
  }
  ……
```

最后，给出一个比较表(其中不使用敏捷方法的数值是根据以前类似项目的数据进行统计得出的)。

表1　敏捷方法和普通方法效率对照表

比较项 / 方法	总体比较			单项比较		
	移植需要的总时间(周)	测试需要总时间(周)	代码质量(bugs /100Kline)	编译一次代码需要时间(小时)	进行一次产品安装需要时间(小时)	进行一次产品测试需要时间(小时)
不使用敏捷方法	8	10	< 10	1.5	1	10
使用敏捷方法	6	7	< 5	0.75	0.25	6
效率提高百分比	25%	30%	50%	50%	75%	40%

从表1的统计可以看出，使用敏捷方法后，项目的单项和整体效率都有了明显的提高，代码质量也提高了很多。在今后的实践中，我们将会在更多的项目中尝试使用敏捷方法。

(下转第132页)

件开发周期各个阶段从关注分离角度需要考虑的问题以及现阶段的研究现状。可以看出，关注分离问题与需求工程之间的关系甚密。虽然在不同的开发背景，不同的阶段，从不同的角度，关注的内容不同，但是需求阶段概念建模对于后继阶段的关注以及交叉关注的识别具有直接的作用。此外尚未有一个统一的模型，能够为软件整个生命周期的关注分离提供支持，并且实现在软件的各个阶段之间能够平滑地过渡。

参 考 文 献

1 Dijkstra E W. A Discipline of Programming. Prentice Hall, Englewood Cliffs, NJ, 1976
2 Kiczales G, et al. Aspect -oriented programming. European Conf. on Object Oriented Programming, Finland. Springer-Verlag LNCS1241, June 1997
3 Tarr P, Ossher H, Harriso W, et al. N degree of separation: Multi-dimensional Separation of concerns. In: proc. of the 21st Intl. Conf. on Software Engineering, May 1999. 107～119
4 IEEE. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std 1471-2000. Approved 21 Sep. 2000. 4
5 Sutton S M Jr, Rouvellou I. Modeling of Software Concerns in Cosmos. In: 1st Intl. Conf. on aspect-Oriented Software Development, Enschede, The Netherlands, April, 2002. 127～133
6 TRESE group. Composing Multiple Concerns Using Composition Filters. ttp://trese. cs. utwente. nl/publications/papers/CF_superimposition_bergmans_aksit. pdf
7 Brito I, Moreira A, Araújo J. A requirements model for quality attributes in all early aspect paper . AOSD, 2002
8 Dingwall-Smith A, Finkelstein A. From Requirements to Monitors by way of Aspects. In: Proc. of AOSD2002
9 Sutton S M Jr. Early-Stage Concern Modeling in all early aspect paper AOSD, 2002
10 Wagelaar D. A Concept-Based Approach for Early Aspect Modelling. In: Proc. of AOSD, 2003
11 Tekinerdogan B. ASAAM: Aspectual Software Architecture Analysis Method in Workshop on Early Aspects. Aspect-Oriented Requirements Engineering and Architecture Design of AOSD, 2003
12 Peri T, et al. N degree of separation: Multi-dimensional Separation of concerns. In: Proc. of the 21st Intl. Conf. on Software Engineering, May 1999. 107～119
13 Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W G. An Overview of AspectJ. In: ECOOP, 2001
14 Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W G. Getting Started with AspectJ. CACM, 2001, 44(10): 59～65
15 Hanenberg S, Bachmendo B, Unland R. An Object Model for General-Purpose Aspect Languages, GCSE. 2001, LNCS, 2186: 80～91
16 Liberherr K, Orleans D, Ovlinger J. Aspect-Oriented Programming with Adaptive Methods. CACM, 2001, 44(10): 39～41
