第 2 章　过程和生命周期的建模

本章将介绍

- 过程的含意
- 软件开发的产品
- 过程和资源
- 软件开发过程的若干模型
- 过程建模的工具和技术

我们在第1章中看到，软件工程是一个创造性的、逐步推进的过程，常常涉及到许多开发不同类型产品的人员。在本章中，我们更深入地探讨这些步骤，研究活动的组织方式，这样我们才能够协调做什么，什么时候去做。本章首先定义过程的含意，以便我们理解对软件开发建模时必须包含什么因素。接着讨论若干类型的软件过程模型。当了解了我们希望使用的模型类型后，再仔细地讨论两种建模技术：静态的和动态的。最后把其中一些技术应用到信息系统和实时系统的例子中。

2.1　过程的含意

当提供一项服务或开发一个产品（无论是开发软件、撰写报告还是商务旅行）时，我们总是按照一系列的步骤来完成任务。通常每次都用同样的顺序完成任务，例如，在房子的管线安装之前通常不能建造石墙，或者在所有的配料混合之前不能烤饼。我们可以把有顺序的任务集合称为过程：即一系列涉及到活动、约束和资源的步骤，它们产生某种类型的有目的的输出。

正如我们在第1章中定义的那样，一个过程常常涉及到一系列的工具和技术。任何过程都具有如下特征：

- 过程规定了所有主要的过程活动。
- 过程使用资源，服从于一组约束（比如进度约束），产生中间结果和最终产品。
- 过程由子过程组成，这些子过程用某种方式链接起来。过程可以定义为分层的过程等级结构，以便每一个子过程具有自己的过程模型。
- 每一个过程活动具有入口和出口标准，这样可知道活动何时开始以及何时结束。
- 活动以一定的顺序加以组织，因此，一个活动相对于其他活动何时完成是很清楚的。
- 每一个过程具有一系列的指导原则，这些指导原则解释了每一个活动的目标。
- 约束或控制可以应用到任何活动、资源或产品中。例如，预算或进度可以限制活动需要的时间长短，工具可以限制资源使用的方式。

当过程涉及到某些产品的开发时，有时把这种过程称为生命周期。因此，软件开发过程有时被称为软件生命周期，因为它从概念到实现、交付、使用和维护描述了软件产品的生命。

过程的重要性在于它使一组活动具有了一致性和结构。当我们知道如何能把事情做好而且希望保证其他人也用同样的方式完成时，这些特性是很有用的。例如，如果Sam是一个好的瓦工，他可以记下他的砌砖过程，这样Sara也就能够学习如何把砖砌好了。他可以考虑人们做事方式的不同；比如，他记下要传授的知识时，使Sara不论用左手还是右手都可以学会怎样砌砖。类似地，软件开发过程也能够用灵活的方式描述，使得人们能使用自己喜欢的技术和工具设计和开发软件；一个过程模型可能要求在编码之前产生设计，但是它可能允许采用许多不同的设计技术。正因如此，过程有助于在不同人员开发的产品和服务中保持一定的一致性和质量。

过程不仅仅是一个程序。在第1章中我们看到一个程序类似于一个菜谱，是把工具和技术组合起来生产产品的一种结构化方式。过程是程序的集合，组合起来以产生满足目标和标准的产品。事实上，在达到我们想要的目标时，过程会建议我们从若干步骤中进行选择。例如，过程要求我们在开始编码之前检查设计组件。可以通过非正式的评审或正式的审查来完成这个任务，每一个活动都有自己的步骤，但是两者都达到了同样的目的。

过程结构用检查、理解、控制和改善组成过程的活动来指导我们的行为。为了了解其过程，我们以如何制作带有巧克力糖衣的巧克力饼为例。这个过程可能包含若干个步骤，比如购买各种配料以及寻找合适的厨房用品。食谱描述了混合配料和烤饼的实际过程。食谱包含活动（比如在把其他的配料混合之前先搅拌鸡蛋）、约束（比如巧克力与糖混合之前先把巧克力加热到融点的温度要求）和资源（比如糖、面粉、鸡蛋和巧克力）。假设Chuck根据这个食谱烤制了巧克力饼。当饼做好以后他品尝了一下，感到饼太甜了。他检查食谱了解到甜是由糖份产生的。然后，他烤了另一份饼，但是这次他在新食谱中减少了糖的用量。他又尝了一下饼，但是现在它没有足够的巧克力味道。于是他的第二版食谱中增加了可可粉的用量，然后又烤了一次。经过几次的反复，他每一次都改变了成份或活动（比如把饼烤得时间长些，或让巧克力与鸡蛋混合之前先冷却下来），Chuck终于得到了他喜欢的饼。如果没有食谱记录这部分过程，Chuck将不能轻易地改变以及评估结果。

过程的重要性还在于它能使我们获取经验并把它传授给别人，正象一个主厨把他们喜欢的食谱传授给他们的同事和朋友一样，杰出的工匠能够把记载的过程和程序传授下来。实际上，学徒和辅导的概念都是基于这样一种思想：共享我们的经验，从而可以把技能从资深人员传授给初学者。

同样地，我们希望从过去的开发项目中学习、记录下运作得最好的、产生高质量软件的实践，并遵循软件开发过程，这样，当我们为顾客开发产品时，就能够理解、控制和改善发生的事情。我们在第1章中看到，软件开发通常涉及到如下阶段：

- 需求分析和定义
- 系统设计
- 程序设计
- 编写程序（程序实现）

- 单元测试
- 集成测试
- 系统测试
- 系统交付
- 维护

每一阶段本身就是一个过程（或过程的集合）。这个过程可以描述为活动的集合。并且每一个活动涉及到约束、输出和资源。例如，需求分析和定义需要用户用某种方式给出功能和特征的说明作为初始输入。本阶段最终的输出是需求集合，但是，当用户和开发人员之间的对话导致了变化和其他方案时，有可能存在中间产品。我们也有一些约束，比如产生需求文档的预算和进度，包含的需求种类的标准以及所使用的表达需求的符号。

本书描述了所有这些阶段。对每一个阶段来说，我们仔细研究所涉及到的过程、资源、活动和输出，我们将了解到它们如何对最终产品（即有用的软件）的质量产生影响。针对每一个开发阶段都有很多方法；活动、资源和输出的每一种配置构成了一个过程，而过程的集合描述了在每一阶段发生了什么。例如，设计可能涉及到一个原型化过程，这里需要研究许多设计决策以便开发人员能够选择合适的途径和复用过程，该复用过程使以前产生的设计组件能引入到当前的设计中。

每一个过程都可以用不同的方式描述，使用文本、图形或两者的结合。软件工程研究人员为这种描述提出了不同的方式，常常组织为包含关键过程特征的原型。在本章的其余部分中，我们研究不同的软件开发过程模型，了解如何对过程活动进行组织以使开发更具效率。

2.2　软件过程模型

软件工程文献介绍了许多软件过程模型。有些是软件开发应该遵循的步骤，而另一些则描述了完成软件开发的实际步骤。理论上，两种类型的模型应当是类似的或相同的，但事实并非如此。建立过程模型并讨论它的子过程有助于开发小组理解理论和实际的差距。

还有很多其他原因也需要对过程建模：

- 当一个开发小组记录下对开发过程的描述时，形成了对软件开发中涉及到的活动、资源和约束的共同理解。
- 建立过程模型有助于开发小组发现过程及其组成部分中的不一致、冗余和遗漏。当注意到和纠正了这些问题以后，过程会变得更加有效，从而把重点放在开发最终产品上。
- 模型应该反映开发的目标，比如构建高质量的软件、在开发的早期发现错误、并且满足所需的预算和开发进度的约束。当模型建立以后，开发小组针对这些目标评价候选活动的合适性。例如，开发小组可以引入需求评审，这样可以在设计开始之前发现和修正需求中的问题。

- 应当根据每一个过程将被使用的特殊情况对其进行裁剪。建立过程模型帮助开发小组理解哪里需要进行裁剪。

每一个软件开发过程模型都含有输入（系统需求）和输出（一个交付的产品）。这些年来已经提出了许多这样的模型。让我们回顾一下若干最流行的模型，以理解它们的共性和区别。

瀑布模型

研究人员们提出的第一个模型是瀑布模型，如图2.1所示，其中各阶段被描述为从一个阶段到另外一个阶段的瀑布（Royce 1970）。该图意味着一个开发阶段必须在另一个开发阶段开始之前完成。这样，当得到了顾客的所有需求、分析了完整性和一致性，并完成了需求文档之后，开发小组才开始进行系统设计活动。瀑布模型对开发中进行的活动表达了一种非常高层次的观点，而且它提出了开发人员应该经过的事件序列。

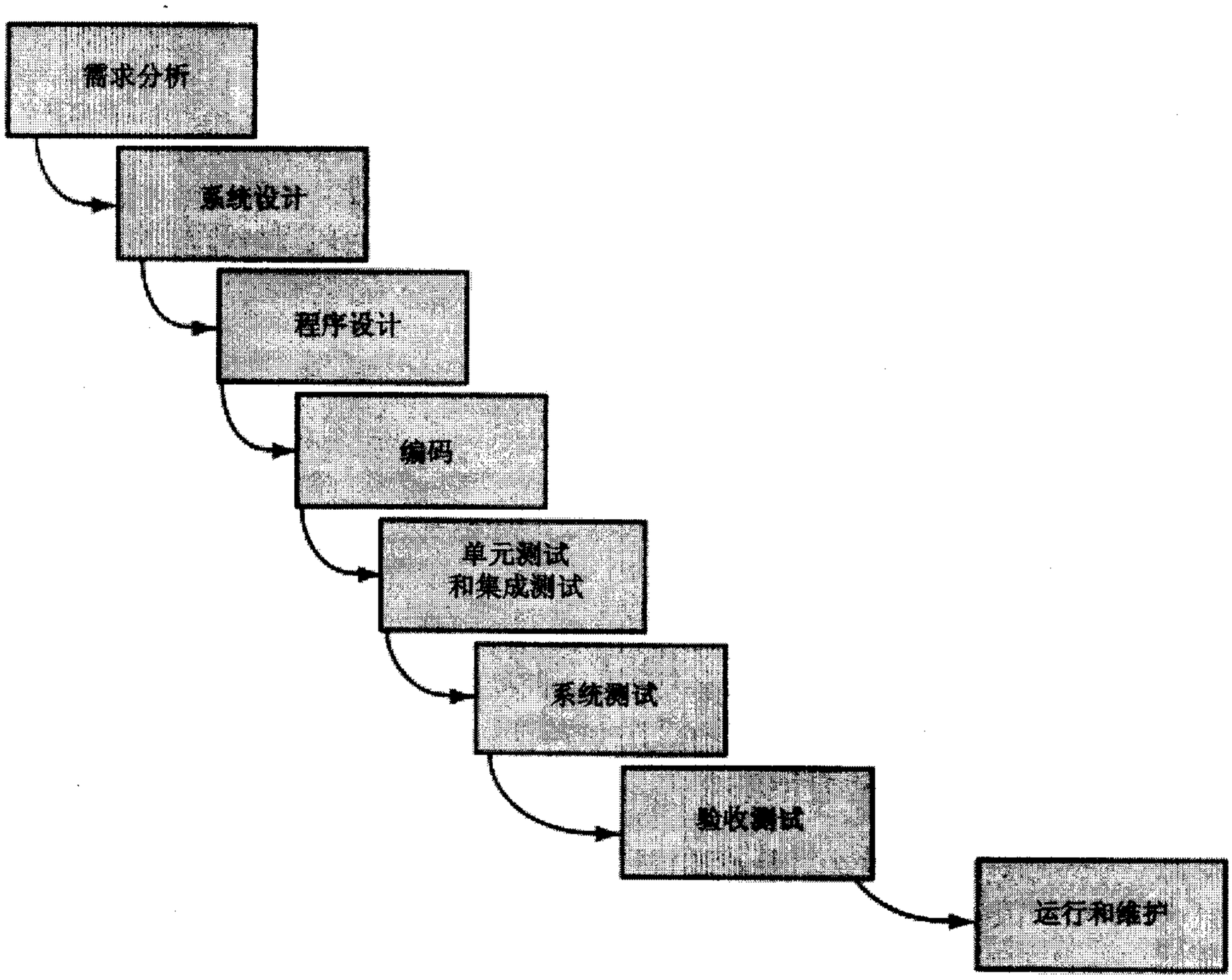

图 2.1 瀑布模型

瀑布模型被用于在各种情况下规定软件开发活动。例如，它是多年来美国国防部软件开发合同中交付使用的基础，在国防部标准2167-A中定义。与每一个过程活动相联系的是里程碑和可移交品，这样，项目经理能够使用模型去度量在给定的时刻项目距完成还有多远。例如，在瀑布模型中“单元测试和集成测试”阶段以“被编写、测试和集成的代码模

块”作为结束的里程碑；中间产品是被测试代码的副本。接着，代码被移交给系统测试人员，以便它可以与其他系统组件（硬件或软件）集成，作为一个整体进行测试。

在开发人员设计他们需要做什么时，瀑布模型是非常有用的。它的简明性使得人们能够轻而易举地向不熟悉软件开发的顾客清楚地作出解释；它清楚地说明了为开始下一阶段的开发需要哪些中间产品。而且，更复杂的模型实际上只不过是瀑布模型的修改，加上反馈循环和其他的活动。

瀑布模型的许多问题已经在各种文献中讨论过了，附注2.1总结了其中两个问题。瀑布模型最大的问题就是它并不能反映实际的代码开发方式。除了非常容易理解的问题之外，软件开发中常常会有许多迭代。通常情况下，软件常用于解决以前从未解决过的问题，或者需要升级解决方案以反映商业环境和操作环境的变化。例如，一个飞机制造商需要针对一种新机身的软件，它比当前的模型更大而且更快，因此，软件开发人员即使在开发航空软件方面具有大量经验，他们仍然面临很多新的挑战。用户和开发人员都不知道影响所希望的结果的所有关键因素，并且正如我们将在第4章中看到的那样，在需求分析方面花费的大量时间都用在了理解系统及其软件所影响的各子项和过程，以及系统和它所处的操作环境之间的关系上。因此，如果不对实际的软件开发过程加以控制的话，结果看起来将会像图2.2那样；当开发人员试图搜集关于问题以及所建议的解决方案如何描述该问题的有关知识的时候，开发人员将会往返于一个活动和另一个活动之间。

