第2章 什么是统一过程

2.1 章节指示图

本章将给出统一过程（UP）非常简练的描述。初学者应该从学习UP的历史开始。如果你已经了解了UP的历史，可以选择跳到2.4节，该节讨论UP和RUP（Rational Unified Process），或者直接阅读2.5节，该节讨论如何在你的项目中运用UP。

我们在UP方面的兴趣（也是本书所涉及的内容）是提供一种过程框架，在其中可以展现OO分析和设计技术。你可以在[Jacobson 1] 中找到UP的完整讨论，在 [Kroll 1]、[Kruchten 2]、[Ambler 1]、[Ambler 2] 和 [Ambler 3] 中找到有关RUP的优秀论述。

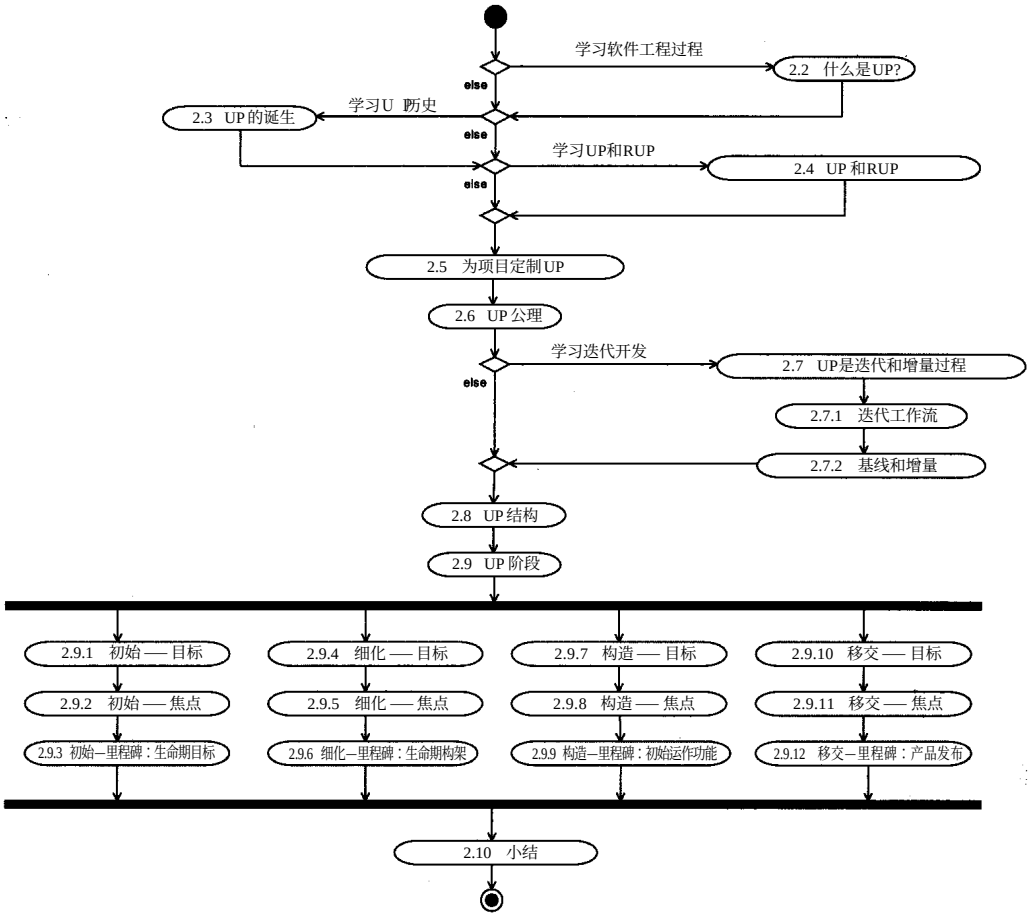

图 2-1

2.2 什么是 UP

软件过程工程（Software Engineering Process，SEP），又称为软件开发过程，定义了开发软件的who、what、when 和how。正如图2-2中说明的那样，SEP是一个把用户需求转换成软件的过程。

