返回总目录

目　　录

第 9 章　使用 UML 的过程 2

9.1　定义和理解软件工程的过程概念 2
9.2　评价软件过程成熟度的标准：CMM 3
9.3　RATIONAL 的统一过程和软件开发的六大经验 6
9.4　过程的两维空间 8
9.5　时间维：阶段和迭代 8
9.6　过程的静态结构 13
9.7　核心工作流程 15
9.8　如何在过程中使用 UML 19
9.9　小　　结 22

第 9 章　使用 UML 的过程

UML 的表示和规则能够用来为系统进行面向对象的建模，但并没有指定应用 UML 的过程和方法，因为它的设计初衷是为了能在尽可能多的领域内得到广泛的应用。尽管如此，要想成功地使用 UML，科学的过程还是必要的，尤其在设计一些需要团队合作的大型系统时。此时必须协调所有人的工作，确保大家向同一方向努力。合理的过程能够有效地测度工作进程，控制和改进工作效率。尤其是软件工程领域的过程更需要加强对可重用性的支持，包括过程本身及其部分（模型、组件、框架等）的重用。

本章首先从一般意义上讨论过程的定义、过程的评估模型，然后具体介绍一种使用 UML 的过程：统一过程。因为该过程目前是由 UML 提出人在开发的，所以与 UML 能够最好地结合。最后，我们讨论使用 UML 的过程的一般特征。

9.1　定义和理解软件工程的过程概念

为软件工程定义一个过程并不容易，需要对软件开发的机制和方法有深入的了解。简单地说，过程描述做什么、怎么做、什么时候做以及为什么要做，描述了一组以某种顺序完成的活动。过程的结果是一组有关系统的文档（模型和其它一些描述），以及对最初问题的解决方案。因为建模语言需要工具的支持，所以过程也需要工具的支持。不过，目前支持过程的工具不如支持建模的工具那么广泛。目前市场上一些面向对象方法如 OMT、Objectory、Booch 和 Fusion 等都被看成是软件开发的过程。

过程描述的一个重要部分是定义如何使用人力、机器、工具和信息等资源的一些规则来完成某个确定的目标，为用户的问题提供解决方案。过程通常被划分成一些嵌套的子过程，在最底层，过程是不可分的原子成份。

过程是软件工程的一种结构化的工作和思考方法，可以从以下几个方面理解过程这一概念：

- 过程的情景：描述使用过程的问题领域。过程必须给出它的使用环境，应用它的问题领域。有时人们并不愿意（甚至不可能）开发或选择一个能够处理所有可能问题的过于通用的过程，最重要的是，过程能够恰当地解决特定问题领域内的某个特定问题。
- 过程的用户：由用户确定如何应用过程。软件工程的过程必须有它的使用指南，指南不仅涉及过程本身，还要涉及潜在的问题解决人员——使用过程的人员。
- 过程的步骤：确定在过程中要采取的步骤，大多数软件开发的过程至少包括三大内容：问题描述、方案设计和实现设计。问题描述发现和描述问题；方案设计给出问题的解决方案，而实现设计面向对象系统设计的软件工程的步骤包括分析、设计和实现。
- 过程的评估：如何评估结果（文档、产品、经验等）。大多数软件开发的过程至少包括三大内容：问题描述、方案设计和实现设计。问题描述用于发现和说明问

题；方案设计给出问题的解决方案，而实现设计面向对象系统设计的软件工程的步骤包括，分析、设计和实现。

9.2　评价软件过程成熟度的标准：CMM

与在开始一个项目时要首先定义它的需求一样，在开始定义软件过程之前也要定义对它的需求。这就是 CMM（软件成熟度建模，Capability Maturity Modeling）所完成的任务。CMM 是由美国国防部资助、美国卡耐基 • 梅隆大学的软件工程研究所提出的，它是大型复杂软件开发过程的框架，CMM 定义了软件过程的五个成熟度等级：初始级，可重复级，已定义级，已管理级和优化级，如图 9-1 所示，表 9-1 是各等级的具体描述和特征。CMM 是一个由低到高逐渐成熟的演进框架，是衡量软件机构过程成熟度的尺度，它的目标是引导软件机构进行软件过程的持续改进。成熟度高的等级有着较高的生产率、较高的产品质量和较低的风险。根据 CMM，一个机构要想达到某个成熟度的过程，必须满足表 9-2 所示的关键过程域（Key Process Areas，KPA）并使之制度化。CMM 现在已被世界上许多机构和组织采纳，来评估和提高他们软件开发过程的质量。

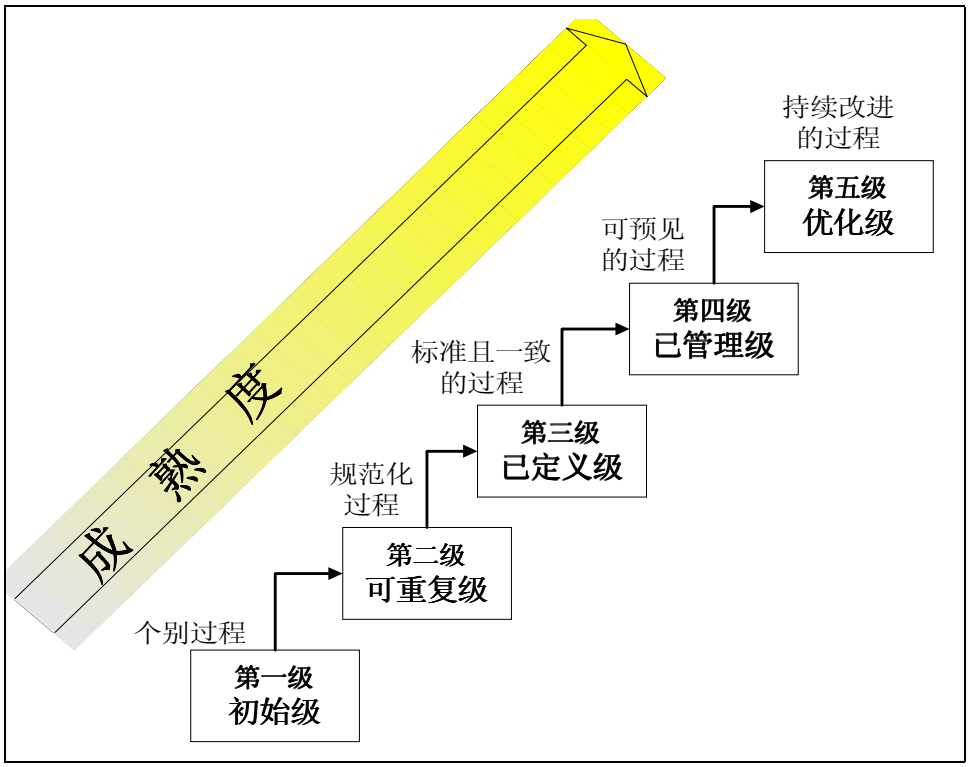

图 9-1　CMM 的五个成熟度等级

表 9-1　CMM 的五个成熟度等级

	等级	描述	特征
1	初始级（Initial）	软件过程是个别的、有时甚至是混乱的，很少有过程是有定义的。成功依赖于个人的努力和个别人的管理能力	● 机构缺乏明文管理办法，软件工作没有稳定的环境，制订了计划又不执行 ● 紧急情况下将已制定的规程丢在一边，急于编码和测试。规定的过程无法克服由于缺乏有效管理带来的不稳定性 ● 个别管理人员能顶住削减过程的压力，但他们离职后，则全然不同 ● 成功依赖于某个有经验的管理人员 ● 软件过程对用户来说是一个黑箱
2	可重复级（Repeatable）	建立了为跟踪成本、进度和功能的基本项目管理过程。基于以往项目的经验，制定了过程实施规范，使类似的项目再次成功	● 新项目的规划和管理是根据以往类似项目的成功经验 ● 通过建立基本的过程管理技术，在项目级提高了项目的能力 ● 软件的需求和可提交的产品都有基线的控制 ● 不同的项目有不同的过程，减少了团队合作和重用的机会 ● 在某些场合向用户提供项目的可见性，通常是对主要项目产品的评审和接受上，允许有限的管理控制
3	已定义级（Defined）	制订管理和开发活动的软件过程的规程，把它们标准化后集成为机构的标准软件过程。针对特定项目，可对标准过程进行剪裁	● 使用标准过程，不同的项目进行不同的剪裁 ● 管理者了解项目的技术进展 ● 定义的过程使用户对项目有更多的可见性，能够准确而快速了解项目最新状态 ● 建立了机构的软件工种过程组（SEPG），负责软件过程活动
4	已管理级（Managed）	对项目的过程活动，包括生产率和质量均制订了度量标准，对软件过程和产品进行量化掌握和管理	● 在所有项目中，对重要的软件过程活动的生产率和质量进行度量 ● 用户在项目开始之前，就对机构 / 团队的软件过程能力以及项目的风险建立准确和量化的了解 ● 对新应用领域的风险是可预知和控制的，可预知产品的质量
5	优化级（Optimized）	来自软件过程以及新的思想和技术的量化反馈持续不断地改进过程	● 集中注意于过程的持续改进 ● 重视探索创新活动，并将成功的创新推广 ● 通过不断查找导致低效的原因，持续地改进软件过程 ● 用户和软件开发人员相互合作，建立有力而成功的关系

