敏捷开发

要点浏览

概念：敏捷软件工程是哲学理念和一系列开发指南的综合。这种哲学理念推崇让客户满意和软件的早期增量发布，小而高度自主的项目团队，非正式的方法，最小化软件工程工作产品以及整体精简开发。开发的指导方针强调超越分析和设计（尽管并不排斥这类活动）的发布，以及开发人员和客户之间主动和持续的沟通。

人员：软件工程师和其他项目利益相关者（经理、客户、最终用户）共同组成敏捷开发团队，这个团队是自我组织的并掌握着自己的命运。敏捷团队鼓励所有参与人员之间的交流与合作。

重要性：孕育着基于计算机的系统和软件产品的现代商业环境，正以飞快的节奏不断变化着，敏捷软件工程提出了针对特定类型软件和软件项目的不同于传统软件工程的合理方案。事实证明，这一方法可以快速交付成功的系统。

步骤：敏捷开发恰当的称呼应当是"类软件工程"，它保留了基本的框架活动：客户沟通、策划、建模、构建和部署，但将其缩减到一个推动项目组朝着构建和交付发展的最小任务集（有人认为这种方法是以牺牲问题分析和方案设计为代价而实现的）。

工作产品：接受敏捷理念的客户和软件工程师有着共同的观点：唯一真正重要的工作产品是在合适的时间提交给客户的可运行软件增量。

质量保证措施：如果敏捷团队认为过程可行，并且开发出的可交付软件增量能使客户满意，则表明敏捷方法已经正确实施。

关键概念
自适应软件开发
敏捷过程
敏捷统一过程
敏捷性
Crystal
DSDM
极限编程
FDD
工业极限编程
精益软件开发
结对编程
项目速度
重构
Scrum
故事
极限编程过程

2001年，Kent Beck和其他16位知名软件开发者、软件工程作家以及软件咨询师[Bec01a]（称为敏捷联盟）共同签署了"敏捷软件开发宣言"。该宣言声明：

我们正在通过亲身实践以及帮助他人实践的方式来揭示更好的软件开发之路，通过这项工作，我们认识到：

个人和这些个人之间的交流胜过了开发过程和工具

可运行的软件胜过了宽泛的文档

客户合作胜过了合同谈判

对变更的良好响应胜过了按部就班地遵循计划

也就是说，虽然上述右边的各项很有价值，但我们认为左边的各项具有更大的价值。

一份宣言通常和一场即将发生的破旧立新的政治运动相联系。从某些方面来讲，敏捷开发确实是这样一场运动。

虽然多年来大家一直都在使用着指导敏捷开发的基本思想，但真正将它们凝聚到一场"运动"中还不到二十年。从本质上讲，敏捷方法[⊖]是为了克服传

⊖ 敏捷方法有时也被称为轻量级方法或精简方法。

统软件工程中认识和实践的弱点而形成的。敏捷开发可以带来多方面的好处，但它并不适用于所有的项目、所有的产品、所有的人和所有的情况。它也并不完全对立于传统软件工程实践，也不能作为超越一切的哲学理念而用于所有软件工作。

在现代经济生活中，通常很难甚至无法预测一个基于计算机的系统（如基于网络的应用）如何随时间推移而演化。市场环境飞快变化，最终用户需求不断变更，新的竞争威胁毫无征兆地出现。在很多情况下，项目实施之前，我们无法充分定义需求。因此，我们必须足够敏捷地去响应不断变化、无法确定的商业环境。

不确定性意味着变更，而变更意味着付出昂贵的成本，特别是在其失去控制或疏于管理的情况下。而敏捷方法最具强制性的特点之一就是它能够通过软件过程来降低由变更所引起的代价。

难道说认识到现实的挑战，我们就完全抛弃那些有价值的软件工程原理、概念、方法和工具吗？绝对不是。和其他所有工程学科一样，软件工程也在持续发展着，我们可以通过改进软件工程本身来适应敏捷带来的挑战。

"敏捷：1；其他：0。" —— Tom DeMarco

Alistair Cockburn[Coc02]在他那本发人深省的敏捷软件开发著作中，论证了本书第2章介绍的惯用过程模型中存在的主要缺陷：忘记了开发计算机软件的人员的弱点。软件工程师不是机器人，他们在工作方式上有很大差别，在技能水平、创造性、服从性、一致性和责任心方面也有巨大差异。一部分人可以通过书面方式很好地沟通，而有些人则不行。Cockburn论证说：过程模型可以"利用纪律或者宽容来处理人的共同弱点"，因而大多数惯用过程模型选择了纪律。他还指出："不能始终一致地做同一件事是人性的弱点，因而高度纪律性的方法学非常脆弱。"

要想让过程模型可用，要么必须提供实际可行的机制来维持必要的纪律，要么必须"宽容"地对待软件工程师。显而易见，宽容实践更易于被接受和保持，但是（正如Cockburn所认同的）可能效率低下。正像人生中的大多数事情一样，必须权衡利弊。

3.1 什么是敏捷

就软件工程工作而言，什么是敏捷？Ivar Jacobson[Jac02a]给出了一个非常有用的论述：

敏捷已经成为当今描述现代软件过程的时髦用词。每个人都是敏捷的。敏捷团队是能够适当响应变化的灵活团队。变化就是软件开发本身，软件构建有变化、团队成员在变化、使用新技术会带来变化，各种变化都会对开发的软件产品以及项目本身造成影响。我们必须接受"支持变化"的思想，它应当根植于软件开发的每一件事中，因为它是软件的心脏与灵魂。敏捷团队意识到软件是由团队中所有人共同开发完成的，这些人的个人技能和合作能力是项目成功的关键所在。

在Jacobson看来，普遍存在的变化是敏捷的基本动力，软件工程师必须加快步伐以适应Jacobson所描述的快速变化。

不要误解，以为敏捷会赋予你随意做出拙劣产品的权利。过程还是需要的，而且纪律是必不可少的。

但是，敏捷不仅仅是有效地响应变化，它还包含着对本章开头所提宣言中哲学观念的信奉。它鼓励能够使沟通（组员之间、技术和商务人员之间、软件工程师和经理之间）更便利的团队结构和协作态度。它强调可运行软件的快速交付而不那么看重中间产品（这并不总是好事情）；它将客户作为开发团队的成员以消除一直普遍存在于多数软件项目中的"区分你我"的态度；它意识到在不确定的世界里计划是有局限性的，项目计划必须是可以灵活调整的。

敏捷可以应用于任何一个软件过程。但是，为了实现这一目标，非常重要的一点是：过程的设计应使项目团队适应于任务，并且使任务流水线化，在了解敏捷开发方法的

流动性的前提下进行计划的制定，消除所有最基本软件产品并精简软件开发过程，强调这样一个增量交付策略：根据具体的产品类型和运行环境，尽可能快地将切实可行的软件交付给用户。

3.2 敏捷及变更的成本费用

"敏捷是动态的、针对特定内容的、主动应对变更以及适应增长特点的。"
—— Steven Goldman等

软件开发的传统方法中（有几十年的开发经验作支持）变化的成本费用随着计划的进展成非线性增长（图3-1，实黑曲线）。这种方法在软件开发团队收集需求时（在项目的早期）相对容易适应变化。应用场景需要修改，功能表应该扩充，或者书面说明书需要编辑。这项工作的费用是最小的，所需的时间不会严重影响项目的结果。但是，如果我们在经过数月的开发时间之后将会怎么样？团队在进行确认测试的过程中（也许是在项目后期的某个活动中），一个重要的利益相关者要求变更一个主要的功能。这一变更需要对软件的体系结构设计进行修改，包括设计和构建三个新组件，修改另外五个组件，设计新的测试等。费用会迅速升级，所需的时间和费用完全是为了保证变化不会引起非预期的副作用，而这方面的开销则是可观的。

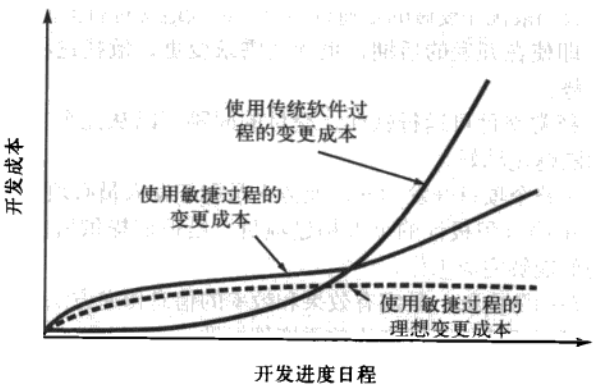

图3-1 变更成本是开发时间的一个函数

KEY POINT
敏捷过程能够降低变更的成本是因为软件产品以增量方式发布，而且在增量内部变更能得到较好的控制。

敏捷的拥护者（例如，[Bec00]，[Amb04]）认为，一个设计良好的敏捷过程"拉平"了变更成本曲线（图3-1，虚线），使软件开发团队在没有超常规的时间和费用影响的情况下，在软件项目后期能够适应各种变化。大家已经学习过，敏捷过程包括增量交付。当增量交付与其他敏捷实践耦合时，例如连续单元测试及结对编程（在本章后面讨论），引起变更的费用会衰减。虽然关于拉平曲线的程度的讨论仍然在进行，但是证据表明[Coc01a]，变更费用显著降低。