图 2-2

统一软件开发过程（Unified Software Development Process，USDP）是一个源于UML作者的SEP。它通常是指统一过程或者UP[Jacobson 1]。我们在整本书中使用术语UP。

UML项目意欲提供可视化语言和软件工程过程。我们现在所知道的是，UML是项目的可视化语言部分 —— UP是过程部分。然而，值得指出的是，UML已经被OMG标准化，但UP却没有。因此，仍然没有标准的SEP来补充UML。

UP的基础是在Ericsson公司（Ericsson方法）、Rational 公司（Rational Objectory Process，1996-1997）和其他最好实践来源所开展的过程工作。同样，它是开发软件实用的、已经经过验证的方法，它整合了其先驱的最好实践。

2.3 UP的诞生

当我们审视描述在图2-3中的UP的历史时，我们可以公平地说，它的发展同一个Ivar Jacobson，的职业生涯紧密相连。实际上，Jacobson 通常被认为是UP之父。这并没有看轻对于UP发展有贡献的其他所有个人（特别是Booch）的工作，而是强调Jacobson 的关键贡献。

UP可以回溯到1967的Ericsson 方法，它把复杂系统的基本步骤构造为一组相互联系的功能块。小的功能块相连以形成更大的功能块以构造完整的系统。该方法的基础是“分而治之”和我们今天所熟知的基于组件的开发（Component-Based Development，CBD）。

尽管整个系统对于部分触及到其组成部分的任何个体来说可能是不可理解的，但是当系统被分成更小的组件时，人们可以理解每个组件提供的服务（用现代术语讲，组件的接口）以及这些组件是如何相互适应的。用UML的语言说，大的功能块被称为子系统，每个子系统是由更小的功能块（组件）所实现。

Ericsson的另一个创新是创建“交通事例”（traffic case）来识别这些功能块，它描述系统是如何被使用的。这些“交通事例”随着时间推移而进化，现在UML中被称为用例。这个过程的结果是展示系统构架，它描述了所有功能块以及它们是如何相互适配的。UML是静态模型的先驱。

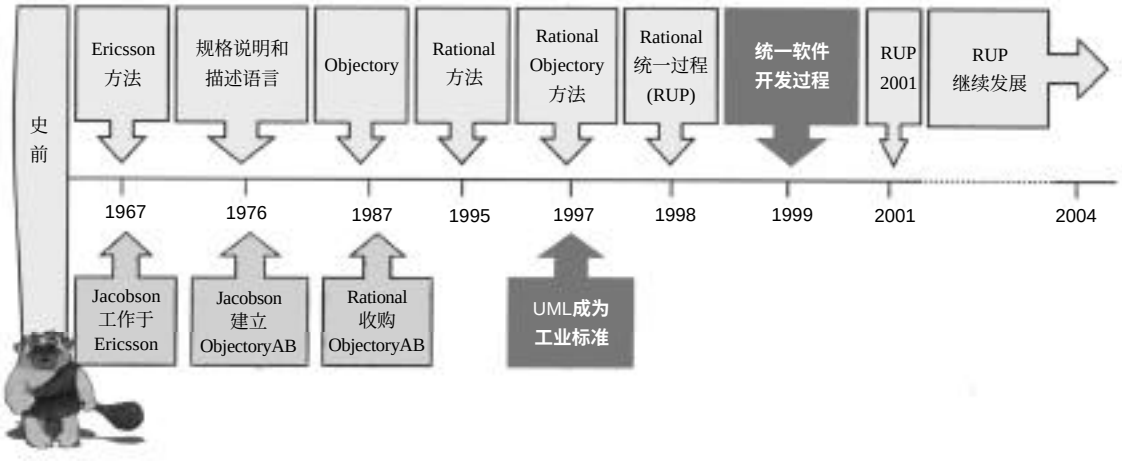

图 2-3

除了需求视图“(交通事例)”和静态视图（构架描述），Ericsson还具有动态视图，其描述所有功能块在时序上是如何通信的。这包括顺序图、通信图和状态图，所有的这些在UML中依然存在，尽管它们是以非常精化的形式出现的。

