计算机科学 2000Vol. 27№. 12

极端编程模型
——新颖的软件工程模型

Extreme Programming Model——New Software Engineering Model

潘秋菱　刘宗田
（合肥工业大学微型计算机应用研究所　合肥 230009）

Abstract Extreme Programming (XP) model is a relatively new software engineering model. With the traditional "Waterfall" development, requirement changes are hard. XP model focuses on the adapting to the requirements change domains. This article introduces the characteristics, key rules and practices of XP model. In the end, it contrasts the XP model with traditional software engineering models, e. g. the "Waterfall" etc

Keywords Extreme programming (XP), Requirement changes, Risk management, Rules and practices, Software engineering model

极端编程模型(Extreme Programming Model, XP模型)，是大约1995年由Kent Beck等提出的一种软件工程模型，Kent Beck和Ward Cunningham从九十年代初起致力于思考使软件开发简易高效的方法。从丰富的开发实践经验中总结出了一组规则，提出了极端编程模型，在软件工程界引起了许多思考和争论。如今，XP模型已成功用于一些努力控制软件开发质量和成本的公司，如Bayerischelandesbank, Credit Swiss Life, DaimlerChrysler, First Union National Bank, Ford Motor Company and UBS[1]，文[2]详细介绍了Chrysler公司应用XP方法的过程及取得的成绩，所有的实践表明XP编程模型在一定的领域中表现得卓有成效。

软件开发方法的研究经过了不同的时代，在60年代末70年代初是一个极端自由的年代，那时的程序员可以随心所欲地进行自己的工作，选用自己喜欢的任何方式。许多程序员非常善于算法复杂的高效的程序，然而有些算法除了作者自己，几乎没有人可以理解，并且软件质量很难保证。以致于随着程序复杂性的增加，若想程序没有bugs地开发运行几乎是一个奇迹。进入70年代后，Dijkstra发表了他的著名的文章"Gotos Considered Harmful"，从此引发了一场争论，争论的最后产生了软件工程的基本思想。人们开始考虑对软件开发进行各种限制，以保证软件的质量。那时，人们相信更大、更多的约束方法，有助于开发出质量稳定、成本可控的软件产品，无纪律的牛仔式的编程方法被视为应该禁止的。80年代中，有了一些开发高质量软件的规则和操作方法，人们开发出的软件比早年的产品具有了更高的可靠性，看起来似乎是如果我们能够提出足够可用的规则来对付我们可能碰到的任何问题的话，就可以完全控制成本，开发出完善的程序。于是人们加入越来越多的规则和方法以覆盖所有可能的问题。然而，90年代以及现在，开发人员发现规则越来越多，方法越来越复杂，需要记录修改的文档也变得繁重不堪，有时甚至失去控制。人们开发出了各种支持开发的工具，这些工具节省了程序员的很多时间，然而，有些工具也非常复杂，掌握工具本身已成为一件困难的工作。于是很多程序员并不完全按照完整的方法进行开发，而使用一些必须的规则，形成了所谓轻量级方法(lightweight methodologies)。与过去的重量级方法(heavyweight methodologies)相比。拥护轻量级开发方法的开发人员只应用一些简易的、必要的规则，抛弃那些复杂的、实际上最后也是很难跟从的规则，更加轻松地开始一个项目。XP编程模型即是一种轻量级编程方法。然而，使用轻量级编程方法并不意味着忘记以往的教训，回复到早期的无序编程状态，而是使用合适的规则使软件开发既可保证一定的可靠性，又有适当的成本。不同的方法提出了不同的规则组合。

"极端编程模型"是一个新近提出来的模型，本文分析了XP模型的特点，简要介绍了它的过程和实践，并与传统的软件工程模型进行了比较。

潘秋菱　博士生，目前研究兴趣为软件工程、Petri网、软件质量保证；刘宗田　博士生导师，研究方向为知识工程，软件工程。

1. 特点

XP 模型是轻量级编程模型的一种，它通过对以往软件开发的标准方法进行重新审视，提出了由一组规则组成的一些简便易行的过程。单独来看每一规则是没有意义的，但所有规则合并起来就展现出一幅完整的景象。这些规则是通过在实践中观察使软件开发高效或缓慢的因素而得出的。它既考虑了保持开发者的活力和创造性，又考虑整个开发应是有组织、有重点的持续过程。

