XP 中若干实践的探讨*)

陈 津 徐宝文

(东南大学计算机科学与工程系 南京210096) (江苏省软件质量研究所 南京210096)

摘 要 XP是适合中小型团队在模糊或迅速变化的需求的情况下使用的轻量软件开发过程。文章首先分析了XP中的几个关键实践:简单设计、测试、结对编程和重构,并针对某些应用背景下可能存在的问题提出了相应的修改建议。然后,以举例的方式具体说明如何运用XP的实践。最后,通过XP与传统软件开发方法的比较,分析了XP的主要特征和适用性。

关键词 软件工程,XP,软件开发

Notes on Several Practices in Extreme Programming

CHEN Jin XU Bao-Wen

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096)

(Jiangsu Institute of Software Quality, Nanjing 210096)

Abstract XP is a lightweight software development process for small and medium teams dealing with vague or quickly changing demands. Firstly several key practices, for instance simple design, tests, pair programming and refactoring, are analyzed in this paper. Some suggestion modifying XP practices is brought forward to tackling some possible problems in practical situation. Then how to use XP practices is explained concretely by an example. At last, XP primary characteristics and applicability are anatomized by contrast with traditional software development process.

Keywords Software engineering, XP, Software development

1 引言

自1968年"软件工程"一词问世以来,软件工程已经成为一门重要的学科。虽然软件危机没有得到彻底解决,但在软件开发方法和技术方面已经取得了很大进步。现在软件开发所面临的商业环境比以前更加纷繁复杂,传统的开发方法已经不能适用于所有情况了。最近由QSM研究机构的Michael Mah所做的研究表明:在统计的1997～1999年的210个项目中,有40%的项目延期,30%的项目超过预算,47%的项目找不到原先的开发团队。造成项目延期和超支的原因从根本上说是传统软件开发方法相对多变的环境来说太笨重了,这些方法要遵循的规则如此之多,以致往往阻碍整体的开发进度。因此常常称之为重量(heavyweight)开发方法。针对重量开发方法的不足,研究人员提出了轻量(lightweight)开发方法或灵活(agile)开发方法[2],如XP(Extreme Programming)、水晶(Crystal)方法、Lean开发方法、Scrum方法以及适应(Adaptive)软件开发方法(ASD)。XP是适用于中小型团队使用的、在需求不明确或者迅速变化的情况下进行软件开发的轻量级方法学[1],是一种比较有影响的方法。

为了应对不断变化的、不确定的、不可预知的商业环境提出的挑战,为了解决软件生产中存在的诸多问题,我们对XP进行了综合性研究。本文首先分析了XP的几项关键实践,并针对相应应用背景中可能存在的问题提出了修改建议,然后举例说明如何将XP的诸多实践应用到一个具体项目中。最后通过与传统软件开发方法的比较论述XP的主要特征并分析XP的适用性。

2 XP实践分析及修改建议

XP中的12项实践并非新创,但是把它们发挥到极致并紧密地结合在一起使用,就产生了前所未有的效果。美国北卡罗来纳州大学计算机科学系对这些实践的接纳程度进行了统计[3]。从数据中我们可以看出学生们对这些实践的接受程度差别很大,从最高的97%到最低的36%。按接受度从高到低依次为:简单设计、编码标准、集体所有权(collective ownership)、测试、持续集成(continuous integration)、结对编程(pair programming)、计划游戏(planning game)、每周工作40小时(40-hour weeks)、现场客户(on-site customer)、小版本(small releases)、隐喻(metaphor)和重构(refactoring)。下面我们重点讨论并分析其中的几项关键实践。

2.1 简单设计

针对经常出现的软件过度开发的问题,XP建议采用简单设计(simple design)。简单设计强迫程序员放弃整体设计的习惯而进行小的增量设计,让系统随着需求的增加而成长。把简单设计节省的时间用于解决困难问题。不要浪费精力去增加将来可能需要的功能,即使现在设计的成本为零,也会由于系统的复杂性而增加理解、测试和修改的成本。这一实践的

*)本文部分得到国家自然科学基金项目、国防"九五"及"十五"重点预研项目、国防武器装备预研基金项目、江苏省自然科学基金项目、江苏省科技攻关项目、武汉大学软件工程国家重点实验室开放基金项目、江苏省计算机信息处理技术重点实验室开放基金项目(苏州大学)的资助。陈 津 硕士生,主要研究领域为软件工程、程序设计语言。徐宝文 教授,博士生导师,主要从事程序设计语言、软件工程、并行与网络软件技术等方向的教学与科研工作。

