第 4 章

Software Engineering: A Practitioner's Approach, Eighth Edition

过程模型

要点浏览

概念：过程模型为软件工程工作提供了特定的路线图，该路线图规定了所有活动的流程、动作、任务、迭代的程度、工作产品及要完成的工作应如何组织。

人员：软件工程师及其管理人员根据他们的要求采用一种过程模型，并遵循该过程模型。此外，软件的需求方也需要参与过程的定义、构建和测试。

重要性：软件过程提高了软件工程活动的稳定性、可控性和有组织性，如果没有过程约束，软件活动将失控并变得混乱。但是，现代软件工程方法必须是“灵活”的，也就是要求软件工程活动、控制以及工作产品适合于项目团队和将要开发的产品。

步骤：过程模型为软件人员提供了开展软件工程工作需要遵循的步骤。

工作产品：从软件工程师的角度来看，工作产品是对过程所定义的活动和任务的格式化描述。

质量保证措施：有大量的软件过程评估机制，开发机构可以评估其软件过程的“成熟度”。然而，表征软件过程有效性的最好指标还是所构建产品的质量、及时性和寿命。

关键概念

面向方面的软件开发
基于构件的开发
并发模型
演化过程模型
形式化方法模型
增量过程模型
个人软件过程
过程建模工具
过程技术
原型开发
螺旋模型
团队软件过程
统一过程
V 模型
瀑布模型

最早提出过程模型是为了改变软件开发的混乱状况，使软件开发更加有序。历史证明，这些模型为软件工程工作提出了大量有用的结构，并为软件团队提供了有效的路线图。尽管如此，软件工程工作及其产品仍然停留在“混乱的边缘”。

在一篇探讨软件世界中有序和混乱之间奇怪关系的论文中，Nogueira 和他的同事指出 [Nog00]：

> 混乱的边缘可定义为“有序和混乱之间的一种自然状态，结构化和反常之间的重大妥协”[Kau95]。混乱的边缘可以被视为一种不稳定和部分结构化的状态……它的不稳定是因为它不停地受到混乱或者完全有序的影响。
>
> 我们通常认为有序是自然的完美状态。这可能是个误区……研究证实，打破平衡的活动会产生创造力、自我组织的过程和更高的回报 [Roo96]。完全的有序意味着缺乏可变性，而可变性在某些不可预测的环境下往往是一种优势。变更通常发生在某些结构中，这些结构使得变更可以被有效组织，但还不是死板得使得变更无法发生。另一方面，太多的混乱会使协调和一致成为不可能。缺少结构并不意味着无序。

上面这段话的哲学思想对于软件工程有着重要的意义。本章所描述的每个过程模型都试图在找出混乱世界中的秩序和适应不断发生的变化这两种要求之间寻求平衡。

4.1　惯用过程模型

惯用过程模型[⊖]力求达到软件开发的结构和秩序，其活动和任务都是按照过程的特定指引顺序进行的。但是，对于富于变化的软件世界，这一模型是否合适呢？如果我们抛弃传统过程模型（以及模型所规定的秩序），以一些不够结构化的模型取而代之，是否会使软件工作无法达到协调和一致？

关键点　过程模型的作用是减少开发新软件产品时出现的混乱。

这些问题无法简单回答，但是软件工程师有很大的选择余地。在接下来的章节中，我们将探讨以秩序和一致性作为主要问题的传统过程方法。我们称为“传统”是因为，它规定了一套过程元素——框架活动、软件工程动作、任务、工作产品、质量保证以及每个项目的变更控制机制。每个过程模型还定义了过程流（也称为工作流）——也就是过程元素相互之间关联的方式。

网络资源　包括最重要的惯用过程模型的“过程模拟比赛”获奖模型可见于http://www.ics.uci.edu/~emilyyo/SimSE/downloads.html。

所有的软件过程模型都支持第2章和第3章中描述的通用框架活动，但是每一个模型都对框架活动有不同的侧重，并且定义了不同的过程流以不同的方式执行每一个框架活动（以及软件工程动作和任务）。

4.1.1　瀑布模型

有时候，当从沟通到部署都采用合理的线性工作流方式的时候，可以清楚地理解问题的需求。这种情况通常发生在需要对一个已经存在的系统进行明确定义的适应性调整或是增强的时候（比如政府修改了法规，导致财务软件必须进行相应修改）；也可能发生在很少数新的开发工作上，但是需求必须是准确定义和相对稳定的。

关键点　惯用过程模型定义了一组规定的过程元素和一个可预测的过程工作流。

41

瀑布模型（waterfall model）又称为经典生命周期（classic life cycle），它提出了一个系统的、顺序的软件开发方法[⊜]，从用户需求规格说明开始，通过策划、建模、构建和部署的过程，最终提供完整的软件支持（图4-1）。

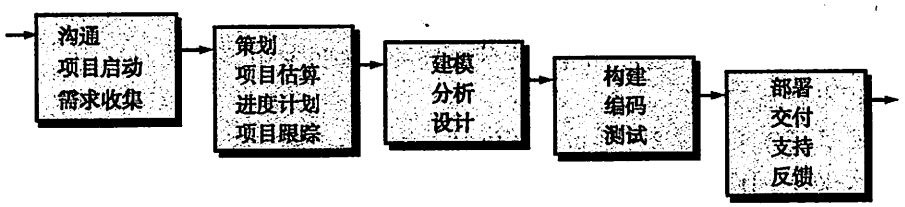

图4-1　瀑布模型

瀑布模型的一个变体称为V模型（V-model）。如图4-2所示，V模型[Buc99]描述了质量保证动作同沟通、建模相关动作以及早期构建相关的动作之间的关系。随着软件团队工作沿着V模型左侧步骤向下推进，基本问题需求逐步细化，形成了对问题及解决方案的详尽且技术性的描述。一

关键点　V模型阐明了验证和确认动作如何与早期工程动作相互关联。

⊖ 惯用过程模型有时称为“传统”过程模型。

⊜ 尽管对Winston Royce [Roy70]提出的最早的瀑布模型进行了改进，加入了“反馈”循环，但绝大多数软件组织在应用该过程模型时都将其视为严格的线性模型。

旦编码结束，团队沿着V模型右侧的步骤向上推进工作，其本质上是执行了一系列测试（质量保证动作），这些测试验证了团队沿着V模型左侧步骤向下推进过程中所生成的每个模型[⊖]。实际上，经典生命周期模型和V模型没有本质区别，V模型提供了一种将验证和确认动作应用于早期软件工程工作中的直观方法。

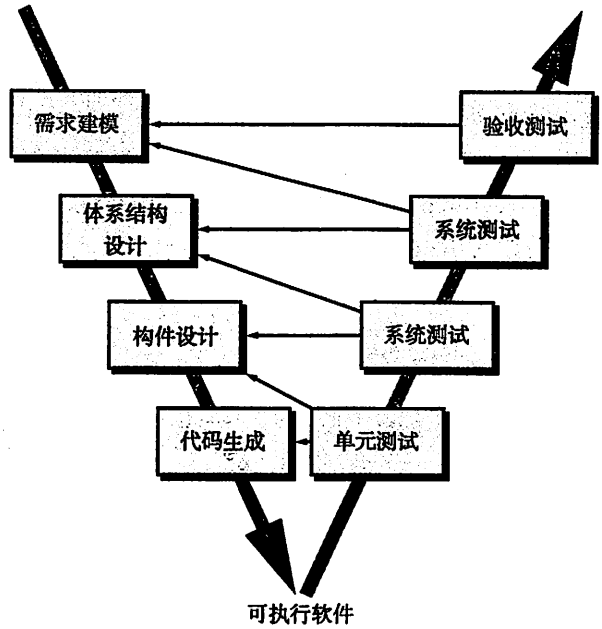

图4-2 V模型

瀑布模型是软件工程最早的范例。尽管如此，在过去的40多年中，对这一过程模型的批评使它最热情的支持者都开始质疑其有效性 [Han95]。在运用瀑布模型的过程中，人们遇到的问题包括：