附注2.1 瀑布模型的缺点

自从引入瀑布模型以来，它已经受到了很多的批评。例如，McCracken 和 Jackson（1981）指出，这个模型给系统开发强加了一种项目管理结构。“无论主张哪种生命周期模型（即使它具有各种变种）能够用在所有的系统开发中，都是不现实的，或是由于对生命周期作了过分基本的假设以致于毫无意义的。”

注意，瀑布模型说明了每一个主要的开发阶段是如何产生某些工件（比如需求、设计或编码）的。其中并没有深入地探讨每一个活动如何把一种工件转化成另外一种工件（例如，从需求转化成设计）。因此，模型并没有向管理者和开发人员提供关于如何处理在开发过程中可能出现的产品和活动的变化的指导。例如，当在编码活动中需求发生变化时，瀑布模型并没有解决设计和编码中随之而来的变化的问题。

Curtis、Krasner、Shen和Iscoe（1987）指出，瀑布模型的主要缺点是它没能把软件看成是一个问题解决的过程。瀑布模型源自硬件领域，表达了软件开发的制造观点。但是，制造是生产某一特定的产品并且重复生产若干次。而软件并不是那样开发出来的；相反，它随着人们对系统的理解和对备选方案的评价的进展而演化。因此，软件开发是一个创造的过程，而不是一个制造的过程。瀑布模型没有告诉我们开发最终产品所需的典型的不断改进的活动。特别是，创造常常涉及到进行不同的尝试、开发和评估原型、评估需求的可行性、比较若干种设计、以及从失败中学习经验，从而最终得到问题的满意解决方案。

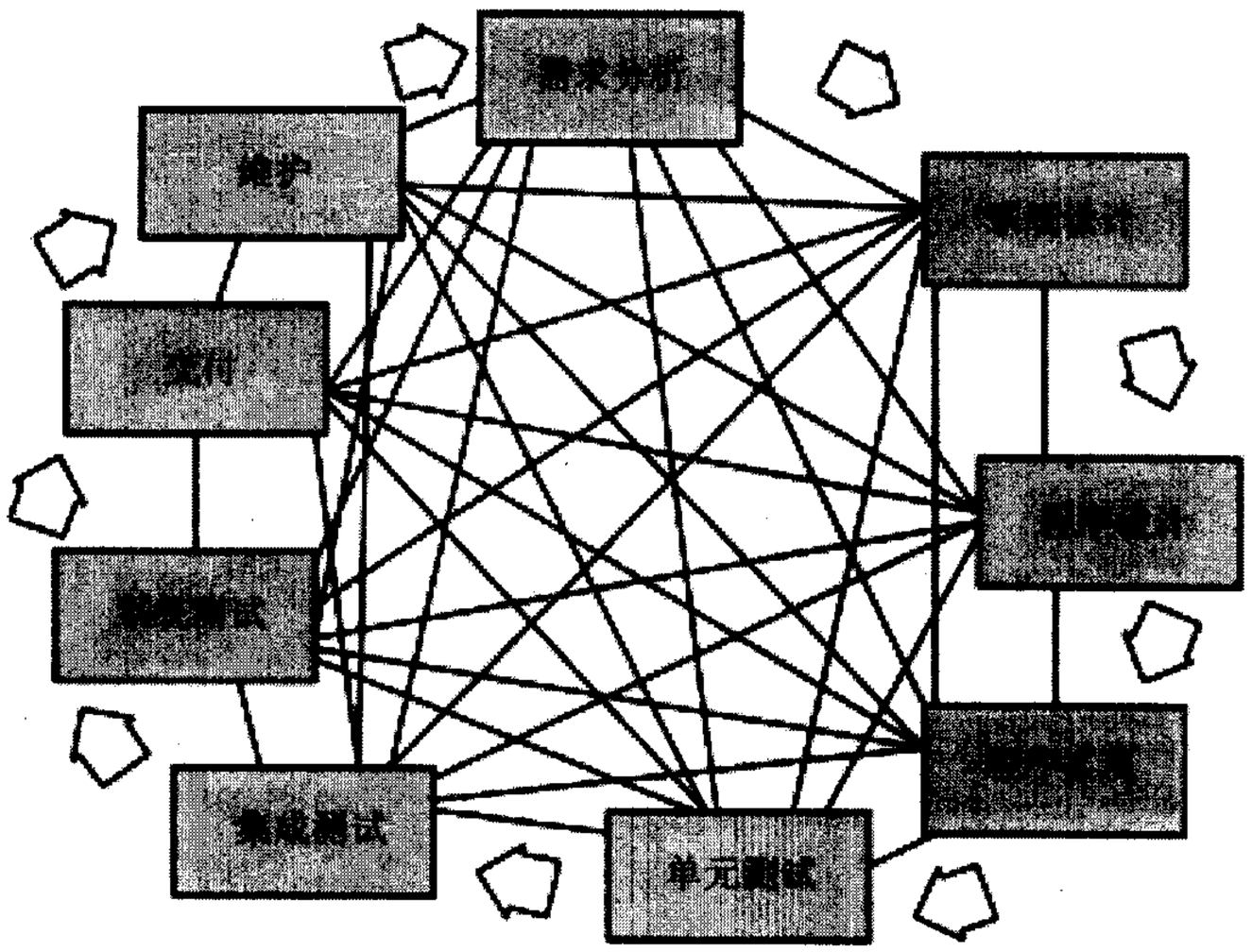

图 2.2 实际的软件开发过程

软件开发过程通过引入加强理解的活动和子过程来帮助控制开发过程中出现的变化。原型化就是这样一个子过程；一个原型就是部分开发的产品，这个产品能使顾客和开发人员检验所建议系统的某些方面，并且判断它对最终的产品是否合适。例如，开发人员可以构建一个系统来实现一小部分关键需求，以确保需求是一致的、可行的和切合实际的。如果需求不令人满意，则可以在需求阶段进行修正，而不是在代价较高的测试阶段。类似地，设计过程中的某些部分也可以进行原型化，如图2.3所示。

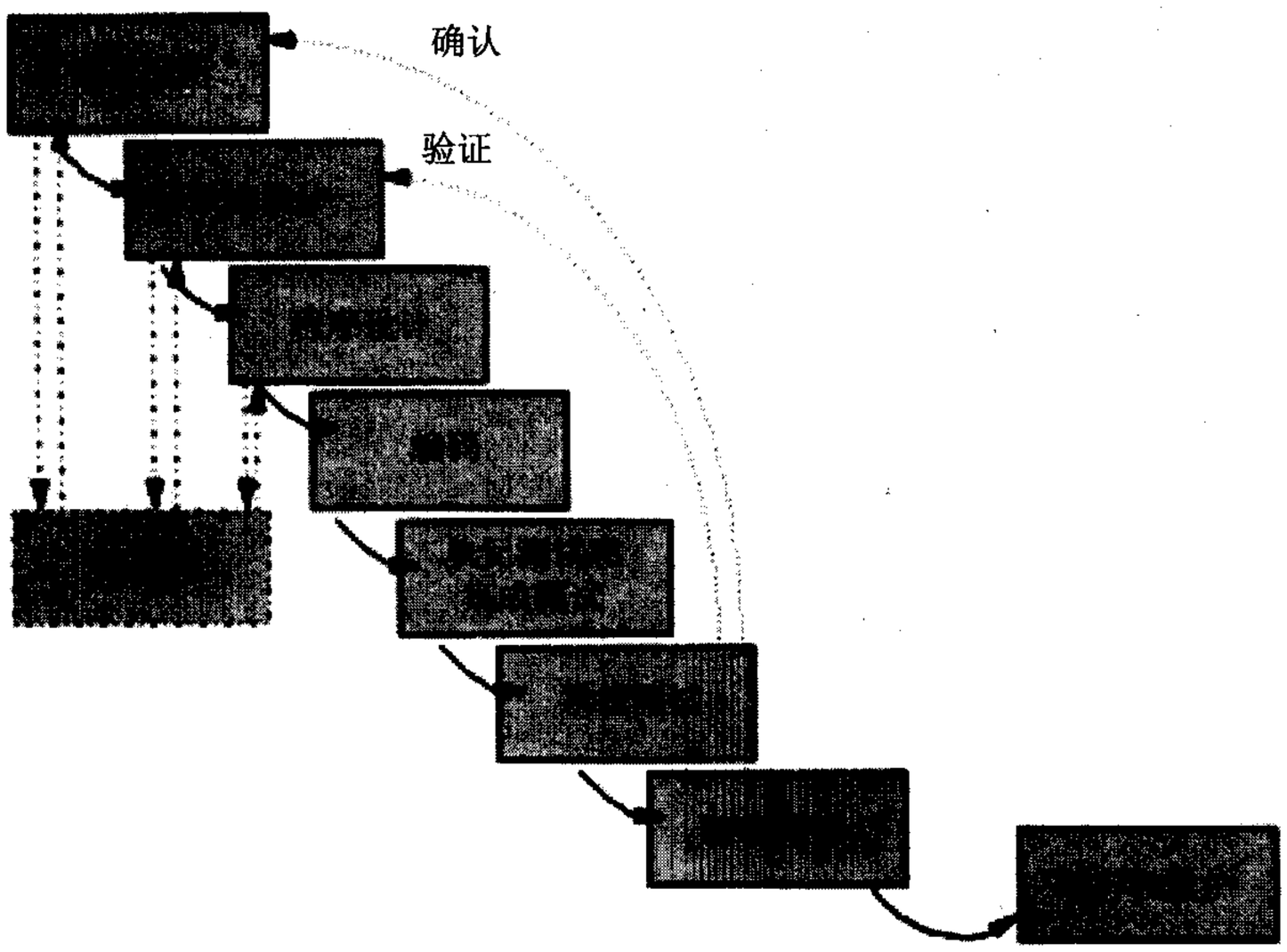

图 2.3 具有原型化的瀑布模型

设计原型化有助于开发人员评估备选的设计方案，并判断对特定问题来说哪一种方案

是最好的。正如我们将在第5章中看到的那样，设计人员可以用若干相差很大的设计来描述需求，从而判定哪一个设计具有最好的特性。例如，可以在一个原型中把网络设计为环形的，另一个是星形的；评价其性能特性以判定在满足性能目标和约束方面哪一种结构更好。

通常情况下，用户界面被作为原型来开发和测试，以便用户了解新的系统的可能形式，并使设计人员能更好地理解用户希望如何与系统交互。因此，在系统测试中需求被正式确认之前，需求中的主要缺陷就已被专门检查和修改过了；确认保证系统实现了所有的需求，这样，每一个系统功能可以回溯到系统说明中的一个特定需求上。系统测试也对需求进行验证；验证确保每一个功能正确运作。也就是说，确认确保开发人员开发的是合适的产品（根据需求说明），而验证检验实现的质量。原型化对验证和确认来说是有用的，但是，我们将在后面的章节中看到，这些活动也可以在开发过程的其他部分出现。

V模型

V模型是瀑布模型的变种，它反映了测试活动与分析和设计的关系（德国国防部1992）。如图2.4所示，编码形成了V的顶点，分析和设计在左边，测试和维护在右边。如我们将在后面的章节中看到的那样，单元和集成测试针对的是程序的正确性。V模型指出，单元和集成测试也可用于验证程序设计。也就是说，在单元和集成测试的过程中，编码人员和测试小组成员应当确保程序设计的所有方面都在代码中得以正确地实现。类似地，系统测试应当验证系统设计，保证系统设计的所有方面都得以正确地实现。由顾客而不是开发人员完成的验收测试把测试步骤与需求说明的每一个元素联系起来，以此来确认需求被正确地实现。这种测试用于在系统被接受和付费之前检验所有的需求是否都完全实现。

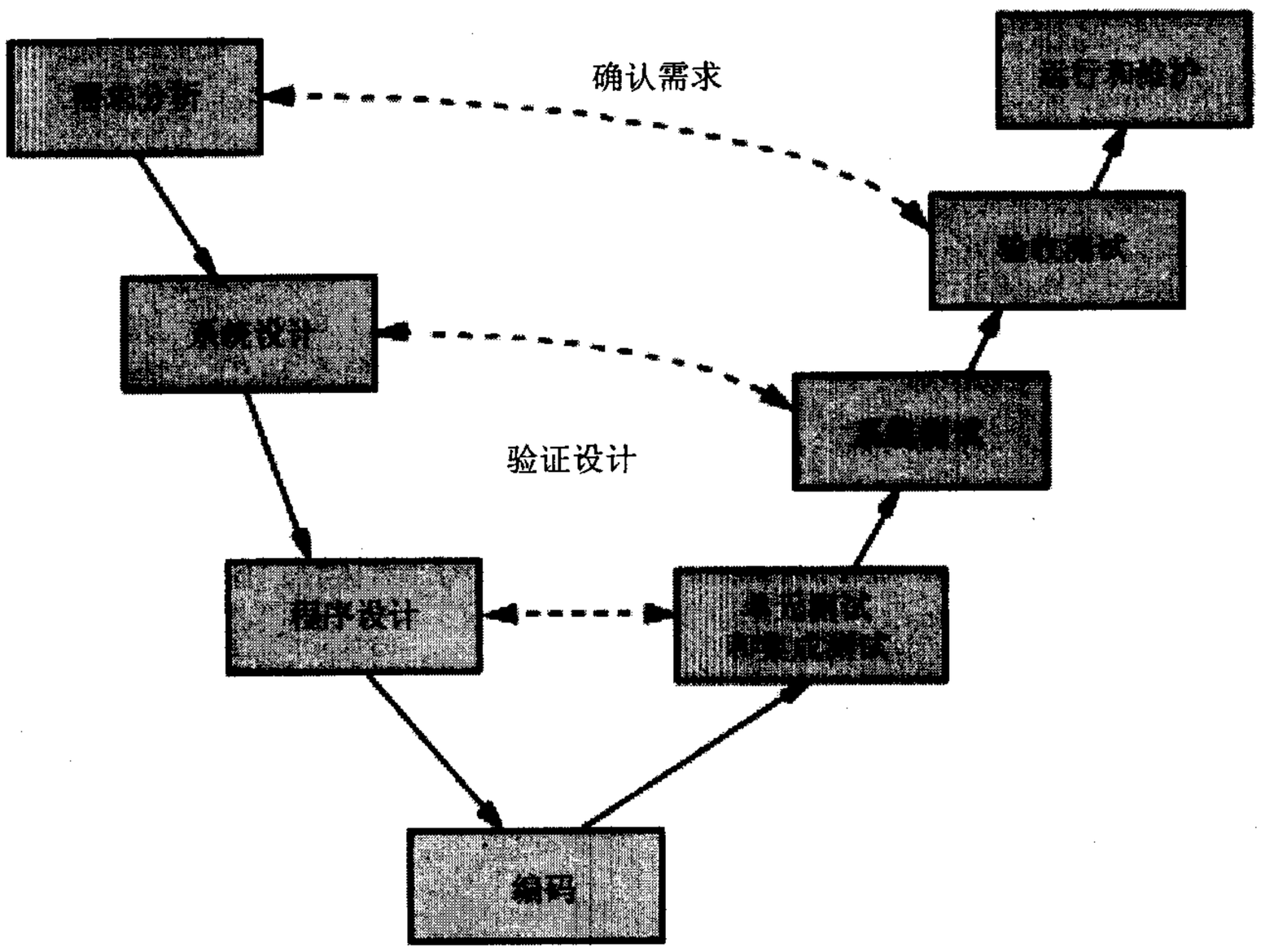

图 2.4　V 模型

该模型中V的左边和右边的连线意味着，如果在验证和确认的过程中发现了问题，那么继续执行右边的测试步骤之前，重新执行V的左边以修正和改进需求、设计和编码。换句话说，V模型使得某些迭代和重新工作更加清淅，而这些工作在瀑布模型中都是隐含的。瀑布模型的中心通常是文档和工件，而V模型的中心则是活动和正确性。