表 9-2　CMM 的关键过程域

关键过程域	目标	等级
需求管理（Requirements Management.）	依据项目的需求，建立和维护与用户之间的共识，并编制文档记录下来	2
软件项目规划（Software Project Planning）	对要完成的工作进行估计和协商，制定工作计划和必要的承诺，并建立文档记录下来，便于跟踪和控制	2
软件项目跟踪和监督（Software Project Tracking And Oversight）	提供对项目实际进展的可见性，将计划实际取得的成果和计划实施情况与计划对照跟踪，确保管理层采取有效和恰当的行动	2
软件子合同管理（Software Subcontract Management）	对符合资格的软件子合同进行选择和有效的管理，跟踪子合同的实际工作进展	2
软件质量保证（Software Quality Assurance）	对软件质量保证活动制订计划，并通知相关人员，对项目的活动和提交的产品进行评审和审计，确认符合所采用的标准、原则和过程	2
软件配置管理（Software Configuration Management）	制订软件配置管理活动的计划，对项目提交的产品的变更加以控制，确保在整个生命周期中，项目所提交产品的集成性	2
机构过程焦点（Organization Process Focus）	开发和维护对机构和项目的软件过程的掌握，协调评估、开发、维护和改进这些过程的活动。这个任务应该由机构内一个永久性的小组来完成	3
机构过程定义（Organization Process Definition）	开发和维护机构的标准过程、以及相关活动，如描述软件生命周期、过程剪裁指南和准则、机构的软件过程数据库和与软件过程有关的文档库	3
培训大纲（Training Program）	确定机构、项目和个人所需要的技能和知识培训，确定目标，然后开发并进行相关的训练达到目标	3
集成化软件管理（Integrated Software Management）	把软件工程和管理活动集成到一个对每个项目都是一致的、经过剪裁的过程中，这个过程是机构的标准软件过程的剪裁版。包括开发项目定义的软件过程，并根据这个过程对项目进行管理	3
软件产品工程（Software Product Engineering）	根据项目定义的软件过程、以及恰当的方法和工具，维护软件产品的一致性	3
组间协调（Intergroup Coordination）	所有相关组都接受顾客的需求，并对组间问题进行标识、追踪和解决，协调组间的合作	3
同行评审（Peer Review）	由开发人员的同行对项目提交的产品进行检查，找出潜在的缺陷以及需要修改的领域	3
定量过程管理（Quantitative Process Management）	为项目的软件过程制订目标，确保实施情况得到量化控制，把机构的标准软件过程定量地表达出来	4
软件质量管理（Software Quality Management）	为软件产品定义质量目标文件，并制订达到这些目标的计划，监督和调整软件计划、产品、活动以及质量目标，确保提交高质量的产品来满足用户需求	4
缺陷分析（Defect Prevention）	分析过去出现的缺陷，找到并消除引起缺陷的原因，防止在将来再次出现	5
技术变更管理（Technology Change Management）	为技术变更制订文档，对新技术进行评估以确定其对产品质量和生产率的影响，并把适合的新技术引入机构的正常活动中	5

过程变更管理 （Process Change Management）	为持续的过程改进的目标制订文档，对项目定义的软件过程进行量化的度量。在整个机构范围内组织软件过程活动，并持续地改进机构的标准软件过程和项目定义的软件过程	5

至少从管理的角度来说，CMM 的强大在于它把成熟和不成熟的软件机构区分开来。不成熟的软件机构通常是被动的，对如何成功开发软件没有任何概念。而成熟的软件机构则理解软件过程，能够判断软件过程以及它们所提交产品的的质量。成熟的软件机构与不成熟的软件机构相比，具有较高的成功率，并且在整个软件开发的生命周期整体成本较低。

为什么一个机构要尽量提高它的软件过程的成熟度呢？因为首先，机构软件过程越成熟，它就更注重管理和项目提交的产品，结果是它生产的产品质量就越好；其次，当机构软件过程成熟度的提高后，它将提高管理控制和对软件过程和产品进行量化管理，结果是所开发项目的风险大幅降低。

为了更好地理解软件过程，图 9-2 描绘了不同的过程方法的范围，显示了开发过程是软件过程的子集，而软件过程又是企业过程的一个子集。图中还表明，影响过程的因素包括机构的文化、架构、所使用的工具、所遵循的标准、公司的立法以及与企业外部交互的过程。

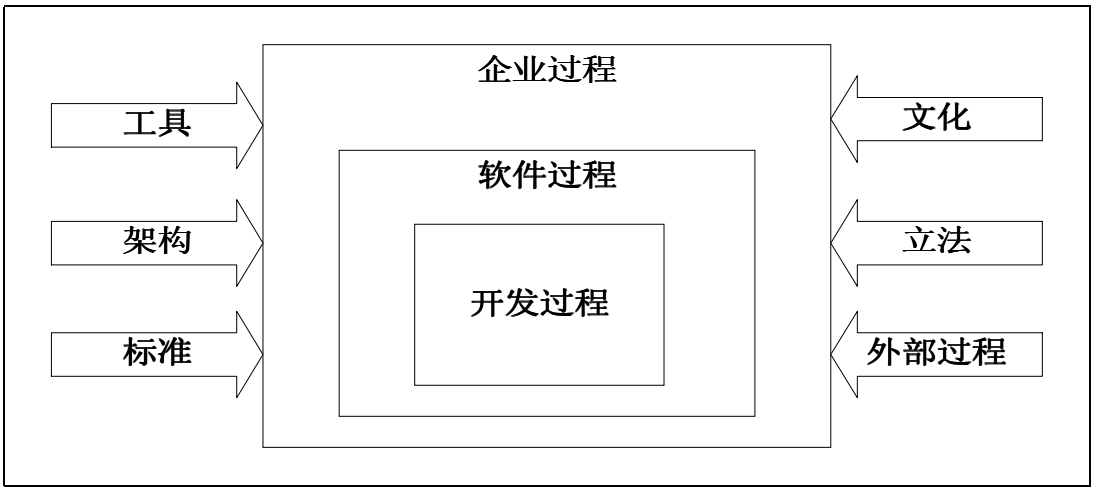

图 9-2　过程的范围

9.3　Rational 的统一过程和软件开发的六大经验

目前比较流行的有几种重要的过程，包括 Rational 的统一过程、OPEN 过程和面向对象软件过程（OOSP）。本书主要对统一过程进行描述，因为它是由提出 UML 的三位方法学家 Booch，Jabson 以 Rational 的 Objectory 为核心提出，在这个过程中使用 UML 是非常自然的。

目前软件开发实践中得到的六大最好经验是：

1．迭代地开发软件

2．管理需求

3．使用基于组件的架构

4．为软件可视化建模

5．验证软件质量

6．控制对软件的变更

Rational 的统一过程的核心是为软件开发团队提供指南、文档模板和工具，以使整个团队能够最有效地利用这六大最好的软件开发经验：

1．迭代地开发软件。当今软件系统十分复杂，很难按照定义整个问题、设计整个系统、建立软件、最后测试产品的顺序线性进行。需要一种迭代的方法，允许通过不断地细化来提高对问题的理解，在多个迭代的基础上递增地得到一个有效的解决方案。 Rational 的统一过程支持迭代地开发，在生命周期的每个阶段都强调风险最高的问题，显著地降低项目的风险系数。这种迭代的方法通过可见的进展情况、可执行的版本来促进端用户的参与和反馈，从而有助于降低开发过程中的风险。而且，因为每个迭代结束时都提交一个可执行的系统版本，使开发团队能够始终把注意力放在产品上，经常性的阶段检查也确保项目按计划进行。最后，迭代方法还很容易容纳需求、特色或规划上的改变。