基础来自于 XP 平滑的变化成本曲线。一旦丧失这一基础，简单设计不但无益而且有害。

在需求快速变化的极端情况下，简单设计的优势最为明显。但是在通常情况下，放弃整体设计一定程度上也存在风险。因此我们认为要根据项目的具体情况来决定简单设计的执行细节。在系统庞大而且复杂的情况下，必须适当考虑整体设计，否则后期的重构成本可能太大从而增加项目失败的风险。

2.2 测试

XP 中包括两类测试[1]：单元测试和功能测试。在 XP 中以测试驱动开发，以测试保证代码质量，测试在所有实践中处于核心地位。XP 中的测试有其自身的特点。

首先，要求在编码前写好测试用例[1]。这样做有几个优点：(1)在编码前写好测试用例实际上就明确定义了接口和相应的动作，使得编程时的思路更清晰，提高了开发质量和速度。(2)有助于设计简单化，因为测试一个简单接口更容易。实际上，测试用例起到了方法的功能说明的作用。(3)如果先编码的话，由于人性中存在盲目自信和懒惰的弱点，程序员往往会认为某些部分肯定符合要求而削弱甚至不做测试工作。(4)在编码前写好测试使得参加项目的全体人员随时都能根据通过测试的多少来确切掌握详细的开发进度。测试成为项目进度的一种度量。

第二，所有的测试必须自动执行且测试的时间不能太长。测试时间太长就不利于持续集成。测试能在无人的情况下执行，提高了测试效率。为控制测试时间，我们建议：(1)不必为每个方法都编写测试用例，选择关键的、易出错的部分编写测试用例。(2)如果测试耗时过长，那么只运行添加或修改部分的测试，而把所有测试的运行放到非工作时间例如晚上下班后进行。

第三，测试必须是独立的[1]。每个测试都不与其它测试交互，以降低测试的复杂度。避免因为一个测试的失败导致相关的其它测试失败，这样就不利于查找出错的原因。

第四，单元测试的编写和执行不是由专门的测试人员完成的，它是编程人员日常工作的一部分。功能测试最好由用户来写，但至少是用户提出由开发人员按要求编写。

测试并不依赖 XP 的其它实践，所以它可以用于任何软件开发项目。在需求快速变化的项目中，测试为系统的修改提供安全性。测试也为程序员提供迅速的反馈，加快变化的速度并使得修改更加积极。根据统计，87%的程序员认为测试用例的执行增强了他们对代码的自信心。在 XP 的课程实验中，学生们最后都认为在编码前写好测试用例是有益的[4]。

2.3 结对编程

结对编程是 XP 中最有争议也最难被程序员接受的实践之一。不同的研究人员对结对编程进行了实验，得出的评估结果却相反[4,5]。这主要是因为结对编程的潜在好处目前尚不清楚，无法对此进行定量的评估。但是，可以肯定的是结对编程的开发成本并没有加倍。

采用结对编程有很多好处。首先，代码具有更高的质量。在输入代码的同时错误往往就被修正过来。而在测试或者实地运行时，确定并修改问题的成本显然要成倍甚至呈指数级地增长。第二，有利于问题的解决，两个人共同做出设计更快更简单更完善。第三，由于结对是动态的，对于系统的每一部分至少有两个人都熟悉详细的设计细节，而文档是达不到同样效果的。同时，结对编程也降低了团队成员变动给项目带来的风险，弥补了文档最小化的缺陷。第四，通过结对编程，新人能够更快更容易地学到更多的设计、编码和当前系统的知识。第五，团队成员间的合作更密切，信息的交流更顺畅。口头交流与文档相比具有不少优点。

对结对编程中比较模糊的问题，我们有如下看法：1. 分工不明确，需要找到一种既高效又容易被接受的工作方式。一个人总是负责编写和输入代码，而对另一个人我们认为可以采用下列几种方式：检查同伴正在编写的代码；考虑现在的设计是否可以简化、改进等，查找相应的文档；考虑下一个需要解决的问题；审查原来编写的代码。2. 对于一些简单的编程，例如标准的数据结构和公认的算法，我们认为就没有必要结对编程了。结对编程要求开放的物理环境的支持，对现在的开发方式是一个挑战。在收益并不清楚的情况下，推行结对编程确实有一定难度，但是也应该看到结对编程具有其它开发方式所不能提供的好处。