1. 实际的项目很少遵守瀑布模型提出的顺序。虽然线性模型可以加入迭代，但是它是用间接的方式实现的，结果是，随着项目组工作的推进，变更可能造成混乱。
2. 客户通常难以清楚地描述所有的需求。而瀑布模型却要求客户明确需求，这就很难适应在许多项目开始阶段必然存在的不确定性。
3. 客户必须要有耐心，因为只有在项目接近尾声的时候，他们才能得到可执行的程序。对于系统中存在的重大缺陷，如果在可执行程序评审之前没有发现，将可能造成惨重损失。

提问 为什么瀑布模型有时候会失效？

42

在分析一个实际项目时，Bradac[Bra94] 发现，经典生命周期模型的线性特性在某些项目中会导致“阻塞状态”，由于任务之间的依赖性，开发团队的一些成员要等待另一些成员工作完成。事实上，花在等待上的时间可能超过花在生产性工作上的时间。在线性过程的开始和结束，这种阻塞状态更容易发生。

目前，软件工作快速进展，经常面临永不停止的变更流，特性、功能

引述 大多数情况下，软件工作遵从骑自行车第一定律：不论你去哪，你都会顶风骑上坡路。
作者不详

⊖ 质量保证动作的详细讨论参见本书第三部分。

和信息内容都会变更，瀑布模型往往并不适合这类工作。尽管如此，在需求已确定的情况下，且工作采用线性的方式完成的时候，瀑布模型是一个很有用的过程模型。

4.1.2　增量过程模型

43

在许多情况下，初始的软件需求有明确的定义，但是整个开发过程却不宜单纯运用线性模型。同时，可能迫切需要为用户迅速提供一套功能有限的软件产品，然后在后续版本中再进行细化和扩展功能。在这种条件下，需要选用一种以增量的形式生产软件产品的过程模型。

关键点　增量模型交付一系列称为增量的版本，随着每个版本的交付，逐步为用户提供更多的功能。

增量模型综合了第 3 章讨论的线性过程流和并行过程流的特征。如图 4-3 所示，随着时间的推移，增量模型在每个阶段都运用线性序列。每个线性序列生产出软件的可交付增量 [McD93]。

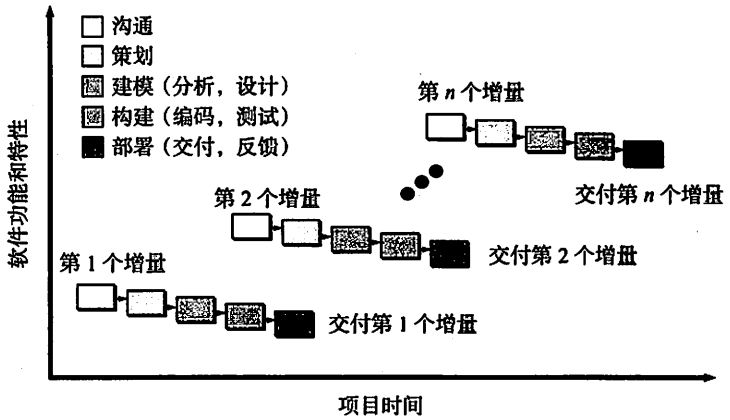

图 4-3　增量模型

例如，采用增量模型开发文字处理软件，在第一个增量中提供基本的文件管理、编辑和文档生成功能；在第二个增量中提供更为复杂的编辑和文档生成功能；在第三个增量中提供拼写和语法检查功能；在第四个增量中提供高级页面排版功能。需要注意的是，任何增量的过程流都可能使用下一节中讨论的原型范型。

运用增量模型的时候，第一个增量往往是核心产品（core product）。也就是满足了基本的需求，但是许多附加的特性（一些是已知的，一些是未知的）没有提供，客户使用该核心产品并进行仔细的评估，然后根据评估结果制定下一个增量计划。这份计划应说明需要对核心产品进行的修改，以便更好地满足客户的要求，也应说明需要增加的特性和功能。每一

44

个增量的交付都会重复这一过程，直到最终产品的产生。

建议　如果你的客户要求你在一个不可能完成的时间提交产品，那么向他建议届时只提交一个或几个增量，此后再提交软件的其他增量。

4.1.3　演化过程模型

软件类似于其他复杂的系统，会随着时间的推移而演化。在开发过程中，商业和产品需求经常发生变化，这将直接导致最终产品难以实现；严格的交付时间使得开发团队不可能圆满完成综合性的软件产品，但是必须交付功能有限的版本以应对竞争或商业压力；虽然能很好地理解核心产品和系统需求，但是产品或系统扩展的细节问题却没有定义。在上述情况和

关键点　演化过程模型中，每个迭代产生软件的一个更完整的版本。

类似情况下，软件开发人员需要一种专门应对不断演变的软件产品的过程模型。

演化模型是迭代的过程模型，这种模型使得软件开发人员能够逐步开发出更完整的软件版本。接下来，将介绍两种常用的演化过程模型。

原型开发。很多时候，客户定义了软件的一些基本任务，但是没有详细定义功能和特性需求。另一种情况下，开发人员可能对算法的效率、操作系统的适用性和人机交互的形式等情况并没有把握。在这些情况和类似情况下，采用**原型开发范型**（prototyping paradigm）是最好的解决办法。

引述 有时候你很想扔掉一款产品，但是通常情况下，你唯一的选择是考虑如何将它卖给客户。
Frederick P. Brooks

虽然原型可以作为一个独立的过程模型，但是更多的时候是作为一种技术，可以在本章讨论的任何一种过程模型中应用。不论人们以什么方式运用它，当需求很模糊的时候，原型开发模型都能帮助软件开发人员和利益相关者更好地理解究竟需要做什么。

原型开发范型（图 4-4）开始于沟通。软件开发人员和其他利益相关者进行会晤，定义软件的整体目标，明确已知的需求，并大致勾画出以后再进一步定义的东西。然后迅速策划一个原型开发迭代并进行建模（以“快速设计”的方式）。快速设计要集中在那些最终用户能够看到的方面（比如人机接口布局或者输出显示格式）。快速设计产生了一个原型。对原型进行部署，然后由利益相关者进行评估。根据利益相关者的反馈信息，进一步精炼软件的需求。在原型系统不断调整以满足各种利益相关者需求的过程中，采用迭代技术，同时也使开发者逐步清楚用户的需求。

建议 如果你的客户有一个合理的要求，但是对细节没有思路，那么不妨先开发一个原型。

理想状况下，原型系统提供了定义软件需求的一种机制。当需要构建可执行的原型系统时，软件开发人员可以利用已有的程序片段或应用工具快速产生可执行的程序。

如果我们的原型达到了上述目的，那么接下来它有什么用呢？Brooks[Bro95] 给出了答案：

在大多数项目中，构建的第一个系统很少是好用的，可能太慢了、太大了、太难用了，或者同时具备上述三点。除了重新开始，没有更好的选择。采用更巧妙的方法来构建一个重新设计的版本，解决上述问题。