2．管理需求。Rational 的统一过程描述了如何启发、组织所需要的功能和约束，以及如何为它们建档；如何跟踪和建档权衡和决策；并且易于表达商业需求和交流。过程所规定的用例和想定已经被证明是表达功能性需求以及保证它们驱动软件的设计、实现和测试，使最终系统更充分地满足用户需要的最佳方法。它们提供了贯穿整个开发和所提交系统的一致而可跟踪的线索。

3．使用基于组件的架构。这个过程侧重在利用资源进行规模开发之前，对架构早期的开发和基线起作用。它描述了如何设计一个能适应变化、直观、有利于系统重用的灵活的架构。Rational 的统一过程支持基于组件的软件开发。组件是完成一个明确功能的非平凡的模块、子系统。Rational 的统一过程提供了一个系统的方法用新的和已有组件定义架构。它们被组装在一个良定的架构中，或者是某个特别的基础结构，或在一个组件的基础结构中，例如 Internet，CORBA 和 COM。

4．为软件建立可视化模型。此过程告诉我们如何可视化地为软件建模来表达架构和组件的结构和行为。这样，就可以把细节隐藏起来，使用图形化的基本模块来写代码。使用抽象可视化有助于进行不同方面的沟通，显示出系统元素是如何组织在一起的；保证各部件与代码一致；维护设计和实现的一致性；促进无二义性的交流。工业标准 统一建模语言（UML）是成功地进行可视化建模的基础。

5．验证软件质量。软件性能和可靠性的低下是影响软件使用的最重要的因素。因此应该根据基于可靠性、功能、应用性能和系统性能的需求对软件的质量进行评估。统一过程将帮助你进行这些类型的规划、设计、实现、执行和评估。质量评估嵌入在整个过程的每个活动中，让所有的人员都参与，使用目标试题和准则，而不再只是事后工作，也不由单独一群人单独完成。

6．控制对软件的变更。对变更进行管理的能力确保每个变化都是可接受的，而在变更是不可避免的环境中，跟踪变更的能力是最根本的。统一过程描述了如何控制、跟踪和监视变更，从而保证迭代开发过程的成功。它也指导人们如何通过控制所有对软件制品（如模型、代码、文档等）的变更，从而隔离来自其它工作空间的变更，以此为每个开发人员建立安全的工作空间。它还描述了如何自动化集成和建立管理，从而使一个团队的工作团

结一致。

9.4　过程的两维空间

统一过程是在两维空间中描述（如图 9-3 所示），或者说过程是沿着两个轴发展：

- 水平轴代表时间，显示了过程动态的一面，是用周期、阶段、迭代、里程碑等名词来描述的；
- 垂直轴代表过程静态的一面：是用活动，产品工人和工作流程描述的。

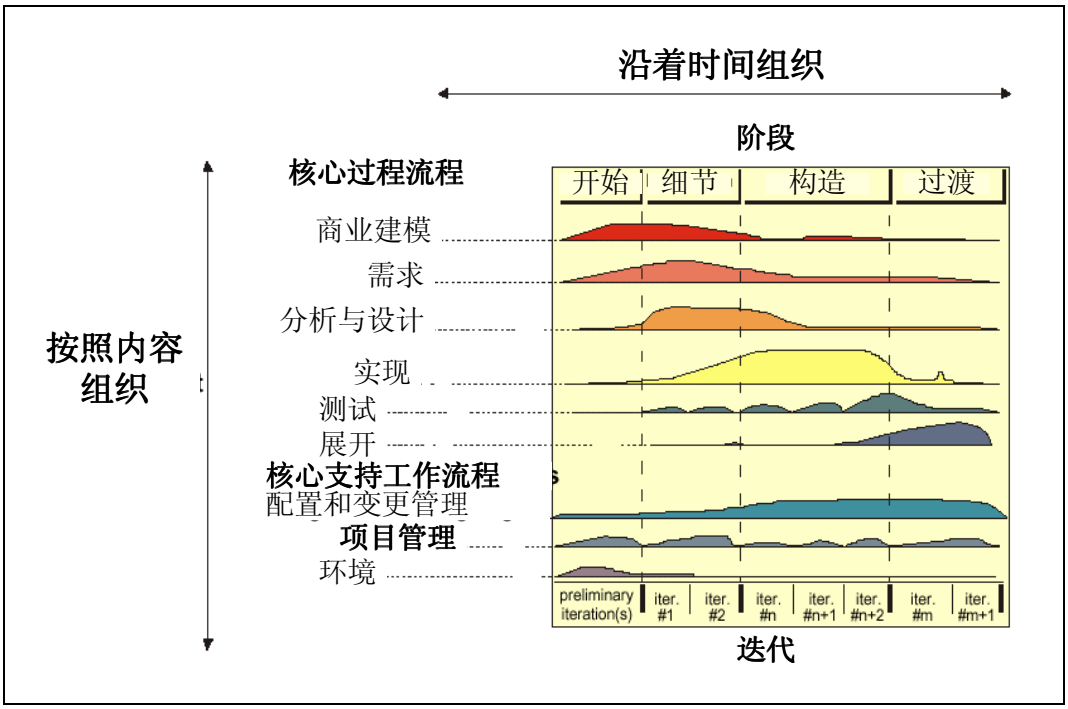

图 9-3　迭代模型图显示了过程在二维空间的结构

9.5　时间维：阶段和迭代

这是过程随着时间的动态组织。把软件的生命划分成一些周期，每个周期都影响新一代产品。Rational 统一过程把一个开发周期划分成四个连续的阶段：

- 开始阶段
- 细节阶段
- 构造阶段
- 过渡阶段

每个阶段的结果都是一个里程碑——是时间上的一点，此时必须做出重要的决策，达到一些重要的目标。每个阶段都有特定的目的（如图 9-4）。

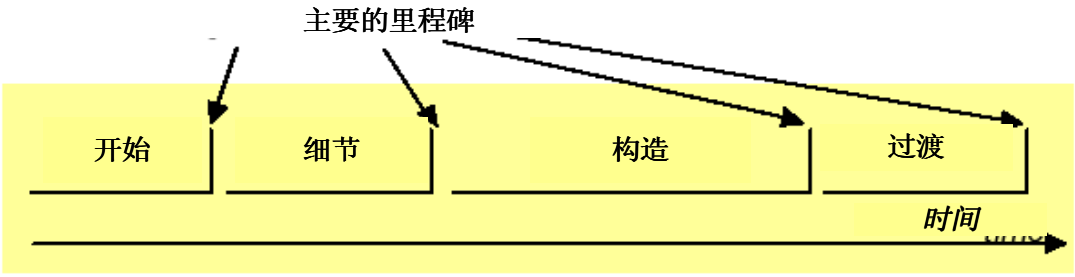

图 9-4　每个阶段的结果都是一个里程碑

9.5.1 开始阶段

在开始阶段要做的工作是，为系统建立商业用例，划定项目领域。为了达到这些目的，必须找出所有系统与之交互的外部实体（角色），并在较高的层次定义这些交互。包括识别所有用户、描述一些有意义的用例。商业用例包括成功的准则、风险评估、对所需要资源的估计以及一个示主要里程碑的重要日期的阶段规划。开始阶段的成果是：

- 前景文档：对核心项目需求、关键性质、主要限制的一个一般性的前景说明；
- 一个最初的用例模型（大约是整个系统的 10%到 20%）；
- 一个最初的项目词汇表（也可以部分地表示成一个领域模型）；
- 一个最初的商业用例，包括商业环境、一些成功的准则（税收预测、市场认知等），以及金融预测；
- 一个最初的风险评估；
- 一个项目规划，明确阶段和迭代；
- 如果需要的话，一个商业模型；
- 一个或多个原型。