3.3 敏捷过程是什么

WebRef
敏捷过程的有关综合文集可在以下网站找到：www.aanpo.org/artides/index。

任何一个敏捷过程都可以由所强调的三个关键假设来识别[Fow02]，这三个假设可适用于大多数软件项目：

1. 提前预测哪些需求是稳定的而哪些需求会变更非常困难。同样，预测项目进行中客户优先级的变更也很困难。

2. 对很多软件来说，设计和构建是交错进行的。也就是，两种活动应当

顺序开展以保证通过构建实施来验证设计模型，而在通过构建验证之前很难估计应该设计到什么程度。

3. 从制定计划的角度来看，分析、设计、构建和测试并不像我们所设想的那么容易预测。

给出这三个假设，同时也就提出一个重要的问题：如何建立能解决不可预测性的过程？正如前文所述，答案就在于过程（对于飞快变化的项目和技术条件）的可适应性。因此，敏捷过程必须具有可适应性。

但是原地踏步式的连续适应性变化收效甚微，因而，敏捷软件过程必须增量地适应。为了达到这一目的，敏捷团队需要客户的反馈（以做出正确的适应性改变），可执行原型或部分实现的可运行系统是客户反馈的最有效媒介。因此，应当使用增量式开发策略，必须在很短的时间间隔内交付软件增量（可执行原型或部分实现的可运行系统）来适应（不可预测的）变更的步伐。这种迭代方法使客户能够：周期性地评价软件增量，向软件项目组提出必要的反馈，影响能够接受反馈的过程的适应性变更。

3.3.1 敏捷原则

尽管敏捷过程能够包容和妥当地处理变更，但我们仍然要认真考察变更的理由。

敏捷联盟[Agi03]为希望达到敏捷的人们定义了12条原则：

1. 我们最优先要做的是通过尽早、持续地交付有价值的软件来使客户满意。

2. 即使在开发的后期，也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。

3. 经常交付可运行软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。

4. 在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。

让软件能够运行是很重要的，但不可忘记还必须使软件具有各种质量属性，其中包括可靠性、可用性以及可维护性。

5. 围绕有积极性的个人构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。

6. 在团队内部，最富有效果和效率的信息传递方法是面对面交谈。

7. 可运行软件是进度的首要度量标准。

8. 敏捷过程提倡可持续的开发速度。责任人（sponsor）、开发者和用户应该能够长期保持稳定的开发速度。

9. 不断地关注优秀的技能和好的设计会增强敏捷能力。

10. 简单（使不必做的工作最大化的艺术）是必要的。

11. 最好的架构、需求和设计出自于自组织团队。

12. 每隔一定时间，团队会反省如何才能更有效地工作，并相应调整自己的行为。

并不是每一个敏捷模型都同等使用这12项原则，一些模型可以选择忽略（或至少淡化）一个或多个原则的重要性。然而，上述原则定义了一种敏捷精神，这种精神在本章中提出的每一个过程模型中得以维护。

3.3.2 敏捷开发的战略

与传统的软件工程过程相反，敏捷软件开发在优越性和适用性方面存在着许多（有时是激烈的）争论。在表达对敏捷拥护者阵营（“敏捷者”）的感想时，Jim Highsmith [Hig02a]（开玩笑地）说明了他那颇为极端的观点：“传统方法学家陷入了误区，乐于生产完美的文档而不是满足业务需要的可运行系统。”而在表述（同样是玩笑性质的）对传统软件工程阵营的立场时，他则给出完全相反的观点：“轻量级方法或者说敏捷方法学家是一群自以为了不起的黑客，他们妄图将其手中的玩具软件放大到企业级软件而制造出一系列轰动。”

在敏捷与软件工程之间做选择不是必须的。自定义一个敏捷软件工程方式是最好的选择。

像所有的软件技术争论一样，这场方法学之争有滑向派别之战的危险。一旦争吵发生，理智的思考就消失了，信仰而不是事实主导着各种决定。

没有人反对敏捷。而真正的问题在于“什么是最佳实现途径”？同等重要的还有，如何构建满足用户当前需要的软件，同时展示具有能满足客户长期需求的扩展能力？

对于这两个问题，还没有绝对正确的答案。即便在敏捷学派内部，针对敏捷问题，也提出了很多有细微差异的过程模型（见3.4节），每个模型内有一组“想法”（敏捷者们不愿称其为“工作任务”），显现出和传统软件工程的显著差异。同时，许多敏捷概念是从优秀的软件工程概念简单地修正而来的。归根结底，兼顾两派的优点则双方都能得到很多好处，而相互诽谤则两败俱伤。

感兴趣的读者可以参看[Hig01]，[Hig02a]和[DeM02]，这些文献针对很多重要技术和方针问题给出了饶有趣味的总结。

3.3.3 人的因素

“敏捷方法源于敏捷性，这种敏捷性取决于体现在团队内部那些不言而喻的知识，而不是写在计划里的知识。”
——Barry Boehm

敏捷软件开发的拥护者们不厌其烦地强调“人的因素”在成功敏捷开发中的重要性。正如Cockburn和Highsmith[Coc01a]所说：“敏捷开发关注个人的才智和技巧，根据特定人员和团队来塑造过程。”这一描述的关键点在于“构造可以满足人员及团队需求的过程模型”，而非其他可选的过程模型[⊖]。

如果敏捷开发团队成员希望努力维护所使用的过程的特性，则该团队成员及团队本身必须具备以下一些特点：

有效的软件团队中，其成员必须具有哪些显著特点？

基本能力。同在传统软件工程中一样，在敏捷开发中，“能力”一词包含了个人内在才能、特定的软件相关技能以及对所选过程的全局知识。关于过程的技能和知识可以而且应该教给敏捷团队的每一位成员。

共同目标。虽然敏捷团队成员能完成不同的任务，为项目提供不同的技能，但是所有人必须瞄准同一个目标，即在承诺的时间内向客户提交可运行的软件增量。为了实现这一目标，项目组还应当做出或大或小的连续的适应性变化，以使过程更适合于团队的需要。

“对一个团队刚刚够用的东西对别的团队来说要么不够用，要么太多。”
——Alistair Cockburn

精诚合作。抛开过程而言，软件工程就是在项目组沟通中评估、分析和使用信息；产生能够帮助所有利益相关者了解项目组工作的信息；构建对客户具有业务价值的软件和相关数据库等信息。为了实现这些任务，项目组成员之间，项目组与所有其他利益相关者之间必须精诚合作。

决策能力。包括敏捷团队在内，任何一个好的软件项目组必须有能够掌握自身命运的自由。这意味着应当赋予项目组在技术和项目问题上的自主决策权。

模糊问题解决能力。软件项目经理应当认识到：敏捷项目组被迫不断面对不确定的事情，被迫不断和变更作斗争。有时，项目组不得不接受今天正在解决的问题明天根本不需解决这样的现实，然而，今后的项目将会从任何解决问题的活动（包括解决错误问题的活动）中学习到经验。

相互信任和尊重。敏捷团队必须成为DeMarco和Lister[DeM98]所说的具有凝聚力的团队（参见第24章），这样的团队展现出的相互信任和尊重使其形成“一个强有力的组织，确保整体的实力大于各部分实力之和”[DeM98]。

⊖ 一些成功的软件工程组织也认识到这一事实，因而忽略他们所选择的过程模型。

KEY POINT
自组织的团队对所承担的工作自行管理。团队做出承诺后制定完成工作的计划。

自组织。自组织在敏捷开发中具有三重含义：（1）敏捷团队组织自身以完成工作；（2）团队组织最能适应当前环境的过程；（3）团队组织最好的进度安排以完成软件增量交付。自组织具有一些技术上的好处，但是更为重要的是它能促进合作，鼓舞士气。本质上，这也就是项目组的自我管理。Ken Schwaber[Sch02]在他的著作中强调以下事情："团队确定他们预期能在迭代内完成多少工作，并承担这些工作。没有什么能比让别人来分派任务更让团队感到沮丧的，也没有什么能让自己负责以履行承诺更让团队倍感鼓舞的了。"

3.4 极限编程

为了更详尽地说明敏捷过程，在此提供一个称为极限编程（eXtreme Programming，XP）的论述，它是敏捷软件开发使用最广泛的一个方法。虽然极限编程相关的思想和方法最早出现于20世纪80年代后期，但具有开创意义的著作由Kent Beck撰写[Bec04a]。近年来，XP的变种，称为工业XP(IXP)被提了出来[Ker05]。IXP细化了XP，目标是在庞大的组织内部使用敏捷过程。