原型化模型

我们已经看到怎样修正瀑布模型的原型化活动以改进对系统的理解。但是原型化不一定只能作为瀑布模型的附属物；如图2.5所示，它本身也是一种有效的过程模型的基础。原型化模型使人们能快速构建整个系统或系统的一部分以理解或澄清问题，因此它与工程化原型具有同样的目的，其中需要对需求或设计进行反复的调查，以确保开发人员、用户和顾客对需要什么和建议什么有一个共同的理解。根据原型化目标的不同，可以消去需求、设计或系统的原型化中的一个或多个循环。不过，总体目标保持不变，即减少开发中的风险和不确定性。

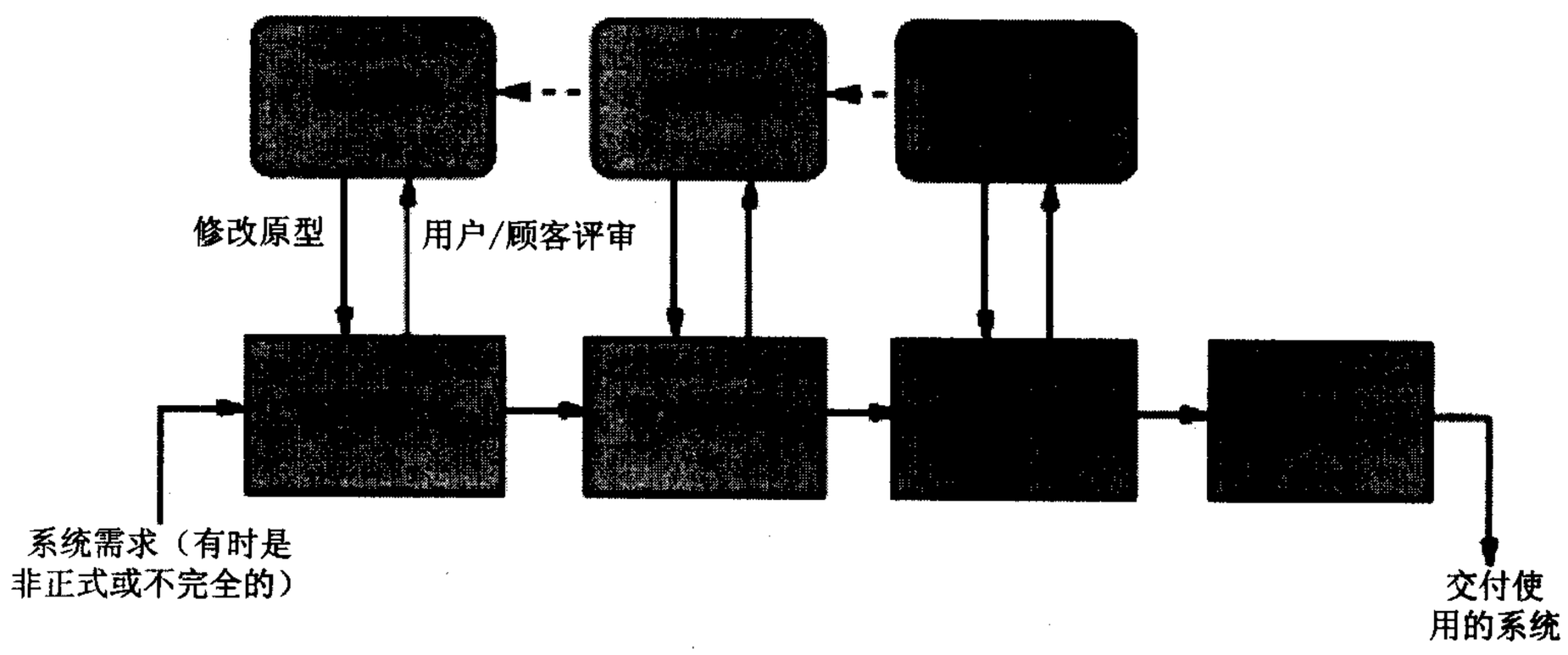

图 2.5　原型化模型

例如，系统开发可能开始于定义需求的集合，这是由顾客和用户提出的。然后，通过让相关团体查看可能的屏幕、表格、报表以及被用户和顾客直接使用的其他系统输出，来探讨不同的方案。当用户和顾客决定他们想要哪种方案的时候，则修正需求。一旦对需求应该是什么样的达成了共识，那么开发人员就可以进行设计了。然后，通过与顾客和用户的协商来探讨不同的设计方案。

我们需要一直修改初始的设计，直到开发人员、用户和顾客对结果很满意为止。事实上，有时考虑不同的设计方案会发现需求中的问题，从而开发人员要返回到需求活动以重新考虑和改变需求说明。最终，对系统进行编码并讨论不同的编码方案，其中可能再次重复需求和设计。

操作说明

对许多系统来说，需求的不确定性导致了以后开发中的变化和问题。Zave（1984）提出了一个过程模型，它允许开发人员和顾客在开发过程早期检验需求和它们的含义，在这个过程模型中他们能够讨论和解决某些不确定性问题。在操作说明模型中，用演示系统行为的方式来评价或执行系统需求。也就是说，一旦指明了需求，就可以用软件包来处理它们，因此，在设计开始之前可以评估它们的含义。例如，如果系统说明要求新建的系统能处理24个用户，那么系统说明的可执行的形式能够帮助分析人员判断用户数目是否给系统增加了太多的性能负担。

这种过程与传统的模型（例如，瀑布模型）不同。瀑布模型把系统的功能与设计分离（也就是说，把系统要做什么与系统如何做分离开来），目的是把顾客的需要与实现分开，而操作说明模型允许把功能和设计合并。图2.6说明了操作说明模型是如何运作的。注意，操作说明模型类似于原型化模型，该过程允许用户和开发人员在早期检查需求。

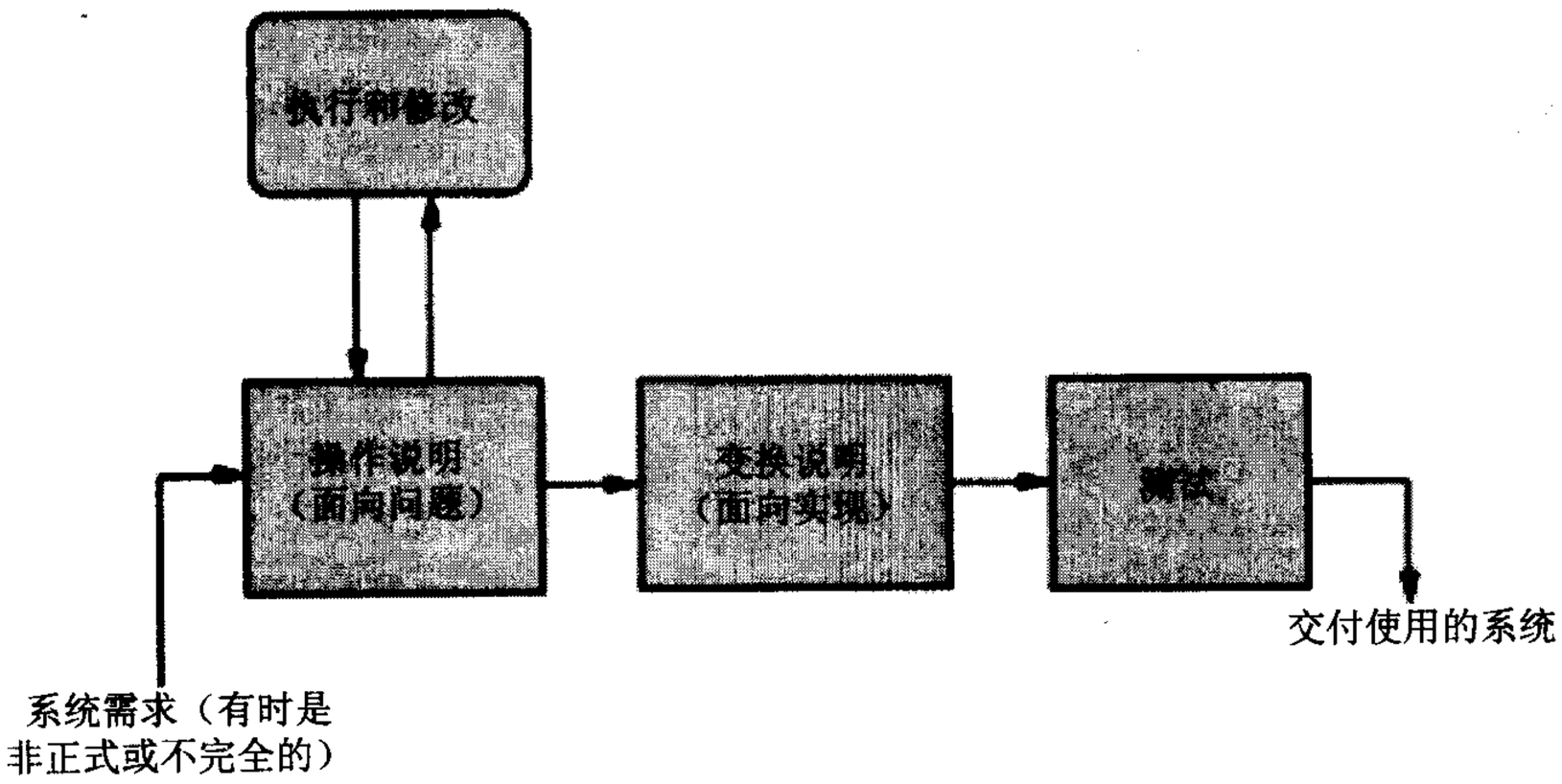

图 2.6　操作说明模型

变换模型

Balzer的变换模型试图通过减少某些主要的开发步骤来减少出错的机会。利用自动化工具的支持，变换过程使用一系列变换把需求说明改变成一个可交付使用的系统（Balzer 1981a）。

变换的形式可以包括：

- 改变数据表示
- 选择算法
- 优化
- 编译

由于从说明到系统的交付使用可以采取很多途径，它们所反映的变换序列和决策都作为正式的开发记录保存。

变换方法具有很好的前景。然而，如图2.7所示，变换方法使用的主要障碍在于需要一份准确表述的形式化说明，这样变换才可以在它上面进行操作。随着形式化说明方法的普遍使用，变换模型将得到更广泛的接受。