里程碑：生命周期目标（Lifecycle Objectives Milestone）

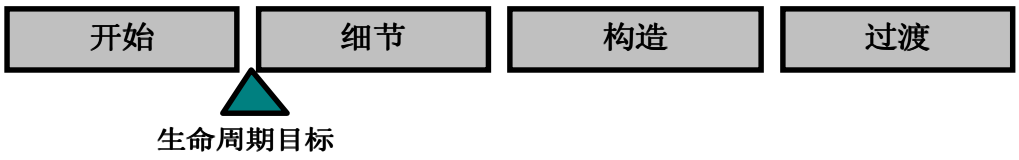

图 9-5　开始阶段的生命周期目标里程碑

在开始阶段的最后，是里程碑生命周期目标里程碑（如图 9-5 所示）。对开始阶段的评估准则是：

- 风险承担人并行地进行领域定义和成本 / 进度估计；
- 需求由主要的用例忠实无二义性地表达出来；
- 成本 / 进度估计、优先级、风险和开发过程的可信度；
- 任何开发出来的架构原型的深度和广度；
- 实际支出与计划支出的比较。

如果项目没有通过这个里程碑，就应该取消项目或重新思考。

9.5.2 细节阶段

细节阶段的目标是分析问题领域，建立一个合理的架构基础，开发项目规划，减少项目中风险最大的元素。为了达到这些目标，必须对系统有一个广泛而不太深入的认识。必须对整个系统的架构有一个理解：它的范围、主要的功能和非功能需求如性能需求等。

很明显，细节阶段是四个阶段中最重要的。在这个阶段的最后，最困难的“工程”就认为已经完成，对项目进行它最重要的结算：决定是否把它提交到构造和过渡阶段。对大多数项目来说，这也对应着从机动、轻捷、风险低的运作进入到高成本、高风险运作的过渡。尽管过程必须能够适应不断的变化，但是细节阶段的活动必须保证架构、需求和规划有足够的稳定性，充分地降低了风险，从而才能够确定地预算出整个开发过程的成本和进度。在细节阶段，根据项目的领域、大小和创新性，可能在一个或多个迭代中，建立一个可执行的架构。这一工作至少要解决开始阶段中找出的最重要用例，而它们通常也揭示了项目的主要技术风险。虽然目标是为产品质量组件建立一个不断演化的原型，但也不排除开发一个或多个用后可能扔掉的原型，来降低某些风险，如设计 / 需求折衷，组件可行性研究，或给投资人、客户或端用户的演示等。

细节阶段的成果是：

- 用例模型（至少完成了 80%）——识别出了所有的用例和角色，以及大多数用例的描述；
- 一些增加的需求，包括非功能性需求以及任何与特定用例无关的需求；
- 软件架构描述；
- 一个可执行的架构原型；
- 一个修正后的风险表和商业用例；
- 一份整个项目的开发规划，包括粗略项目规划，显示“迭代”以及对每个迭代的评估准则；
- 一个更新过的开发用例，指定要使用的过程；
- 一份初步的用户手册（可选）。

里程碑：生命周期架构

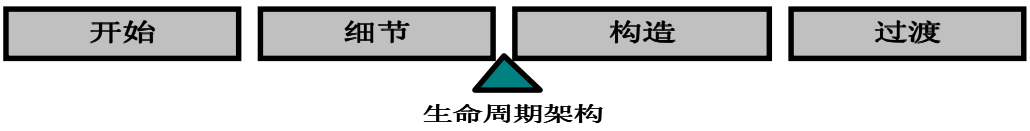

图 9-6　细节阶段的里程碑生命周期架构

在细节阶段的最后是第二个重要的项目里程碑，生命周期架构里程碑（如图 9-6 所示）。此时，检查系统详细的目标的领域，架构的选择以及对主要风险的解决。

细节阶段主要的评估准则包括对以下问题的回答：

- 产品的前景是否稳定；
- 架构是否稳定；
- 可执行的演示是否强调了主要的风险元素，并且已经解决？
- 构造阶段的规划是否已经足够详细和准确？是否有可信的评估支持？

- 如果用当前的规划来开发整个系统，并且使用当前的架构的话，是否所有的风险承担人对当前的前景都达成一致？
- 是否实际的资源支出与计划的支出都是可接受的？

如果项目不能通过这个里程碑，则将终止或重新考虑。

9.5.3 构造阶段

在构造阶段，将开发所有其它的组件和应用部件，并对它们进行测试，并集成到产品中。从某种意义上说，构造阶段是一个制造过程，重点是管理资源，控制运作，优化成本、进度和质量。从这个意义来说，管理层是从开始和细节阶段对智力产品的开发转向构造和过渡阶段对可应用产品的开发。

许多项目都很大，可以分散地并行增量地构造。这些并行活动可以显著地加速可应用版本的开发速度，但也增加了资源管理和工作流程同步的复杂性。一个健壮的架构加上一个易于理解的规划是息息相关的。换言之，架构最重要的一个质量因素就是它容易构造，这就是为什么在细节阶段强调架构和规划平衡开发的原因之一。

构造阶段的产品是一个可以立即提交给它的端用户使用的产品，它至少应该包括：

- 在足够的平台上集成的软件产品；
- 用户手册；
- 对当前版本的描述。

里程碑：初始运行能力

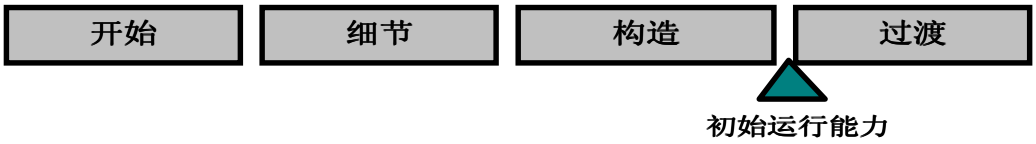

图 9-7　构造阶段的里程碑初始运行能力

在构造阶段的最后是第三个重要的项目里程碑：初始运行能力里程碑（如图 9-7 所示）。此时，要决定软件、节点和用户是否已经准备好运行，并且项目没有出现任何高风险问题。这个版本通常叫做“beta”版。

构造阶段的评估准则包括对以下问题的回答：

- 这个产品版本是否足够稳定和成熟，可以在用户群中发布吗？
- 是否所有的风险承担人都已经准备好把它提交给用户？
- 实际的资源支出和计划的支出是否仍然可接受？

如果项目没有达到这个里程碑，那么必须把过渡推迟发布。

9.5.4 过渡阶段

过渡阶段的目的是，把软件产品过渡给用户群。一旦产品提交给最终用户，通常会产生新的要求，如继续开发新版本，修正一些问题，或者完成某些被推迟的功能部件。当基线足够成熟，能够向最终用户领域发布时，就进入了过渡阶段。这通常需要系统的一些可以使用的子集已经达到一定的质量要求，并且有用户文档，从而使过渡产生积极的效果。包括：

- “beta 测试”确认新的系统达到用户的预期；
- 与它要取代的旧系统并行操作；
- 对运行的数据库进行转换；
- 训练用户和维护人员；
- 向市场、分销商和销售人员进行新产品的展示。

过渡阶段侧重把软件提交给用户的活动，这个阶段包括几个典型的迭代，如 beta 版本，通用版本以及错误修正和增强版本。在开发面向用户的文档、训练用户、支持用户早期使用产品以及对用户的反馈作出响应等都需要可观的精力。但是，在生命周期的这一点上，用户反馈可能主要限于产品调整、配置、安装和可用性问题上。过渡阶段的主要目标包括：

- 使用户可自我帮助；
- 使风险承担人合作，使展开基线完整，并与前景评估准则一致。

这一阶段根据产品的不同，可以非常简单，也可以极其复杂。例如，一种已有的桌面软件的新版本可以很简单，而替换国家的航空控制系统就非常复杂。

里程碑：产品发布

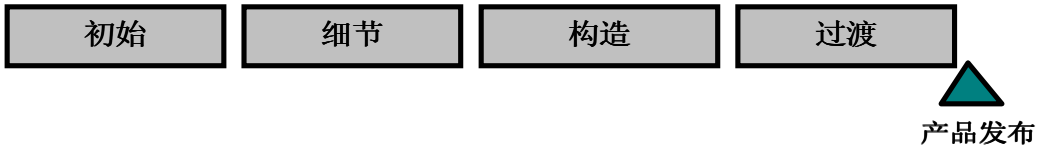

图 9-8　过渡阶段的里程碑产品发布

在过渡阶段的最后是第四个重要的项目里程碑，即产品发布里程碑（如图 9-8 所示）。在这一点上，要确定是否已经达到目标，能否开始另一个开发周期。有些情况情况下，这个里程碑会与下一周期的开始阶段的里程碑一致。过渡阶段主要的评估准则包括对以下问题的回答：