3.4.1 极限编程的权值

Beck[Bec04a]为实施XP的全部工作定义了五个有重要意义的要素，即沟通、简明、反馈、鼓励和尊重。这五个要素中的每一个都是完成特定的XP活动、动作和任务的驱动力。

为了在软件工程师和其他利益相关者之间获得有效的沟通（例如，为软件建立所需的特性和功能），XP强调在用户和开发者之间进行紧密的、非正规的（口头的）合作，建立交流重要理念的有效隐喻[⊖]，连续的反馈，避免以大量的文档作为交流媒介。

ADVICE
只要有可能就应作到简明，不过必须认识到总是做重构会消耗大量的时间和资源。

为了做到简明，XP限制开发者只对即时需求做设计，而不考虑长远需求。这样做的目的是为了使代码设计简单化。如果设计需要改进，那么以后能够实现重构[⊖]。

反馈来自于以下三项：已实现的软件本身、客户和其他软件团队成员。通过设计和完成一个有效的测试策略（第17～20章），软件（通过测试结果）给敏捷团队提供反馈信息。XP使用单元测试作为主要的测试策略。每进行一级开发，开发团队就设计一个单元测试来测试每个操作是否按照规定功能完成。当一个增量提交给客户时，经由增量完成的用户故事或用例（第5章）就作为用于验收测试的一个基础。软件完成输出、功能和用例行为的程度构成了一种反馈。最后，当新需求作为迭代计划的一部分而提出时，团队就把费用和进度影响的反馈信息提供给客户。

"我们能开发多么小的软件，又能开发多么大的软件呢？XP可以给出回答。"
——匿名

Beck[Bec04a]指出与某个XP实践密切相关的就是需要鼓励。更贴切一些的词或许可以称之为纪律。例如，为将来的需求做设计有着显而易见的压力。多数软件团队慑服于此，辨解说"为明天做设计"可以在长跑中节省时间和精力。一个敏捷XP团队必须有为今天做设计的纪律（鼓励），认识到将来的需求可能会有显著的变化，从而需要对设计和已完成代码进行返工。

通过遵循以上这些权值，敏捷团队还应在团队成员之间，在其他利益相关者和团队成员之间，间接地，包括软件本身，灌输相互尊重的思想。当团队成功交付了软件增量时，他们对XP过程的尊重也会增加。

⊖ 在XP中，隐喻就是"每个人（客户、设计人员以及管理者）可以讲述的系统如何工作的故事"[Bec04a]。

⊖ 重构可以让软件工程师在不改变外部功能和行为的情况下改进设计的内部结构（或是源代码）。实际上，重构可以用来提高设计的效能、可读性或性能，或是改进实现设计的代码。

3.4.2 极限编程过程

WebRef
XP规则的很好综述可见于：www.extremeprogramming.org/rules.html。

XP使用面向对象方法作为推荐的开发范型（见本书附录2），它包含了策划、设计、编码和测试4个框架活动的规则和实践。图3-2描述了XP过程，并指出与各框架活动相关的关键概念和任务。下面将概括XP关键的活动。

策划。策划活动（也称为策划比赛）开始于倾听，这是一个需求获取活动，该活动要使XP团队技术成员理解软件的商业背景以及充分感受要求的输出和主要特征及主要功能。倾听产生一系列“故事”（也称为“用户故事”），描述即将建立的软件的需要的输出、特征以及功能。每个故事（类似于第5章讲述的用例）由客户书写并置于一张索引卡上，客户根据对应特征或功能的综合业务价值标明故事的权值（即优先级）[⊖]。XP团队成员评估每一个故事并给出以开发周数为度量单位的成本。如果某个故事的成本超过了3个开发周，则将请客户把该故事进一步细分，重新赋予权值并计算成本。重要的是应注意到新故事可以在任何时刻书写。

什么是XP故事？

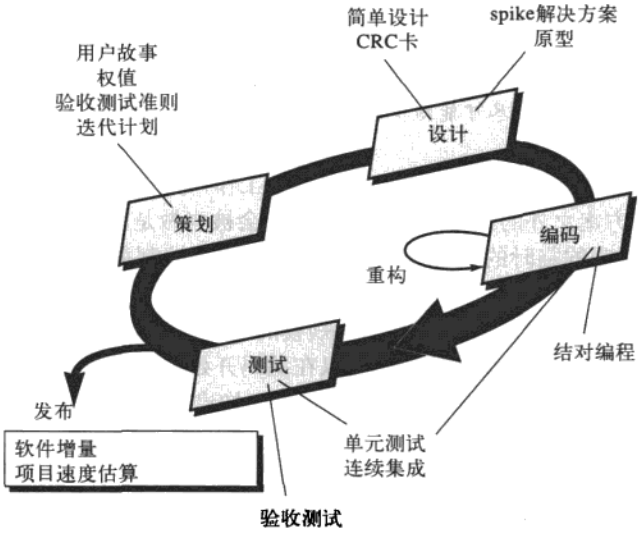

图3-2 极限编程过程

WebRef
很好的“策划比赛”可以参见c2.com/cgi/wiki?planningGame。

客户和XP团队共同决定如何把故事分组并置于XP团队将要开发的下一个发行版本中（下一个软件增量）。一旦认可对下一个发布版本的基本承诺（就包括的故事、交付日期和其他项目事项），XP团队将以下述三种方式之一对有待开发的故事进行排序：（1）所有选定故事将在几周之内尽快实现；（2）具有最高权值的故事将移到进度表的前面并首先实现；（3）高风险故事将首先实现。

KEY POINT
项目速度是团队生产力的精妙度量。

项目的第一个发行版本（也称为一个软件增量）交付之后，XP团队计算项目的速度。简而言之，项目速度是第一个发行版本中实现的客户故事个数。项目速度将用于：（1）帮助估计后续发行版本的发布日期和进度安排；（2）确定是否对整个开发项目中的所有故事有过分承诺。一旦发生过分承诺，则调整软件发行版本的内容或者改变最终交付日期。

⊖ 一个故事的权值也可能取决于其他故事的存在。

在开发过程中，客户可以增加故事，改变故事的权值，分解或者去掉故事。接下来由XP团队重新考虑所有剩余的发行版本并相应修改计划。

设计。XP设计严格遵循KIS（Keep It Simple，保持简洁）原则，即使用简单而不是复杂的表述。另外，设计为故事提供不多也不少的实现原则，不鼓励额外功能性（因开发者假定以后会用到）设计[⊖]。

XP鼓励使用CRC卡（第7章）作为在面向对象环境中考虑软件的有效机制。CRC（类-责任-协作者）卡确定和组织与当前软件增量相关的面向对象的类[⊜]。XP团队使用类似于第8章描述的过程来管理设计工作。CRC卡也是作为XP过程一部分的唯一的设计工作产品。

ADVICE
XP并不强调设计的重要性。这一点不是所有人都会同意的。事实上，有时设计还是应该强调的。

WebRef
重构技术及其工具可在以下站点找到：www.refactoring.com。

如果在某个故事设计中碰到困难，XP推荐立即建立这部分设计的可执行原型，实现并评估设计原型（被称为Spike解决方案），其目的是在真正的实现开始时降低风险，对可能存在设计问题的故事确认其最初的估计。

前一小节提到，XP鼓励既是构建技术又是设计优化方法的“重构”，Fowler[Fow00]描述“重构”如下：

重构是以不改变代码外部行为而改进其内部结构的方式来修改软件系统的过程。这是一种净化代码[并修改或简化内部设计]以尽可能减少引入错误的严格方法。实质上，重构就是在编码完成之后改进代码设计。

因为XP设计实际上不使用符号并且几乎不产生工作产品，如果有的话也只不过是生成除CRC卡和Spike解决方案之外的工作产品，所以设计会被当做是可以并且应当在构建过程中连续修改的临时的人工产品。重构的目的是控制那些由于提出“可以根本改进设计”的小设计修改而造成的（代码）改动[Fow00]。然而应当注意的是，重构所需的工作量随着应用软件规模的增长而急剧增长。

KEY POINT
重构能够改进设计（或源代码）的内部结构，但并未改变其外部功能或行为。

XP的中心观念是设计可以在编码开始前后同时进行，重构意味着设计随着系统的构建而连续进行。实际上，构建活动本身将给XP团队提供关于如何改进设计的指导。

编码。XP推荐在故事开发和初步设计完成之后，团队不是直接开始编码，而是开发一系列用于检测本次（软件增量）[⊜]发布的包括所有故事的单元测试，一旦建立了单元测试[⊗]，开发者就更能够集中精力于必须实现的内容以通过单元测试。不需要加任何额外的东西（KIS，保持简洁）。一旦编码完成，就可以立即完成单元测试，从而向开发者提供即时的反馈。

WebRef
有关XP的有用信息可以在以下网址找到：www.xprogramming.com。

XP编码活动中的关键概念（也是讨论最多的方面）之一是结对编程。XP建议两个人面对同一台计算机共同为一个故事开发代码。这一方案提供了实时解决问题（两个人总比一个人强）和实时质量保证的机制（在代码写出后及时得到复审），同时也使得开发者能集中精力于手头的问题。实施中不同成员担任的角色略有不同，例如，一名成员考虑设计特定部分的编码细节，而另一名成员确保编码遵循特定的标准或者（XP所要求的那些），确保故事相关的代码能够通过相对于故事而开发的单元测试。

? 何谓结对编程？

⊖ 虽然复杂的设计表示和术语可以加以简化，但每一个软件工程方法都应遵循这些设计准则。

⊜ 面向对象类将在本书附录2、第8章及整个第二部分讨论。

⊜ 这一方法类似于在学习开始之前得到考试题目，这样会使学生很容易地将注意力集中于要提的问题。

⊗ 单元测试将于第17章中详细讨论，集中于单个软件构件，检查构件接口、数据结构及功能，其目的在于发现构件内的错误

许多软件团队中聚集了个人主义者。如果要使结对编程工作有效，就必须设法改变团队的文化。

在XP中怎样应用单元测试?