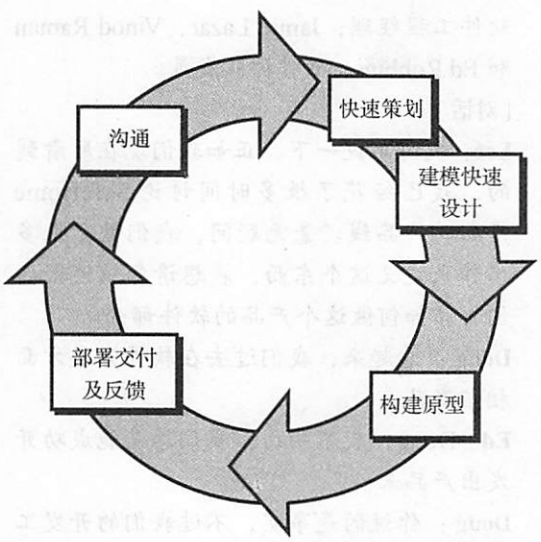

图 4-4 原型开发范型

原型可以作为第一个系统，也就是 Brooks 推荐我们扔掉的系统。但这可能是一种理想的方式。尽管许多原型系统是临时系统并且会被废弃，但其他一些原型系统将会演化为实际系统。

建议 面对把一个粗糙的原型系统变为工作产品的压力时，如果采取抵制的态度，那么结果往往是产品质量受到损害。

利益相关者和软件工程师确实都喜欢原型开发范型。客户对实际的系统有了直观的认识，开发者也迅速建立了一些东西。但是，原型开发也存在一些问题，原因如下。

1. 利益相关者看到了软件的工作版本，却未察觉到整个软件是随意搭成的，也未察觉到为了尽快完成软件，开发者没有考虑整体软件质

量和长期的可维护性。当开发者告诉客户整个系统需要重建以提高软件质量的时候，利益相关者会不愿意，并且要求对软件稍加修改使其变为一个可运行的产品。在绝大多数的情况下，软件开发管理层会做出妥协。

2. 作为一名软件工程师，为了使一个原型快速运行起来，往往在实现过程中采用折衷的手段。他们经常会使用不合适的操作系统或程序设计语言，仅仅因为当时可用或他们对此较为熟悉。他们也经常会采用一种低效的算法，仅为了证明系统的能力。时间长了，软件开发人员可能会适应这些选择，而忽略了这些选择其实并不合适的理由，结果使并不完美的选择成了系统的组成部分。

46

尽管问题会发生，但原型开发对于软件工程来说仍是一个有效的范型。关键是要在游戏开始的时候制定规则，也就是说，所有利益相关者必须承认原型是为定义需求服务的。然后丢弃原型（至少是部分丢弃），实际的软件系统是以质量第一为目标而开发的。

SafeHome　选择过程模型　第 1 部分

[场景] CPI 公司软件工程部会议室。该（虚构的）公司专注于开发家用和商用的消费产品。

[人物] Lee Warren，工程经理；Doug Miller，软件工程经理；Jamie Lazar、Vinod Raman 和 Ed Robbins，软件团队成员。

[对话]

Lee：我简单说一下。正如我们现在所看到的，我已经花了很多时间讨论 SafeHome 产品的产品线。毫无疑问，我们做了很多工作来定义这个东西，我想请各位谈谈你们打算如何做这个产品的软件部分。

Doug：看起来，我们过去在软件开发方面相当混乱。

Ed：Doug，我不明白，我们总是能成功开发出产品来。

Doug：你说的是事实，不过我们的开发工作并不是一帆风顺，并且我们这次要做的项目看起来比以前做的任何项目都要庞大和复杂。

Jamie：没有你说的那么严重，但是我同意你的看法。我们过去混乱的项目开发方法这次行不通了，特别是这次我们的时间很紧。

Doug（笑）：我希望我们的开发方法更专业一些。我上星期参加了一个培训班，学了很多关于软件工程的知识。我们现在需要一个过程。

Jamie（皱眉）：我的工作是编程，不是文书。

Doug：在你反对我之前，请先尝试一下。我想说的是……（Doug 开始讲述第 3 章讲述的过程框架和本章到目前为止讲到的惯用过程模型）

Doug：所以，似乎线性模型并不适合我们……它假设我们此刻明确了所有的需求，而事实上并不是这样。

Vinod：同意你的观点。线性模型太 IT 化了……也许适合于开发一套库存管理系统或者什么，但是不适合我们的 SafeHome 产品。

Doug：对。

Ed：原型开发方法听起来不错，正适合我们现在的处境。

Vinod：有个问题，我担心它不够结构化。

Doug：别担心。我们还有许多其他选择。我希望在座的各位选出最适合我们小组和我们这个项目的开发范型。

螺旋模型。最早由 Barry Boehm[Boe88] 提出，螺旋模型是一种演进式软件过程模型。它结合了原型的迭代性质和瀑布模型的可控性和系统性特点。它具有快速开发越来越完善的软件版本的潜力。Boehm[Boe01a] 如下这样描述螺旋模型：

关键点 螺旋模型能运用在应用系统开发的整个生命周期，从概念开发到维护。 47

螺旋模型是一种风险驱动型的过程模型生成器，对于软件集中的系统，它可以指导多个利益相关者的协同工作。它有两个显著的特点。一是采用循环的方式逐步加深系统定义和实现的深度，同时降低风险。二是确定一系列里程碑作为支撑点，确保利益相关者认可是可行的且可令各方满意的系统解决方案。

螺旋模型将软件开发为一系列演进版本。在早期的迭代中，软件可能是一个理论模型或是原型。在后来的迭代中，会产生一系列逐渐完整的系统版本。

螺旋模型被分割成一系列由软件工程团队定义的框架活动。为了讲解方便，我们使用前文讨论的通用框架活动[⊖]。如图 4-5 所示，每个框架活动代表螺旋上的一个片段。随着演进过程开始，从圆心开始顺时针方向，软件团队执行螺旋上的一圈所表示的活动。在每次演进的时候，都要考虑风险（第 35 章）。每个演进过程还要标记里程碑——沿着螺旋路径达到的工作产品和条件的结合体。

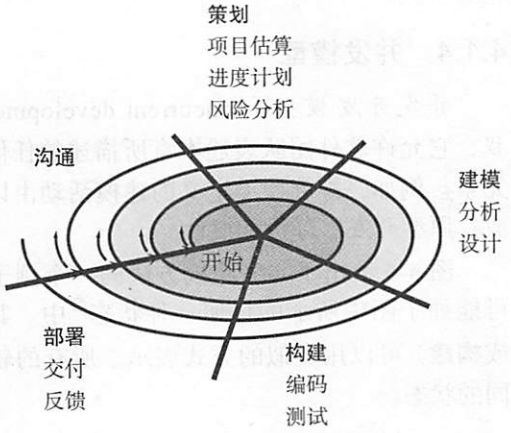

图 4-5 典型的螺旋模型

网络资源 可以在如下网址获得关于螺旋模型的有用信息：www.sei.cmu.edu/publications/documents/00.reports/00sr008.html。 48

螺旋的第一圈一般开发出产品的规格说明，接下来开发产品的原型系统，并在每次迭代中逐步完善，开发不同的软件版本。螺旋的每圈都会跨过策划区域，此时，需调整项目计划，并根据交付后用户的反馈调整预算和进度。另外，项目经理还会调整完成软件开发需要迭代的次数。

其他过程模型在软件交付后就结束了。螺旋模型则不同，它应用在计算机软件的整个生命周期。因此，螺旋上的第一圈可能表示“概念开发项目”，它起始于螺旋的中心，经过多个迭代[⊜]，直到概念开发的结束。如果这个概念将被开发成为实际的产品，那么该过程将继续沿着螺旋向外伸展，成为“新产品开发项目”。新产品将沿着螺旋通过一系列的迭代不断演进。最后，可以用一圈螺旋表示“产品提高项目”。本质上，当螺旋模型以这种方式进行下去的时候，它将永远保持可操作性，直到软件产品的生命周期结束。过程经常会处于休止状态，但每当有变更时，过程总能够在合适的入口点启动（如产品提高）。