XP 是面向客户的开发模型，重点强调用户的满意程度。在用户需要的时候提供给用户满意的程序。其目的在于提高开发过程中对需求改变的适应能力，即使是在开发的后期，也可较大程度地适应用户的改变。同时，XP 指出了一种简单但高效的团队开发风格，认为开发组不仅只是软件开发员，同时还包括管理人员和客户。询问问题、商讨项目范围和进度以及创建功能测试需求等都需要多方人员的紧密协作。在 XP 的思想理念下，管理人员、客户和开发人员作为工作组的一员，目标都是为了生成出高质量的产品。

XP 主要在四个方面突出了自身的特点，致力于提高软件的开发能力[2]：交流（Communication）、简洁（Simplicity）、反馈（Feedback）和进取（Aggressiveness）。XP 编程方式促使软件开发人员和客户以及相互之间保持随时频繁的交流，所有项目的展开都建立在用户的参与和项目组讨论基础上。XP 编程方式努力产生简洁清晰的设计。XP 尽可能早地将结果产品送交用户并根据用户的反馈进行修改。极端编程模式使开发小组对用户需求和使用技术的改变总是充满信心。

XP 编程方法与传统的软件工程模型具有很大的不同，所以应严格注意其适用范围。

2. 适用范围

XP 编程模式是针对用户需求经常变化的软件开发提出的编程模式。有时用户决定需要新软件时并不能确知自己的详细需求，也许有时可以预期到系统的功能会随时间不断改变。类似情形下，XP 编程模型就会展现出其他传统的软件过程方法所不可比拟的优点。

XP 的提出针对的另一问题是软件开发所承担的风险。风险就是带有负面效果的情况产生的可能性。软件的风险包括开发、使用和维护三个方面。如果用户需要一个有特殊要求的新系统，这个系统的开发就具有一定的风险，如果开发该软件的小组不了解这种系统，风险就会大一些，如果这种系统对整个软件开发企业都是全新的话，风险更大。XP 编程模式的建立就是为了减小风险，增加成功的可能性。

XP 编程模型适合于中小型的开发小组，并需要小组内部有良好的交流和改进机制。XP 不适于庞大的项目组，一般来说，对于需求不断变化的或是高风险的项目，运用 XP 方法进行开发的较小的开发小组将总是比庞大的项目组拥有更高的效率。同时，XP 模型要求使用面向对象的开发方法[3]，并对测试提出了较高的要求。

XP 项目组无一例外地比相似情况下其它软件工程方法拥有更高的生产率，但这并不是 XP 编程方法的主要目的，XP 的真正目标是在用户需要的时候可以提交使用户满意的系统。对于注重用户满意程度的企业，XP 模型是一个很值得考虑的软件过程模型。

3. 规则与实践

规则与实践是 XP 模型的观点、内容同时也是操作方法。以下依照传统的计划、设计、编码、测试及其它顺序来介绍 XP 模型的主要规则和实践，但是，各阶段都是反复进行的，而且，正如其名字，从对它的实践的描述中看到很多引人注目的想法[5]。

3.1 计划

计划阶段有如下规则：

1）“用户故事” 在 XP 模型中它代替了传统的用户需求。由用户用自己领域词汇书写。它用来预计项目花费时间和承担的风险，并作为编写功能测试的依据。用户故事和传统用户需求的最大不同是对细节的描述，用户故事提供细节的程度仅到可在一定风险范围内预期完成该功能所需时间即可。与用户需求的另一点不同的是用户故事将注意力放在用户需要上，不考虑任何技术细节。

2）“计划游戏” 用户根据编程人员的估算决定每次产品发放的功能范围与时间。在每一个开发循环内，编程人员只开发用户所指定的功能（即用户故事）。生成各方（开发者、用户、管理层）同意的工作时间表，

3）“小型发放”（Small Releases） 项目开始后，在几个月内就进入产品原型生产，但这时并未完成全部开发需求。只开发计划游戏制定的此次发放的功能，从这时起，经常地进行新的发放——间隔范围可以用天到月之间的时间单位表示。

3.2 设计

软件设计遵照以下规则进行：

1）“简单的设计” 作尽可能简单的设计。设计中不含任何重复的程序、含有尽量少的类与方法。只有确定需要的模块才作为设计的对象，在任一个时刻，设计的产品都运行所有的测试，从而向编程人员提供他们

所需的一切测试反馈信息。

2)"隐喻" 用户与编程人员用"隐喻"或一组"隐喻"来定义产品系统,命名设计的类等,利于理解,交流与重用。

3)"CRC 卡片设计" 应用 CRC(Class,Responsibilities,and Collaboration,类,功能,协作关系)卡片进行设计使设计者完全脱离程序思路,充分发挥面向对象的功能。CRC 设计方法与传统方法的最大不同的是没有完整书面的设计方案。一切包含在卡片之中。