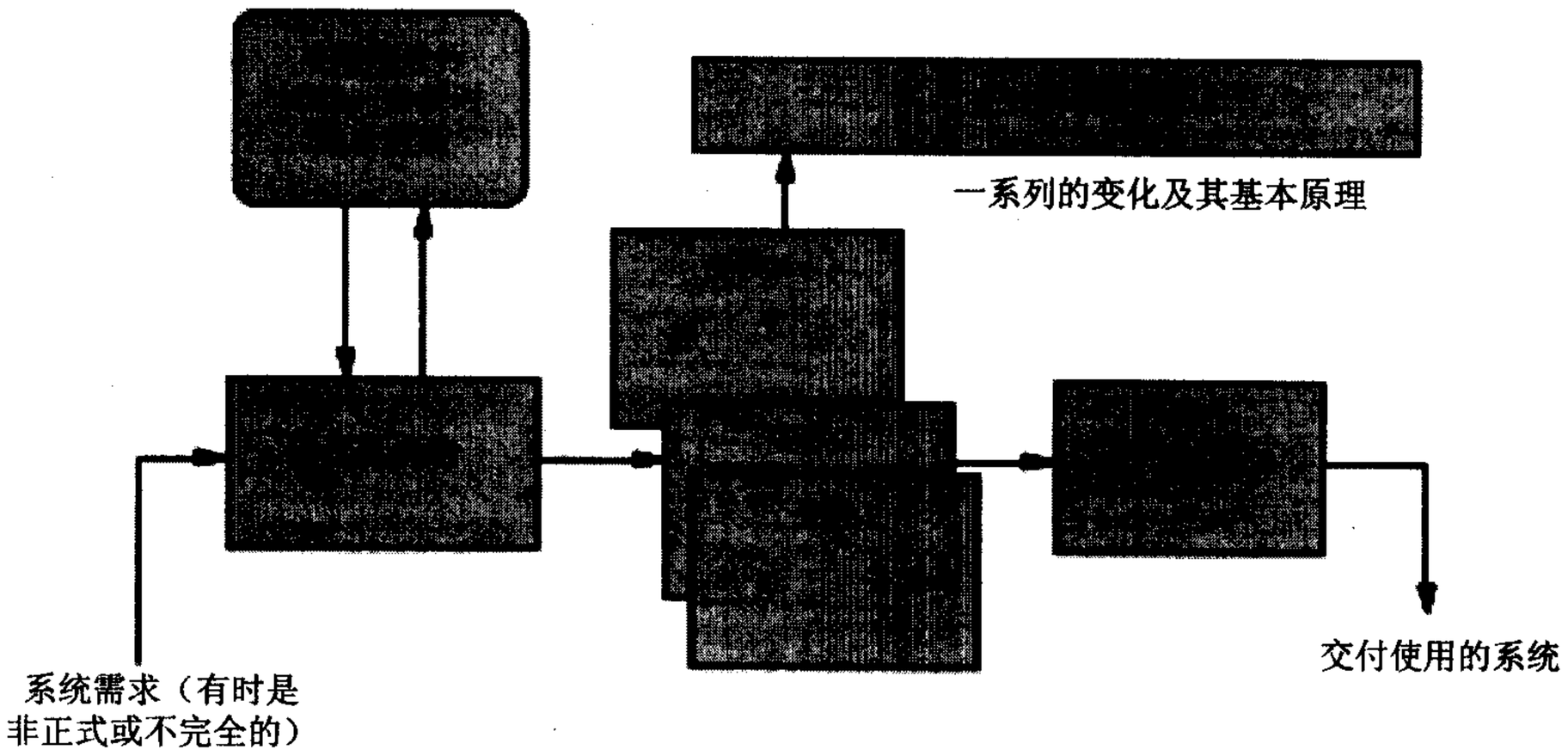

图 2.7　变换模型

阶段化开发：增量和迭代

在软件开发的早期，顾客愿意为软件系统的交付等待很长的时间。有时从写下需求文档到系统交付使用会经过若干年，称为循环周期。然而，今天的商业环境不再能容忍长期的延迟。软件有助于使产品在市场中显得突出，而顾客总是在寻找新的品质和功能。例如，1996年，惠普公司80%的年收入来源于过去两年的产品。因此，人们开发了新的过程模型来帮助减少循环周期。

减少循环周期的一种方法是使用阶段化开发，如图2.8所示。设计系统时使其能一部分一部分地移交，使用户在使用部分功能的同时其余部分仍然在开发过程中。这样，常常会有两个系统并行工作，即产品系统和开发系统。产品系统（或称操作的系统）是一个被顾客和用户使用的系统，而开发系统是准备用来替换现行产品系统的下一个版本。通常，我们用它们的发布版本号代表一个系统：开发人员构建版本1（Release 1），测试它，然后把它交给用户作为第一个可操作的发布版本。然后，当用户使用版本1的时候，开发人员正在构建版本2（Release 2）。这样，在操作版本n时，开发人员总是在构建版本n+1。

开发人员有很多种方法决定如何组织开发以得到发布版本。两种最常用的方法是增量开发和迭代开发。在增量开发中，需求文档中指明的系统按功能划分为子系统。定义发布时首先是定义一个小的、具有一定功能的子系统，然后在每一个新的发布中增加新的功能。图2.9的上面部分显示了增量开发如何逐步在每一个新的发布版本中构建直到功能完整。

迭代开发是在一开始就移交一个完整的系统，然后在每一个新的发布版本中改变每个子系统的功能。图2.9的下面部分列举了一个迭代开发中的三个发布版本。

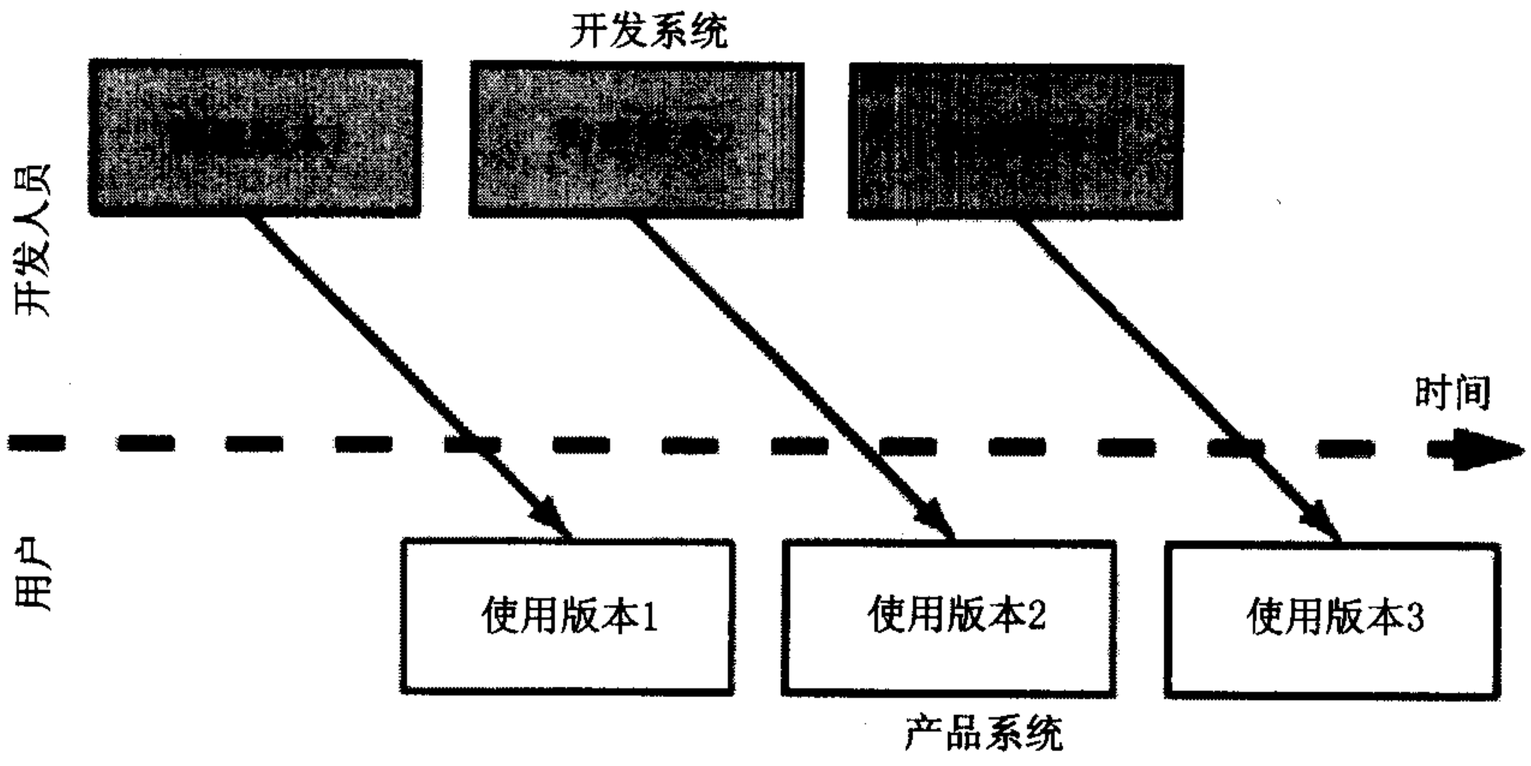

图 2.8　阶段化开发模型

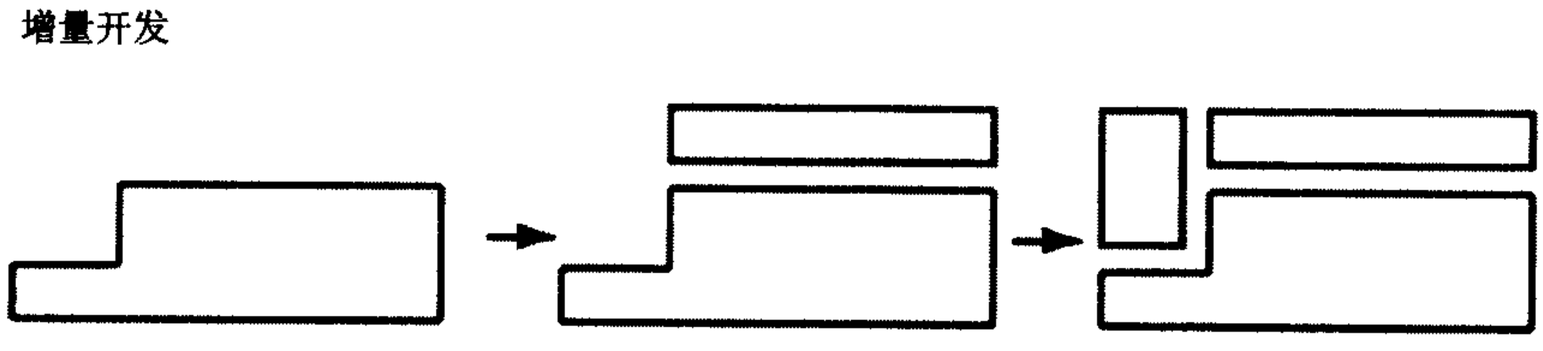

图 2.9　增量和迭代模型

为了理解增量开发和迭代开发之间的区别，考虑一个文字处理软件包。假设这个软件包实现三种类型的功能：创建文本、组织文本（比如：剪切和粘贴）以及文本格式化（比如：使用不同的字体大小和类型）。如果用增量开发构建这样一个系统，我们可能在版本1（Release 1）中仅仅实现创建功能，然后在版本2（Release 2）中实现创建和组织功能，最后在版本3（Release 3）中实现创建、组织和格式化功能。然而，如果使用迭代开发方法，我们可以在发布版本1中实现三种不同类型功能的初始版本。例如：我们可以创建文本，然后剪切并粘贴它，但是剪切和粘贴功能可能不够灵活快捷。因此，在下一次迭代，即版本2（Release 2）中，我们有相同的功能，但是已经改善了性能；现在剪切和粘贴功能能够很方便快捷地实现。每一个版本都以某种形式改进了前一个版本。

在现实中，许多机构综合运用迭代开发和增量开发方法，一个新的发布版本可能包含新的功能，但是已存在的功能也可以得到完善。这种形式的阶段化开发方法是合乎人们需要的，原因如下：

1. 培训可以在早期的版本中开始，甚至在某些功能还未实现时就可以开始了。培训过

程使得开发人员能观察功能的执行情况，从而为后面发布的版本提出改进的建议。通过这种方式，开发人员可以对用户更加负责。

2. 可以为那些以前从未实现的功能提前开拓市场。
3. 当来自使用系统的报告中有未预料到的问题时，不断地发布新版本使得开发人员能全面快速地修正这些问题。
4. 开发小组可以把不同的发布版本针对不同的专业领域。例如，一个发布的版本可以利用用户界面的专业技术把系统从命令驱动变成点击界面，另外一个发布的版本可以集中提高系统性能。

螺旋模型

Boehm（1988）从涉及到的风险的角度来看待软件开发过程，提出用螺旋模型把开发活动和风险管理结合起来，以降低和控制风险。图2.10所示的螺旋模型在某种意义上类似于图2.9所示的迭代开发。它以需求分析和一个最初的开发计划（包括人员、设计和开发环境的预算、约束和备选方案）为开始，在产生“操作概念”文档以从更高的层次描述系统如何工作之前，该过程插入一个评估风险和选择原型的步骤。从该文档中，指定一系列需求，并对之进行详细检查，以确保需求尽可能的完善和一致。因此，操作的概念是第一次迭代的产品，而需求则是第二次迭代的主要产品。在第三次迭代中，系统开发产生设计，而第四次迭代能够进行测试。

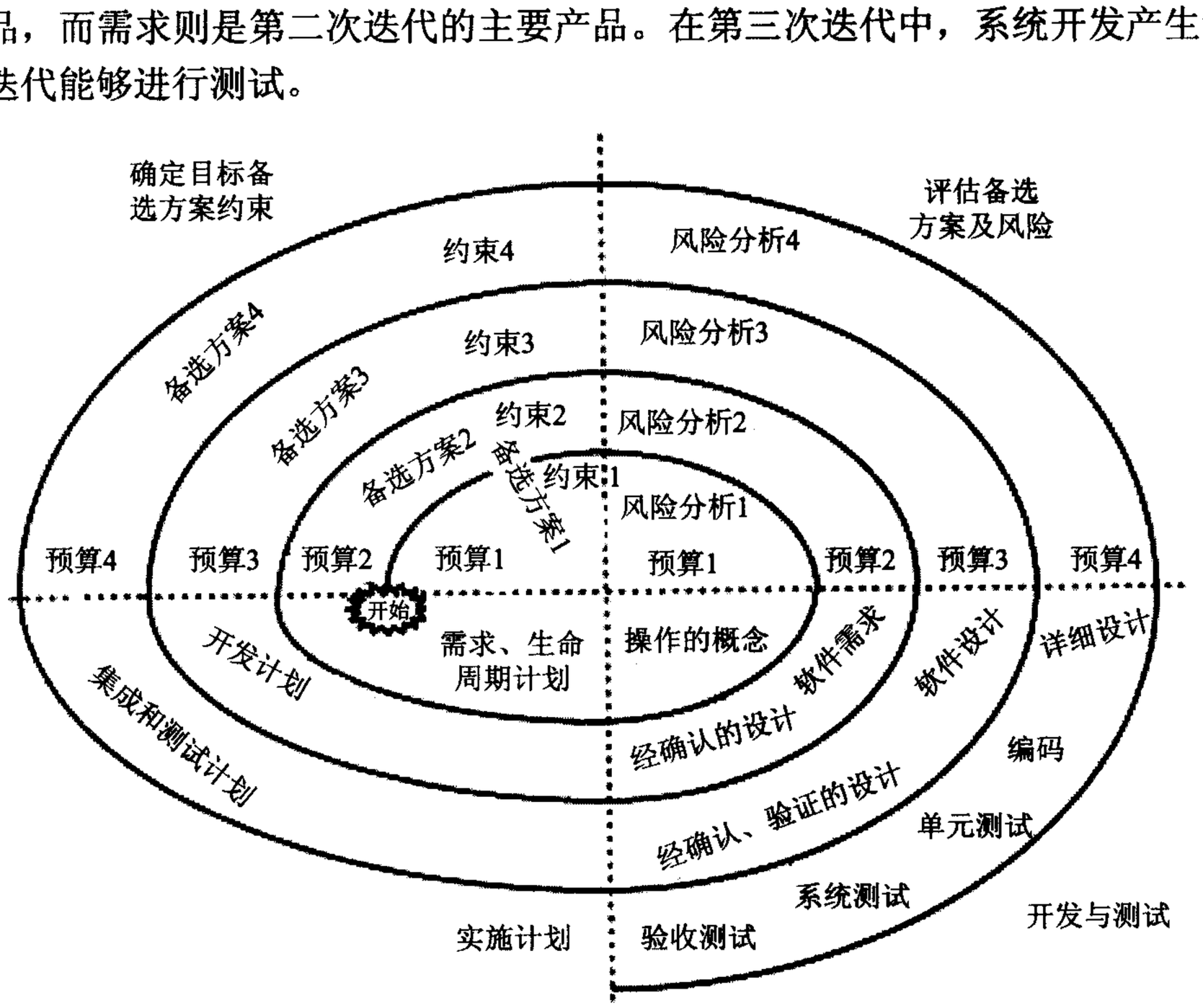

图 2.10　螺旋模型

在每一次迭代中，从需求和约束的角度出发进行风险分析来权衡不同的方案，而原型化方法是在决定某一特定选择之前验证可行性或期望度。当确定了风险以后，项目经理必

须决定如何消除或降低风险。例如，设计人员并不能确定用户是喜欢某一种界面还是喜欢另外一种。此时他们可能会选择阻碍新系统有效使用的界面。为了降低这种风险，设计人员可以把界面原型化并运行该原型以测试用户更喜欢哪一种界面。甚至在设计中选择包含两种不同的界面，这样用户能够在他们登录的时候选择其中一个界面。像预算和进度这样的约束有助于决定要选择哪一种风险管理策略。第3章将进一步地讨论风险管理。