建议 如果你的项目要求固定预算开发（通常不是一个好主意），那么螺旋模型会带来问题：每一圈完成的时候，都将重新计划和修改项目开销。

螺旋模型是开发大型系统和软件的很实际的方法。由于软件随着过程的推进而变化，因此在每一个演进层次上，开发者和客户都可以更好地理解和应对风险。螺旋模型把原型作为降低风险的机制，更重要的是，开发

⊖ 这里讨论的螺旋模型有别于 Boehm 提出的模型。原始的螺旋模型可参见 [Boe88]。关于 Boehm 的螺旋模型的更多更新的讨论可参见 [Boe98]。

⊜ 沿轴指向中心的箭头区分开了部署和沟通两个区域，表明沿同一个螺旋路径存在潜在的局部迭代。

人员可以在产品演进的任何阶段使用原型方法。它保留了经典生命周期模型中系统逐步细化的方法，但是把它纳入一种迭代框架之中，这种迭代方式与真实世界更加吻合。螺旋模型要求在项目的所有阶段始终考虑技术风险，如果适当地应用该方法，就能够在风险变为难题之前将其化解。

与其他范型一样，螺旋模型也并不是包治百病的灵丹妙药。很难使客户（特别是以合同的形式）相信演进的方法是可控的。它依赖大量的风险评估专家来保证成功。如果存在较大的风险没有被发现和管理，就肯定会发生问题。

引述 我今天只走这么远，只有明天才能为我指明方向。

Dave Matthews Band

4.1.4 并发模型

并发开发模型（concurrent development model）有时也叫作并发工程，它允许软件团队表述本章所描述的任何过程模型中的迭代元素和并发元素。例如，螺旋模型定义的建模活动由以下一种或几种软件工程动作完成：原型开发、分析和设计。[⊖]

关键点 必须将项目计划看成是活的文档，必须经常对进度进行评估并考虑变更情况，从而对其进行修改。

49

图 4-6 给出了并发建模方法的一个例子。在某一特定时间，**建模**活动可能处于图中所示的任何一种状态[⊜]中。其他活动、动作或任务（如**沟通**或**构建**）可以用类似的方式表示。所有的软件工程活动同时存在并处于不同的状态。

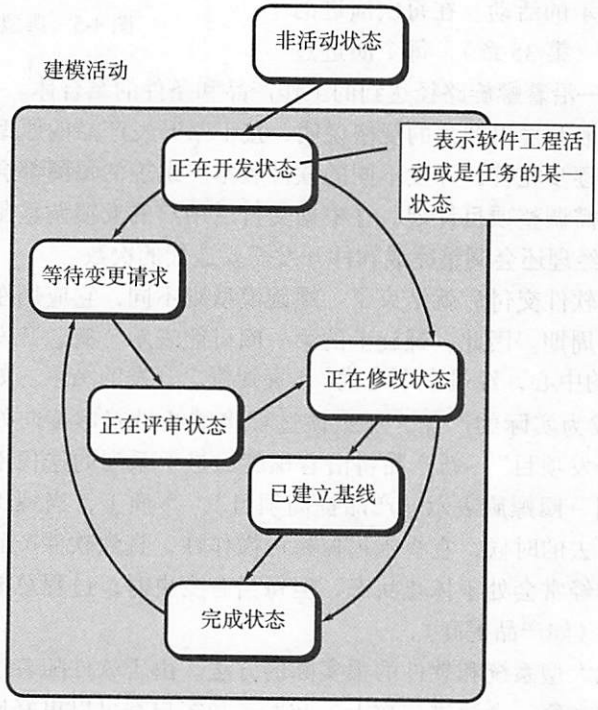

图 4-6　并发过程模型的一个元素

⊖ 需要注意的是，分析和设计都是很复杂的任务，需要进行大量的讨论。本书的第 2 部分将详细讨论这些问题。

⊜ 状态是外部可见的某种行为模式。

例如，在项目的早期，沟通活动（图中并未标明）完成了第一个迭代，停留在**等待变更**状态。建模活动（初始沟通完成后，一直停留在**非活动**状态）现在转换到**正在开发**状态。然而，如果客户要求必须完成需求变更，那么建模活动就会从**正在开发**状态转换到**等待变更**状态。

并发建模定义了一系列事件，这些事件将触发软件工程活动、动作或者任务的状态转换。例如，设计的早期阶段（建模活动期间发生的主要软件工程动作）发现了需求模型中的不一致性，于是产生了分析模型修正事件，该事件将触发需求分析动作从**完成**状态转换到**等待变更**状态。

并发建模可用于所有类型的软件开发，它能够提供精确的项目当前状态图。它不是把软件工程活动、动作和任务局限在一个事件的序列，而是定义了一个过程网络。网络上每个活动、动作和任务与其他活动、动作和任务同时存在。过程网络中某一点产生的事件可以触发与每一个活动相关的状态的转换。

建议 并发模型更适合于不同软件工程团队共同开发的产品工程项目。

引述 确保你所在组织的每一个过程都有客户，如果没有客户，过程就会变成空中楼阁，失去目标。

V. Daniel Hunt

SafeHome 选择过程模型 第2部分

[**场景**] CPI公司软件工程部会议室，该公司生产家用和商用消费类产品。

[**人物**] Lee Warren，工程经理；Doug Miller，软件工程经理；Vinod和Jamie，软件工程团队成员。

[**对话**]

（Doug介绍了一些可选的演化模型）

Jamie：我现在有了一些想法。增量模型挺有意义的。我很喜欢螺旋模型，听起来很实用。

Vinod：我赞成。我们交付一个增量产品，听取用户的反馈意见，再重新计划，然后交付另一个增量。这样做也符合产品的特性。我们能够迅速投入市场，然后在每个版本或者说在每个增量中添加功能。

Lee：等等，Doug，你的意思是说我们在螺旋的每一轮都重新生成计划？这样不好，我们需要一个计划，一个进度，然后严格遵守这个计划。

Doug：你的思想太陈旧了，Lee。就像他们说的，我们要现实。我认为，随着我们认识的深入和情况的变化来调整计划更好。这是一种更符合实际的方式。如果制定了不符合实际的计划，这个计划还有什么意义？

Lee（皱眉）：我同意这种看法，可是高管人员不喜欢这种方式，他们喜欢确定的计划。

Doug（笑）：老兄，你应该给他们上一课。

4.1.5 演化过程的最终评述

我们注意到，现代计算机软件总是在持续改变，这些变更通常要求在非常短的期限内实现，并且要充分满足客户–用户的要求。许多情况下，及时投入市场是最重要的管理要求。如果市场时间错过了，软件项目自身可能会变得毫无意义。[⊖]

50
~
51

⊖ 需要注意的是，尽管及时投入市场很重要，但最早进入市场并不保证一定成功。事实上，许多非常成功的软件是第二个或是第三个进入市场的（他们的成功在于吸取了前人的教训）。

演化过程模型就是为了解决上述问题的，但是，作为一类通用的过程模型，它们也有缺点。Nogueira 和他的同事对此概括如下 [Nog00]：

尽管演化软件过程毫无疑问具有一定的优势，但我们还是有一些担忧。首先，由于构建产品需要的周期数目不确定，原型开发（和其他更加复杂的演化过程）给项目计划带来了困难……

