极限编程中的知识创造

杨烜会[1] 刘震宇[1] 王劲波[2]
[1](厦门大学管理科学系,福建厦门 361005)
[2](厦门大学计划统计系,福建厦门 361005)
E-mail:xhyang@xmu.edu.cn

摘 要 通过识别极限编程中的知识创造活动,分析了极限编程的知识创造机理和特点,从而解释了极限编程的有效性。

关键词 极限编程 知识创造 SECI模型

文章编号 1002-8331(2006)29-0009-03 **文献标识码** A **中图分类号** TP312

Knowledge Creation in Extreme Programming

YANG Xuan-hui[1] LIU Zhen-yu[1] WANG Jin-bo[2]
[1](Dept. of Management Science, Xiamen University, Xiamen, Fujian 361005)
[2](Dept. of Planning Statistics, Xiamen University, Xiamen, Fujian 361005)

Abstract: Based on knowledge management theory, this paper profiles the eXtreme Programming method by identifying knowledge creating activities in it, illustrates the mechanism of knowledge creation in it, and analyzes the effectiveness of it.

Keywords: eXtreme Programming(XP), knowledge creation, SECI model

1 概述

软件工程的热点——敏捷开发方法,特别是极限编程(eXtreme Programming, XP)方法受到越来越多的关注。就技术层面而言极限编程并无突破,但文献表明它可以提高开发效率和代码质量[1]。本文从知识管理(Knowledge Management, KM)视角出发,用'知识创造'理论来解释极限编程为何能显著改善开发过程。

本文包含以下内容:文献回顾部分介绍了极限编程和知识管理等概念,以及使用知识创造理论分析极限编程的必要性;实践分析部分剖析了极限编程中的知识创造机理和有效性;总结部分讨论了极限编程中知识创造的特点;展望部分列举后续研究的方向。

2 文献回顾

2.1 极限编程

K.Beck在1999年最早表述了极限编程方法[2]。极限编程把多项软件工程最佳实践,如回归测试、代码评审、持续集成等,有机地组合在一起以期发挥最大的效果。国内外的研究人员和实践者由此掀起了研究热潮,有大量的应用案例和适应性改进发表[1]。一个有趣的研究是R.Miller针对Beck研究中过分专注于编程所做的改进——把原有实践扩充成19个并按实践主体分类(见表1)[3]。

尽管极限编程的研究很活跃,但大多是描述性的工作,即定义极限编程过程或报告极限编程应用。目前的研究没有从根本上说明极限编程为何有效,没有详细分析极限编程的机理。本文认为从知识管理视角出发可以找到答案。

表1 极限编程实践

(J/P/C/M分别代表Joint/Programmer/Customer/Management)

主体	名称	内容
共同	J1 迭代	每一到三周交付覆盖新特性的工作系统
	J2 公共词汇表	建立项目词汇集合
	J3 开放式空间	移除隔断式家具,采用开放式布局
	J4 简要回顾	经常反思开发的得失体会
程序员	P1 测试驱动开发	先编写测试代码或脚本,再编写实现代码,然后验证
	P2 结对编程	两人结成对工作,通过共同思考、讨论和检查来编程
	P3 重构	通过改写既有代码改善结构和性能
	P4 代码集体所有	每个人都有权力存取代码,有责任改进代码程序
	P5 持续集成	把工作代码频繁集成到工作系统中并编译、测试
	P6 过犹不及	只做必需做的事,不为'可能'的需要浪费资源
用户	C1 讲述故事	在开发现场描述系统的功能特性
	C2 规划版本	规划版本的特性集合和任务优先级
	C3 接收测试	通过测试确认项目进度和特性质量
	C4 频繁发布	尽快将工作版本投入使用以获得反馈
项目经理	M1 确认职责	帮助用户和程序员承担合适的任务
	M2 空中掩护	处理行政行杂务,保证项目成员专心工作
	M3 季度评审	确保项目进展情况在多个项目组中的传播
	M4 镜子	用度量和文书工作勾勒项目进展
	M5 持续前进	保证项目成员能持续工作

2.2 软件开发与知识管理

软件是产品和过程的统一体,从知识的角度来看:软件是编码化了的知识;软件开发是知识密集型活动。知识管理在软件开发中起着重要的作用,"整个软件开发的过程可以总结为:获取明确的和隐含的知识,并把这些知识具体化到软件之中";"软件开发的关键在于协调各种不同知识的学习"[4]。

传统软件工程方法(瀑布模型、螺旋模型等)已包含了知识

基金项目:国家自然科学基金资助项目(编号:70372070)