本章中出现的过程模型仅仅是实际使用和讨论的模型的一小部分。其他过程模型可以根据用户、客户和开发人员的需要进行定义和剪裁。正如在附注2.2中提到的那样，我们应该真正地把开发过程当作一个过程模型的集合，而不是集中于某一个模型或观点。

附注2.2　过程模型的集合

我们在附注2.1中看到，开发过程是一个问题求解活动，但是流行的过程模型很少包含问题的求解。Curtis、Krasner 和 Iscoe （1988）对17个大型项目进行了现场调查，以判断过程模型中应获取哪些问题求解的因素来帮助我们理解软件开发。特别是，他们研究了影响项目结果的行为因素和组织因素。结果，他们提出了一个软件开发的分层行为模型，其中包含5个关键因素：商业环境、公司、项目、开发小组和个人。个人的观点提供关于认知和动机的信息，项目和开发小组的观点告诉我们小组的动态。公司和商业环境提供了可能影响生产率和质量的组织行为的信息。这个模型并不是传统过程模型的替代，而是对传统模型的补充，它与传统模型无关，提供了行为是如何影响创建和生产活动的信息。

随着开发人员和顾客对问题的了解，他们把领域知识、技术和业务结合起来，从而得到一个合适的解决方案。通过把开发看作是互相协作的过程的集合，可以了解学习、技术交流、客户交互和需求协商的效果。不过，当前描述一系列开发任务的模型“在下述几个方面是没有用的：分析项目成员必须了解多少新信息、如何协商那些有分歧的需求、项目小组如何解决体系结构的冲突，以及这些因素和类似因素对项目内在的不确定性和风险有何影响”（Curtis、Krasner和Iscoe 1988）。然而，当我们引入认知的、社会的和有组织的过程模型时，我们开始了解瓶颈和低效率的原因。正是这种洞察力使得管理者能够理解和控制开发过程。而通过贯穿于模型各层行为的总体效果，可以了解每个模型对另一个模型因素的影响。

不论使用什么样的过程模型，对所有模型来说许多活动都是共有的。在后面的章节中讨论软件工程时，我们将仔细讨论每一个开发活动，了解它所涉及的内容，并找出使用什么样的工具和技术能够使我们的工作更加有效率。

2.3　过程建模工具和技术

一旦决定了在过程模型中获取什么内容，就会有很多的建模工具和技术可供选择；从前面章节的模型描述中已经看到了若干建模的方法。合适的技术取决于你的目标和想要的

工作方式。特别是，标记符号的选择依赖于你想在模型中获取什么。符号可以是文本方式的，它把过程表达为函数；也可以是图形的，它把过程描述成由正方形和箭头组成的分层结构；也可以是图形和文本组合的，它把图形化的描述与表格和函数组合起来，以详细描述高层的说明。许多过程的标记符号也可用来表达需求和设计；我们将在后面的章节中讨论它们。

本章中，标记符号仅次于模型的类型，对于模型我们集中于两种主要的类别：静态的和动态的。静态模型描述了过程，显示了输入转化为输出的过程。动态模型能够执行过程，这样用户能够看到中间过程和最后的结果是如何转化的。

静态模型：Lai符号

对一个过程静态地建模有多种方法。在20世纪90年代早期，Lai（1991）设计了一种全面的过程符号，这种符号能够让人们在任何细节的层次上对任何过程建模。它定义了一个范式，在这个范式中人员执行角色而资源执行活动，最终导致产品的产生。这个过程模型显示了角色、活动和工件之间的相互关系，而状态表显示了在给定的时间内每个工件的完成情况的信息。

特别地，过程元素有以下7种类型。

1. 活动：过程中将要发生的事情。该元素与下面几个因素有关：该活动之前和之后发生的事情、所需的资源、什么触发了活动的开始、约束活动的规则、如何描述算法和得到的经验教训，以及如何把该活动与项目小组联系起来。
2. 顺序：活动的顺序。顺序可用触发器、程序结构、变换、排序或满足的条件来描述。
3. 过程模型：是关于系统的兴趣的观点。部分过程可以表示为分离的模型，用来预测过程行为或分析一定的特性。
4. 资源：必需的事项、工具或人员。资源可以包括设备、时间、办公空间、人员、技术等。过程模型确定每个活动需要的各种资源的数量。
5. 控制：对执行过程的外部影响。控制可以是手动的或自动的、人工的或机械的。
6. 策略：指导原则。这种高层的过程约束影响过程的执行，它可能包含一个规定的开发过程、必须使用的工具或强制性的管理模式。
7. 组织结构：过程代理者的等级结构，使实际分组与逻辑分组及相关角色相对应。从实际分组到逻辑分组的映射必须足够灵活以反映实际环境的变化。

过程描述本身具有若干层次的抽象，包括引导资源用于构造特定模块的软件开发过程，以及类似于螺旋或瀑布模型的泛型模型。Lai符号包括若干模板，例如工件定义模板（Artifact Definition Template），该模板记录特定的工件信息。

Lai的方法可以用来对软件开发过程建模。在本章后面的部分中，我们利用它对开发过程中涉及到的风险建模。不过，为了展示它在获取复杂活动多个方面上的使用和能力，我们把它应用到开车这样一个相对简单并且熟悉的过程中。表2.1是对这个过程中的关键资源（汽车）的描述。

表2.1　工件“CAR”的工作定义表（Lai 1991）

名称	car	
提要	这是代表汽车类的工件	
复杂性类型	合成	
数据类型	(car c,user-defined)	
工件状态列表		
parked	((state_of(car.engine)=off) (state_of(car.gear)=park) (state_of(car.speed)=stand))	车没有开动，引擎没有发动
initiated	((state_of(car.engine)=on) (state_of(car.key_hole) = has-key) (state_of(car-driver(car.)) = in-car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))	车没有开动，但是引擎发动
moving	((state_of(car.engine) = on) (state_of(car.keyhole) = has-key) (state_of(car-driver(car.)) = driving) ((state_of(car.gear) = drive)or (state_of (car.gear) = reverse)) ((state_of(car.speed) = stand)or (state_of(car.speed) = slow) or (state_of(car.speed) = medium) or (state_of (car.speed) = high))	车向前或向后开动
子工件列表		
	doors	汽车的四扇门
	engine	汽车的引擎
	keyhole	汽车的点火钥匙孔
	gear	汽车的传动装置
	speed	汽车的速度
关系列表		
car-key	这是汽车与钥匙之间的关系	
car-driver	这是汽车和司机之间的关系	

其他的模板定义了关系、过程状态、操作、分析、活动和角色。图形范式表示了元素之间的相互关系，以获取主要的关系和次要的关系。例如，图2.11说明了启动汽车的过程。“初始”框表示入口条件，“停车”框代表出口条件。条件框中的左列列出了工件，右列是工件的状态。

变换图是对过程模型的补充，它展示了状态彼此之间是如何联系的。例如，图2.12说明了汽车状态的变换。

Lai符号很好地说明了如何把多个结构和策略用于获取大量有关软件开发过程的信息。而且，就像汽车例子中所展示的那样，它也可用来组织和描述有关用户需求的过程信息。

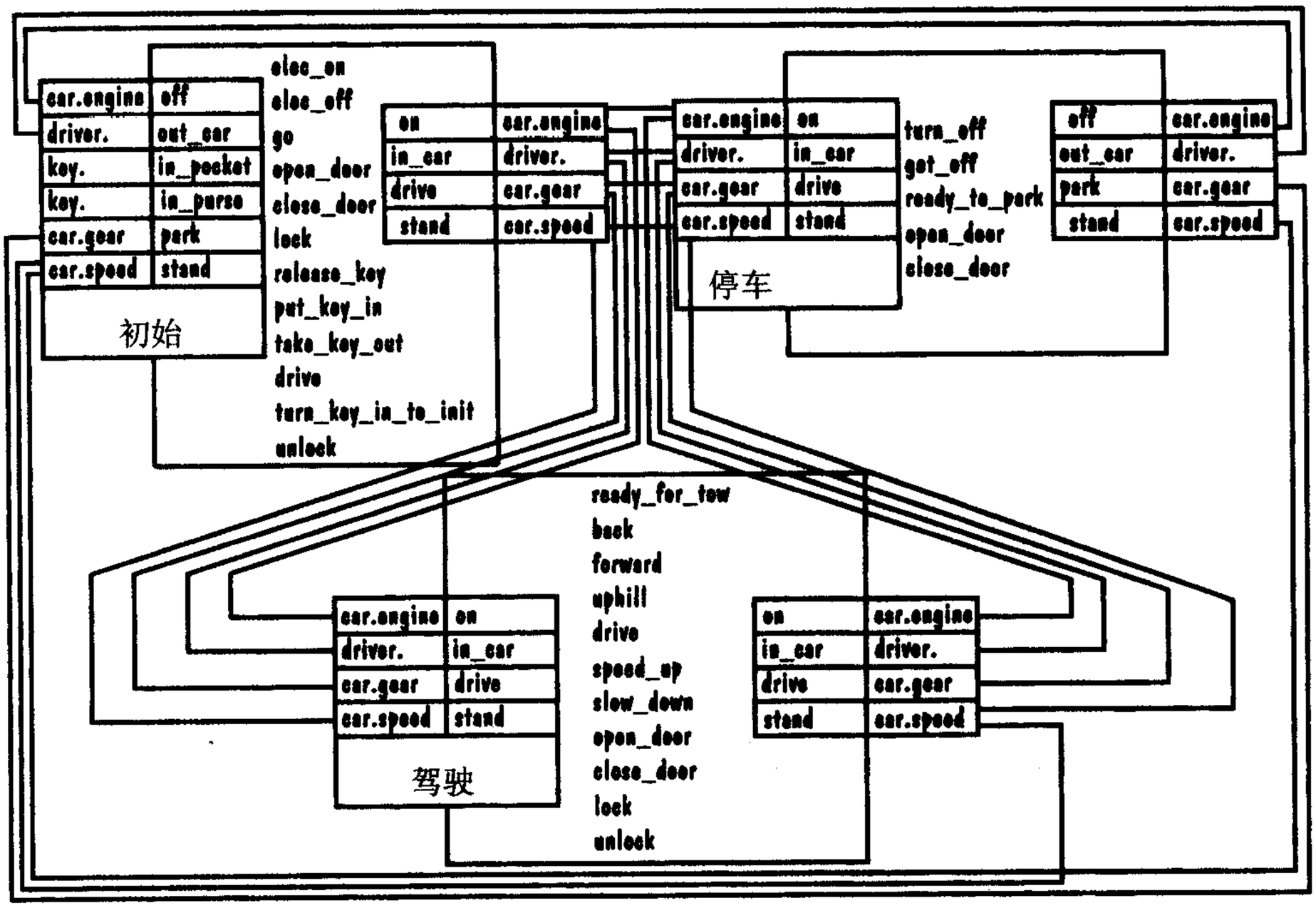

图 2.11　启动汽车的过程（Lai 1991）

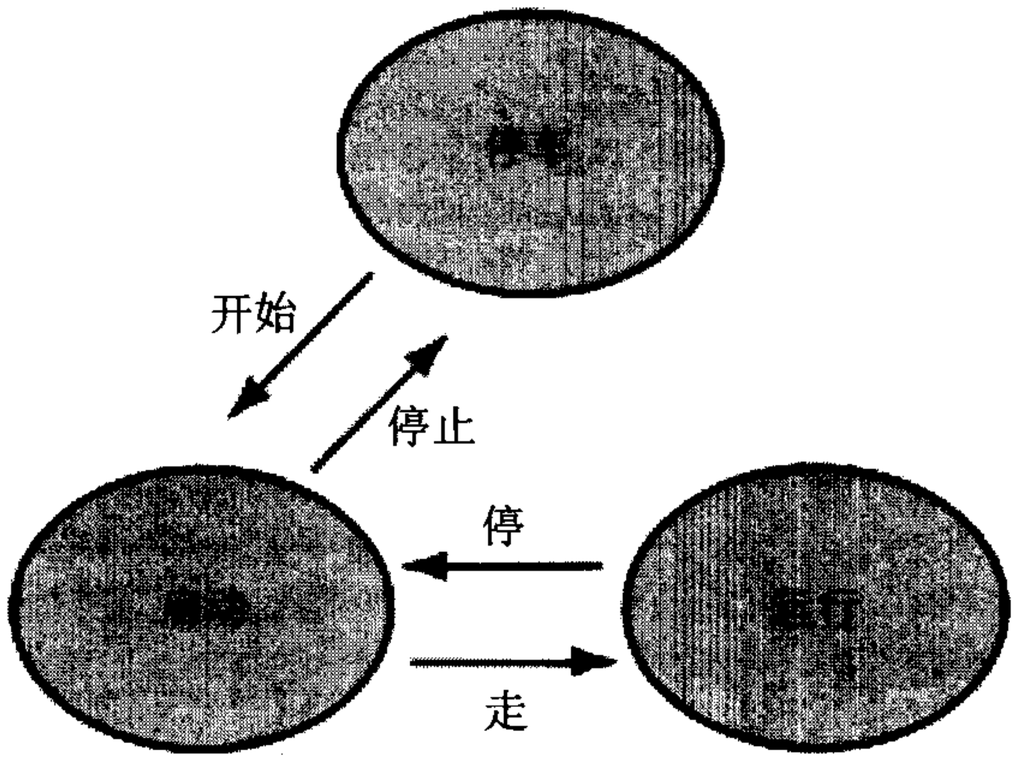

图 2.12　汽车的变换图（Lai 1991）

动态模型：系统动态

过程模型的一个有价值的特性是处理过程的能力，因此当活动发生的时候我们可以观察资源和产品发生了什么情况。换句话说，我们希望给出过程的一个模型，然后在软件向我们展示资源流如何经过活动成为输出时进行观察。这种动态过程的观点使我们能模拟过程，并在真正扩充资源之前进行修改。例如，我们可以使用动态过程模型来帮助我们判断我们需要多少测试人员，以及必须何时启动测试以按进度完成测试。类似地，我们能通过引入或排除活动来了解它们对工作量和进度的影响。例如，我们可以增加一个代码评审活动，对评审过程中发现多少错误作出假设，从而判断评审是否明显地缩短了测试时间。