其次，演化软件过程没有确定演化的最快速度。如果演化的速度太快，完全没有间歇时间，项目肯定会陷入混乱；反之，如果演化速度太慢，则会影响生产率……

再次，演化软件过程应该侧重于灵活性和可延展性，而不是高质量。这种说法听起来很惊人。

确实，一个强调灵活性、可扩展性和开发速度而不是高质量的软件过程听起来令人震惊。可是，很多广为人们尊重的软件工程专家都这样建议（例如 [You95]、[Bac97]）。

演化模型的初衷是采用迭代或者增量的方式开发高质量软件[⊖]。可是，用演化模型也可以做到强调灵活性、可延展性和开发速度。软件团队及其经理所面临的挑战就是在这些严格的项目、产品参数与客户（软件质量的最终仲裁者）满意度之间找到一个合理的平衡点。

提问　演化过程模型具有哪些不为人知的弱点？

4.2　专用过程模型

专用过程模型具有前面章节中提到的传统过程模型的一些特点，但是，专用过程模型往往应用面较窄且较专一，只适用于某些特定的软件工程方法。[⊖]

52

4.2.1　基于构件的开发

商业现货（Commercial Off-The-Shelf，COTS）软件构件由厂家作为产品供应，通过良好定义的接口提供特定的功能，这些构件能够集成到正在构建的软件中。*基于构件的开发模型*（component-based development model）具有许多螺旋模型的特点。它本质上是演化模型 [Nie92]，需要以迭代方式构建软件。不同之处在于，基于构件的开发模型采用预先打包的软件构件来开发应用系统。

网络资源　基于构件开发的相关信息可以参见 www.cbd-hq.com。

建模和构建活动开始于识别可选构件。这些构件有些设计成传统的软件模块，有些设计成面向对象的类或类包[⊜]。若不考虑构件的开发技术，则基于构件开发模型由以下步骤组成（采用演化方法）：

1. 对于该问题的应用领域研究和评估可用的基于构件的产品。
2. 考虑构件集成的问题。
3. 设计软件架构以容纳这些构件。
4. 将构件集成到架构中。
5. 进行充分的测试以保证功能正常。

⊖ 在这里，软件质量的含义非常广泛，不仅包括客户满意度，还包括本书第二部分讲的各种技术指标。

⊖ 在某些情况下，这些专用过程模型也许应该更确切地称为技术的集合或“方法论”，是为了实现某一特定的软件开发目标而制定的。但它们确实也提出了一种过程。

⊜ 附录 2 讨论了面向对象概念，该概念的使用贯穿了本书第二部分。在这里，类封装了一组数据以及处理数据的过程。类包是一组共同产生某种结果的相关类的集合。

基于构件的开发模型能够使软件复用，从而为软件工程师带来极大收益。如果构件复用已经成为你所在的软件工程团队文化的一部分，那么将会缩短开发周期并减少项目开发费用。基于构件的软件开发将在第 14 章进行详细讨论。

4.2.2 形式化方法模型

形式化方法模型（formal methods model）的主要活动是生成计算机软件形式化的数学规格说明。形式化方法使软件开发人员可以应用严格的数学符号来说明、开发和验证基于计算机的系统。这种方法的一个变形是*净室软件工程*（cleanroom software engineering）[Mil87, Dye92]，这一软件工程方法目前已应用于一些软件开发机构。 53

形式化方法（附录 3）提供了一种机制，使得在软件开发中可以避免一些问题，而这些问题在使用其他软件工程模型时是难以解决的。使用形式化方法时，歧义性问题、不完整问题、不一致问题等都能够更容易地被发现和改正——不是依靠特定的评审，而是应用数学分析的方法。在设计阶段，形式化方法是程序验证的基础，使软件开发人员能够发现和改正一些常常被忽略的问题。

虽然形式化方法不是一种主流的方法，但它的意义在于可以提供无缺陷的软件。尽管如此，人们还是对在商业环境中应用形式化方法有怀疑，这表现在：

- 目前，形式化模型开发非常耗时，成本也很高。
- 只有极少数程序员具有应用形式化方法的背景，因此需要大量的培训。
- 对于技术水平不高的客户，很难用这种模型进行沟通。

提问 既然形式化方法能够保证软件的正确性，为什么没有被广泛应用呢？

尽管有这些疑虑，但软件开发者中还是有很多形式化方法的追随者，比如有人用其开发那些高度关注安全的软件（如飞行器和医疗设施），或者开发那些一旦出错就将导致重大经济损失的软件。

4.2.3 面向方面的软件开发

不论选择什么软件过程，复杂软件都无一例外地实现了一套局部化的特性、功能和信息内容。这些局部的软件特性被做成构件（例如面向对象的类），然后在系统架构中使用。随着现代计算机系统变得更加复杂，某些*关注点*——客户需要的属性或者技术兴趣点——已经体现在整个架构设计中。有些关注点是系统的高层属性（例如安全性、容错能力），另一些关注点影响了系统的功能（例如商业规则的应用），还有一些关注点是系统性的（例如任务同步或内存管理）。

网络资源 关于 AOP 的资源和信息可以参见 aosd.net。

如果某个关注点涉及系统多个方面的功能、特性和信息，那么这些关注点通常称为*横切关注点*（crosscutting concern）。*方面的需求*（aspectual requirement）定义那些对整个软件体系结构产生影响的横切关注点。*面向方面的软件开发*（Aspect-Oriented Software Development，AOSD）通常称为*面向方面编程*（Aspect-Oriented Programming，AOP）或者*面向方面构件工程*（Aspect-Oriented Component Engineering，AOCE）[Gru02]，它是相对较新的一种软件工程模型，为定义、说明、设计和构建方面（aspect）提供过程和方法——“是对横切关注点进行局部表示的一种机制，超越了子程序和继承方法”[Elr01]。

关键点 AOSD 定义了“方面”，表示用户跨越多个系统功能、特性和信息的关注点。 54

与众不同的面向方面的过程还不成熟。尽管如此，这种过程模型看似具备了演化模型和并发过程模型的共同特点。演化模型适合定义和构建方面；而并发开发的并行特点很重要，因为方面是独立于局部的软件构件开发的，并且对这些构件的开发有直接影响。因此，在构建方面和构件的过程活动之间建立起异步的通信非常重要。

关于面向方面的软件开发最好查阅相关专著中的详细讨论。感兴趣的读者可以参看[Ras11]、[Saf08]、[Cla05]、[Fil05]、[Jac04]和[Gra03]。

软件工具　过程管理

[目标] 辅助定义、执行和管理传统过程模型。

[机制] 过程管理工具帮助软件组织或团队定义完整的软件过程模型（框架活动、动作、任务、质量保证检查点、里程碑和工作产品）。而且，该工具为软件工程师的技术工作提供路线图，为经理们跟踪和控制软件过程提供模板。

[代表性工具]⊖

- GDPA。一个研究性的过程定义工具包，该工具包由德国的 Bremen 大学（www.informatik.uni-bremen.de/uniform/gdpa/home.htm）开发，它提供了大量的过程建模和管理功能。
- ALM Studio。由 Kovair 公司（http://www.kovair.com/）开发的一套工具包，用于过程定义、需求管理、问题解决、项目策划和跟踪。
- ProVision BPMx。由 OpenText（http://bps.opentext.com）开发，它提供了很多代表性工具，可以辅助过程定义和工作流自动化。

以下网站提供了很多与软件过程相关的各种很有价值的工具：www.computer.org/portal/web/swebok/html/ch10。

4.3　统一过程

Ivar Jacobson、Grady Booch 和 James Rumbaugh[Jac99] 在他们关于统一过程（unified process）的影响深远的著作中，讨论了关于有必要建立一种“用例驱动，以架构为核心，迭代并且增量”的软件过程的问题，并阐述如下：