- 用户是否满意？
- 是否能够接受实际的和计划的资源支出？

9.5.5　迭代

在 Rational 统一过程中的每个阶段都可以继续细分成迭代。迭代是一个完整的开发循环，它的结果是可执行产品的一个版本（内部或外部的），是正在开发的最终产品的一个子集。它从一个迭代到另一个迭代，不断递增地成长，直到最后成为最终系统。

迭代方法的优点。相比于传统的瀑布过程，迭代过程具有以下优点：

- 在早期就降低风险；
- 对变化更加可管理；
- 更高的重用性；
- 项目团队在开发过程中可以学习；
- 更好的整体质量。

9.6　过程的静态结构

过程定义谁在做什么，怎么做，什么时候做。统一过程是用四个主要的建模元素来表达的：

- 工人，“谁”；
- 活动，“怎么做”；
- 产品，“做什么”；
- 工作流程，“什么时候做”。

9.6.1　活动、产品和工人

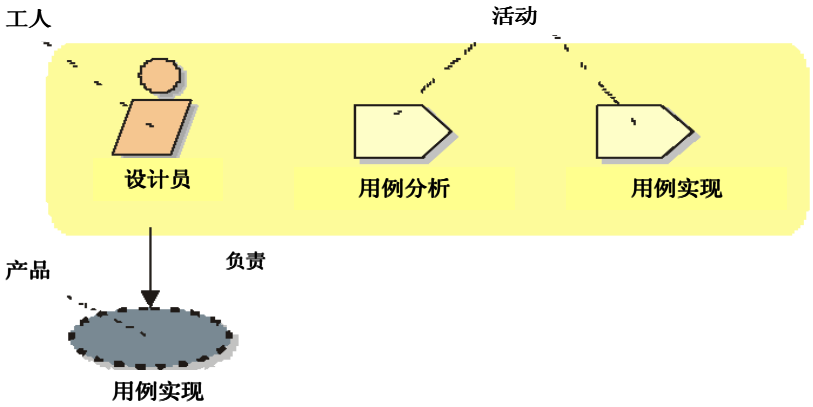

图 9-9　工人、活动和产品

如图 9-9 所示。

- 工人。工人定义了个体或一个作为团队工作小组的行为和责任。可以把工人看做是个体在项目中戴的一顶“帽子”，一个人可以戴好几顶不同帽子。这一点非常重要，因为很自然地会把工人当作个体或团队本身，但在统一过程中，工人还定义了个体应该进行的工作。分配给工人的责任包括完成某组活动，以及是一组产品的主人。

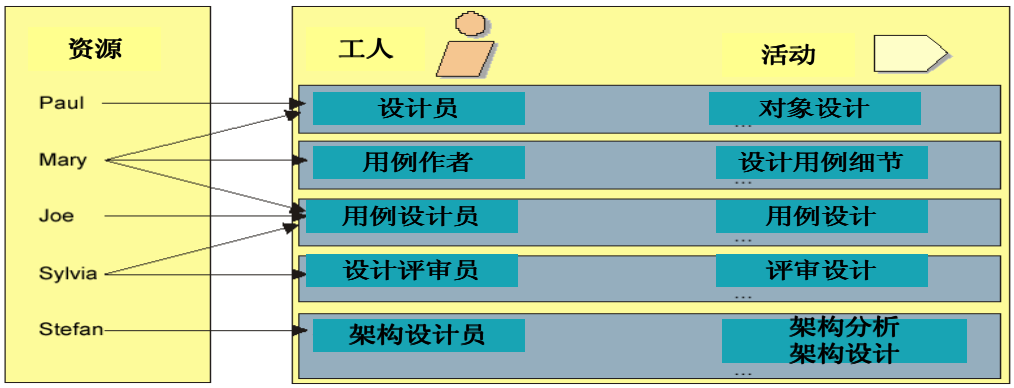

图 9-10　人和工人

- 活动。某个工人的活动是承担这一角色的人必须完成的一组工作。活动有明确的意图，通常用创建或更新某些产品来表示，这些产品包括模型、类、规划等。每个活动都分配给一个特定的工人。活动的力度通常从几个小时到几天不等，通过涉及一个工人，影响一个或数量很小的产品。活动应该可以作为规划和进展的元素；如果太小，它就被忽略；如果太大，就不得不用活动的一部分来表示进展（如图 9-10 所示）。活动的例子包括：
- 规划一个迭代，是工人：项目经理的活动；
- 找出用例和角色，是工人：系统分析员的活动；
- 审查设计，是工人：设计审查员的活动；
- 执行性能测试，是工人：性能测试员的活动。
- 产品。产品是一个过程所生产、修改或使用的一片信息。产品是项目切实的成果，是项目为生产出最终的产品而制造或使用的东西。产品是工人为了完成一个活动而用的输入，也是活动的输出或成果。用面向对象设计的名词来说，活动是在活动对象（工人）上的操作，产品是活动的参数。 产品的形式有：
- 模型，如用例模型或设计模型
- 模型元素，也就是模型里的一个元素，如类、用例或子系统
- 文档，如商业用例或软件架构文档
- 源代码
- 可执行程序

9.6.2　工作流程

仅仅把所有的工人、活动和产品都列举出来还不够组成过程，还需要一种有效的方式，把产生有价值结果的活动序列描述出来，并显示工人之间的交互。工作流程是一个活动序列，产生一个可观察到值的结果。用 UML 的话来说，可以用一个序列图、协作图或一个活动图来表示工作流程。这里我们用活动图来描述（如图 9-11 所示）。

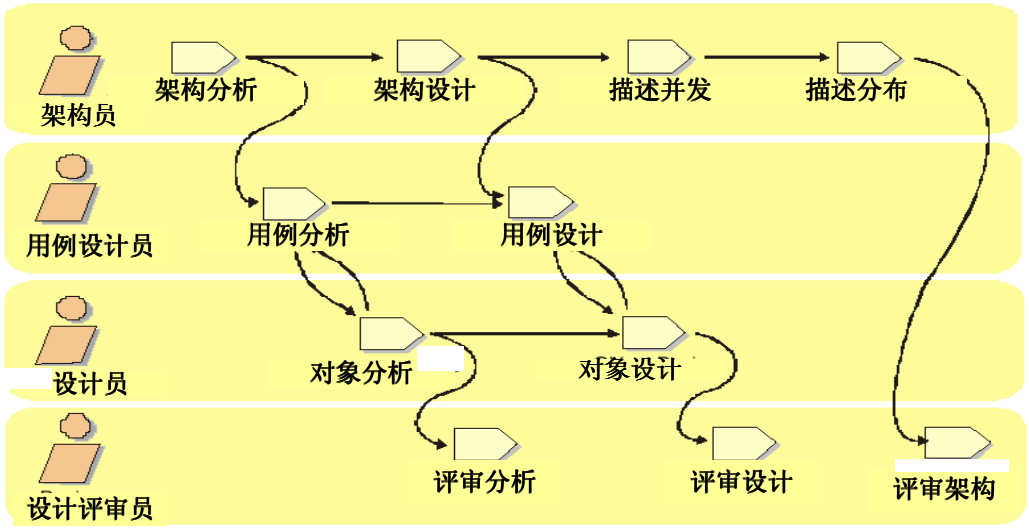

图 9-11　工作流程的例子

注意，要想表示出活动之间所有的相关性是不可能，也是不实际的，通常，两个活动之间的相关性会比图中显示出来的更紧密地交织在一起，尤其是当它们涉及同一个工人或同一个个体时。人不是机器，不能把工作流程解释成由人机械执行的程序。