作者简介:杨烜会(1974-),男,博士研究生,主要研究方向为软件工程,信息系统等;刘震宇,教授、博导、博士;王劲波,讲师、博士研究生。

管理活动,例如需求管理是知识获取和整理;配置管理则是知识存储和分发。软件企业也使用知识管理系统(Knowledge Management System,KMS),如微软和 Infosys 公司的知识管理设施。然而,这些传统方法不能满足实际需要——项目失败率依然居高不下,知识共享遭到抵制[5]。主要原因在于这些方法局限于静态的、编码化的知识管理方式。

有效的软件工程方法必须使用更加动态的、过程化的知识管理理论来指导开发实践。本文认为极限编程正是遵循了这种思路才取得了显著的效果,但目前的研究还没有涉及极限编程中的知识创造,所以要引入知识创造理论。

2.3 知识创造理论

基于知识的理论(Knowledge-Based View,KBV)认为企业要通过创造和应用知识来保持竞争优势。该理论的一个基础框架是野中郁次郎的知识创造理论[6]。

野中的理论前提是知识可以按认知论(Epistemology)分为隐性知识(Tacit knowledge)和显性知识(Explicit knowledge)。显性知识指可以通过文字或系统化语言表达的知识,以数据、公式、说明书、手册等形式共享。隐性知识指深藏于实践中、难以言明和模仿的知识,在没有知识携带者参与的情况下,这类知识难于交流、理解和共享。

野中的理论认为组织是知识创造和放大的场所,知识不断地在个人/小组/公司甚至跨公司间传播、碰撞;知识通过在隐性/显性间转化来创造新的知识;知识转化过程有四种形式(见图 1):

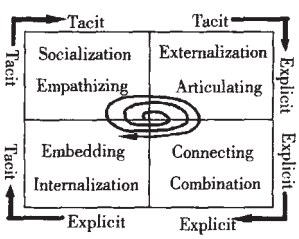

图 1 知识创造螺旋(SECI 模型)

(1)群化(Socialization):通过共同经历交流隐性知识。获取隐性知识的关键是观察和模仿,而非语言。

(2)外化(Externalization):将隐性知识用显性形式表达。转化工具有隐喻、类比、概念和模型等。它是知识创造中至关重要的环节。

(3)融合(Combination):将各种显性知识组合和系统化。通过文字或电子文档等形式对显性知识进行筛选、分析、组合和补充。

(4)内化(Internalization):将显性知识形象化和具体化。显性知识被个人吸收、消化,并升华成各自的隐性知识。

2.4 使用知识创造理论研究极限编程

野中的理论是目前知识管理中最全面、解释力最强的理论,使用它来研究极限编程有以下原因:

(1)野中的理论认为知识是"人们调整个人信仰趋向'真理'的动态过程",它注重隐性知识的价值。极限编程认为个人的隐性知识是软件开发的关键。

(2)野中的理论强调能动的创造,在使用中创造而不是先创造后使用。极限编程承认需求无法事先获取,解决方案也要不断调整。

(3)野中的理论认为组织可以放大个人知识。极限编程执着于建立共享的团队文化。

(4)野中的理论聚焦于研发活动,和软件开发有密切的关系。极限编程的一个来源(Scrum 软件开发方法)就出自野中 1986 年发表的文章[2]。

使用知识管理理论研究极限编程的工作目前非常少,本文希望能填补这个空白。

3 实践分析

3.1 极限编程中的软件开发知识

本文认为"软件开发知识"包含:(1)业务系统的知识;(2)编程和计算的知识;(3)软件项目管理的知识。它们分别对应软件开发范围、软件解决方案和软件开发活动。

这里对技术与知识的关系稍加说明。技术是知识的子集。技术是显性的,如业务描述、编程语言语法、质量保证规程等。知识包含技术,但它还包含决定获取何种技术、如何高效使用技术、如何培养技术能力等。软件开发知识是动态的,而不仅仅是技术的合集。知识附着在个人身上,在开发过程中创造、分享和使用。从隐性/显性纬度来看,软件开发技术是显性的,而软件开发知识无论是量还是质都更偏向于隐性知识。

极限编程把三种软件开发知识交织起来完成开发任务。极限编程本身是一种项目管理方法,但它包括重构技术、结对编程技术、持续集成技术等多项技术。极限编程实践中涉及的知识很广泛。从主体来看,它把用户也纳入范围,涉及所有主体的知识;从认知论来看,它包含这些人员所具有的隐性知识和显性知识;从存量来看,它不但包括项目初始时的知识,而且包括开发过程中所创造的知识。