> 当前，软件朝着更大、更复杂的系统发展。部分原因在于计算机的计算能力逐年递增，使得人们对其的期望值增大。另一方面，还是受 Internet 应用膨胀的影响，促使各类信息交换……我们从软件版本的提升中学会了如何改进产品，使我们对更复杂软件的胃口越来越大。同时希望软件更好地满足我们的需要，结果导致软件更加复杂。简而言之，我们想要的越来越多。

55

在某种程度上，统一过程尝试着从传统的软件过程中挖掘最好的特征和性质，但是以敏捷软件开发（第 5 章）中许多最好的原则来实现。统一过程认识到与客户沟通以及从用户的角度描述系统（即用例）⊖并保持该描述的一致性的重要性。它强调软件体系结构的重要

⊖ 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。大多数情况下，工具的名字由各自的开发者注册为商标。

⊖ 用例（use case）（第 5 章）是一种文字描述或模板，从用户的角度描述系统功能和特性。用例由用户来写，并作为创建更为复杂的分析模型的基础。

作用，并“帮助架构师专注于正确的目标，例如可理解性、对未来变更的可适应性以及复用”[Jac99]。它建立了迭代的、增量的过程流，提供了演进的特性，这对现代软件开发非常重要。

4.3.1 统一过程的简史

20世纪90年代早期，James Rumbaugh [Rum91]、Grady Booch [Boo94]和Ivar Jacobson [Jac92]开始研究“统一方法”，他们的目标是结合各自面向对象分析和设计方法中最好的特点，并吸收其他面向对象模型专家提出的其他特点（例如[Wir90]）。他们的成果就是UML——统一建模语言（unified modeling language），这种语言包含了大量用于面向对象系统建模和开发的符号。到了1997年，UML已经变成了面向对象软件开发的行业标准。

UML作为需求模型和设计模型的表示方式，其应用贯穿本书第二部分。附录1针对不熟悉UML基本概念和建模规则的人给出了介绍性的指导。有关UML的全面介绍可以参考有关UML的材料，附录1中列出了相关书籍。

4.3.2 统一过程的阶段[⊖]

在第3章，我们讨论了5种通用的框架活动，并认为它们可以用来描述任何软件过程模型。统一过程也不例外。图4-7描述了统一过程（Unified Process，UP）的“阶段”，并将他们与第1章及本章前面部分讨论的通用活动进行了对照。

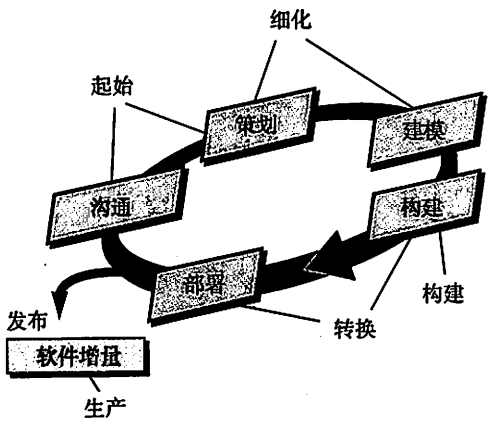

图4-7 统一过程

UP的起始阶段（inception phase）包括客户沟通和策划活动。通过与利益相关者协作定义软件的业务需求，提出系统大致的架构，并制定开发计划以保证项目开发具有迭代和增量的特性。该阶段识别基本的业务需求，并初步用用例（第8章）描述每一类用户所需要的主要特性和功能。此时的体系结构仅是主要子系统及其功能、特性的试探性概括。随后，体系结构将被细化和扩充成为一组模型，以描述系统的不同视图。策划阶段将识别各种资源，评估主要风险，制定进度计划，并为其在软件增量开发的各个阶段中的应用建立基础。

关键点 UP阶段的目的与本书中定义的通用框架活动的目的是类似的。

细化阶段（Elaboration Phase）包括沟通和通用过程模型的建模活动（图4-7）。细化阶段扩展了初始阶段定义的用例，并扩展了体系结构以包括软件的五种视图——用例模型、需求模型、设计模型、实现模型和部署模型。在某些情况下，细化阶段建立了一个“可执行的体系结构基线”[Arl02]，这是建立可执行系统的“第一步”（first cut）[⊖]。体系结构基线证明了体系结构的可实现性，但没有提供系统使用时所需的所有功能和特性。另

56

57

⊖ 统一过程有时也用Rational公司（后来被IBM收购）的名字命名，称为Rational统一过程（Rational Unified Process，RUP），Rational公司是早期开发和细化该统一过程的主要投资方，并建立了支持该过程的完整环境（工具及技术）。

⊖ 需要指出的是，体系结构基线不是原型系统，因为它并不会被抛弃，而是在下一个UP阶段进一步充实。

外，在细化的最终阶段将评审项目计划以确保项目的范围、风险和交付日期的合理性。该阶段通常要对项目计划进行修订。

网络资源：敏捷开发中统一过程的有趣讨论参见www.ambysoft.com/unifiedprocess/agileUP.html。

UP 的*构建阶段*（construction phase）与通用软件过程中的构建活动相同。构建阶段采用体系结构模型作为输入，开发或是获取软件构件，使得最终用户能够操作用例。为达到上述目的，要对在细化阶段开始的需求模型和设计模型加以完善，以反映出软件增量的最终版本。软件增量（例如发布的版本）所要求的必须具备的特性和功能在源代码中实现。随着构件的实现，对每一个构件设计并实施单元测试[⊖]。另外，还实施了其他集成活动（构件组装和集成测试）。用例用于导出一组验收测试，以便在下一个 UP 阶段开始前执行。

UP 的*转换阶段*（transition phase）包括通用构建活动的后期阶段以及通用部署（交付和反馈）活动的第一部分。软件被提交给最终用户进行 Beta 测试，用户反馈报告缺陷及必要的变更。另外，软件开发团队创建系统发布所必需的支持信息（如用户手册、问题解决指南及安装步骤）。在转换阶段结束时，软件增量成为可用的发布版本。

UP 的*生产阶段*（production phase）与通用过程的部署活动一致。在该阶段，对持续使用的软件进行监控，提供运行环境（基础设施）的支持，提交并评估缺陷报告和变更请求。

有可能在构建、转换和生产阶段的同时，下一个软件增量的工作已经开始。这就意味着五个 UP 阶段并不是顺序进行，而是阶段性地并发进行。

软件工程的工作流分布在所有 UP 阶段。在 UP 中，工作流类似于任务集（参见第 3 章的描述）。也就是说，工作流识别了完成一个重要的软件工程活动的必要任务，以及在成功完成任务之后所产生的工作产品。需要注意的是，并不是工作流所识别的每一个任务都在所有的项目中应用。软件开发团队应根据各自的需要适当调整过程（动作、任务、子任务及工
58 作产品）。

4.4　个人过程模型和团队过程模型

引述：成功者不过是养成了成功人士做事的习惯。
Dexter Yager

好的软件过程需要贴近于过程的执行人员。如果一个软件过程模型是为社团或是软件组织制定的，那么，只有对其进行充分改进，并使其真正满足实施软件工程项目的要求，该模型才能有效。理想情况下，软件开发人员会建立最适合他们的过程，并同时能够与开发团队及整个组织的要求相吻合。换句话说，开发团队将制定其自己的过程，同时满足个人的小范围的要求和企业的大范围要求。Watts Humphrey([Hum97] 和 [Hum00]) 认为，有可能建立“个人软件过程”和“团队软件过程”。虽然二者都需要艰苦努力、培训和协调，但都是可以做到的。[⊖]