建立动态过程模型有很多方法。Forrester于20世纪50年代提出了系统动态方法，该方法已经在模拟不同的过程（包括生态、经济和政治系统）中发挥了很大的作用。Abdel-Hamid和Madnick已经把系统动力学应用到了软件开发中，使项目经理能够在把过程选择强加给开发人员之前“测试出”他们的过程选择（Abdel-Hamid 1989； Abdel-Hamid和Madnick 1991）。

为了了解系统动态是如何运作的，我们来看看软件开发过程是如何影响生产率的。可以构建不同活动的描述性模型，这些活动涉及到开发人员的时间，然后观察模型中的变动怎样增加或减少了设计、编写和测试代码所用的时间。首先，我们必须判断哪些因素影响了总生产率。图2.13描述了Abdel-Hamid对这些因素的理解。箭头指出了一个因素中的变化如何影响另一个因素中的变化。例如，如果在分配的项目人员中有经验的职员所占比例从1/4增加到1/2，那么，我们将期望平均生产率会有所提高。类似地，职员所占的比例越大（反映在职员规模上），那么用于项目小组成员交流的时间就越多（交流耗费）。

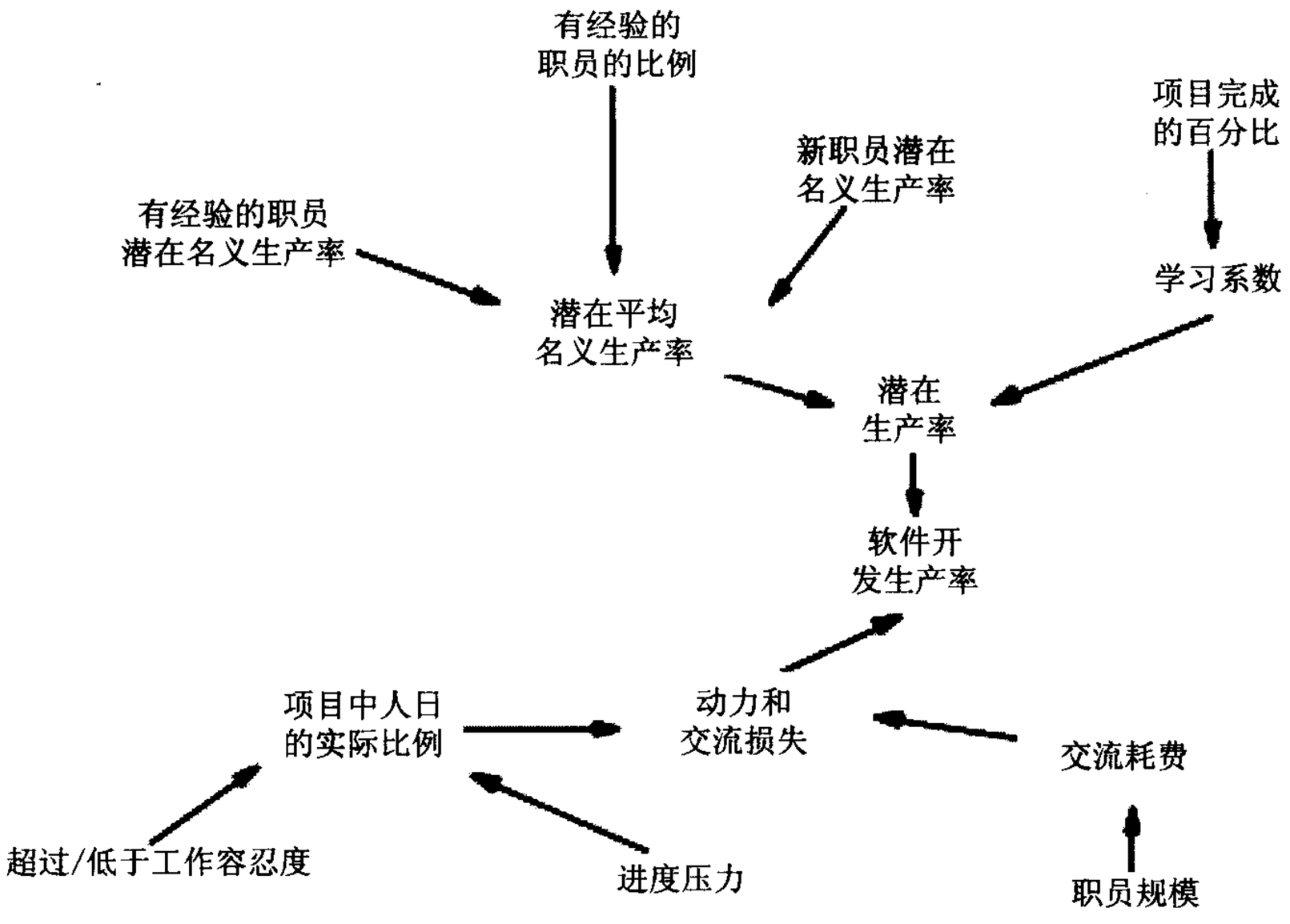

图 2.13　影响生产率的因素的模型（Abdel-Hamid 1996）

图2.13说明潜在平均名义生产率受下面三种因素影响：有经验的职员的生产率、有经验的职员的比例以及新职员的生产率。同时，新职员必须了解这个项目；项目完成的部分越多，则新职员在他们能够成为项目小组中有生产力的成员之前就必须了解得越多。

其他因素也会影响开发的总生产率。首先，必须考虑每个开发人员每天用于开发项目的时间比例。进度压力影响这个比例，开发人员对工作负担的容忍度也影响这个比例。职员规模也影响生产率，但是职员越多，则用于项目小组成员交流信息的时间就越多。交流、动力以及图2.13的上半部分所表示的潜在生产率，三者的组合给出了一种普遍的软件开发生产率关系。

因此，使用系统动态方法的第一步是基于经验证据、研究报告和直觉的组合来确定这

些关系。接着量化这些关系。量化可以包含直接的关系，比如职员规模和交流之间的关系。我们知道，如果有n个人分配到一个项目组，那么可能有$n(n-1)/2$对人员必须彼此交流和协调。对某些关系，尤其是涉及到影响整个时间资源的那些关系来说，我们必须进行一些分配来描述资源的增加和减少。例如，在一个项目中很少会出现每个人都在第一天开始工作的情况。系统分析员首先开始工作，而只有在大部分需求和设计组件文档化之后编程人员才加入项目。因此，这种分配就描述了资源的增加和减少（或者甚至是波动，比如在假期或夏天休假时的出勤率）。

一个系统动态模型是可扩展的也是复杂的。例如，Abdel-Hamid的软件开发模型包含100多个因果连接；图2.14简要地说明了他定义的关系。他定义了影响生产率的4个主要方面：软件产品、人力资源管理、计划和控制。产品包括质量保证、学习和开发速度等问题。人力资源描述了雇用、人事变动率和经验。计划关心进度安排和由此引起的压力，而控制描述了过程度量和完成项目所需的工作量。

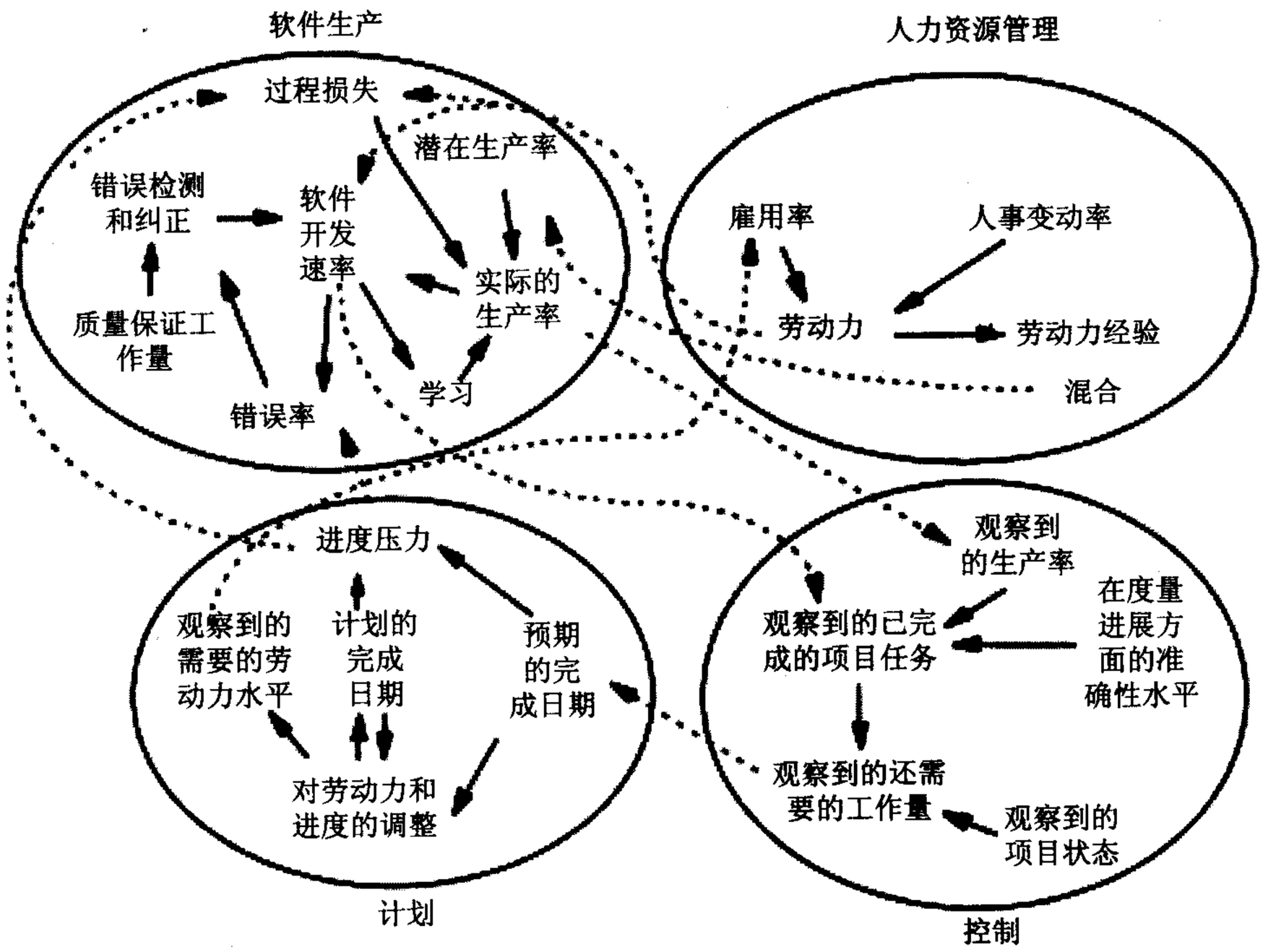

图 2.14　软件开发的结构（Abdel-Hamid 1996）

由于连接的数目可能相当大，系统动态模型由某些软件来支持，这些软件获取连接和定量描述，然后模拟全局过程或某些子过程。

系统动态模型的威力很大，但是使用这个方法必须小心谨慎。模拟的结果依赖于量化的关系，而量化的关系常常是启发式的或含糊不清的，而不是明确地基于实验的研究。然而，正如我们在后面的章节中将要看到的那样，一个关于开发各方面的管理信息的历史数据库能够加强我们对关系的理解，从而对动力学模型的结果更有信心。

附注2.3　过程编程

在20世纪80年代中期，Osterweil（1987）提出软件开发过程应当使用数学描述的方法进行说明。这就是说，如果我们充分理解了一个过程，我们就能够编写程序来描述这个过程，然后运行这个程序实现这个过程。过程编程的目标就是减少不确定性，人们不仅通过充分理解一个过程来编写软件以抓住它的本质、而且通过把这个过程转化成问题的确定的解决方案，从而实现这个目标。