3.2 极限编程实践

野中的理论按本体论(Ontology)把知识主体分成个人和组织两个层面,本文把极限编程中的个人层面的主体分成三类:用户、程序员和项目经理。极限编程中的用户指软件的最终使用者;极限编程中没有分析师和设计师角色,程序员负责按业务需求创造出最后的工作系统;极限编程中的项目经理也和传统的管理人员不同,他们的作用是服务而非监督。在极限编程实践中,软件开发知识在三类主体间流动,通过两种形态转化来创造新的知识。下面具体分析每项实践中包含的知识创造活动。

3.2.1 共同实践

共同实践涉及所有项目成员并贯穿项目始终,这些实践有助于建立共同经历和创造公共知识。

在迭代实践中,三类主体以隐性和显性的知识存量为基础合作,个人知识的在项目组内互相作用凝聚成显性的、编码化的知识体——目标工作系统。迭代控制知识创造螺旋的节奏,它把知识创造分割成多个步骤来进行,每次迭代至少覆盖知识创造螺旋一周,包含所有的四种知识转化。

公共词汇表是显性知识。在公共词汇表实践中,知识主体通过外化构造出模型、隐喻来描述系统;也可通过融合显性知识形成核心词汇集合作为指南。通过使用公共词汇表,各类认知主体把词汇表中的知识内化到个体身上。

开放式空间为知识创造准备物理和心理环境。采用开放式家具布局,可以让项目人员肩并肩地工作,消除听力和视力所及范围内知识流动的物理阻隔。开放式空间让行为观察和身体语言交流更方便,加速了群化式的知识传播;开放式空间也让

非正式讨论更容易，像‘头脑风暴’这样的外化可随时进行。开放式空间营造集体主义式开放的心理氛围，对于群化和外化是重要的支持。开放式空间在某些情况下会干扰内化。

梳理积累的知识是简要回顾要达到的效果。用户、程序员和项目经理都需要外化刚刚获得的业务流程、编程思想、项目进度等知识。及时地把这些知识表达出来并交流可以创造出新知识，特别是关于项目管理的知识。用户、程序员和项目经理也可以外化得到的知识融合成备忘录等文档形式。

3.2.2 程序员实践

程序员使用编程语言对业务知识进行编码，知识创造从学习业务知识开始。程序员还需要校准知识创造活动。

测试驱动开发包括单元测试和功能测试。测试是一种外化工具，程序员通过协助用户编写功能测试外化业务知识；通过编写单元测试外化设计策略。通过执行测试，程序员不断地验证自己的猜想，内化对功能和设计的理解。

结对编程支持群化、外化和内化。极限编程中的代码全部由程序员结对编写，业务知识通过群化式的生产最终外化成代码。程序员也通过编写代码来内化对业务规则和编程技巧的理解。结对编程时，程序员用问答来外化编程经验；用观察来群化编程知识。有意识的、经常性的轮换加强了知识共享和扩散。结对编程也会创造出关于团队合作的知识，比如程序员的工作习惯和专长分布等知识，这些知识通过逸事或传说等形式外化。结对编程实践也内化了合作能力。此外，由于测试驱动开发和重构都是通过结对完成的，所以结对编程中的知识创造也在这些实践中发生。

重构是知识校准过程，其知识创造以组合为主。重构过程中，体系结构被优化，蕴涵在代码中的知识被重新安排。尽管没有产生大量的新知识，但知识的质量被改善。

知识共享是知识创造的基础。结对编程让隐性知识得以共享，而代码集体所有则分享显性知识。集体所有权实际上是一种集体知识创造责任——每个人都要参与知识创造，每个人都有责任提高知识的质量。

知识创造需要不断地调整，而调整需要反馈，持续集成让反馈更及时，让问题更早暴露，同时让分散于各处的知识更早融合。

过分强调设计的精巧会让知识创造成为智力游戏，过犹不及提示程序员把知识创造活动聚焦于客户的商业价值。

3.2.3 用户实践

知识创造的一个方面是重要方面是问题发现。用户实践定义目标软件系统的商业价值。

业务流程知识可能是显性的，如提货单样本；也可能是隐性的，如流程加速技巧；甚至可能是未知的，需要在实践中创造或再造。用户通过讲述故事来外化流程知识。故事的模糊性更容易将隐性知识析出。讲述和倾听构成交互的过程，程序员和用户的知识在这里融合，最后整理成功能测试。

在规划版本实践中，用户把业务知识外化成特性集合，通过融合程序员对这些故事（任务）工作量的估计，提出版本规划。业务优先级别的判定依赖于用户的隐性知识。

知识创造需要连续反馈，接收测试提供反馈。它和单元测试分别反映系统的内外两个侧面。用户通过两种活动来履行接收测试：协助书写功能测试脚本和执行这些测试。书写测试脚本外化了对目标系统的功能预期，执行测试则内化了开发进度方面的知识。