4.4.1　个人软件过程

每个开发人员都采用某种过程来开发计算机软件。这种过程也许是随机的，也许是特定

⊖ 有关软件测试（包括单元测试）的深入讨论参见第 22 ～ 26 章。

⊖ 需要指出的是，敏捷方法的支持者（第 5 章）同样认为过程应贴近团队。他们只不过提出了达到同样目的的另外一种解决方法。

的，可能每天都会改变，可能不够高效、不够有效甚至不成功，但不管怎样，过程都是存在的。Watts Humphrey[Hum05] 建议为了改变无效的个人过程，开发人员必须经过四个阶段，每个阶段都需要培训和认真操作。个人软件过程（Personal Software Process，PSP）强调对产品以及产品质量的个人测量。并且，PSP 让第一线工作人员负责项目计划的制定（如估算和进度安排），并授权给他们来控制所有开发的软件产品的质量。PSP 过程模型定义了以下五个框架工作活动。

网络资源 有关PSP的大量资源可见于http://www.sei.cmu.edu/tsp/tools/academic/。

策划。它将需求活动分离出来，并估算项目的规模和所需资源，并且进行缺陷评估（估计此项工作的缺陷数目）。所有的度量都用工作表或是模板记录。最后，识别开发任务并建立项目进度计划。

高层设计。建立每个构件的外部规格说明并完成构件设计。如果有不确定的需求，则构建原型系统。所有问题都要记录和跟踪。

59

高层设计评审。使用正式的验证方法（附录 3）来发现设计中的错误。对所有的重要任务和工作结果都进行度量。

开发。细化和评审构件级设计。完成编码，对代码进行评审，并进行编译和测试。对所有的重要任务和工作结果都进行度量。

提问 PSP中用到了哪些框架活动?

后验。根据收集到的度量和测量结果（需要进行大量数据的统计分析）确定过程的有效性。度量和测量结果为提高过程的有效性提供指导。

PSP 强调尽早发现错误，并且分析易犯错误的种类，后者和前者同样重要。这是通过对软件开发人员提交的所有产品进行严格评估实现的。

关键点 PSP强调对所犯的错误类型进行记录和分析，以便制定消除错误的策略。

PSP 代表的是一种严格有序的、基于度量的软件工程方法，这可能对许多第一线工作人员产生文化冲击。但是，如果将 PSP 恰当地介绍给软件工程师 [Hum96]，那么软件工程的生产率和软件质量将大幅度提高 [Fer97]。然而 PSP 没有被工业界广泛地加以采纳。遗憾的是，其原因更主要的在于人的本性和软件开发组织的惯性，而非 PSP 方法本身的好坏。PSP 是对能力的极大挑战，并且需要得到一定程度的承诺（包括第一线工作人员及其经理），这种支持通常很难得到。PSP 的培训相对时间较长，价格较高。由于文化的关系，对很多软件人员来说都难以达到所需的度量水平。

PSP 能否用作个人的有效软件过程呢？答案尚不明确。但是，即使 PSP 没有得到完全的采纳，但它所引进的许多个人过程改进的理念也值得学习。

4.4.2 团队软件过程

考虑到很多行业级软件项目都由项目团队开发，Watts Humphrey 吸取了引入 PSP 的经验教训，并提出了团队软件过程（Team Software Process，TSP）。TSP 的目标是建立一个能够“自我管理”的项目团队，团队能自我组织进行高质量的软件开发。Humphrey [Hum98] 为 TSP 定义了以下目标：

网络资源 关于采用PSP和TSP构建优秀团队的信息可参考www.sei.cmu.edu/tsp/。

- 建立自我管理团队来策划和跟踪其工作、确定目标、建立团队自己的过程和计划。团队既可以是纯粹的软件开发队伍，也可以是集成的产品队伍（Integrated Product Team，IPT），可以由 3 ～ 20 名工程师组成。
- 指示管理人员如何指导和激励其团队，并保持团队的最佳表现。

60

- 使 CMM[⊖]第 5 级的行为常规化，并依此约束员工，这样可加速软件过程改进。
- 为高成熟度的软件组织提供改进指导。
- 协助大学教授行业级团队技能。

建议 为了组建有自我管理能力的团队，必须能够做到内部相互配合，并与外部建立良好沟通。

一个自我管理的团队对其整体目标有一致的理解。它定义了每个团队成员的角色和责任；跟踪量化的项目数据（包括生产率和质量）；确定适合该项目的团队过程和执行该过程的具体策略；定义适合团队软件工程工作的本地标准；持续评估风险并采取风险规避措施；跟踪、管理和报告项目状态。

TSP 定义了以下框架活动：项目启动、高层设计、实现、集成和测试以及后验。正如在 PSP 中与其对应的活动（注意这里用的术语是不一样的），这些活动使整个团队按规范的方式计划、设计和构建软件，同时量化地评测软件过程和产品。后验用于确定过程改进的步骤。

关键点 TSP 脚本定义了团队过程的组成部分，以及过程中的活动。

TSP 使用大量的脚本、表格和标准等来指导其团队成员的工作。脚本（script）定义了特定的过程活动（如项目启动、设计、实现、集成和系统测试、后验），以及作为团队过程一部分的其他更详细的工作职能（如开发计划的制定、需求开发、软件配置管理及单元测试）。

TSP 认为，最好的软件团队需要具有自我管理的能力[⊖]。团队成员制定项目目标，调整过程以满足需要，控制进度计划，并通过度量及对所收集的度量值的分析，不断改进团队的软件工程方法。

与 PSP 类似，TSP 也是一个严格的软件工程过程，在提高生产率和质量方面可以取得明显的和量化的效果。开发团队必须对过程做出全面的承诺，并且经过严格的训练以保证方法得以很好地应用。

4.5 过程技术

前面章节中讨论的一些过程模型必须加以调整才能应用于特定的软件项目团队。为了利于模型调整，人们开发了过程技术工具（process technology tool）以帮助软件开发组织分析

61

现有过程、组织工作任务、控制并监控过程进度和管理技术质量。

过程技术工具帮助软件开发组织对第 3 章中讨论的过程框架、任务集和普适性活动建造自动化的模型。模型通常用网络图表示，可以通过分析定义典型的工作流，并识别有可能减少开发时间或费用的其他可选择的过程结构。

一旦创建了可接受的过程，其他过程技术工具可用来分配、监控甚至控制过程模型中定义的所有软件工程活动、动作和任务。每个项目组成员都可以利用这种工具建立需要完成的工作任务检查单、工作产品检查单和需要执行的质量保证活动检查单。过程技术工具也可以和其他适用于特定工作任务的软件工程工具配合使用。

⊖ 能力成熟度模型（CMM）是一种衡量软件过程效率的技术，将在第 37 章进行讨论。

⊖ 第 5 章讨论了敏捷模型中的关键因素“自管理”团队的重要性。

软件工具 过程建模工具

[目标] 软件组织必须首先理解软件过程，然后才能对其进行改进。过程建模工具（也称为过程技术工具或是过程管理工具）用来表示过程的关键环节，以便更好地理解过程。这些工具也可以提供对于过程描述的链接，以协助过程的参与人员了解并掌握所需完成的操作及工作任务。过程建模工具还提供了与其他工具的链接，以支持所定义的过程活动。

[机制] 这类工具辅助开发团队定义一个特定过程模型的组成部分（如动作、任务、工作产品、质量保证点等），为每个过程元素的描述和内容定义提供详细的指导，并管理过程执行。在某些情况下，过程技术工具包括标准的项目管理任务，如估算、进度计划、跟踪和控制。

[代表性工具] ⊖

- Igrafx Process Tools。对软件过程进行映射、度量和建模的一组工具（http://www.igrafx.com/）。
- Adeptia BPM Server。用来管理、增强自动化、优化业务过程的工具（www.adeptia.com）。
- ALM Studio Suite。一个重点关注对沟通和建模活动进行管理的工具集（http://www.kovair.com）。