2.4 重构

重构是改进代码结构但不改变功能的过程，其目的是增加可理解性、可维护性和灵活性。在 XP 中的重构常常是由于简单设计引起的。所以，我们认为为了最大程度地降低成本，要努力在设计和重构之间取得最佳的平衡。为此应该适当考虑系统的整体设计以避免由于缺乏整体设计而带来的不必要的重构。但是，对于大型项目重构可能是避免不了的。在项目开始初期，由于代码量小而无需重构。当代码量达到一定程度时，重构的问题就凸现出来了。重构具有相当的风险，而自动测试能够给程序员提供迅速的反馈，避免在重构的过程中引入缺陷。所以，应该避免同时重构生产代码和测试代码，以免难以分析出错的原因。

软件开发是一项实践性很强的活动，所有的规则都必须经过实践的考验。这就需要软件开发人员大胆尝试、细心求证，使得 XP 的实践在项目开发中不断得到完善和发展。

3 实例研究

现在我们以某市电信业务综合管理系统为例简要说明一下 XP 若干实践的应用。该项目从1997年启动，到2001年系统才达到基本稳定，而那时早已超过合同期限。项目的开发过程及最后完成情况都不尽如人意。其原因主要有两个。从用户方角度，系统开发期间正值电信改革步伐最大的时期，不管是机构、人员还是业务都有很大变动。开发方在项目初期提出的整体实施方案还没过半年就因为变动太大而再也无人问津。实际上，整个项目除了代码就没有其它有价值的文档。从开发方角度，到项目结束时，初期参与开发的几个核心人员中，只剩下项目经理。造成这种情况的原因，究其根本不外乎一个“变”字。而这种小团队面对快速的业务及环境变化的情形，正是适合以 XP 方式进行软件开发的典型情况。

因而我们以此为蓝本，论述一下如何执行 XP 的实践。该系统的隐喻(metaphor)是：为市局及全市几十个电信分局、支局提供所有电信业务的受理、取消、计费、统计、分派调度和集中监控等功能的计算机综合管理系统。整个系统把电信局所有的生产单位，包括几十个营业厅、市话班、线路班、数据业务班、程控交换班及市局计算机中心联系在一起。由系统按照业务流程的规定对业务进行综合处理。项目开发地在用户方的计算机中心，以后的系统维护也由该部门负责，由该中心的计算机专业人员充当 XP 中现场用户(on-site customer)的角色是最恰当的。规划对策(planning game)的过程是：先由相关

业务部门提出需求，然后由计算机中心集中整理并根据业务的紧迫性和重要性确定优先级，最后以用户故事(user story)的形式提交给开发方。由开发人员估算完成用户故事所需的时间，最后由现场用户决定系统版本的组成。包含最有价值的业务需求的小版本(small releases)对于提高用户满意度和及时获得反馈都很有帮助。由于现场用户具有计算机专业知识，编写功能测试的难度不大。虽然由于系统存在不少GUI类而使得测试很困难，但是考虑到电信业务系统对稳定性、可靠性、准确性的要求甚高，编写相应的单元测试和功能测试还是很有必要的，而且开发的测试套件在其它类似的项目中具有通用性。为了降低GUI类测试的难度，可以把业务逻辑从GUI类中尽可能分离出来，或者求助于GUI的测试工具。在这种人员和业务变动很大的情况下，结对编程的优势就显现出来了。加入团队的新成员在结对中迅速熟悉系统和编程环境，在工作的同时就接受了培训。虽然这会降低结对中熟练程序员的工作效率，但项目和整个团队却因此而受益。专门准备一台计算机做持续集成(continuous integration)，上面存放最新的系统版本和测试用例。每次加入新功能必须首先通过自己编写的单元测试，然后与最新的版本集成并100%通过所有的测试用例，一次集成才算结束。

类似的情况都可以尝试用XP的方式进行开发。但是，在一个项目中引入XP并非易事，除了需要细化以上的实践以外，还需注意逐步推进，尊重所有项目人员的意见并且要根据实际情况对XP进行一些适当的修改[8]，以达到项目开发的最佳状态。

4 XP与传统软件开发方法的比较和分析

为了分析XP的特征和适用性，我们有必要把XP与传统软件开发方法做一下比较。

第一，应用开发方法的前提不同。传统的软件开发方法基于这样的假定：修改程序的成本随时间的推移而以指数的方式上升。而应用XP的前提却是：变化的成本与时间之间不是以指数级的关系而是以多项式的关系上升，甚至逐步趋向一个定值。如果一个项目不具备平滑的变化成本曲线，那么就不适合应用XP的开发方法。