OO 软件工程的下一个主要发展是国际标准组织CCITT的规格说明和描述语言（Specification and Description Language，SDL）的发布。SDL是第一个基于对象的可视化建模语言，1992年，它被扩展成为使用类和继承的面向对象的语言。该语言被设计用于捕获通信系统的行为。系统被建模为一组功能块，它们通过相互之间发送信号进行通信。SDL-92 是第一个对象建模标准并一直沿用至今。

1987，Jacobson 在斯德哥尔摩成立了Objectory AB。该公司开发并出售工基于Ericsson方法的软件工程过程，被称为Objectory（Object Factory）。Objectory SEP 由一组文档、相当特殊的工具和一些源于 Objectory AB可能必需的咨询服务组成。

这个时期的最重要的创新是Objectory SEP自身是按照系统方法来开发的。过程的工作流（需求、分析、设计、实现和测试）是用一组图来表达的。换言之，Objectory过程是像软件系统一样被建模和开发的。这为将来过程的开发铺平了道路。Objectory和UP一样是一个过程框架，在被用于特定项目之前需要充分定制。Objectory过程产品与为各种类型软件开发过程制作的一些模板一并推出，但是这些模板总要进一步大量剪裁。Jacobson 认识到所有软件开发项目都是不同的，因此“放之四海皆准”的SEP 不是真正可行的或所期望的。

1995年，Rational收购了Objectory AB，Jacobson 致力于 Rational 已有的大量过程相关的统一工作。4+1的构架视图——四种不同的视图（逻辑、进程、物理和部署视图）加上统一的用例视图被开发出来。这些仍然是形成系统架构UP方法的基础。此外，迭代开发被形式化为阶段序列（初始、细化、构造和移交）它符合瀑布生命周期的原则以及迭代和增量开发的动态反馈。这个工作的主要参与者是 Walker Royce、Rich Reitmann、Grady Booch（Booch方法的发明人）和 Philippe Krutchen。特别地，Booch关于构架的经验和强大思想被整合进了Rational方法（请参见[Booch 1] 以了解对于他的优秀思想的讨论）。

Rational Objectory Process（ROP）是Objectory和Rational的过程工作统一的结晶。值得一提的是，

ROP 改进了Objectory的薄弱环节——需求（而不是用例）实现、测试、项目管理、部署、配置管理和开发环境。风险被作为ROP的驱动之一而引进，构架被定义并被形式化为“构架描述”的可交付产品。在此其间，Booch、Jacobson和Rumbaugh在Rational开发了UML。这成为 ROP 建模语言，ROP 本身以此来表达。

1997年后，Rational 收购了很多公司，引入了在需求捕获、配置管理、测试等领域的专门技术。这导致了1998年Rational Unified Process（RUP）的发布。从此，RUP发布了多个版本，每个发布都胜过先前的版本。请参见www.rational.com和[Kruchten 1]以获得更详细的信息。

1999年，一本重要的书《统一软件开发过程》(《The Unified Software Develepment Process》)[Jacobson 1]出版了，它描述了统一过程。尽管RUP是Rational的过程产品，但UP是源于UML作者的开放SEP。毫不奇怪，UP和RUP是非常相似的。本书中，我们采用UP而不是 RUP，因为UP是一个开放的 SEP，对所有人来说都是可理解的，并且不束缚于任何特定产品或者供应商。

2.4 UP和RUP

RUP是IBMUP的商业版本，2000年，IBM 收购了Rational公司。它提供了所有的标准、工具以及其他必需物，这些东西是UP中没有包括的，并且需要你自己提供的。RUP 提供了丰富的、基于 web 的环境，对于每种工具，提供了完整的过程文档以及“工具导师”