频繁发布的作用在于控制反馈的频率。在极限编程中，发布的版本几乎就是最后的目标系统。通过频繁发布，项目人员对项目进展进行内化；项目人员基于工作系统这个原型来展开讨论，从而外化对系统认识深入后所获得的知识。

3.2.4 项目经理实践

在极限编程团队里，经理的职责在于协调与服务，而不是“科学管理”式的命令与控制。

软件开发团队需要自治。极限编程中项目经理协调责任确认，而不是指派任务。项目经理通过确认职责来引导知识工人充分发挥能动性。特别是在项目组建初期，项目经理需要协助项目成员把责任确认机制内化，使每个成员都可以熟练地选择合适自己的任务。

在项目开发过程中有很多琐碎的行政事物，如人员招聘，对外汇报等。项目经理要保证知识工人专注于系统开发，所以需要提供空中掩护使他们免除干扰。空中掩护的目的在于保证知识创造活动的稳定进行。

季度评审是项目间的知识交流活动，它被用来支持软件开发组织同时开展多个项目。知识创造并不是封闭体系，项目经理需要与外部知识源沟通以获得必要的支持。项目经理要扮演知识中介或‘边界拓展者’。项目经理要通过多种方式和外界交流，要把项目进展外化成项目简报，要和其它经理一道融合项目进度报告。

项目经理要在项目过程中观察和度量项目进展，要像镜子一样对项目进行反省。极限编程中项目经理一般要承担编程工作，项目中的各种知识经由群化过程流过项目管理。项目经理更要把项目各方面的情况外化成显性知识，并要像枢纽一样融合各处的知识。

持续前进有助于提高知识创造的产量和质量。对于软件业这样高压力的工作，如何激励和保持知识工人的创造性是个巨大的挑战，持续前进不失为一个有效的方法。

4 总结

上文逐项分析了极限编程实践中的知识创造活动，从中可以看出极限编程中的知识创造有以下特点：

（1）充分的知识共享。极限编程利用公共词汇表、代码集体所有等实践促进显性知识共享；利用结对编程、简要回顾等实践促进隐性知识流动。极限编程中用户的业务知识直接转移给程序员并最终外化成代码，这比传统方法中由分析师把需求加工成规范，由设计师生成设计方案再转移给程序员更加高效。

（2）有机的知识组织。极限编程的各项实践很少只包含一种知识转化，很少只涉及个别人员。极限编程通过制度性安排，如开放式空间、结对编程等方式在开发组织中造成知识冗余和技能多样化，利用自治团队的能动作用创造知识。共同实践和项目经理实践努力培养并维护知识创造的氛围和节奏。

（3）动态的知识创造。极限编程对于外化工具的使用是非常灵活的，如测试代码承担了知识的表达和共享、公共词汇表甚至可以用来表达设计。极限编程通过及时的高频反馈不断调整知识创造过程，如测试驱动开发、持续集成、简要回顾和镜子等实践。极限编程还利用重构不断提升知识质量。