XP验收测试是由用户故事驱动的。

当结对的两人完成其工作，他们所开发的代码将与其他人的工作集成起来。有些情况下，这种集成作为集成团队的日常工作实施。还有一些情况下，结对者自己负责集成，这种“连续集成”策略有助于避免兼容性和接口问题，建立能及早发现错误的“冒烟测试”环境（第17章）。

测试。正如已经指出的，在编码开始之前建立单元测试是XP方法的关键因素。所建立的单元测试应当使用一个可以自动实施的框架（因此易于执行并可重复），这种方式支持代码修改之后即时的回归测试策略（第17章）（会经常发生，为XP提供重构支持）。

一旦将个人的单元测试组织到一个“通用测试集”[Wel99]，每天都可以进行系统的集成和确认测试。这可以为XP团队提供连续的进展指示，也可在一旦发生问题的时候及早提出预警。Wells[Wel99]指出：“每几个小时修改一些小问题，比仅仅在最后截止期之前修正大问题要节省时间。”

XP验收测试，也称为客户测试，由客户规定技术条件，并且着眼于客户可见的、可评审的系统级的特征和功能。验收测试根据本次软件发布中所实现的用户故事而确定。

3.4.3 工业极限编程

Joshua Kerievsky[Ker05]这样描述工业极限编程（IXP）：“IXP是XP的一种有机进化。它由XP的最低限要求、以客户为中心、测试驱动精神组成。IXP与原来的XP的主要差别在于其管理具有更大的包容性，它扩大了用户角色，升级了技术实践。”IXP合并了六个新实践，这些新实践的设计是为了有助于确保在一个庞大的组织内一些重要项目中XP工作成功地实施。

为了生成IXP，在XP上附加了哪些新的实践?

能力是你能够做什么，激励决定了你做什么，而态度决定了你做得怎样。——Lou Holtz

准备评估。在IXP项目开始执行前，组织机构应进行准备评估（readiness assessment）。评估应确定是否（1）存在支持IXP的适合的开发环境，（2）开发团队由合适的利益相关者组成，（3）组织机构具有清晰的质量大纲并且支持连续的改进，（4）组织文化会支持一个敏捷团队的新的权值，（5）组成较为广泛的项目社区。

项目社区。经典XP建议选择适合的人员组成敏捷团队可以确保成功。就是说团队成员必须经过良好的训练，具有良好的适应性和技能，以及适宜的性格为自组织团队做出贡献。当在一个大型组织内将XP应用于重要的项目，团队的概念就变成社区。一个社区可能拥有一个技术专家和处于项目成功核心地位的客户们，以及其他利益相关者（例如，法律人员、质量检验员、生产或销售人员），“他们通常位于IXP计划的周边，但在项目中他们扮演着重要的角色”[Ker05]。在IXP内，应明确定义社区成员和他们的角色，应建立社区成员之间交流和合作机制。

项目承租。IXP团队通过对项目本身进行评估来确定对于项目的合适的商业调整是否存在，以及是否可以进一步深化组织机构的全部目标和目的。承租也要检查项目环境来决定项目如何完成，如何扩展，或者如何替代现在的系统或过程。

测试驱动管理。一个IXP项目需要可测量的标准来评估项目的状态和迄今为止的进展情况。测试驱动管理建立一系列可测量的“目标”[Ker05]，然后定义一些机制来确定目标是否可以实现。

回顾。IXP团队在一个软件增量交付之后要实施特定的技术评审（第15章）。这种评审称为回顾，它在软件增量过程中以及/或者全部软件的发布过程中复查“问题、事件以及教训”[Ker05]。这样做的目的是为了改善IXP过程。

持续学习。由于学习是持续过程改进中至关重要的组成部分，因此，鼓励（可能激励）

XP团队的成员去学习新的方法和技术来提高软件产品质量。

除了以上讨论的六个新实践，IXP还修改了大量已有的XP实践。故事驱动开发（SDD）主张验收测试的故事写在所有代码生成之前。领域驱动设计（DDD）是XP中“系统隐喻”概念的改进。DDD[Eva03]建议渐进建立域模型，“域模型可以精确表示领域专家如何考虑课题”[Ker05]。结对（pairing）扩展了XP结对编程的概念，包括了管理者和其他利益相关者，目的是提高那些可能不直接参与技术开发的XP团队成员间的知识共享程度。迭代可用性（interative usability）并不鼓励前载接口（front-loaded interface）部件设计，其用意在于支持可用性设计，从而有利于软件增量交付以及用户与在研软件的交互。

IXP对其他XP实践进行了少量的修改，并重新确定某些角色和责任，使他们担负起大型组织重要项目的责任。关于IXP的进一步讨论，可以访问http://industrialxp.org。

3.4.4 关于XP的争论

所有的新模型和方法都会刺激有价值的讨论和某些情况下的热烈争论。这些现象对于极限编程来说都发生了。在一本有趣的书中研究了XP的功效，Stephens和Rosenberg[Ste03]认为许多XP实践是有价值的，但是其他的则属炒作过度，少部分是有疑问的。作者提出XP实践的依存性既具有优点也存在缺点。由于许多组织机构采用的只是XP实践的一部分，这就减弱了整个过程的效力。支持者称XP是持续发展的，许多受到批评的问题的产生可以看做是XP实践的成熟。在一直困扰的问题中对XP的批评意见有[⊖]：

是什么问题导致了XP的争论？

- *需求易变。*因为客户是XP团队的成员，对需求的改变不是正式地提出。结果是，项目的范围会发生变化，早期的工作或许得进行修改来适应当前的要求。拥护者认为这种情况的发生是由于不顾所应用的过程以及XP为控制范围渐变的机制所导致的。
- *矛盾的客户需求。*许多项目都有众多客户，每个客户都有自己的一套需求。在XP中，团队自身需要吸纳不同客户的要求，这项工作可能超出了自己的职权范围。
- *需求的非正规表示。*在XP中，用户故事和验收测试是对需求的唯一明确的表现形式。批评者指出需要更为正规的模型或规格说明来保证遗漏、不一致以及错误在系统建立前就被发现。拥护者则认为，需求的变化特性在其发展时使这些模型和规格说明变得过时了。
- *正规设计的缺乏。*XP削弱了对体系结构的设计的要求，在许多案例中，都建议所有类型的设计都不必那么正规。批评者主张当开发复杂的系统时，必须强调设计要保证软件的体系结构能够展示其质量和可维护性。XP的拥护者指出XP过程的增量特性限制了复杂性（简单性是其核心价值），因此降低了扩展设计的必要性。

应该注意到，每一个软件过程都有缺陷，而且许多软件机构已经成功地使用了XP。关键是认识到一个过程的弱点在哪里，然后使其适应于组织机构的特定要求。

考虑敏捷软件开发 SAFEHOME

[场景] Doug Miller的办公室。

[人物] Doug Miller，软件工程经理；Jamie Lazor和Vinod Raman，软件团队成员。

[对话]