回首1999年，可以合理地认为RUP仅是UP的业务实现。然而，RUP从那以后发展了很多，现在它在很多方面扩展了UP。今天，我们应该视UP 为开放的、一般情况，而视RUP为扩展和覆写UP特征的特定业务子类。但是RUP与UP仍然保持很多相似点而不是不同点。主要不同点在于完整性和细节上，而不在语义或者思想方面。OO分析和设计的基本工作流是极其相似的，UP观点描述对于RUP的用户同样适用。在本书中采用UP，我们让文本适合于大多数没有使用RUP 的OO分析师和设计师，以及少数使用过RUP 的专家。

UP和RUP都建模软件开发过程的who、when 和 what，但是它们有所不同。RUP最新版本与UP具有一些术语和语法差异，尽管过程元素的语义仍保持相同。

图2-4表示RUP过程图标是如何映射到我们在本书中所使用的UP图标。注意在RUP图标和原始UP图标之间存在«trace»关系。在UML中，«trace»关系是模型元素之间特殊类型的依赖，表示«trace»关系的开始端元素是箭头所指端元素的历史发展。这非常好地描述了UP和RUP模型元素。

为建模SEP的“who”，UP 引入了工作者（worker）的概念。其描述项目中由个体或团队所扮演的角色。每个工作者由很多个体或者团队所实现，而每个团队或者个体作为多个工作者而进行工作。在RUP中，工作者实际上被称为“角色”，但是语义保持相同。

UP把“what”建模为活动和制品。活动是项目中由个体或者团队执行的任务。这些个体或者团队在执行特定活动时，总是采用特定的角色，因此，对于任何活动，UP（和 RUP）告诉我们参与该活动的工作者（角色）根据需要，活动可以被分解成更小的活动。制品是项目的输入和输出，它们可以是源代码、可执行程序、标准、文档等。制品可以具有很多不同的图标，这取决于他们是什么内容，在图2-4中，我们使用了通用的文档图标。

UP把“when”建模为工作流。这些是由工作者所执行的相关活动的序列。在RUP 中，高级工作流，例如，需求或者测试，被赋予特殊的名称，纪律（discipline）。工作流被分解成一个或者多个工

作流细节，描述工作流内的活动、角色和制品。在UP中，只能通过名称引用这些工作流细节，但是在RUP中，他们被赋予的不同的自己的图标。

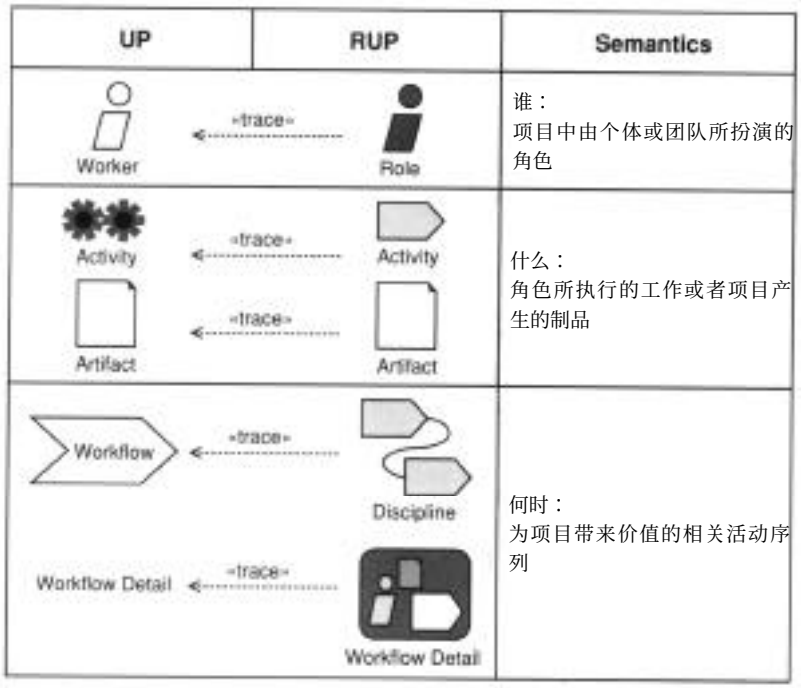

图 2-4

2.5 为项目定制UP

UP是通用的软件开发过程，必须为你的组织定制，然后为你的每个项目定制。所有的软件项目都不同，SEP“放之四海皆准”的方法不能很好地工作。定制过程包含定义和协调：

- 内部标准；
- 文档模板；
- 工具——编译、配置管理工具等；
- 数据库——缺陷跟踪、项目跟踪等；
- 生命周期变更——例如，关键安全系统采用的、更加复杂的质量控制措施；