如果过程编程是可能的，那么我们就能够管理所有过程活动的可见性、把所有活动自动化、轻松地协调以及改变所有活动。因此，过程编程有可能成为生产软件的自动化环境的基础。

然而，Curtis、Krasner、Shen和Iscoe（1987）指出，Osterweil对计算机编程的比喻没有抓住开发过程的内在变化。当写完一个计算机程序之后，程序员假定实现环境正确地运作；操作系统、数据库管理和硬件都是可靠的和正确的，因此在计算机对指令的响应中几乎不存在变化。但是当一个过程程序对项目小组成员发布一个指令时，在执行任务的方式上和产生的结果上存在着很大的区别。正如我们将在第3章中看到的那样，在技能、经验、工作习惯、对客户需求的理解和很多其他因素上的不同可能极大地增加变化。Curtis和他的同事建议，过程编程应当仅限于那些存在很小变化的情况。而且他们还指出，Osterweil的例子仅仅提供了关于任务顺序的信息；过程程序并没有警告管理者那些迫在眉睫的问题。“一个创造性的智力任务网的协调看起来并没有通过当前过程编程的实现而得到很大的改善，因为协调最重要的因素是保证在所有交互的代理人的头脑中，对系统如何操作有同样的模型”（Curtis等人 1987）。

2.4　实际的过程建模

很长时间以来，过程建模一直是软件工程研究的焦点。但是，它的实用性如何呢？有研究人员报告称，正确使用过程建模为理解过程和揭示不一致性提供了很大的好处。例如，Barghouti、Rosenblum、Belanger和Alliegro（1995）进行了两个个案研究，判断在大型组织中使用过程模型的可行性、实用性和限制。本节将讨论他们所做的工作以及发现。

Marvel的个案研究

在他们的研究中，研究人员都使用MSL（即Marvel说明语言）来定义过程，然后给它产生一个Marvel过程执行环境（Kaiser、Feiler和Popovich 1988；Barghouti和Kaiser 1991）。MSL使用3种主要的结构——类、规则和工具封装——来产生一个3部分的过程描述：

1. 基于规则的过程行为说明。
2. 模型信息过程的面向对象的定义。
3. 用来执行该过程的外部软件工具和Marvel之间接口的一组封装。

第一个个案研究是AT&T呼叫处理网络，该网络处理电话呼叫，而一个分离的信令网负责给这些呼叫安排路由并平衡网络负载。Marvel用于描述信号错误处理过程，该过程负责检测、服务以及解决信息网络的问题。Workcenter 1监控该网络、检测错误并把错误分配给其他两个工作中心中的一个进行处理。Workcenter 2处理需要详细分析的软件错误或人为错误，Workcenter 3处理硬件的错误。图2.15描述了这个过程。双虚线指出哪一个活动使用了椭圆所代表的工具或数据库。长方形表示任务或活动，而菱形代表判定。箭头指出控制的流向。如你所见，该图提供了一个概要，其详细程度不足以用来捕捉基本的过程元素。

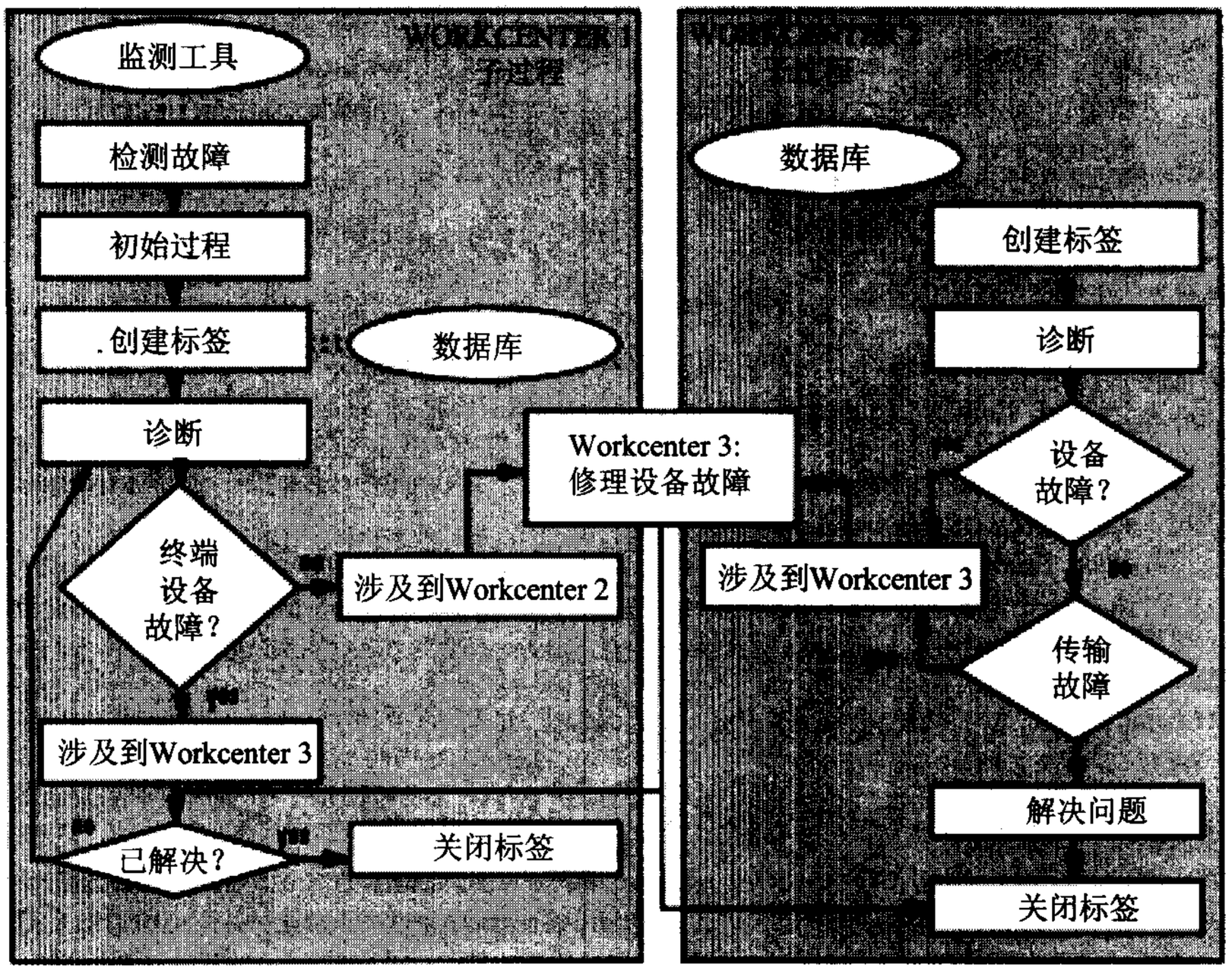

图 2.15　信号错误处理过程（Barghouti 等人 1995）

相应地，每个实体和工作中心都用MSL来建模。图2.16说明了如何进行这些工作。图的上部定义了类TICKET，这里ticket表示一旦出现错误时就记下的一个故障标签（或问题报告）。正像我们在有关测试的章节中将看到的那样，故障标签用于跟踪一个问题——从它的出现到解决。整个网络用22个这样的MSL类来表示；一个过程创建的或需要的所有信息都包含在其中。

接着，模型描述了信号错误处理过程的各个方面。图2.16的下部是一个MSL规则，它与图2.15中标有“诊断”的方框基本对应。因此，MSL描述了诊断问题的规则；它由每一个打开的标签激发。当过程模型完成之后，则需要21个MSL规则来描述这个系统。

第二个个案研究的是AT&T 5ESS交换软件的部分软件维护过程。第一个个案研究中目标是过程的改善，而第二个个案研究的目标只是通过在MSL中获取过程步骤和交互而把它们文档化。该模型包含有25个类和26个规则。

```
TICKET:: superclass ENTITY
 status : (initial, open, referred_out, referral_done,
                    closed, fixed) = initial;
 diagnostics     : (terminal, non_terminal, none) = none;
 level           : integer;
 description     : text;
 referred_to     : link WORKCENTER;
 referrals       : set_of link TICKET;
 process         : link PROC_INST;
end

diagnose [?t: TICKET]:
 (exists PROC_INST ?p suchthat (linkto [?t.process ?p]))
 :
(and (?t.status = open)(?t.diagnostics = none))
{TICKET_UTIL diagnose ?t.Name}
(and (?t.diagnostics = terminal)
        (?p.last_task = diagnose)
        (?p.next_task = refer_to_WC3));
(and (?t.diagnostics = non_terminal)
        (?p.last_task = diagnose)
        (?p.next_task = refer_to_WC2));
```

故障标签的类定义

诊断标签的规则

图 2.16　Marvel 命令的例子（Barghouti 等人 1995）

对每一个模型，用MSL的过程描述来生成“过程执行环境”，它产生一个数据库，数据库中是信息模型类的实例。然后，研究人员模拟若干场景以验证模型的实现是否像期望的那样。在模拟的过程中，他们收集计时和资源利用的数据，为分析过程的性能提供基础。通过改变这些规则并重复执行一个场景，对计时进行比较和对照，从而在不进行大的资源投资的情况下产生明显的过程改善。

建模和仿真实验对早期的问题确定和解决是有帮助的。例如，软件维护过程定义揭示了当前过程文档中的三种问题：没有任务输入和输出、模棱两可的输入输出标准，以及无效的过程定义。信号错误模型的仿真揭示了各工作中心单独描述中的低效率。

Barghouti和他的同事指出了把过程建模问题划分成两个部分（对信息建模和对行为建模）的重要性。通过这种区分，产生的模型清楚而简洁。他们还指出，计算机密集的活动比人员密集的活动更容易建模，而Curtis和他的同事也注意到了这一经验。

过程建模工具和技术中有价值的属性

有很多过程建模的工具和技术，研究人员正不断地努力工作，以判断对于给定条件哪些工具和技术是最合适的。但是，无论是哪一种技术，都存在一些有价值的特征。Curtis、Kellner和Over（1992）确定了5类有价值的属性。

1. 有利于人们的理解和交流。技术应该使用大多数顾客和开发人员能理解的方式表示过程，鼓励过程的交流并对其形式和改善达成一致。技术应该包含足够的信息，允许一名或多名人员实际执行该过程。而模型或工具应当成为培训的基础。
2. 支持过程改进。技术应当确定一个开发或维护过程的基本组件。它应当允许在后面的项目中复用过程或子过程、比较备选方案，并且在实际实现过程之前估计变化的影响。类似地，技术应该有助于为过程选择工具和技术、有助于鼓励结构化的学习、

有助于支持过程的持续演化。

3. 支持过程管理。技术应该允许过程与项目有关。这样，开发人员和顾客应该能够推测软件创建或演化的属性。技术还应该支持计划和预测、监控和管理过程，以及度量关键的过程特性。
4. 在实现过程时提供自动化的指导。技术应该定义所有的或部分的软件开发环境、提供指导和建议，并保留可复用的过程表示供以后使用。
5. 支持自动化的过程执行。技术应该自动执行全部的或者是部分过程、支持协同工作、获取相关的测量数据，以及加强规则以保证过程的完整性。

当为开发项目选择过程建模技术时，这些特性可以作为有用的指导。如果你的机构试图将其过程标准化，那么其中特别重要的是第4个特征；工具能够提示开发人员下一步做什么，并提供入口和检查点以确保在采取下一步骤之前工艺满足了一定的标准。例如，工具能够检查一组代码组件，评估它的规模和结构。如果规模或结构超出了预设的限制，那么，必须在测试开始之前通知开发人员，而某些组件可能被重新检查，也可能要重新设计。

2.5 信息系统的例子

让我们考虑用哪一种开发过程来支持以下的信息系统，即Piccadilly电视广告程序的例子。考虑到该程序对何时能够销售何种类型的广告有许多约束，而规则可能随着广告标准局及其他规则团体的规章而变化。因此，我们希望建立一个容易维护和改变的软件系统。当建立系统的时候甚至约束也可能发生变化。

瀑布模型对于我们的系统来说可能太严格了，因为在需求分析阶段完成之后它几乎不允许变化。原型化方法对开发用户界面来说可能是有用的，因此，在我们的模型中可能想引入某种类型的原型。但是大部分的不确定性在于广告规则和商业约束。我们希望使用这样的过程模型：当系统演化的时候，这个模型仍可以使用和复用。对建立Piccadilly系统来说，螺旋模型的变种可能是一个很好的选择，因为它鼓励我们重新审视我们的假设、分析我们的风险以及原型化各种系统特性。如螺旋模型的左上1/4部分所显示的那样，对备选方案的反复评价有助于我们把灵活性融入到我们的需求和设计之中。

Boehm的螺旋表示是高层次的，它没有足够的细节来指导分析人员、设计人员、编码人员和测试人员的活动。不过，有很多技术和工具可以在很精细的层次上表示过程模型。技术和工具的选择部分地依赖于个人的偏好和经验，部分依赖于是否适合所表示过程的类型。让我们来了解一下如何使用Lai符号来表示部分Piccadilly系统的开发过程。

由于希望使用螺旋模型来帮助我们管理风险，我们必须在过程模型中引入“风险”的特征。这就是说，风险是我们必须描述的工件，从而我们能够在螺旋的每一次迭代中测量和跟踪风险。每一个可能的问题都具有相关的风险，可以从两方面考虑风险，即概率和严重性。概率就是某个特定问题出现的可能性。而严重性就是它将要给系统带来的影响。例如，对于建立Piccadilly系统所使用的开发方法，假设我们正在考虑Piccadilly系统开发方法的培训不充分问题。我们可能决定使用面向对象的方法，但是我们会发现，分配到项目组

的开发人员有很少的面向对象的经验或者甚至没有。这个问题发生的概率可能很小，因为所有的新雇员都会被送去参加4周的面向对象开发的强化课程。另一方面，如果这样的问题真的发生了，那么它将严重地影响开发小组在指定的时间内完成该软件。因此，这个问题发生的概率很低，但是它的严重性很大。

我们可以用Lai工件表来表示这些风险的情况，如表2.2所示。这里，风险是工件，它具有子工件概率和严重性。为简单起见，我们为每一个子工件选择两个状态：概率的低和高，严重性的小和大。事实上，每一个子产品都具有很多的状态（像非常小、很小、有些小、中等、有些高、很高、非常高等），导致工件本身产生许多不同的状态。