（敲门，Jamie和Vinod来到Doug的办公室）

Jamie：Doug，有时间吗？

⊖ 针对XP的评价意见可在以下网站获得：www.softwarereality.com/ExtremeProgramming.jsp。

Doug：当然，Jamie，什么事？

Jamie：我们考虑过昨天讨论的过程了……就是我们打算为这个新的SafeHome项目选什么过程。

Doug：哦？

Vinod：我和在其他公司的一位朋友聊，他告诉我极限编程。那是一种敏捷过程模型，听说过吗？

Doug：听说过，有好处也有坏处。

Jamie：对，看起来很适合我们。可以使软件开发更快，用结对编程来达到实时质量检查……我想这一定很酷。

Doug：它确实有很多实实在在的好主意。比如，我喜欢其中的结对编程概念，还有利益相关者参加项目组的想法。

Jamie：哦？你是说市场部将和项目组一起工作？

Doug（点头）：他们也是利益相关者，不是吗？

Jamie：哇，他们会每隔5分钟就提出一些变更。

Vinod：不要紧。我的朋友说XP项目有包容变更的方法。

Doug：所以你俩认为我们应当使用XP？

Jamie：绝对值得考虑。

Doug：我同意。既然我们选择了增量模型方法，那就没有理由不利用XP带来的好处。

Vinod：Doug，刚才你说“有好处也有坏处”，坏处是什么？

Doug：我不喜欢XP不重视分析和设计……简而言之就是直接编码。（团队成员相视而笑。）

Doug：那你们同意用XP方法吗？

Jamie：（代表二人说）我们干的就是编码！

Doug（大笑）：没错，但我希望看到你花少量时间编码和重新编码，而花多一点时间分析我们应当做什么并设计一个可用系统。

Vinod：或许我们可以二者兼用，带有一定纪律性的敏捷。

Doug：我想我们能行，Vinod，实际上我坚信这样的做法没问题。

3.5 其他敏捷过程模型

“我们这个专业实施方法论就如同14岁的少年穿衣服一样，还不成熟。”
—— Stephen Hawrysh和Jim Ruprecht

软件工程的历史是由散乱着的几十个废弃的过程描述和方法学、建模方法和表示法、工具以及技术所构成，每一个都是轰轰烈烈地冒出来，接着又被新的（期望是）更好的所替代。随着敏捷过程模型的大范围推广，每一种模型都在争取得到软件开发界的认可，敏捷运动正在遵循着同样的历史步伐[⊖]。

就像前一节提到的，在所有敏捷过程模型中使用最广泛的就是XP。但是也提出许多其他敏捷过程模型，并且也在行业中使用。最普遍的有：

- 自适应软件开发（ASD）
- Scrum
- 动态系统开发方法（DSDM）

⊖ 这并不是坏事。在某个模型或方法被当做事实上的标准之前，都在尽力争取软件工程师群体的人心。最终胜利者将发展成为最佳实践，而失败者将销声匿迹或是被融入取胜的模型。

- Crystal
- 特征驱动开发（FDD）
- 精益软件开发（LSD）
- 敏捷建模（AM）
- 敏捷统一过程（AUP）

在以下的几节中，我们将简要介绍这几个敏捷过程模型。一个重要的说明是，所有敏捷方法（或多或少地）都遵循敏捷软件开发宣言以及3.3.1节中提到的那些原则。更为详尽的细节可参考在每个小节或评述中的参考资料，进入Wikipedia参看“敏捷软件开发”[⊖]。

3.5.1 自适应软件开发

WebRef

ASD方面的材料可见于：www.adaptivesd.com。

自适应软件开发（Adaptive Software Development，ASD）是由Jim Highsmith[Hig00]提出的，它可作为构建复杂软件和系统的一项技术，其基本概念着眼于人员协作和团队自我组织。

Highsmith认为一个基于协作的敏捷、自适应性开发方法是“我们复杂交互作用中如同纪律和工程的秩序之源”，他给ASD“生命周期”（图3-3）的定义包含思考、协作和学习三个阶段。

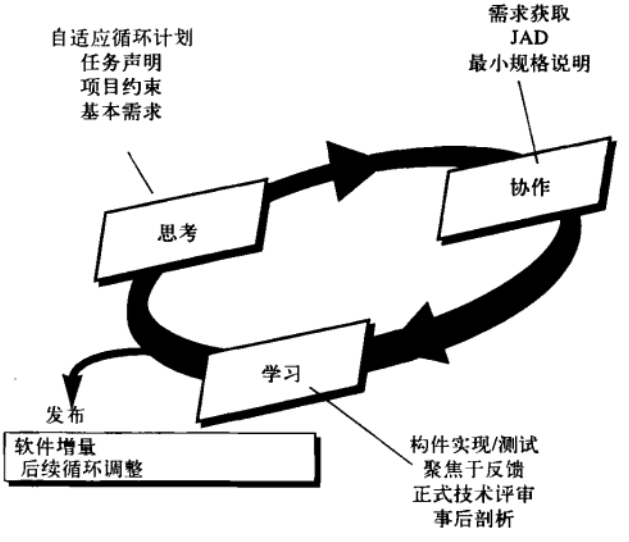

图3-3 自适应软件开发

只是在抛弃了“我们和他们”的观念后，与客户的有效协作关系才会建立起来。

思考阶段中，启动项目并完成自适应周期策划。自适应周期策划通过使用项目启动信息——客户任务描述、项目约束（如交付日期或用户描述）和基本需求——来确定项目所要求的一套软件增量发布周期。

无论是多么有远见和完整的周期计划，总是要发生变化的。基于第一个周期完成后所获得的信息，对计划进行评审和调整，使计划可以更好地适应一个ASD团队正在工作的现状。

有积极性的人员以超越其聪明才智和独创的方式共同工作，协作方法是所

⊖ 请参考http://en.wikipedia.org/wiki/Agile_software_development#Agile_methods。

有敏捷方法中不断重现的主旋律。但协作并不容易。它不仅强调沟通和团队合作，而且也不排斥个人的作用，因为个人的创造力在协作思考中起着重要作用。更重要的是信任，一起工作的人们必须相互信任，才能够：（1）毫无恶意地做出评论；（2）毫无怨言地相互帮助；（3）尽最大努力工作；（4）拥有解决手头工作的技能；（5）用有效的方式沟通问题和事务。

KEY POINT
ASD强调把学习当做达到"自我组织"团队的关键元素。

当ASD团队成员开始开发构成自适应周期的构件时，其重点是朝着完成周期的方向学习尽可能多的东西。事实上，Highsmith[Hig00]认为软件开发人员常常高估自己（对技术、过程和项目）的理解力，这样的学习将帮助他们改进其真正的理解水平。ASD团队通过以下三种方式学习：客户反馈意见（第5章），技术评审（第15章），事后剖析。

ASD理念无论其使用什么过程模型都具有重要的存在价值。ASD整体上强调软件项目团队具有自我组织的动态性、人与人的协作、个人以及团队的学习，从而使团队更有可能取得成功。

3.5.2 Scrum

WebRef
关于Scrum的有用信息和资源可见于：www.controlchaos.com。

Scrum（得名于橄榄球比赛[⊖]）是Jeff Sutherland和他的团队在20世纪90年代早期发展的一种敏捷过程模型，近年来，Schwaber和Beedle[Sch01a]对其做了进一步的发展。

Scrum原则与敏捷宣言是一致的，应用Scrum原则指导过程中的开发活动，过程由"需求、分析、设计、演化和交付"等框架性活动组成。每一个框架活动中，发生于一个过程模式（在下文中讨论）中的工作任务称为一个冲刺（sprint）。冲刺中进行的工作（每一个框架活动中的冲刺的数目根据产品复杂度和规模大小而有所不同）适应于当前的问题，由Scrum团队规定并常常作实时修改。Scrum过程的全局流程如图3-4所示。

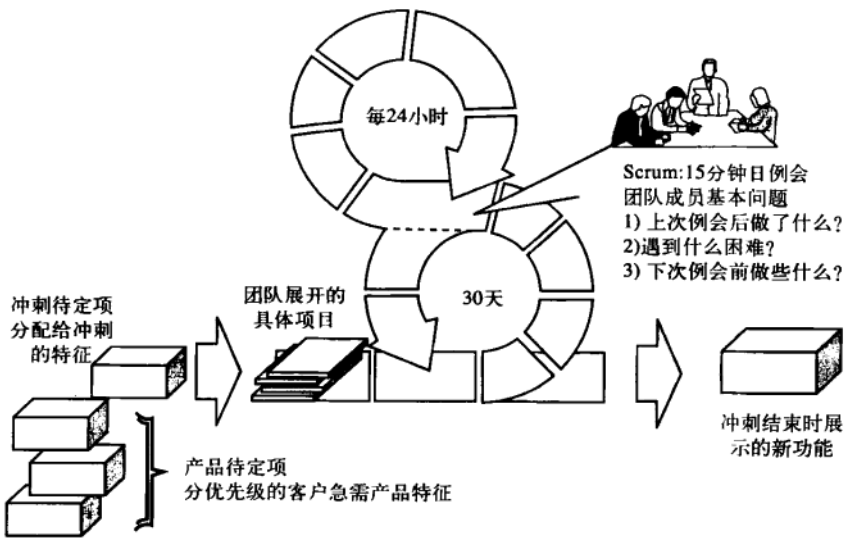

图3-4 Scrum过程流

Scrum强调使用一组"软件过程模式"[Noy02]，这些过程模式被证实在时间紧张的需求

⊖ 一组球员围着球共同努力（有时具有暴力性），带球扑地。

KEY POINT
Scrum是由含有以下内容的一组过程模式构成：强调项目优先次序、分离的工作单元、沟通以及频繁的客户反馈。

变化的和业务关键的项目中是有效的。每一个过程模式定义一系列开发活动：

待定项（backlog）——一个能为用户提供商业价值的项目需求或特征的优先级列表。待定项中可以随时加入新项（这就是变更的引入）。产品经理根据需要评估待定项并修改优先级。