17 http://www. ccs. neu. edu/home/lieber/
18 Hyperspace Web site. http://www. research. ibm. com/hyperspace
19 Chun-Carroll M C, Wright J, Ying A T T. Visual Separation of Concerns through Multidimensional Program Storage. In: Proc. of the 2nd Intl. Conf. on AOSD, Boston, MA, USA, 2003
20 曹东刚，梅宏. 面向 Aspect 的程序设计——种新的编程范型. 计算机科学，2003，30(9)：5～10
21 Aspect browser. On http://www-cse. ucsd. edu/users/wgg/Software/AB/
22 Hannemann J, Kiczales G. Overcoming the Prevalent Decomposition of Legacy Code. In: Workshop on Advanced Separation of Concerns at the Intl. Conf. on Software Engineering (ICSE), 2001 in Toronto
23 Iwamoto M, Zhao J. Refactoring Aspect-Oriented Programs. In: 4th AOSD Modeling With UML Workshop, UML'2003, San Francisco, California, USA, Oct. 2003
24 Miller G. Refactoring with Aspects in XP. LNCS 2675, 2003. 343～346
25 Rashid A. A hybrid Approach to separation of concerns: The Story of SADES in REFLECTION 2001 LNCS 2192, Springer-Verlag Berlin Heidelberg, 2001. 231～249
26 IBM Concern Manipulation Environment On. http://www. research. ibm. com/cme/
27 Demeter L K J. http://www. ccs. neu. edu/research/demeter/index. html
28 Lieberherr K J, Silva-Lepe I, Xiao C. Adaptive object-oriented programming using graph-based customization. CACM, May 1994, 37(5): 94～101
29 Harrison W H, Ossher H L, Tarr P L. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. IBM Watson Research Center RC22685 (W0212-147), Dec. 2002
30 Aldrich J. Challenge Problems for Separation of Concerns. In: Proc. of the OOPSLA 2000 Workshop on Advanced Separation of Concerns, Oct. 2000

(上接第128页)

结束语 目前软件业竞争激烈，而且机会稍纵即逝，如何利用有限的资源进行最有效、快捷的开发，就成了其中的焦点问题。本文介绍的敏捷开发方法是众多方法中比较有效的一种，尤其是对中小型项目的开发。但是它也有着自己的一些局限。我们在使用敏捷方法开发的过程中也有一些教训。比如最开始我们希望使用结对编程，并且要求每半天就要提交一次可编译的代码。结果发现组员的负担相当大，大量时间耗费在保证代码可编译性上。后来将时间调整为每天一次，而且取消了结对编程，改为小结交流，效率明显提高了很多。如何保证开发中类似的用于项目控制的开销"刚刚好"，是应用敏捷方法需要考虑的重要问题。

敏捷方法对需求不确定或常常变更的情形是有效的。如果要采用适应性方法，你就需要信任你的开发人员，并让他们参与(技术)决策。适应性过程的成功依赖于你对你的开发人员的信任。如果你认为你的开发人员素质不够，那么就应采用传统的预设性方法。

敏捷方法一般适用于小规模过程，比如 XP 最好用于10人以下，水晶方法最多也只能管理50人左右的项目组。如果项目组规模较大，则敏捷方法恐怕不适合。

总之，没有哪一种开放方法是适用于所有项目开发的，敏捷方法给传统软件开发带来了一种新的思路和开发模式，但也有其适用范围，在实际开发过程中，需要根据实际项目的需要来选择合适的开发方法，并尽最大可能发挥人的创造性和潜能，利用不同人的不同特点，充分沟通，这才是我们应该从敏捷方法中真正要学习的。

参 考 文 献

1 Cockburn A. Agile Software Development. AddisonWesley, 2001
2 Flower M. New Methodology. http://www. martinfowler. com/articles/newMethodology. html, 2003
3 http://www. agilealliance. org, 2003
4 钱乐秋，张敬周，朱三元. Agile 方法研究综述. http://www. iturls. com/Articles/doc/Agile_Summarize. pdf, 2003
5 周芝英. 现代软件工程(第二版)第一册. 科学出版社，1999
6 http://www. agilechina. org, 2003