第二，XP之所以称为轻量开发方法是因为相对于传统的开发方法，XP的文档量已经缩到了极限。除了代码以外，唯一的文档就是描述用户故事的索引卡片。轻量是对开发人员而言的，而对用户来说XP并不轻量。在XP中以面对面的口头交流代替文档，具有诸多优点。面对面的直接交流显然要比读写文档的方法更快更透彻地理解交流的内容，而又不会造成误解和歧义。当面的交流直接针对某个问题，而不要在一堆文档中查找，其目的性、针对性更强。总之，口头的交流既降低了开发成本又加快了项目进度。但同时也不可避免地存在易失、可追溯性差的缺点。如果软件的某部分长期没有改动，同时相关的人员变动很大的话，仅仅凭借生产代码和测试用例，就很难获得所需的信息了。因此我们认为需要对XP做一点修改，如果系统的某个部分一定时间没有改动或以后改动的可能性不大，又或者在相关人员变动时，要把关键的信息记录下来。这就需要项目管理人员和开发人员很好地把握时机，既不增加开发成本又要避免口头交流带来的弊端。此外，随着团队规模的不断增长，口头交流就变得越来越不可行了。这也是XP团队保持中小型规模的原因。

第三，传统的开发方法通过各种手段来避免开发过程中的改变。比如，在项目初期把用户需求做得尽可能详细。所以，传统的软件开发过程适用于缓慢变化的商业环境提出稳定需求的情况。但是，用户往往不能在项目开发初期一次性明确地告诉开发人员所有他们想要的东西，而且由于竞争的加剧，他们还时常需要改变业务需求。针对这种情况，XP提出：拥抱变化[1]。由于有一系列的措施在修改的同时保证软件的高质量，XP的开发人员才能迅速应对环境变化和需求变化。对变化采取截然不同的策略的原因是基于前面提到的对变化的成本的不同曲线。

第四，XP的一个独特之处是把设计、编码、测试阶段混合在一起。XP并不像传统开发方法一样强迫团队按照阶段的顺序进行开发，团队可以随时从事肯定是针对一个阶段的任何特定任务，甚至平行进行也可以。换句话说，在XP中水平方向不需要代表一条线性的时间轴[7]。

以上的比较和分析能够使我们更清晰地认识XP的特征和适用性，但是要彻底剖析XP需要更多的来自实践的经验和数据。

结束语 在应对需求模糊或者迅速变化的情况方面，XP开辟了一条新路。XP中的实践并无奇特之处，甚至有些实践在单独使用时存在缺陷，但是当XP把它们发挥到极限并结合在一起，由于相互的支持和补充而形成了强大的合力，给软件开发过程带来了彻底的变革。虽然对XP的研究已取得不少成果，但是作为一种新生的、富有前途的软件开发方法，XP绝非完美无暇，还需要深入地研究它。我们的最终目标是不断完善和发展XP，使XP具有更广泛的适用性，让更多的软件开发和项目管理人员理解并实践XP，推动整个软件产业健康快速的发展。

参考文献

1 Beck K. Extreme Programming Explained: Embrace Change. Addison-Wesley. 1999

2 Highsmith J, Cockburn A. Agile Software Development: The Business of Innovation. IEEE Computer, 2001, 34(9): 120～127

3 Shukla A, Williams L. Adapting Extreme Programming For A Core Software Engineering Course. In: Proc. of the 15th Conf. on Software Engineering Education and Training, 2002. 184～191

4 Müller M M, Tichy W F. Case Study: Extreme Programming in a University Environment. In: Proc. of the 23rd Intl. Conf. on Software Engineering, 2001. 537～544

5 Paulk N C. Extreme programming from a CMM perspective. IEEE Software. 2001, 18(6): 19～26

6 Poole C J, Murphy T, Huisman J W, Higgins A. Extreme Maintenance. In: Proc. of IEEE Intl. Conf. on Software Maintenance, 2001. 301～309

7 Beck K. Embracing change with extreme programming. IEEE Computer, 1999, 32(10): 70～77

8 Nawrocki J, Jasiński M, Walter B, Wojciechowski A. Extreme programming modified: embrace requirements engineering practices. In: Proc. of IEEE Joint Intl. Conf. on Requirements Engineering. 2002. 303～310