冲刺（sprint）——由一些工作单元组成，这些工作单元是达到待定项中定义的需求所必需的，并且必须能在预定的时间段（time-box[⊖]）内（一般情况下为30天）完成。冲刺过程中不允许有变更（例如，积压工作项）。因此，冲刺给开发团队成员的工作提供了短期但稳定的环境。

Scrum例会——Scrum团队每天召开的短会（一般情况为15分钟），会上所有成员要回答三个问题[Noy02]：

- 上次例会后做了什么？
- 遇到了什么困难？
- 下次例会前计划做些什么？

团队领导（也称为Scrum主持人）主持会议并评价每个团队成员的表现。Scrum会议帮助团队尽早发现潜在的问题。同时，每日例会导致“知识社会化交流”[Bee99]并进一步促进自我组织团队的建设。

演示——向客户交付软件增量，这可以试用并评价向用户演示所实现的功能并由客户对其评价。需提醒的很重要的一点是，演示不需要包含计划内的所有功能，但是规定该时间段内的可交付功能必须完成。

Beedle和他的同事[Bee99]在他们所发表的关于这些模式的综合讨论中提到：“Scrum预先假定混乱的存在……”。Scrum过程模式保证软件开发团队在无法消除不确定的世界里能成功地工作。

3.5.3 动态系统开发方法

WebRef
有关DSDM的有用信息见于：www.dsdm.org。

动态系统开发方法（Dynamic System Development Method，DSDM）[Sta97]是一种敏捷软件开发方法，该方法提供一种框架，使其“通过在可控项目环境中使用增量原型开发模式完全满足对时间有约束的系统的构建和维护”[CCS02]。DSDM建议借用修改版Pareto原则的哲学观念。这种情况下，如果交付整个应用系统需用100%时间，那么80%的应用系统可以用20%的时间交付。

DSDM使用迭代软件过程，每一个迭代都遵循80%原则，即每个增量只完成能够保证顺利进入下一增量的工作，剩余的细节则可以在知道更多业务需求或者提出并同意变更之后完成。

DSDM协会（www.dsdm.org）是一个由一些共同充当DSDM方法管理者的成员公司所组成的全球性组织。协会定义了一个称为DSDM生命周期的敏捷过程模型。该生命周期定义了3个不同的迭代周期，前面还加了两个生命周期活动：

可行性研究——建立要开发应用系统的业务需求和相关约束，并评估该应用系统采用DSDM过程是否可行。

业务研究——建立能为应用系统提供业务价值所需要的功能和信息需求；同时，确定基本的应用系统架构并识别软件的可维护性需求。

功能模型迭代——为客户开发一系列证明其功能的增量原型（注意：所有DSDM原型都倾向于逐渐发展成可交付的应用系统）。这一迭代的意图是在用户使用原型系统时诱导出反馈信

⊖ Time-box是一个项目管理术语（见本书第4部分），指的是分配给完成某一任务的时间段。

息以获取其他的需求。

设计和构建迭代——在功能模型迭代中，重新构建原型以确保每一个原型都以工程化方式实现，并能为最终用户提供可操作的业务价值。有些情况下，功能模型迭代、设计和构建迭代可同步进行。

实现——将最终软件增量（一个可操作的原型）置于操作环境中。应当注意：（1）增量不见得100%完成；（2）增量置于操作环境以后可能需要改变。在这两种情况下，DSDM开发转向功能模型迭代后继续进行。

KEY POINT
DSDM是一个过程框架，它可采用其他敏捷方法（如XP）的策略。

DSDM和XP(3.4节)可以结合使用，这种组合方法用具体实践（XP）定义固定的过程模型(DSDM生命周期)，这些具体实践是构建软件增量所必需的。另外，ASD协作和自我组织团队的概念也可运用于这种组合过程模型。

3.5.4 Crystal

KEY POINT
Crystal是一个具有相同“遗传密码”的过程系列，只是针对项目的特点采用了不同的方法。

Alistair Cockburn[Coc05]和Jim Highsmith[Hig02b]建立了Crystal敏捷方法系列[⊖]，其目的是发展一种提倡“机动性的”软件开发方法，Cockburn将软件开发刻画为：“一种资源有限、合作完成的发明和交流活动，其首要目标是交付有用的、可工作的软件，其第二目标是为下一次行动做准备”[Coc02]。

为实现机动性，Cockburn和Highsmith定义了一系列方法学，它们包含具有共性的核心元素，以及独一无二的角色、过程模式、工作产品和实践。Crystal系列实际上是一组经过证明对不同类型项目都非常有效的敏捷过程。它的目的是使得敏捷团队可以根据其项目和环境选择最合适的Crystal系列成员。

3.5.5 特征驱动开发

特征驱动开发（Feature Driven Development, FDD）最初由Peter Coad及其同事[Coa99]作为面向对象软件工程的实用过程模型而构思的。Stephen Palmer和John Felsing[Pal02]扩展并改进了Coad的工作，描述了一个可用于中、大型软件项目的适应性敏捷过程。

WebRef
有关FDD的许多文章和文献可见于：www.featuredriven-development.com/。

像其他敏捷方法一样，FDD采用这样一个基本原理：（1）重点强调在FDD团队成员间的合作；（2）使用基于特征分解随后集成软件增量的方法管理问题和计划复杂性，并且（3）使用口头的、图解的以及基于文本的方法交流技术细节。FDD重点强调通过鼓励增量开发策略、使用设计和代码检查、应用软件质量保证审查（第16章）、收集度量、使用模板等活动（用于分析，设计和构建）来确保软件质量。

在FDD环境中，特征“是可以在2周或更短时间实现的具有客户价值的功能”[Coa99]。强调特征的定义是为了如下好处：

- 特征是小块可交付功能，用户可以更容易地对其进行描述、轻松地理解他们之间的相互关系，更好地评审以发现歧义性、错误和遗漏。
- 特征可以组织为具有层次关系的业务相关的分组。
- 由于特征是FDD可以交付的软件增量，团队每两周便可开发出可供使用的特征。
- 由于特征很小，其设计和代码表示都可以很容易、很有效地检查。
- 项目计划、进度和跟踪都由特征层次驱动，而不是可任意调整的软件工程任务集。

Coad及其同事[Coa99]建议使用以下模板定义特征：

<action> the **<result>** **<by | for | of | to>** a(n) **<object>**

⊖ Crystal（水晶）这一名称是取自地质学水晶的特征，每一种水晶都有独特的颜色、形状和硬度。

这里的<object>是“一个人、地点或事件（包括角色、时刻或时间间隔或者类似目录项的描述）”。例如，一个电子商务应用项目的特征可能如下所示：

将产品加入购物卡

显示产品详细技术说明

为顾客存储购物信息

一个特征集将相关特征分在一个业务相关的类别中，定义[Coa99]如下：

<action><-ing> a(n) <object>

例如，出售一件商品（Making a product sale）是一个特征集，它包含上面提到的及其他特征。

FDD方法定义五种“协作”[Coa99]框架活动（FDD中称为“过程”），如图3-5所示。

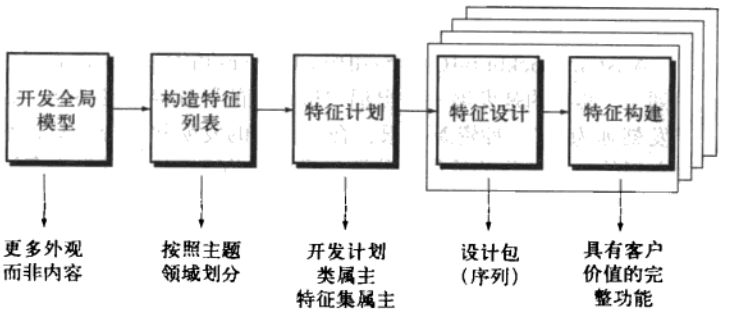

图3-5 特征驱动开发[Coa99]（经过授权）

和其他敏捷方法相比，FDD更加强调项目管理原则和技术。随着项目规模和复杂度增长，项目管理常常是不充分的，对于开发者、管理者和其他的利益相关者而言，非常有必要理解项目状态（已经完成了什么，遇到了什么问题）。如果最后期限压力很大，则确定软件增量（特征）是否能如期完成非常重要。FDD在特征设计和实现阶段定义了6个里程碑以达到上述目标，分别为：“设计走查、设计、设计检查、编码、代码检查、促进构建”[Coa99]。

3.5.6 精益软件开发

精益软件开发（Lean Software Development，LSD）在软件工程领域适应于精益制造的原则。精益原则鼓励LSD过程（[Pop03]，[Pop06a]）消除耗损、把质量体现于产品、创造知识、遵从承诺、快速交付、尊重成员以及整体优化。