4)"透彻解决" 在设计阶段如果有技术和其他难以确定的问题,宁可派几人编制必要的系统进行实际测试,以提高预期的可靠性,同时降低可能的风险。

5)"不及早加入" 只按时间表规定完成所需功能,决不加入现在不需要的功能。

6)"循环制造" 通过对现有设计的改变,使系统的设计实现演化。这个过程中,所有的测试用例都必须正确运行,以确保原有系统的完好无损。

3.3 编码

编码时的规则有:

1)"编码标准" 编码必须应用相同的标准和书写格式。

2)"双人编程" 所有的产品程序都必须由两个开发人员在同一个显示器、用同一个键盘、及用同一个鼠标写出。

3)"不断集成" 所有新程序都在几个小时之内被集成到现有的系统之中。在进行集成时,系统一定要从新构建;同时,一切测试必须通过,否则,新的改变就被舍弃。

4)"顺序发放" 同一时刻只允许一组编程小组往源代码库中集成、测试、发放新内容,保证每次发放的都是最新版本。

5)"共同拥有" 任何时候,只要有机会,任何编程人员都应该改进系统中的任何程序。不会有人成为改变中的"瓶颈"。

6)"最后优化" 首先使它工作,使它正确,最后提高速度。

3.4 测试

XP 模型大大加强了对测试的要求和测试的技术性及自动性[6]。

1)"单元测试" 是 XP 模型的基石。编程人员每时每刻都在写单元测试用例。但测试风格与以往有一些不同,首先编程人员必须先写好单元测试用例,才能开始设计开发。其次必须测试系统中所有的类,必须创立或下载单元测试框架进行自动测试。

2)"功能测试" 用户则为每个开发循环的"故事"写功能测试用例。创立自动功能测试以保证可以对所有改变进行重新测试,功能测试一旦失败需要建立一组单位测试重新监控测试过程。这些测试用例都必须正确地运行并收集起来作为总体测试用例。

3.5 其他

其他包含一些管理操作型的规则实践方法。

1)"驻点用户" 用户成员全工作时间与开发小组在一起。

2)"清晨站立会议" 每天早晨准时开始项目组全体成员围站讨论遇到的问题,解法及注意事项,避免长时间讨论,不强调人员到齐。作到有问题发言,无问题静听,从而增强了交流并提高了效率。

3)"40 小时工作周" 不允许任何人连续两周超时工作,就算是频繁的独立超时工作,也将被视为存在某种必须加以解决的深层的问题。

4)"开放的工作空间" 整个开发小组在同一个大层间内工作。层间的四周建立工作小间,每个小间之内,两个开发人员在设立在中央的一台计算机上工作。

5)"仅仅是规则" 每个人都必须承诺遵守所有的规则,才能成为"极端"团队的开发人员,也就是说,团队中所有成员必须遵守团队开发的所有规则,否则就必须离开这个团队。但是,它们仅仅是规则。这个开发团队可以随时改变这些规则,先决条件是,他们必须就如何评价这些改变所带来的效果取得一致的意见。

4. 与传统软件工程模型的比较

历史最长、应用最直观的软件工程模型为 1970 年 Winston Royce 提出的"瀑布模型"(WFM)[7]。它提出软件生命周期的各个组成部分:分析、设计、编码、测试、维护。各自的定义任务及相互关系,是以后各种软件过程模型演变的基础。在实践中,其开发过程基本上就按软件生命周期模型的各个组成部分顺序地进行,被认为是一个线性模型[8]。WFM 强调开发的阶段性,强调早期计划及需求调查,强调产品测试。对混乱的、不成熟的软件开发方式进行了约束,是六十年代末出现的软件危机的有效解决方案。WFM 模型至今仍然是世界范围内应用最广泛的模型。

然而,WFM 模型也有很多缺点,随着人们软件开发技术的不断进步,也积累了丰富的开发经验,成长出许多优秀的软件技术和管理人员,业界对 WFM 的批评逐渐增多,甚至于这些批评已经引起了对其效力的怀疑[9]。主要的问题有以下几点:

a)依赖于早期进行的唯一一次需求调查,不能适应需求的变化;然而,在实际工作中,很难要求用户一阶段内清晰完全地概括所有需求。不但增加了用户的负担,同时,不能正视实际中在项目的最初阶段总是存在一些不确定性的事实。结果(也是常常发生的情况)