关于定制过程的细节超出了本书范围，但它描述在[Rumbaugh 1]。

尽管RUP比UP完整得多，它仍然必需以相同的方式进行定制。然而，需要定制的工作量比从原始UP开始要省力得多。实际上，任何软件工程过程，通常需要投入一定量的时间和资金进行定制，你必须为帮助你进行定制的来自SEP供应商的咨询服务做好预算。

2.6 UP公理

UP 具有三个基本公理，它们是：

- 用例和风险驱动；
- 构架中心的；
- 迭代和增量的。

我们将在第4章深入学习用例，但现在我们说用例是捕获需求的方法。因此我们准确地说：UP是需求驱动的。

风险是UP的另一个驱动，因为如果你不主动攻击风险，风险就会主动攻击你！从事软件开发项目的任何人毫无疑问地同意这个观点。UP通过对风险分析预测并关注软件构造。然而这实际是项目经理和构架师的工作，因此我们在本书中不涉及它的任何细节。

开发软件系统的UP方法是开发和演进一个健壮的系统构架。构架描述了策略：系统是如何被分成组件，这些组件是如何交互和部署在硬件上。显然，高质量系统构架将产生高质量系统，而不是很少谋划的、堆砌在一起的源代码的集合。

最后，UP是迭代和增量的。UP的迭代表示我们把项目划分成小的子项目（迭代），提交系统的功能块或者增量，最终产生完整的功能系统。换言之，我们通过逐步精化过程构造我们的系统以达到我们的最终目标。同古老的瀑布生命周期的或多或少具有严格序列的分析、设计和构造相比，这有很大的不同。实际上，我们在整个项目的过程中将多次返回到关键UP工作流，如分析。

2.7 UP是迭代和增量过程

为了理解UP，我们必须理解迭代。迭代的思想很简单历史表明：通常说，人类发现小问题比大问题容易解决。因此我们把软件开发项目划分成许多更小的“迷你项目”，使之更容易管理和成功完成。每个“迷你项目”是一个迭代。要点是，每个迭代包含正常软件开发项目的所有元素：

- 计划
- 分析和设计
- 构造
- 集成和测试
- 内部或者外部发布

每次迭代产生包括最终系统的部分完成的版本和任何相关的项目文档的基线。通过逐步迭代基线之间相互构建，直到完成最终系统。

两个连续基线之间的差异被称为增量，这就是UP为什么被称为迭代和增量的生命周期。

你将在2.8节中看到，迭代被编组成为阶段。阶段提供UP的宏结构。

2.7.1 迭代工作流

在每个迭代中，有5个核心工作流，说明需要作什么以及需要什么工作技能。除了5个核心工作流外，还有其他工作流如计划、评估以及与特定迭代相关的任何其他的工作。然而，UP中不包括这些。5种核心工作流是：

- 需求——捕获系统应该作什么；
- 分析——精化和结构化需求；
- 设计——在系统构架内实现需求；
- 实现——构造软件；

• 测试——验证实现是否如期望那样工作。

迭代的一些可能工作流图解在图2-5中。我们将在本书的后面看到更详细的需求、分析、设计和实现的工作流（测试工作流不在涉及的范围之内）

尽管每次迭代可以包含5种核心工作流，但是特定工作流的重点依赖于项目生命周期中的迭代发生的位置。

把项目划分成一系列迭代，允许我们对项目进行灵活计划。最简单的方法是按照时间顺序的迭代序列，一个接一个。然而，常常可能并行安排迭代。这意味着需要理解每次迭代的制品之间的依赖，需要有方法指导基于构架和模型的并行迭代。并行迭代的好处是缩短面市时间，可以更好地利用团队，但是必须仔细计划。