每一个原则都适应于软件过程。例如，在一个敏捷软件计划中消除耗损可以解释为[Das05]：（1）不加入无关的特性和功能，（2）评估任何一个新的需求对费用和进度的影响，（3）去掉任何多余的过程步骤，（4）建立能改进团队成员获取信息渠道的机制，（5）保证测试能够发现尽可能多的错误，（6）缩短获取需求所需的时间，以及缩短获得影响软件或影响创建软件的步骤的结论的时间，（7）将信息传送给过程中所有利益相关者的方式流线化。

对LSD细节以及完成过程的实际的指导原则的讨论，可以参看[Pop06a]和[Pop06b]。

3.5.7 敏捷建模

很多情况下，软件工程师必须构建大型的、业务关键型的系统，这种系统的范围和复杂性必须通过建模方式保证以下事项：（1）所有参与者可以更好地理解要做什么；（2）有效地将问题分解给要解决它的人；（3）对正在设计和构建的系统质量进行评估。

WebRef
有关敏捷建模更为全面的信息可见于：www.agilemodeling.com。

过去的30多年来，人们提出了许多用于分析和设计（架构和构件级）的软件工程建模方法和表示方法，这些方法有很重要的价值，但事实证明它们都很难使用也很难维持（尤其在跨多个项目时），部分问题则是建模方法的“负担”，这里的负担指所需符号的数量、所建议采用模型形式化的程度、用于大型项目时模型的大小和变更发生时维护的难度。就算没有其他原因，仅仅考虑项目智能化的可管理性，分析和设计建模对大型项目就具有重要意义。敏捷方法软件工程建模能否为解决这些问题提供可选方案呢？


几天前我去药店打算买些感冒药，可是不容易啊。看到有太多的药可选，往前走走，是一种速效的，还有一种是长效的。究竟选哪个好呢？是图眼前还是图长远呢？——Jerry Seinfeld

在敏捷建模官方网站上，Scott Ambler[Amb02a]用以下方式描述敏捷建模（Agile Modeling，AM）：

> AM是一种用于对基于软件的系统实施有效建模和文档编制的基于实践的方法学。AM是可以有效并以轻量级方式用于软件开发项目中软件建模的标准、原则和实践。由于敏捷模型只是大致完善，而不要求完美，因此敏捷模型比传统的模型更有效。

AM采纳了与敏捷宣言一致的全部标准。敏捷建模的指导思想认为，一个敏捷团队必须有做出可能导致否决设计和重新构建等决定的勇气，也必须有意识到技术不能解决所有问题、应当尊重和采纳业务专家和其他相关人员意见的谦逊作风。

虽然AM提出一系列的“核心”和“补充”建模原则，但其中独具特色的是[Amb02a]：


对所有软件工程工作而言，“轻装”都是适合的指导思想。我们只需要那些有价值的模型，不要多，也不要少。

有目的的模型。在构建模型之前，使用AM的开发者心中应当有明确的目标（如与客户沟通信息，或有助于更好地理解软件的某些方面），一旦确定模型的目标，该用哪种类型的表达方式以及所需要的具体细节程度都是显而易见的。

使用多个模型。描述软件可以使用多种不同的模型和表示法，大多数项目只用到其中很小的部分就够了。AM建议从需要的角度看，每一种模型应当表达系统的不同侧面，并且应使用能够为那些预期的读者提供有价值的模型。

轻装上阵。随着软件工程工作的进展，只保留那些能提供长期价值的模型，抛弃其余的模型。保留下来的每一个工作产品都必须随着变更而进行维护，这些描述工作将使整个团队进度变慢。Ambler[Amb02a]提示说“每次决定保留一个模型，你都要在团队中以抽象方式使用信息的便利性与敏捷性方面做权衡（即团队内部、团队与利益相关者增强沟通）。

内容重于表述形式。建模应当向预期的读者分享重要的信息。一个有用的内容很少，但语法完美的模型不如一个有缺陷但能向读者提供有用内容的模型更有价值。

理解模型及工具。理解每一个模型及其构建工具的优缺点。

适应本地需要。建模方法应该适应敏捷团队的需要。

当前软件工程领域界大都已采用了统一建模语言（UML）[⊖]作为首选的方法来表示分析和设计模型。统一过程（第2章）已经为UML提供了一个框架。Scott Ambler[Amb06]已经开发出集成了其敏捷模型原理的UP的简单版本。

3.5.8 敏捷统一过程

敏捷统一过程（Agile Unified Process，AUP）采用了一个“全局串行”以及“局部迭代”[Amb06]的原理来构建基于计算机的系统。采用经典UP阶跃性活动——开始、加工、构建以

⊖ 在本书附录1中提供了UML的简明教程。

及变迁，UP提供一系列覆盖（例如，软件工程活动的一个线性序列），能够使团队为软件项目构想出一个全面的过程流。然而，在每一个活动里，一个团队迭代使用敏捷，并且将有意义的软件增量尽可能快地交付给最终用户。每个AUP迭代执行以下活动[Amb06]：

建模。UML建立了对商业和问题域的表述。然而，为了保持敏捷，这些模型应当“足够好”，[Amb06]使团队继续前进。

- 实现。将模型翻译成源代码。
- 测试。像XP，团队设计和执行一系列的测试来发现错误以保证源代码满足需求。
- 部署。就像第一章和第二章中讨论过的一般过程活动，这部分的部署重点仍然是对软件增量的交付以及获取最终用户的反馈信息。
- 配置及项目管理。在AUP中，配置管理（第22章）着眼于变更管理、风险管理以及对开发团队的任一常效产品[⊖]的控制。项目管理追踪和控制开发团队的活动情况和工作进展。
- 环境管理。环境管理协调过程基础设施，包括标准、工具以及适用于开发团队的支持技术。

虽然AUP与统一建模语言有历史上和技术上的关联，但是很重要的一点必须注意，UML模型可以与任一敏捷过程模型相结合（3.5节）。

SOFTWARE TOOLS

敏捷开发

目的：敏捷开发工具的目标是辅助软件开发一个或多个方面，强调便利地快速构建可执行软件。这些工具也可以用于惯用（传统）过程模型（见第2章）的开发。

机制：工具的机制各不相同。通常，敏捷工具集包括项目计划、用例开发和需求收集、快速设计、代码生成和测试的自动支持。

代表性工具：[⊜]

注：由于敏捷开发是一个热门话题，大多数软件工具供应商都声称出售支持敏捷方法的工具。下面工具具有的特性对敏捷项目非常有用。

OnTime，由Axosoft开发（www.axosoft.com），提供对各种技术活动的敏捷过程管理支持。

Ideogramic UML，由Ideogramic开发（www.ideogramic.com），是特别为敏捷过程开发的UML工具集。

Together Tool Set，由Borland销售（www.borland.com），提供支持XP和其他敏捷过程中许多技术活动的工具包。

3.6 敏捷过程工具集

KEY POINT 这里所提到的敏捷过程“工具集”更多地是从人员的方面，而不是从技术方面支持敏捷过程。

敏捷哲学的拥护者指出，自动软件工具（例如，设计工具）应当被看做是对开发团队活动小小的补充，而不是团队成功的关键。然而，Alistair Cockburn[Coc04]建议，工具是有益处的，“敏捷团队强调使用工具可以达到快速理解的目的。有些工具是社会性的，甚至开始于租赁阶段。有些工具是技术性的，可以帮助使用者团队模拟物理现状。很多工具是物理性的，允许人们在工作场所操作这些工具。”

由于找到工具需要的人（租用者）、团队内部的合作、利益相关者间的相

⊖ 常效工作产品指的是被管理的团队在不同阶段开发出的模型、文档或测试用例，这些产品在软件增量交付后并不废弃。

⊜ 这里提到的工具只是此类工具的例子，并不代表本书支持采用这些工具。在大多数情况下，工具名称被各自的开发者注册为商标。

互沟通以及间接管理，事实上都是敏捷过程模型的关键元素，因此Cockburn指出，用于解决这些问题的“工具”是敏捷的至关重要的成功因素。例如，一个租用的“工具”或许正是寻找一个期望的团队成员花极少的时间与一个现有团队成员结对编程所需要的。这种“适合”能够立即判定。

协作和沟通“工具”通常技术含量较低并且与可以提供信息以及协调敏捷开发人员的任何机制相结合（“这些机制可以是物理上的近距离性、白色书写板、海报、索引卡以及可粘贴留言条”）[Coc04]。积极的沟通是通过团队能动性获得的（例如，结对编程），而被动的沟通是通过“信息辐射体”实现的（例如，一台平面显示器可以显示一个增量不同组件的全部状态）。项目管理工具降低了Gantt（甘特）图的重要性，使用挣值图（earned value chart）来取代Gantt图或“对过去增量测试图……还有其他工具可用来优化敏捷团队工作的环境（例如，更有效的会议区），通过培养社交互动（例如，配置团队）、物理设备（例如，电子白色书写板）、过程增强（例如，结对编程或时间盒）”[Coc04]等提高团队文化。