下一节我们将讨论在过程中最基本的工作流程类型，称谓核心工作流程。

9.7　核心工作流程

统一过程中有九种核心过程工作流程，把工人和活动划分成不同的逻辑组合。

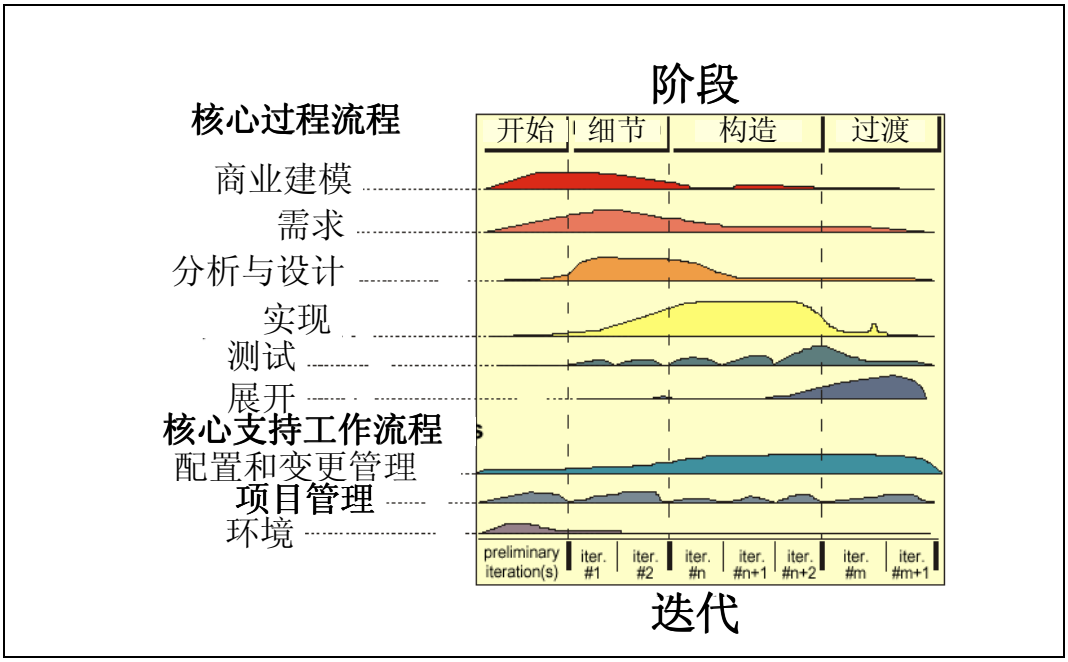

图 9-12　九个核心过程工作流程

核心过程工作流程被划分成六个核心“工程”工作流程（如图 9-12）：

1．商业建模工作流程
2．需求工作流程
3．分析与设计工作流程
4．实现工作流程
5．测试工作流程
6．展开工作流程

其它三个核心“支持”工作流程是：

1．项目管理工作流程
2．配置和变化管理工作流程
3．环境工作流程

尽管六个核心工程工作流程的名字会使人想起传统瀑布过程中的几个顺序阶段，但我

们必须记住，迭代过程的阶段是不同的，在整个生命周期中，这些工作流程被一遍一遍地访问。在一个项目实际的完整工作流程中，这九个核心工作流程是相互交错的，并且在每个迭代中都以不同的重点和强度重复它们。

9.7.1 商业建模

大多数商业工程项目的主要问题在于，软件工程和商业工程这两个社会无法恰当地相互交流，这就导致商业工程的输出无法被正确地用作软件开发的输入，反过来也是一样。Rational 统一过程通过为两大社会提供一个共同的语言和过程，同时说明如何创建和维护商业和软件模型之间直接的可跟踪性来解决这个问题。

在商业建模中，我们用所谓的商业用例来为商业过程建档，这就确保所有的风险承担人对机构到底需要支持什么样商业过程达成共识。对商业用例的分析是为了理解商业到底是如何支持商业过程的，这是用商业对象模型来建档的。但许多项目也许选择不做商业.建模。

9.7.2 需求

需求工作流程的目标是描述系统应该“做什么”，并且允许开发人员和客户都同意这一个描述。为了达到这个目的，我们启发、组织所需要的功能和约束，并编制文档把它们记录下来；跟踪并记录所做的折衷和决策。创建一个前景文档，启发风险承担人。找出代表用户以及其它可能与正在开发系统交互的角色，找出代表系统行为的用例。因为是按照角色的需要开发用例的，所以系统似乎与用户更有关系。图 9-13 是一个例子，显示的是一个回收机器系统的用例模型。

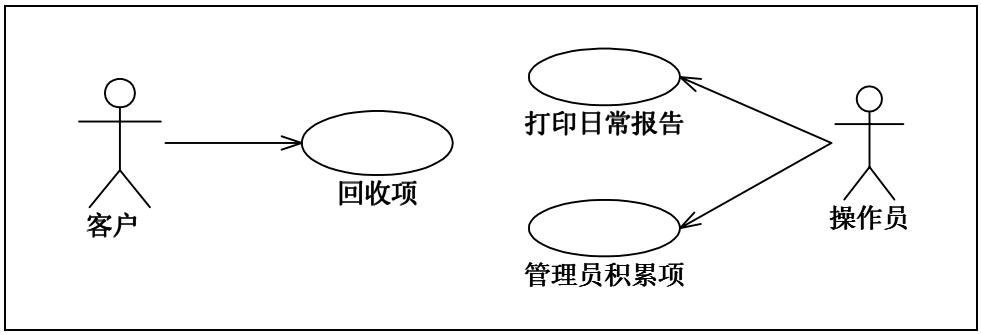

图 9-13　有角色和用例的用例模型的例子

每个用户都详细描述，用例描述显示了系统是如何一步一步与角色交互的，以及系统都做了些什么。非功能需求是用辅助规格说明。用例函数是贯穿系统开发周期的统一线索，在需求获取、分析与设计、测试中使用同一个用例模型。

9.7.3 分析和设计

分析与设计工作流程的目标是说明在实现阶段是“如何”实现系统的。建立的系统要：

- 在特定环境下，完成用例说明中所指定的任务和功能；
- 满足对它的所有需求；
- 被结构化，确保它是健壮的（如果功能需求发生变化时，很容易改变）。

分析与设计的结果是设计模型以及可选的一个分析模型。设计模型是源代码的一个抽象；也就是说，设计模型的作用是一个“蓝图”，描述了如何对源代码进行结构化和收发室。设计模型由设计类组成，通过定义好的接口，它们被结构化为设计包和设计子系统，它们代表在实现中，什么将变成组件。设计模型中还包含有关设计类的对象如何协作完成用例的描述。图 9-14 是上面图中的回收机器系统的设计模型的一部分。

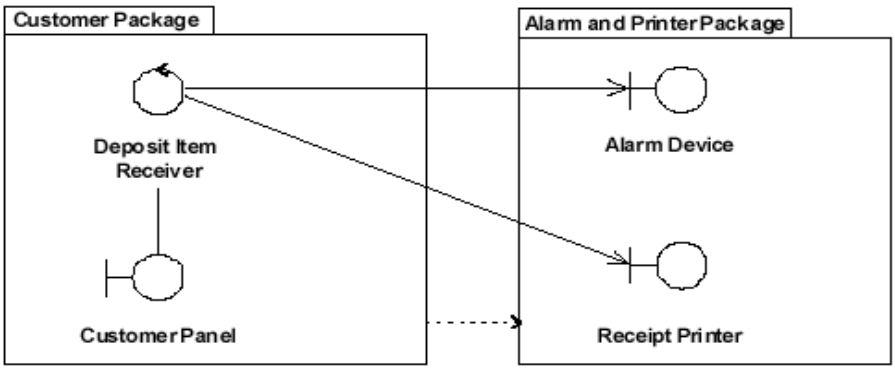

图 9-14　设计模型的一部分，有通信设计类和包组合设计类

设计活动以架构为中心，构建和确认这个架构是早期迭代最主要的重点。架构是用一种架构视图表示的，这些视图表达了主要的结构设计决策。从本质上，架构视图是整个设计的抽象或简化，丢掉细节，使重要的特征明显地表现出来。不仅对于开发好的设计模型而言，对于提高系统开发过程中任何模型的质量来说，架构都是一个非常重要的载体。

9.7.4　实现

实现的目的是：

- 通过分层次地组织实现子系统来定义代码的结构；
- 用组件（源文件，可执行文件等）来实现类；
- 把所开发的组件按照单元来测试；
- 把每个实现人员（团队）工作的结果集成到一个可执行的系统中。