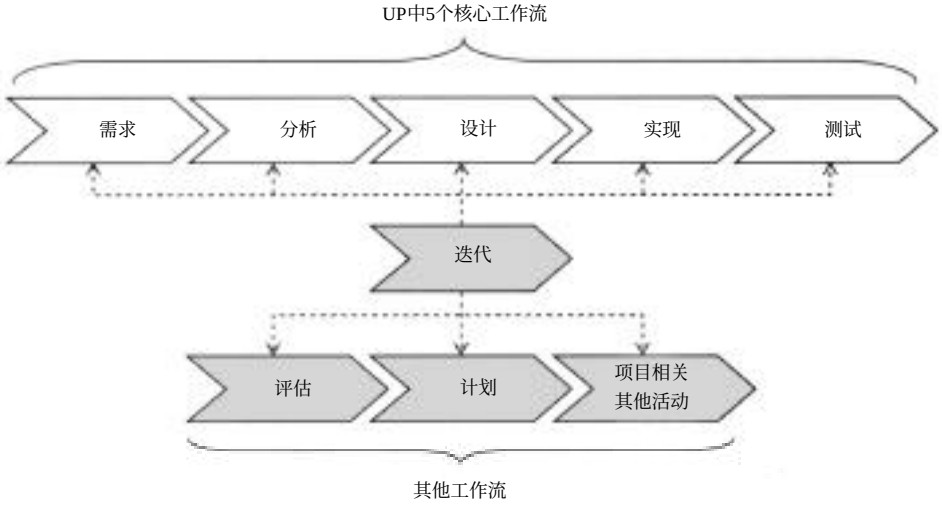

图 2-5

2.7.2 基线和增量

每次UP迭代产生一个基线。每次迭代产生一组已评审和已批准制品的内部（或者外部）发布。每个基线：

• 提供进一步评审和开发的一致基础；

• 只能通过正式的配置和变更管理过程才能变更。

然而，增量仅是一个基线和下一个基线之间的差异。它们组成了通向最终、可交付系统的台阶。

2.8 UP结构

图2-6表示UP的结构。项目生命周期被划分成四个阶段——初始、细化、构造和移交。每个阶段结束都有一个重要的里程碑。在每个阶段中，存在一个或者多个迭代，在每次迭代中，我们可以执行5种核心工作流和任何额外的工作流。每个阶段迭代的精确数目依赖于项目的规模，但是每次迭代不应该超过2-3个月。例如，一个中等规模的项目大约持续18个月。