是，改变总是引起混乱。

b)用户要有足够的耐心，直到项目开发的后期用户才能够见到具体的项目。这时，如果发现项目与自己的想法不一致，或开发方与用户之间产生了误解，更改是十分不方便的。如果交付用户时发现了较严重的错误，将是灾难性的。

c)由于是单一流程，开发中的经验教训不能反馈应用于本产品的过程。

d)风险往往迟至后期的开发阶段才显露，因而失去及早纠正的机会。

e)开发人员经常进行不必要的等待。Bradac 通过对实际项目的分析提出：这种经典的 WFM 会导致开发组的一些成员必须等待另一些成员完成依赖的工作，而实际上，等待耗费的时间很可能超过开发项目的时间，尤其是在项目开始和结束的阶段[10]。

针对瀑布模型的缺点，以后提出了各种各样的软件过程模型，包括演化模型(the Evolutionary model)、渐增模型(the Incremental Model)、原型系统(the Prototyping Model)等。这些模型在开发模式上都采取分批循环开发的办法，每循环开发一部分的功能，它们成为这个产品的原型的新增功能。于是，设计就不断地演化出新的系统。实际上，这些模型可看作是重复执行的多个"瀑布模型"。"螺旋模型"(the Spiral Model)是 Barry Boehm 1988 年提出的[11]。基本的做法是在"瀑布模型"的每一个开发阶段之前，引入非常严格的风险识别、风险分析和风险控制，直到采取了消除风险的措施之后，才开始计划下一阶段的开发工作，否则，项目就很可能被取消。另外，如果有充足的把握判断遗留的风险已降低到一定的程度，项目管理人员可作出决定让余下的开发工作采用另外的生命周期模型，如"渐进模型"，"瀑布模型"，或自定的混合模型。

这些传统软件过程模型都强调计划准确，文档编写详细、准确，并将文档作为控制的主要手段。而 XP 模型从实践及统计数据中提出文档编写的新模式，从上节对规则的描述可以看出，XP 模型只提供必须用到的信息，努力减轻开发人员及用户的负担，正视开发的所有改变，将开发视为下断集成，下断重复的过程。同时，鼓励开发人员协作、创新，并通过严格自动的测试对整个项目进行控制。XP 模型中包含了一些从管理的观点考虑的制度和方法，提出了诸如驻点用户，双人编程等概念，强调开发组的团体意识，认为人人都应对项目的所有程序随时修改改进。重新看待需求的改变，从理论及实践等各方面既努力减轻开发人员的负担，又努力保证客户的满意程度。

总结 XP 编程模式是为了克服传统的瀑布流模型等不适于软件开发过程中需求的变化而提出的新颖的软件过程模型，目的在于提高开发过程中对需求改变的适应能力，同时，降低软件开发风险。XP 是面向客户的开发模型，真正目标是在用户需要的时候可以提交使用户满意的系统，重点强调用户的满意程度。

显然，XP 模型在实际应用中还有许多问题和细节，也引起了较大的争议，网上还出现了专门的讨论组[12]，用户故事、CRC 设计方法、双人编程、测试、数据库的设计等都是讨论组争论的焦点，然而，随着人们编程经验的丰富，加上对模型的不断验证和改进，XP 模型会在一定的范围内发挥更大的作用。

参 考 文 献

1 Available at: http://www extremeprogramming.org

2 Anderson A, et al. Extreme Programming/Chrysler C3 Payroll. Distributed Computing, 1998(Oct.)

3 Available at: http://c2.com/cgi/wiki? Extreme Values

4 Beck K. Smalltalk Best Practice Patterns. Prentice Hall, Upper Saddle River, NJ, 1997

5 Beck K. Extreme Programming Explained. Embrace Change Addison-Wesley, Reading, MA, Oct. 1999

6 Jeffries R E. extreme Testing Software Testing & Quality Engineering, 1999(March)

7 Royce W W. Managing the Development of Large Software Systems: Concepts and Techniques. Proc WESCON, Aug. 1970

8 Hanna M. Farewell to Waterfalls. Software Magazine, 1995(May), 38～46

9 Pressman R S. Software Engineering, A Practitioner's Approach. McGraw-Hill, 1997

10 Bradac M, et al. Prototyping a Process Monitoring Experiment. IEEE Trans. Software Engineering, 1994, 20(10): 774～784

11 Boehm B. A Spiral Model for Software Development and Enhancement. Computer, 1988, 21(5): 61～72

12 Available at: http://www.egroups com/group/extreme-programming