系统是通过实现组件实现的。Rational 统一过程描述了如何重用已有组件，或用良定的责任实现新的组件，使系统更易于维护，提高重用的可能性。组件被结构为实现子系统。子系统的形式是目的，加上结构或管理信息。例如，在文件系统中，可以把子系统创建成一个目录或文件夹，或是 C++ 或 Ada 中的 Rational/Apex 子系统，或者是用 Java 的包。

9.7.5　测试

测试的目的是：

- 验证对象之间的交互；
- 验证是否恰当地集成了软件的所有组件；
- 在发布软件之前，找到错误并改正。

Rational 统一过程就是用一个迭代的方法，在整个项目过程都在进行测试。这些可以迟早发现错误，大幅度降低修改错误的成本。测试是沿着三个质量维进行的：可靠性、功

能性、应用性能和系统性能。对每一质量维，过程都描述了如果通过规划、设计、实现、执行和评估的测试生命周期。同时，还描述了“什么时候”以及“怎么”进行自动测试的策略。使用迭代的方法时，为了在每个迭代结束和产品新版本开始时能够进行回归测试，测试自动化尤其重要。

9.7.6 展开

展开工作流程的目的是为了成功地开发出产品版本，并把软件提交给它的端用户。它包括一系列的活动：

- 制作软件的外部版本；
- 把软件打包；
- 分发软件；
- 为用户提供帮助和支持。

在许多情况下，还包括：

- 规划和实施 beta 测试；
- 移植现有软件或数据；
- 正式接受。

尽管展开活动大部分围绕着过渡阶段，但许多活动还需要在早些阶段就进行准备。

9.7.7 项目管理

软件项目管理是平衡竞争目标、管理风险和战胜困难、成功地提交一个满足客户（付钱的人）和用户需要的产品的艺术。众多项目都无法使客户和用户满意的事实说明了这一任务的艰巨性。这一工作流程侧重于一个迭代开发过程的特定内容，目标是：

- 为管理软件密集型项目提供框架；
- 为项目的规划、人员管理、运行和监督提供实用的指南；
- 提供一个管理风险的框架。

这个工作流程并不是成功的秘诀，它只是提供了一个能够有效提高项目质量的项目管理方法。

9.7.8 配置和变化管理

在配置和变化管理这个工作流程中，描述了如何控制由共同完成同一项目的许多人制造出来的大量产品。控制有助于避免混乱，保证各个产品不会因为以下问题产生冲突：

- 同时更新——当一个或多个工人单独地做同一个产品时，最后一个人可能会破坏前面人的工作；
- 有限的通知——当多个人共享的几个产品中的错误被更正时，可能有些人没有得到有关修改的通知；
- 多版本—— 大多数大程序都是以版本不断进化的形式开发的。一个版本可能提供给用户使用，另一个用来测试，而第三个用来继续开发。如果在其中任一版本中发现问题，就需要把修改的信息传播给其它版本。

在这个工作流程中，将尽可能对重复工作、无效的改变进行控制和监视，以避免由此

产生的混乱。它主要提供一些管理多个软件版本的指南、跟踪在继续开发版本、保证按照用户定义的规格说明进行开发、强制采取节点专用开发政策。它对如何并行开发、在多节点上开发以及使建设过程自动化进行了描述。这些在迭代开发过程中尤其重要。另外，这个工作流程还描述了如果进行审核跟踪，把谁、什么时候、为什么对任何产品做的什么修改记录下来。另外，它还包括改变请求管理，也就是如何报告和管理故障，以及如何使用故障数据来跟踪进展和趋势。

9.7.9 环境

环境工作流程的目的是为软件开发组织提供软件开发环境——提供开发团队需要的过程和工具支持。这个工作流程的重点是在项目的情景下，配置过程的活动；另外就是描述如何在一个机构内实现过程。环境工件流程还包含一个开发工具包，提供一些定制过程的指南、模板和工具。

9.8 如何在过程中使用 UML

9.8.1 以架构为中心

使用 UML 的过程是以架构为中心的一个过程。也就是说，一个确定的基本系统架构是非常重要的，并且在过程的早期就要建立这个架构。系统架构是由不同模型的一组视图表达的，一般包括逻辑视图、并发视图、组件视图和展开视图，而用例视图则把这四种视图联系在一起，如图 9-15 所示。

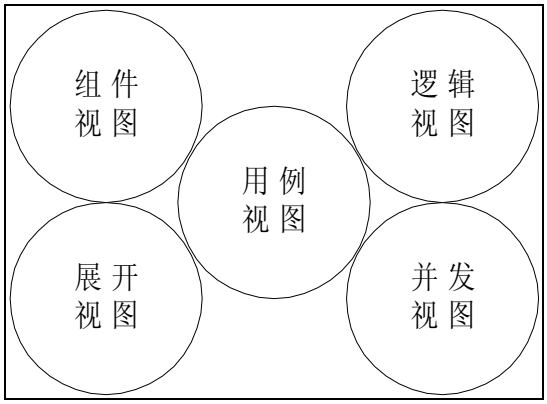

图 9-15 UML 的视图体现了系统的架构

架构是系统的映射，它定义了系统的不同组成部分、它们之间的关系和交互、通信机制、以及如何一些整体规则，如何修改系统组件、如何添加新组件等。好的架构强调系统的功能和非功能方面。

9.8.2 用例驱动

在进行一个大型面向对象项目时，我们典型地是从收集需求开始的用例技术开始，然后分析和设计的类图技术，最后主要的工作是编写代码。过程的每个小步骤都是迭代的，

但总体来看又遵循需求、分析、设计和编码这几个主要步骤。由于 UML 包含了对系统功能的描述，所以它们影响了所有的阶段和视图，如图 9-16 所示。用例把需求、分析、设计、实现和测试这些工作流程绑定在一起。

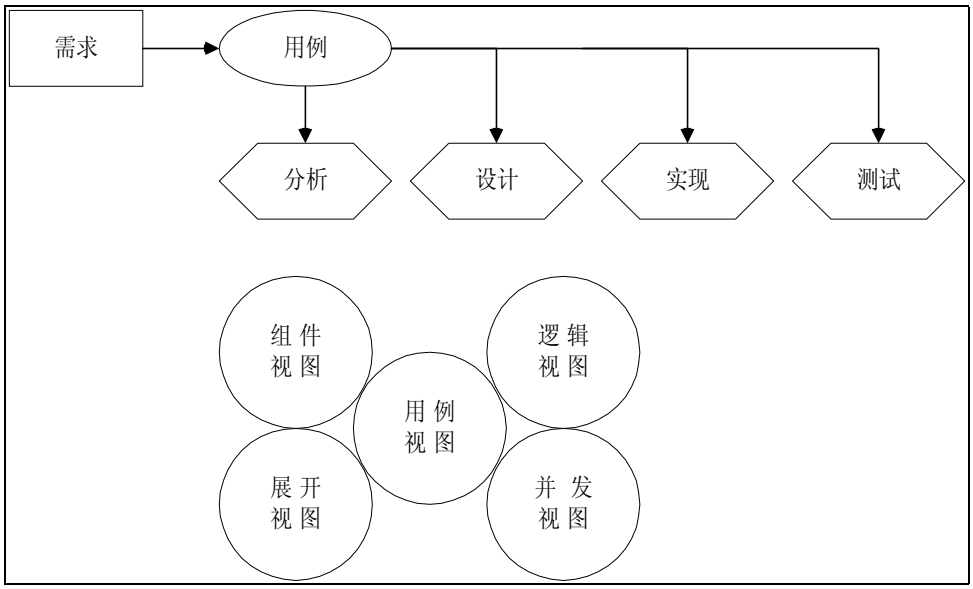

图 9-16　用例影响过程的每一阶段和视图

在分析阶段，使用用例来描述所要求的功能，并由客户确定这些功能；在设计和实现阶段，必须实现用例；最后，在测试阶段，由用例对系统进行验证；它们是测试用例的基础。

9.8.3　UML 对迭代开发过程的支持

在图 9-17 中表示了 UML 主要的图之间的关系，箭头表示“输入”关系。该图从更深一层表明了面向对象建模的基础，UML 不同模型之间的关系反映了面向对象建模的迭代特性。