极限编程实践能灵活地进行知识创造并广泛地分享知识，所以它能有效地组织软件开发。

（下转 15 页）

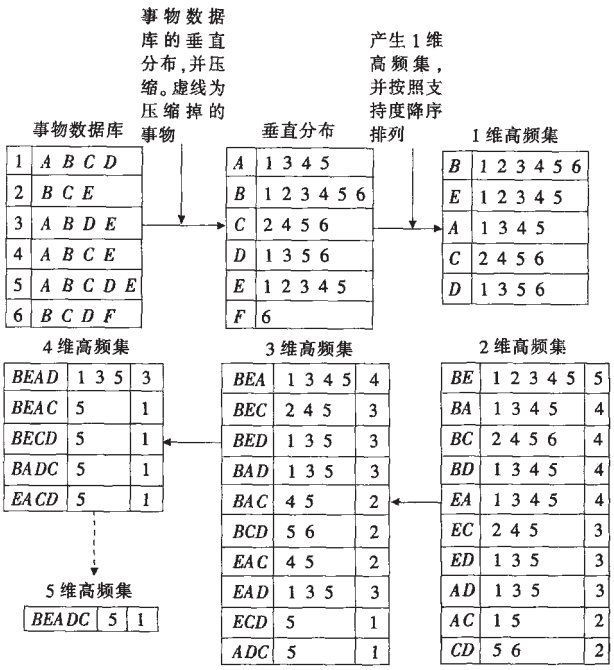

图 5　降序 join 半概念格挖掘频繁项的过程

```
(4)  L_i=output(Tran);  //输出 i 维频繁项集 L_i
(5)  for(j=1;j<=|Tran|;j++){  //得到 i+1 维高频集
(6)    [C(j) W(j)]=generate(Tran[j]);
       //返回以 Tran[j]为前缀的 i+1 维候选集及每个候选项的权值 W
(7)    [C(j) W(j)]=DSort(C, W, MIS);
       //对以 Tran[j]为前缀的 i+1 维候选集按降序排列,当支持度相同时,按照字母序排列,把小于最小支持度 MIS 的项删除。得到以 Tran[j]为前缀的 i+1 维高频集。
       }
(8)    Updata(Tran, C, W); //更新事务数据库
(9)    i++;
  }
```

3.3　算法的说明

通过上面的说明,可以清晰地看到,在排序加权 join 半概念格中[B]的等价类含有的项集数目要比在完全概念格中[B]等价类中含有的的项集数目少的多(注:事务数据库中项越多裁减掉的项集也就越多),所以与之相比,首先,在同等条件下我们的算法所需的空间要比 Eclat 算法小;其次,由于同类中含的项集少(少的程度取决于具体问题),算法在挖掘子概念格时其时间开销要小。

4　结束语

本文通过对 Eclat 算法进行深入分析,对完全概念格按照支持度进行了删除,得到了一个向下封闭的降序加权 join 半概念格,基于降序加权 join 半概念格和降序加权 join 半子概念格对 Eclat 算法进行了改进,给出了一个快速的关联规则挖掘算法,经过分析,该算法与 Eclat 算法相比,效率更高。当然,在最坏情况下同类中含的项集差别不大,这里的算法与 Eclat 算法相差不大,这种情况发生的几率比较小。

(收稿日期:2006 年 8 月)

参考文献

1.Zaki M J.Scalable algorithm for association mining[J].IEEE Trans Knowledge and Data Engineering, 2000; 12(3): 372~390
2.Agrawal R, Imielinski T, Swami A.Mining association rules between sets of items in large database[C].In: Proc of the ACM-SIGMOD conf on Management of Data, ACM Press, 1993: 207~216
3.Han J, Pei J, Yin Y.Mining frequent patterns without candidate generation[C].In: proc of the ACM-SIGMOND conf on management of data.Dallas ACM Press, 2000: 1~12
4.高飞,谢维信.发现含有第一类项目约束的频繁集的快速算法[J].计算机研究与发展, 2001; 38(11): 295~301
5.杜剑锋,李宏,陈松乔.等单调和反单调约束条件下关联规则的挖掘算法[J].计算机科学, 2005; 32(6): 142~144
6.高飞.关联规则挖掘算法研究[D].西安:西安电子科技大学, 2001-10
7.Du Boucher-Ryan, Derek Bridge.Collaborative Recommending using Formal Concept Analysis[J].Knowledge-Based Systems, 2006: 1~7
8.Taweechai Ouypornkochagorn, Kitsana Waiyamai.Formal Concept Mining: A Statistic-Based Approach for Pertinent Concept Lattice Construction[J].Lecture Notes in Computer Science, 2004: 195~212
9.Karell Bertet, Jean-Marc Ogier.Graphic Recognition: The Concept Lattice Approach[J].Lecture Notes in Computer Science, 2004: 265~279
10.秦昆.基于形式概念分析的图像数据挖掘研究[D].武汉:武汉大学, 2004

(上接 11 页)

5　展望

本文对极限编程中的知识创造活动做了分析,这是使用知识管理原理分析极限编程的第一步,后续的研究工作可在多方面展开:对比传统软件工程方法和极限编程中的知识管理活动;利用知识管理原理弥补极限编程在外包和分布式开发等条件下的不足;讨论软件企业采用极限编程进行知识管理的战略等。(收稿日期:2006 年 8 月)

参考文献

1.G Succi, M Marchesi.极限编程研究[M].张辉译.北京:人民邮电出版社, 2002
2.K Beck.Embracing Change with Extreme Programming[J].IEEE Software, 1999; 32(10): 70~77
3.R.Miller.Demystifying Extreme Programming:'XP distilled' Revisited[EB/OL].http://www-128.ibm.com/developerworks/java/library/j-xp0813
4.P.McBreen.软件工艺[M].熊节译.北京:人民邮电出版社, 2004
5.K C Desouze.Barriers to Effective Use of Knowledge management Systems in Software Engineering[J].Communications of the ACM, 2003; 46(1): 99~101
6.I Nonaka, H Takeuchi.The Knowledge-Creating Company[M].New York: Oxford University Press, 1995