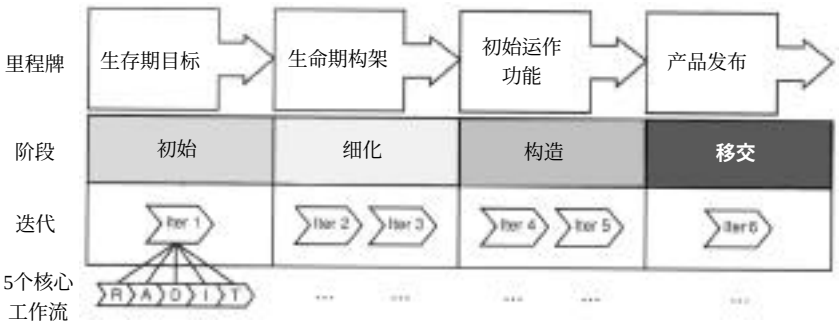

图 2-6

图2-6中，UP由四个阶段的序列组成。每个阶段由主要里程碑所终止。

- 初始 —— 生命周期目标；
- 细化 —— 生命周期构架；
- 构造 —— 初始运作功能；
- 移交 —— 产品发布。

随着项目按UP的阶段进展，每个核心工作流的工作量发生变化。

图2-7是理解UP工作流的钥匙。沿着顶部而下，是阶段。沿着左边而右是9种主要工作流。沿着底部，是一些迭代。曲线表示随着项目阶段进展核心工作流中相对工作量。

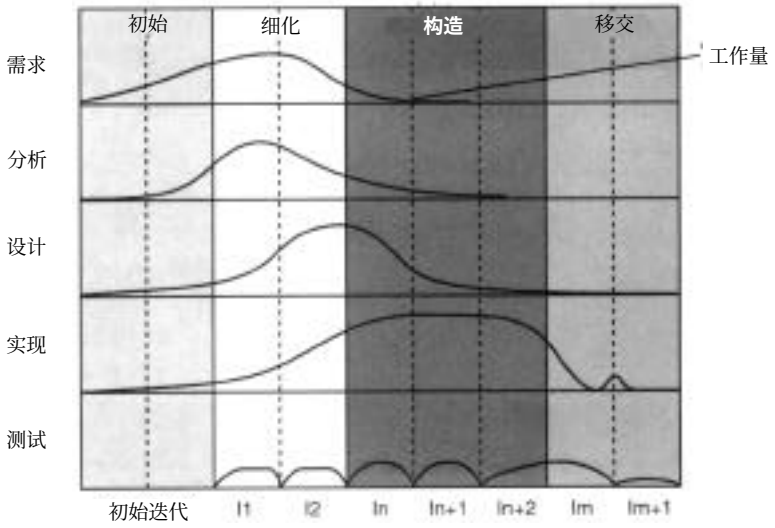

图2-7　源自 [Jacobson 1] 的图 1-5，获得 Addison-Wesley 的授权

像图2-7演示的那样，初始阶段中，大部分工作是需求和分析。细化阶段中，重点是需求、分析和一些设计。显然，在构造阶段，重点是设计和实现。最后，在移交阶段，重点是实现和测试。

UP的重要特征之一是基于目标的过程，而不是基于交付的过程。每个阶段结束于里程碑，里程碑是包含一组满足的条件，这些条件包含创建特定提交物，而不依赖于项目的特定需要。

在本章余下的部分，我们将简要概述每个UP阶段。

2.9 UP阶段

每个阶段具有目标、带有一个或者多个核心工作流的重要活动焦点和里程碑。这将是我们研究阶段的框架。

2.9.1 初始—目标

初始的目标是“获得项目的基础”初始包括：

- 建立可行性—它包括一些验证技术决策的技术原型或者验证业务需求的概念验证模型；
- 创建业务用例，演示项目将移交合格的业务利益；
- 捕获基本需求，帮助界定系统范围；
- 识别关键风险。

该阶段的主要工作者是项目经理和系统构架师。

2.9.2 初始—焦点

初始的主要重点是需求和分析工作流。然而，如果决定构造技术或者概念验证原型，可能要作一些设计和实现。测试工作流对于本阶段不适用，因为仅有的软件制品是将要抛弃的原型。

2.9.3 初始—里程碑：生命期目标

当很多SEP关注主要制品的创建时，UP采用面向目标的不同方法。每个里程碑设定必须完成的特定目标，这个目标必须在里程碑被认为已经抵达之前完成。一些目标可能产生特定制品，而一些不是。

初始的里程碑是生命期目标。达到该里程碑必须满足的条件在表2-1 中列出。我们建议你创建一组提交物以实现这些条件。然而，请记住，当它对于项目增加真正价值时，才创建该提交物。

表 2-1

满足的条件	提交的产品
利益相关人同意项目目标	描述项目的主要需求、特征和约束的愿景文档
已经定义系统范围，获得利益相关人认可	初始的用例模型（仅是完整的10%~20%）
已经捕获关键需求，获得利益相关人认可	项目词汇表
利益相关人认可成本和进度评估	初始项目计划
项目经理已经提出业务用例	业务用例
项目经理已经作过风险评估	风险评估文档和数据库
通过技术研究和/或原型确认可行性	一个或者多个可抛弃原型
构架轮廓出现	初始的构架文档

2.9.4 细化—目标

细化的目标概括如下：

- 创建可执行的构架基线；
- 精化风险评估；
- 定义质量属性（缺陷发现率、可接受的缺陷密度等）；
- 捕获 80% 的功能性需求用例（你将在第3 章和第 4 章中看到，这确切包含什么）