表2.2　工件“风险”的定义表格

名称	Risk(ProblemX)	
过程提要	这是表示问题X会发生的风险的工件，它对开发过程的某些方面有负面影响	
复杂性类型	合成	
数据类型	(risk_s,user_defined)	
工作状态列表		
low	((state_of(probability.x) = low) (state_of(severity.x) = small))	问题的概率低，问题的严重性影响小
high-medium	((state_of(probability.x) = low) (state_of(severity.x) = large))	问题的概率低，问题的严重性影响大
low-medium	((state_of(probability.x) = high) (state_of(severity.x) = small))	问题的概率高，问题的严重性影响小
high	((state_of(probability.x) = high) (state_of(severity.x) = large))	问题的概率高，问题的严重性影响大
子工件列表		
	probability.x	问题X发生的可能性
	severity.x	问题X对工程的影响的严重性

用同样的方法，我们可以定义开发过程的其他方面，并使用图表说明活动和它们之间的相互关系。用这种方法对过程建模有很多优点，而且它建立了对开发需要什么的共同理解。如果用户和开发人员参与定义和描述Piccadilly的开发过程，那么，每一个人都期望了解开发过程涉及到什么活动、它们产生什么、以及何时能够得到每个工件。特别地，可以使用螺旋模型和风险表格的组合周期性地评价风险。围绕螺旋模型的每一次旋转，要重新评估和表述每一个风险的概率和严重性；当风险高得不可接受时，可以修正过程模型以引入风险缓和及风险降低技术，我们将在第3章中了解这些内容。

2.6　实时系统的例子

Ariane-5软件涉及到Ariane-4的软件的复用。复用的目的是为了减少风险、提高生产率和质量。这样，开发新的Ariane软件的任何过程模型都应当包含复用活动。特别地，过程模型必须包含检查可复用组件的质量的活动，以确保复用的软件在新系统设计的环境下正

常运作。

这样一个过程模型看起来可能像图2.17的简化模型。模型中的方框表示活动。从左边进入方框的箭头是资源，在右边离开方框的箭头是输出。而从顶部进入的箭头是控制或约束，比如进度、预算或标准。那些从下部进入的箭头是有助于实现活动的机制，比如工具、数据库或技术。

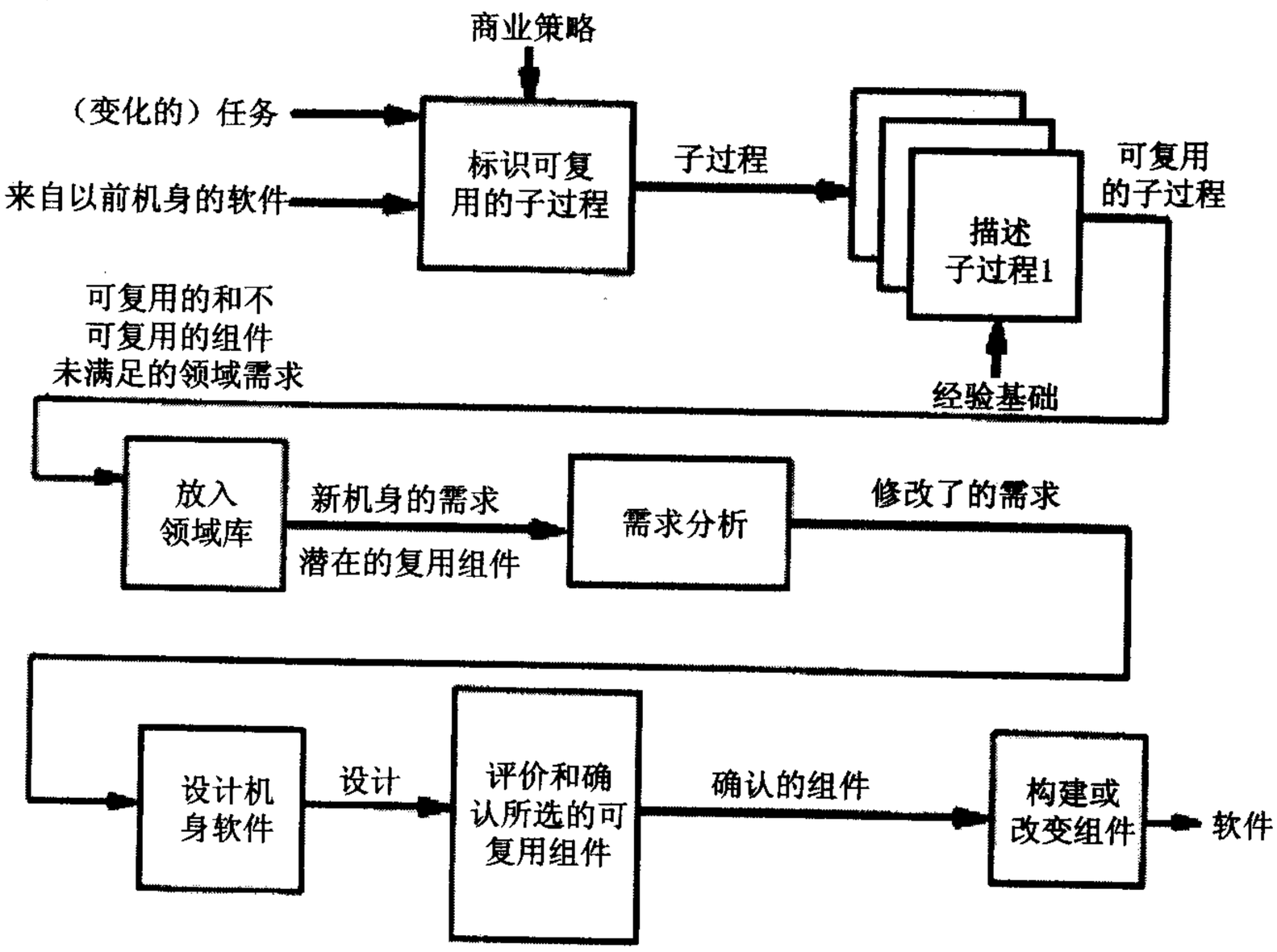

图 2.17　新机身软件的复用过程模型

Ariane-4复用的过程开始于软件任务，即控制一个新的火箭，以及来自以前的机身的软件、没有达到的需求，以及可用的其他资源（比如购买的软件或其他项目的复用库）提供的软件组件。基于航空建造者的商业策略，开发人员可以标识可复用的子过程，描述它们（可能与过去的经验相联系的概念）并把它们放在库中为需求分析人员所用。可复用的过程常常包括可复用的组件（即可复用的需求、设计或代码组件，甚至是测试实例、过程描述以及其他的文档和工件）。

接着，需求分析人员检查新机身的需求以及可以从库中得到的复用部件。他们产生一个修改了的需求，其中包括新的和可复用的需求。一旦完成设计之后，就评估所有的复用设计组件，以确认它们是正确的，并且与新设计的部分和在需求中描述的系统的全局目标是一致的。最后，使用这些确认的组件来建立或改动软件，从而生成最终的系统。正如我们将在后面的章节中看到的那样，这样一个过程原本是有可能防止Ariane-5坠毁的。

2.7　本章对你意味着什么

本章中，我们看到软件开发过程涉及到活动、资源和产品。当你与一个小组一起工作

的时候，过程模型对指导你的行为是有益的。当你设计和建立新系统的时候，详细的过程模型告诉你如何与你的同事协调和分工合作。我们也看到过程模型包括组织的、功能的、行为的和其他的方面，从而使你能够集中于开发过程的特定方面以增强你的理解或指导你的行动。

2.8　本章对开发小组意味着什么

对开发小组来说，过程模型也具有明显的优势。一个良好的模型向每一个小组成员表明发生什么活动、何时发生以及由谁进行该活动，从而明确责任分工。另外，为了满足项目的预算和进度，项目管理者可以使用过程工具来执行过程、模拟活动以及跟踪资源，以决定最佳的人员或活动配置。这种仿真在资源实际提交之前完成，由于无需反馈或纠正错误，从而节省了时间和费用。事实上，可以在过程模型中引入迭代和增量开发，这样小组可以从原型中学习或对变化的需求做出反应，从而仍能达到合适的期限。

2.9　本章对研究人员意味着什么

过程建模是软件工程中人们比较有研究兴趣的领域。许多软件开发人员感到，通过使用好的过程，开发产品的质量可以得到保证。这样，研究人员有许多领域可以探讨：

- 过程表示　如何用能够被开发人员理解的方式记录过程。
- 过程模型　如何使用一组合适的活动、资源、产品和工具来描述过程。
- 过程建模支持工具　如何制定或模拟一个过程模型，从而可以评估资源的利用率、使用率和性能。
- 过程度量和评估　在特定的时间和环境下，如何判断哪些活动、资源、子过程和模型类型对于生产高质量产品是最好的。

许多工作需要与过程改进的研究相互配合，我们将在13章中讨论过程改进。

2.10　学期项目

在FCO的Loan Arranger系统的开发过程中，现在处于早期阶段，还没有得到该系统的一组全面需求。现在所有的只是系统功能的概要，以及如何使用该系统来支持FCO业务的概念。你还不熟悉概要中使用的许多术语，因此，可以要求客户代表准备一份词汇表。这些术语的描述见表2.3。

表2.3 Loan Arranger的术语表

借方：借方就是来自贷方的钱的接受者。借方可以共同接受贷款；也就是说，每一份贷款可能有多个借方。每一个借方有一个相关的名字和一个惟一的借方标识码

借方风险：与任何借方有关的风险因素都是基于借方的偿还历史的。没有任何贷款的借方被分配给一个50的借方风险因子。当借方按时偿还贷款时风险因子减少，而当借方推迟偿还或不偿还时则风险因子增加。借方的风险用下面的公式来计算：

风险=50－(10×贷款信誉良好的年数)＋(20×推迟贷款的年数)＋(30×不偿还贷款的年数)

例如，借方有三次贷款。第一次贷款发生在两年以前，所有偿还按时完成。那么，那次贷款就是信誉良好的，持续时间为两年。第二次和第三次贷款已经分别有四年和五年的历史，直到目前为止每一次贷款都信誉良好，这两笔贷款都仅推迟一年时间。因此，风险是

$$50-(10\times 2)+[20\times(1+1)]+(30\times 0)=70$$

最大的风险值是100，最小的风险值是1

群组：群组就是作为单一的单元销售给投资者的贷款集合。与每一个群组相关的是在群组中的贷款总量、群组中的贷款有效期（即借方仍然偿还的那些贷款）、与购买该群组有关的风险估计，以及借方偿还了所有贷款时的利润

群组风险：贷款群组的风险是根据群组中贷款的风险加权平均值来衡量的，每一个贷款的风险都根据其贷款额来衡量。要计算n个贷款的加权平均值，假设每个贷款Li有剩余的本金Pi和贷款风险Ri。那么加权平均值就是

$$\frac{\sum_{i=1}^{n} PiRi}{\sum_{i=1}^{n} Pi}$$

贴现：贴现就是FCO愿意把贷款贷给投资者的价格。它根据下面的公式来计算：

贴现＝剩余的本金×[利率×(0.2＋(0.005×(101－贷款风险)))]

利率类型：贷款中的利率是固定的或可调整的。固定利率的贷款（称为FM）在抵押期间具有相同的利率。可调整利率的贷款（称为ARM）基于美国财政部提供的财政指数，每年变化利率

投资者：投资者就是有兴趣从FCO那里购买群组贷款的个人或组织

投资请求：投资者提交投资请求，指定在即将进行的投资上的最大的风险程度、在群组中所要求的最小的利润，以及必须偿还群组中贷款的最长期限

贷方：贷方是向借方提供贷款的机构。一个贷方可以没有、有一个或多个贷款

贷方信息：贷方信息是从外部获得的描述性数据。贷方信息不能被改变或删除。下面的信息与每一个贷方相关：贷方名称（机构）、贷方联系人（在该机构中的人员）、联系人的电话号码、以及惟一的贷方标识码。一旦加入到系统，则贷方的记录可以被编辑，但不能被删除

贷方机构：贷方机构是贷方的同义词。详见贷方

（续表）

贷款：贷款是描述与贷款相关的标识信息的信息集合。下面的信息与每次贷款有关：贷款额、利率、利率类型（可调整的或固定的）、结算日期（即借方最初从贷方获得贷款的日期）、期限（用年数来表示）、借方、贷方、贷款类型（巨额的或常规的）以及财产（用财产的地点标识）。贷款必须只与一个贷方相关并且只与一个借方相关。另外，每一次贷款由贷款风险和贷款状态标识

贷款分析员：贷款分析员是FCO的职业雇员，被培训使用Loan Arranger系统来管理贷款或把贷款分成群组。贷款分析员对贷款或借款的术语非常熟悉，但是他们手边并没有相关信息来评估一次贷款或一组贷款

贷款风险：每一笔贷款与一定水平的贷款风险相关，风险用从1到100的整数来表示。1代表最低风险的贷款，也就是说借方不可能推迟或不还贷款。100表示最高的风险，也就是说确信借方不还这个贷款

贷款状态：贷款可以具有下面三种状态：良好、推迟或不还。如果借方到目前为止偿还了所有贷款，则其贷款就是良好的状态。如果借方偿还了最后一笔贷款，但不是在规定的期限完成的，那么其贷款处于推迟的状态。如果借方的最后一笔偿还在超过期限10天之后还没有收到，则贷款是不还的状态

贷款类型：贷款可能是巨额的抵押贷款，其资产的价值超过275 000美元；或常规的抵押贷款，其资产值是等于或少于275 000美元

有价证券：即由FCO购买的可用在群组中的贷款的集合。由Loan Arranger维护的库包含了所有有价证券的有关信息

以上信息澄清了一些概念，但是仍不是一组好的需求。不过，已经能够对开发如何进行作出一些初步的决定。回顾在本章中所给出的过程，决定哪种过程适于开发Loan Arranger系统，列出每一个过程对于Loan Arranger系统的优点和缺点。

2.11　主要参考文献

作为第五届国际软件过程讨论会的结果，由Kellner主持的一个工作小组系统地阐述了标准问题，用以评估和比较一些更流行的过程建模技术。他们把标准问题设计得足够复杂，使之能够测试如下几个方面的技术能力：

- 抽象的多个层次
- 控制流、顺序以及对顺序的约束
- 判定点
- 迭代和对早期步骤的反馈
- 用户的创造性
- 对象和信息管理，以及贯穿过程的流
- 对象的结构、属性和相互关系
- 特定任务的组织责任
- 信息传播的物理通信机制
- 过程度量