4.6 产品和过程

如果过程很薄弱，则最终产品必将受到影响。但是对分过程的过分依赖也是很危险的。Margaret Davis[Dav95a] 在多年前写的一篇简短的文章里对产品和过程的双重性进行了以下评述：

大约每十年或五年，软件界都会对“问题”重新定义，其重点由产品问题转向了过程问题。因此，我们逐步采纳了结构化程序设计语言（产品）、结构化分析方法（过程）和数据封装（产品），到现在重点是卡内基·梅隆大学软件工程研究所提出的能力成熟度模型（过程）（随后逐步采纳面向对象方法和敏捷软件开发）。 62

钟摆的自然趋势是停留在两个极端的中点，与之类似，软件界的关注点也不断地摆动，当上一次摆动失败时，就会有新的力量加入，促使它摆向另一个方向。这些摆动是非常有害的，因为它们可能从根本上改变了工作内容及工作方法，使软件工程实践人员陷入混乱。而且这些摆动并没有解决问题，只是把产品和过程分裂开来而不是作为辩证统一的一体，因此注定要失败。

这种二象性在科学界早有先例，当某一个理论不能对观测到的相互矛盾的结果做出合理解释时，就会出现二象性理论。由 Louis de Broglie 于 20 世纪 20 年代提出的光的波粒二象性就是一个很好的例子。我相信，我们对软件组成部分和开发过程的观测证明了软件具有过程和产品的二象性。如果仅仅将软件看作一个过程或是一个产品，那就永远都不能正确地理解软件，包括其背景、应用、意义和价值。

所有的人类活动都可以看成一个过程，我们每一个人都从这些活动中获得对自我价值的认识，这些活动所产生的结果可以被许多人反复地在不同的情况下使用。也就是说，我们是

⊖ 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。大多数情况下，工具的名字由各自的开发者注册为商标。

从我们自己或是他人对我们产品的复用中得到满足的。

因此，将复用目标融入软件开发，这不仅潜在地增加了软件专业人员从工作中获得的满足感，也增加了接受“产品和过程二象性”这一观点的紧迫性。对于一个可复用的部件，如果仅仅从产品或是仅仅从过程的角度考虑，都不利于软件开发，这种片面的观点或者影响了人们对产品的应用环境和应用方法的认识，或者忽略了该产品还可以作为其他开发活动的输入这一事实。因此，片面地强调某一方面的观点会极大地降低软件复用的可能性，也会大大减少工作的成就感。

正如从最终产品获得满足一样，人们在创造性的过程中得到了同样的（甚至更大的）成就感。艺术家不仅仅对装裱好的画卷感到高兴，更在每一笔绘画的过程中享受乐趣；作家不仅欣赏出版的书籍，更为每一个苦思得到的比喻而欣喜。一个具有创造性的专业软件人员也应该从过程中获得满足，其程度不亚于最终的产品。产品和过程的二象性已经成为保留推动
63 软件工程不断进步的创造性人才的一个重要因素。

4.7　小结

传统软件过程模型已经使用了多年，力图给软件开发带来秩序和结构。每一个模型都建议了一种不同的过程流，但所有模型都实现同样的一组通用框架活动：沟通、策划、建模、构建和部署。

像瀑布模型和V模型这类顺序过程模型是最经典的软件工程模型，顺序过程模型建议采用线性过程流，这在软件世界里通常与当代的软件开发现实情况不符（例如持续的变更、演化的系统、紧迫的开发时间）。但线性过程模型确实适用于需求定义清楚且稳定的软件开发。

增量过程模型采用迭代的方式工作，能够快速地生成一个软件版本。演化过程模型认识到大多数软件工程项目的迭代、递增特性，其设计的目的是为了适应变更。演化模型，例如原型开发及螺旋模型，会快速地产生增量的工作产品（或是软件的工作版本）。这些模型可以应用于所有的软件工程活动——从概念开发到长期的软件维护。

并发过程模型为软件团队提供了过程模型中的重叠和并发元素的描述方法。专用模型主要包括基于构件的模型，强调软件构件的重用和组装；形式化方法模型提倡采用数学方法进行软件开发与验证；面向方面的模型的目的是解决跨越整个软件体系结构的横切关注问题。统一过程模型是一种“用例驱动、以体系结构为核心、迭代及增量”的软件过程框架，由UML方法和工具支持。

软件过程的个人模型和团队模型都强调成功软件过程的关键因素：测量、策划和自我管理。

习题与思考题

4.1　详细描述三个适于采用瀑布模型的软件项目。

4.2　详细描述三个适于采用原型模型的软件项目。

4.3　如果将原型变成一个可发布的系统或者产品，应该如何调整过程？

4.4　详细描述三个适于采用增量模型的软件项目。

64 4.5　当沿着螺旋过程流发展的时候，你对正在开发或者维护的软件的看法是什么？

4.6　可以合用几种过程模型吗？如果可以，举例说明。

4.7 并发过程模型定义了一套“状态”，用你自己的话描述一下这些状态表示什么，并指出他们在并发过程模型中的作用。

4.8 开发质量“足够好”的软件，其优点和缺点是什么？也就是说，当我们追求开发速度胜过产品质量的时候，会产生什么后果？

4.9 详细描述三个适于采用基于构件模型的软件项目。

4.10 我们可以证明一个软件构件甚至整个程序的正确性，可是为什么并不是每个人都这样做？

4.11 统一过程和 UML 是同一概念吗？解释你的答案。

扩展阅读与信息资源

第 2 章的扩展阅读部分提到的大部分资料都详细地介绍了传统过程模型。

Cynkovic 和 Larsson（《Building Reliable Component-Based Systems》，Addison-Wesley，2002）以及 Heineman 和 Council（《Component-Based Software Engineering》，Addison-Wesley，2001）描述了实现基于构件系统的过程需求。Jacobson 和 Ng（《Aspect-Oriented Software Development with Use Cases》，Addison-Wesley，2005）以及 Filman 和他的同事（《Aspect-Oriented Software Development》，Addison-Wesley，2004）讨论了面向方面过程的独特性质。Monin 和 Hinchey（《Understanding Formal Methods》，Springer，2003）提供了有价值的介绍，Baco 和他的同事（《Formal Methods》，Springer，2009）讨论了目前的最新水平和新方向。

Kenett 和 Baker（《Software Process Quality：Management and Control》，Marcel Dekker，1999）以及 Chrissis、Konrad 和 Shrum（《CMMI for Development：Guidelines for Process Integration and Product Improvement》，3rd ed.，Addison-Wesley，2011）考虑了高质量的管理和过程设计是如何相互影响的。

除了 Jacobson、Rumbaugh 和 Booch 的有关统一过程的书籍 [Jac99] 以外，Shuja 和 Krebs（《IBM Rational Unified Process Reference and Certification Guide》，IRM Press，2008）、Arlow 和 Neustadt（《UML 2 and the Unified Process》，Addison-Wesley，2005）、Kroll 和 Kruchten（《The Rational Unified Process Made Easy》，Addison-Wesley，2003）以及 Farve（《UML and the Unified Process》，IRM Press，2003）等人的书籍提供了很好的补充信息。Gibbs（《Project Management with the IBM Rational Unified Process》，IBM Press，2006）讨论了统一过程中的项目管理问题。Dennis、Wixom 和 Tegarden（《System Analysis and Design with UML》，4th ed.，Wiley，2012）解决涉及 UP 的编程和业务过程建模问题。

网上有大量关于软件过程模型的信息源，与软件过程有关的最新参考文献可以在 SEPA 网站 www.mhhe.com/pressman 找到。