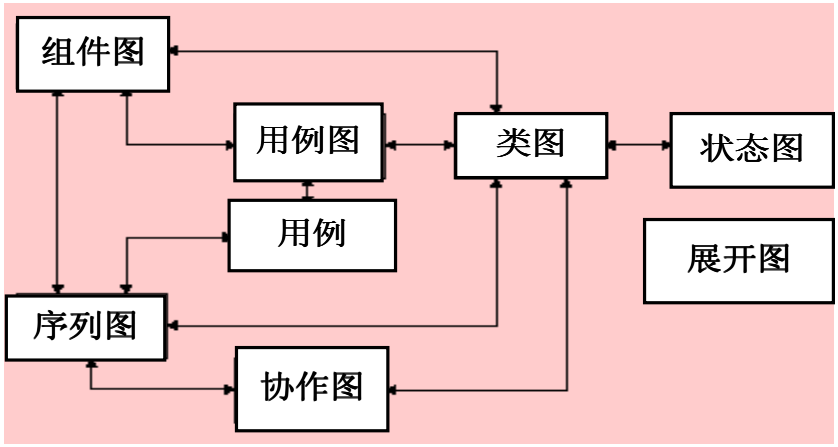

图 9-17　从迭代的角度理解 UML 的建模技术

图 9-18 显示另一种略微不同的构造过程，一种顺序的过程。其中，矩形之间的线代

表“由……建档”关系。例如状态图是用来为对象图建档的。该图还表明源代码是用来为类图中的类建档的。

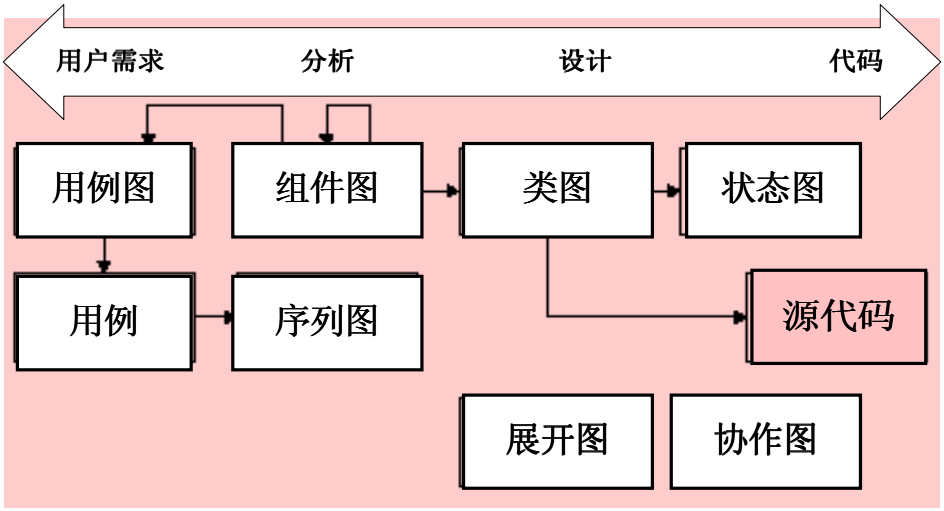

图 9-18　从顺序的角度理解 UML 的建模技术

图 9-17 和 9-18 显示了面向对象建模过程一个有趣的性质：从大的角度来看是一个顺序的过程，而从小的角度来看是一个迭代的过程。

9.8.4　UML 的图与工作流程和模型之间的关系

模型是对系统架构进行可视化、指定、构造和编制文档的手段和工具。在统一过程中的每个工作流程都有相应的模型来描述，对应每个流程可以有一个或多个模型。而这些模型就是用 UML 的图来表达的，UML 的图为模型提供了视图。见图 9-19。

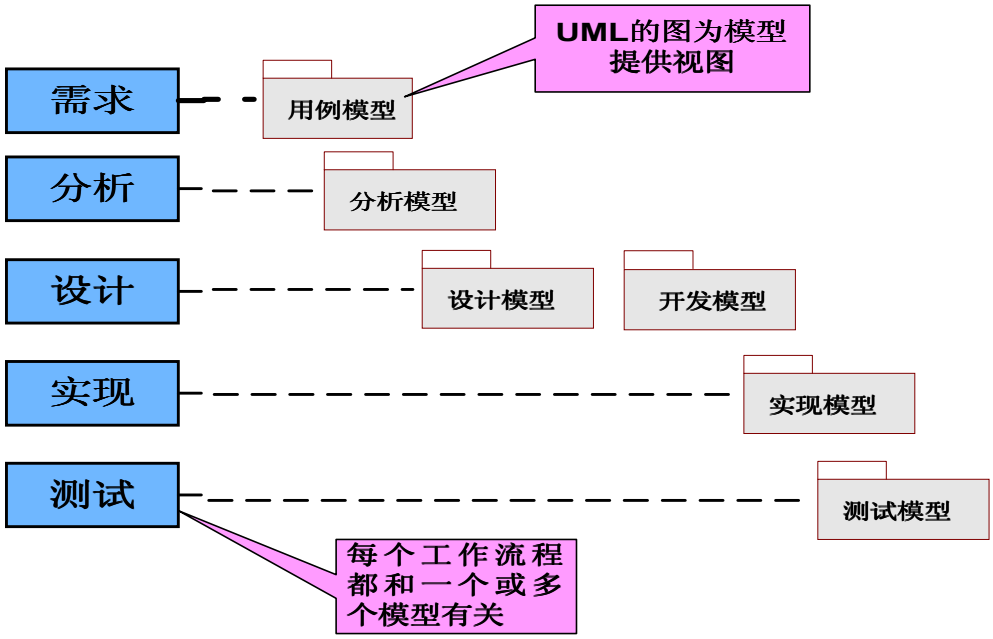

图 9-19　工作流程和模型

每个模型都是用一种或多种 UML 图来描述的，它们之间的对应如下：

- 用例模型。用用例图、顺序图、协作图、状态图和活动图描述。
- 分析模型。用类图和对象图（包括子系统和包）、顺序图、协作图、状态图和活动图描述。
- 设计模型。用类图和对象图（包括子系统和包）、顺序图、协作图、状态图和活动图描述。
- 开发模型。可用展开图（包括活动类和组件）、顺序图、协作图描述。
- 实现模型。可用组件图、顺序图和协作图描述。
- 测试模型。测试模型引用了所有其它模型，所以它使用它们对应的所有图。

9.9　小　　结

因为 UML 只是一种建模语言，没有过程，所以本章对使用 UML 的过程进行了讨论。简单地说，过程描述做什么、怎么做、什么时候做以及为什么要做，描述了一组以某种顺序完成的活动。过程的结果是一组有关系统的文档（模型和其它一些描述），以及对最初问题的解决方案。因为建模语言需要工具的支持，所以过程也需要工具的支持。不过，目前支持过程的工具不如支持建模的工具那么广泛。

首先介绍了 SEI 著名的 CMM 模型，初始级，可重复级，已定义级，已管理级和优化级。CMM 是一个由低到高逐渐成熟的演进框架，是衡量软件机构过程成熟度的尺度，它的目标是引导软件机构进行软件过程的持续改进。成熟度高的等级有着较高的生产率、较高的产品质量和较低的风险。根据 CMM，一个机构要想达到某个成熟度的过程，必须满足关键过程域并使之制度化。

目前市场上有几个流行的过程，我们介绍了 Rational 公司提出的统一过程，因为它是 UML 的三个创始人在 Rational 旗下提出的，所以它与 UML 的结合最为紧密。统一过程从时间和机构角度，对软件过程的活动进行组织。水平轴是时间维，是过程随着时间的动态组织。它把软件的生命划分成一些周期，每个周期都影响新一代产品。统一过程把一个开发周期划分成四个连续的阶段：开始阶段、细节阶段、构造阶段和过渡阶段。每个阶段的结果都是一个里程碑——是时间上的一点，此时必须做出重要的决策，达到一些重要的目标。每个阶段都有特定的目的。垂直轴是代表过程静态的一面：是用活动、产品、工人和工作流程描述的。

最后，我们讨论了使用 UML 的过程必须具备的特征，这就是以架构为中心、用例驱动、支持迭代和递增地开发过程。我们分析了 UML 对以上这些特征的支持。用例指定了功能，而架构指定了形式，必须平衡用例和架构。