所有这些都是工具吗？如果它们能够促进敏捷团队成员的工作以及提高最终产品的质量，它们就是工具。

3.7 小结

在现代经济中，市场条件变化十分迅速，客户和最终用户的要求在演变，新一轮竞争威胁会没有任何征兆地出现。从业者必须使软件工程工作保持敏捷——要限定过程应是灵活机动的、有适应能力的和精益的以适应现代商务的需求。

软件工程的敏捷理念强调4个关键问题：自我组织团队对所开展工作具有控制力的重要性；团队成员之间以及开发参与者与客户之间的交流与合作；对“变更代表机遇”的认识；以及强调快速软件交付以让客户满意。敏捷过程模型能解决上述这些问题。

极限编程（XP）是应用最广泛的敏捷过程。按照计划、设计、编码和测试四个框架活动组织，XP建议一系列新颖和有力的技术，保证敏捷团队创建能体现客户指定优先级特征和功能的频繁软件发布。

其他敏捷过程模型也强调人员合作和团队自组织，只是定义自己的框架活动，选择不同的侧重点。例如，ASD使用迭代过程，该过程由自适应周期计划、相对严格的需求收集方法和一个迭代开发周期构成，其中迭代开发周期包括客户关注点组和正式技术评审作为实时反馈机制。Scrum强调一系列软件过程模式的使用，这些模式已被证实对时间紧迫、需求变更和业务重要的项目非常有效。每一个过程模式定义一系列开发任务并允许Scrum团队以适应其项目要求的方式构建过程。动态系统开发方法（DSDM）倡导时间调度的使用，认为对每一个软件增量所需的必要工作仅仅是能够方便地进入下一次增量开发。Crystal是一系列敏捷过程模型，可用于具有特定特征的项目。

特征驱动开发（FDD）在某种程度上比其他敏捷方法更“形式化”，但是通过使项目开发团队专注于特征的开发来维持敏捷性——可在2周或更短时间内实现对客户有价值的功能。精益软件开发（LSD）是在软件工程界采用精益制造的原则。敏捷建模（AM）认为建模对于所有的系统都是必要的，但是模型的复杂度、类型和规模必须根据所构建的软件来调节。敏捷统一过程（AUP）采用“全局串行”以及“局部迭代”的原则来构建软件。

习题与思考题

3.1 重新阅读一遍本章开头的“敏捷软件开发宣言”，你能否想出一种情况，此时4个权值中的一个或多

个将使软件开发团队陷入麻烦?

3.2 用自己的语言描述（用于软件项目的）敏捷性?

3.3 为什么迭代过程更容易管理变更？是不是本章所讨论的每一个敏捷过程都是迭代的？只用一次迭代就能完成项目的敏捷过程是否存在？解释你的答案。

3.4 是否每一个敏捷过程都可以用第2章所提及的通用框架性活动来描述？建一张表，将通用活动和每个敏捷过程所定义的活动对应起来。

3.5 试着再加上一条“敏捷性原则”，以便帮助软件工程团队更具有机动性。

3.6 选择3.3.1节提到的一条敏捷性原则，讨论本章所描述的各过程模型是否符合该原则。[注意：我只是给出了这些过程模型的一个概述，因此，确定一个或多个模型是否符合某个原则或许不可能，除非你做过额外的研究（对这个问题并不需要）。]

3.7 为什么需求变化这么大？人们终究无法确定他们想要什么吗?

3.8 许多敏捷过程模型推荐面对面交流，实际上，现在软件开发团队成员及其客户在地理上是相互分散的。你是否认为这意味着这种地理上的分散应当避免？能否想出一个办法克服这个问题。

3.9 写出一个描述多数网页浏览器所具有的“我的最爱”或“书签”特征的XP用户故事?

3.10 什么是XP的“Spike解决方案”？

3.11 用自己的语言描述XP的重构和结对编程的概念。

3.12 增加阅读，然后描述什么是时间盒？它是如何帮助ASD团队在短时期内传递软件增量的?

3.13 80%的DSDM原则和ASD定义的时间盒可以得到相同的结论吗?

3.14 使用第2章描述的过程模式模板，为3.5.2节所描述的任一Scrum模式开发一种过程模式。

3.15 为什么称Crystal是一个敏捷方法系列?

3.16 使用3.5.5节所描述的FDD特征模板，定义Web浏览器的特征集，并为该特征集开发一系列特征。

3.17 访问敏捷建模官方站点（www.agilemodeling.com），给出所有核心和补充AM原则的列表。

3.18 3.6节中给出的工具集为敏捷方法的“软件”方面提供了很多支持。由于交流非常重要，推荐一种实际工具集可以用来增强敏捷团队中客户间的交流。

推荐读物与阅读信息

敏捷软件开发的全部理论和基本原则在本章提供的很多参考书中都有更深层的论述。另外，在Shaw和Warden（《The Art of Agile Development》，O'Reilly Media,Inc.,2008），Hunt（《Agile Software Construction》，Springer，2005），以及Carmichael和Haywood（《Better Software Faster》，Prentice-Hall，2002）的书中给出了关于主题的有用的讨论。在Aguanno（《Managing Agile Project》，Mult-media Publications，2005），Highsmith（《Agile Project Management: Creating Innovation Products》，Addison-Wesley,2004),以及Larman（《Agile and Iterative Development: A Manager's Guide》，Addison-Wesley,2003)的书中,给出了关于管理的概述及项目管理问题的思考。在 Highsmith （《Agile Software Development Ecosystem,》 Addison-Wesley,2002) 提出了一份关于敏捷原则、过程和实践的调查。由Booch及其同事的书中（《Balancing Agile and Discipline》，Addison-Wesley,2004),给出了关于敏捷和规则间平衡的颇有价值的讨论。

Martin（《Clean Code：A Handbook of Agile Software Craftmanship》，Prentice-Hall, 2009）指出了在敏捷软件工程环境下开发“简洁程序代码”所需敏捷原则、模式和实践。Leffingwell（《Scaling Software Agility：Best Practices for Large Enterprises》，Addison-Wesley, 2007）讨论了在大型计划中按比例提高敏捷实践的策略。Lippert 和Rook（《Refactoring in Large Software Projects：Performing Complex Restructurings Successfully》，Wiley，2006）讨论了在大型、复杂系统中重构的使用。Stamelos

和Sfetsos（《Agile Software Development Quality Assurance》，IGI Global，2007）讨论了符合敏捷理论的SQA技术。

在过去十年间，很多书中描述了极限编程。Beck（《Extreme Programming Explained：Embrace Change》，2d ed. Addison-Wesley, 2004）的书是论述得比较全面的。另外，Jeffries及其同事（《Extreme Programming Installed》，Addison-Wesley, 2000）、Succi和Marchesi（《Extreme Programming Examined》，Addison-Wesley, 2001）、Newkirk和Martin（《Extreme Programming in Practice》，Addison-Wesley, 2001）、Auer及其同事（《Extreme Programming Applied: Play to Win》，Addison-Wesley, 2001）的著作中，就如何更好地应用XP给出了关键的有指导意义的讨论。McBreen（《Questioning Extreme Programming》，Addison-Wesley, 2003）则以挑剔的眼光审视XP，定义了何时、何地更为适用。对结对编程的深入考虑可阅读McBreen（《Pair Programming Illuminated》，Addison-Wesley, 2003）。

Highsmith[HIG00]深入强调了ASD。Schwaber（《The Enterprise and Scrum》，Microsoft Press。2007）讨论了对主营业务产生影响的计划中使用Scrum。Schwaber及Beedle（《Agile Software Development with SCRUM》，Prentice-Hall, 2001）发表了对Scrum方法的深入探讨。关于DSDM方法的有价值的论述可以在DSDM Consortium（《DSDM: The Method in Practice》，2nd ed., Person Education,2003）及Stapleton（《DSDM: The Method in Practice》，Addison-Wesley, 1997）的书中找到。Cockburn（《Crystal Clear》，Addison-Wesley, 2005）对过程中的Crystal系列进行了精彩的论述。Palmer和Felsing[PAL02]发表了关于FDD的详细研究，Carmichael和Haywood（《Better Software Faster》，Prentice-Hall, 2002）发表了另一项关于FDD的研究，其中包含了对该过程机制的一步步的描述。Poppendieck和Poppendieck（《Lean Development: An Agile Toolkit for Software Development Managers, Addison-Wesley, 2003）给出管理和控制敏捷项目的指导。在Ambler和Jeffries（《Agile Modeling》，Wiley, 2002）深入探讨了AM。

Internet网上可以获得关于敏捷软件开发的更为广泛的信息资源。在SEPA网站http://www.mhhe.com/engcs/compsci/pressman/professional/olc/ser.htm上可以找到最新的敏捷过程参考文献。