• 为构造阶段创建详细计划；

• 计划包括资源、时间、设备、人员和成本标价。

细化阶段的主要目的是创建可执行构架基线。这是真正可执行系统，它是根据特定构架而构造的。它不是原型（原型被抛弃了,）而是所期望系统的“ 初步”随着项目的推进，这个初始的可执行构架基础将被加入，并且在构造和移交阶段演进为最终可提交系统。因为未来的阶段以细化的结果作为基础，所以这可能是最关键的阶段。实际上，本书非常关注细化的活动。

2.9.5 细化 — 焦点

在细化阶段，每个核心工作流关注如下：

• 需求 — 精化系统范围和需求；

• 分析 — 确定了需要构造什么；

• 设计 — 创建稳定的构架；

• 实现 — 构造构架基线；

• 测试 — 测试构架基线。

显然，细化关注需求、分析和设计工作流，在本阶段后期，当可执行构架基线将要产生时，实现变得很重要。

2.9.6 细化 — 里程碑：生命期构架

生命期构架的里程碑。该里程碑必须满足的条件列举在表-2中。

表 2-2

满足的条件	提交的产品
已经创建有弹性的、健壮的、可执行的构架基线	可执行构架基线
可执行构架基线演示了重要风险已被识别并且被解决	UML 静态模型
	UML动态模型
	UML用例模型
产品的愿景已经稳定	愿景文档
风险评估已被审阅和修正	已更新的风险评估
已修正业务用例，获得利益相关人的认可	已更新的用例
已创建足够详细的项目计划，使得下一阶段的时间、资金和资源的估算符合实际	已更新的项目计划
利益相关人同意项目计划	
按照项目计划，用例模型已经被修正	已更新用例
利益相关人同意继续项目	签字（sign-off）文档

2.9.7 构造 — 目标

构造的目标是完成所有的需求、分析和设计。把在细化阶段产生的构架基线演进成最终系统。构造的主要问题是维护系统构架的完整性。一旦交付压力存在，便急于编码，并偷工减料，这导致破坏构架愿景、低质量和高维护成本的最终系统的产生，这种情况很常见。显而易见，应该避免这种结果。

2.9.8 构造 — 焦点

本阶段的主要工作流是实现。在其他工作流中，充分的工作已经完成需求捕获、分析和设计。测

试同样变得更加重要—因为每个新的增量都在最近基线的之上构造的，同时需要单元测试和集成测试。我们总结了每个工作流中经历的工作：

- 需求 —— 揭示任何遗漏的需求；
- 分析 —— 完成分析模型；
- 设计 —— 完成设计模型；
- 实现 —— 构造初始运作功能；
- 测试 —— 测试初始运作功能。

2.9.9 构造 —— 里程碑：初始运作功能

从本质上讲，里程碑非常简单—软件系统已经完成，准备在用户场地进行beta测试。该里程碑的满足条件在表2-3 中给出。

表 2-3

满足的条件	提交的产品
软件产品足够稳定、具有足够的品质可以部署到用户社区	软件产品 UML模型 测试套件
利益相关人认可并且作好了把软件移交到他们环境中的准备	用户手册 发布的描述
实际的支出与计划支出是可接受的	项目计划

2.9.10 移交 —— 目标

当beta测试完成，移交阶段开始，系统最终部署。这包括修复了在beta 阶段中发现的任何缺陷，为软件在所有用户场地首次展示作好准备。下面，我们总结了该阶段的目标：

- 修复缺陷；
- 为用户场地准备新软件；
- 在用户场地剪裁软件；
- 如果不可预见的问题出现，修改软件；
- 创作用户手册和其他文档；
- 提供用户咨询；
- 产生项目后评审。

2.9.11 移交 —— 焦点

重点在实现和测试工作流。已做了充分的设计去纠正在beta测试中发现的任何设计错误。希望，到了项目生命周期的这个阶段，需求和分析工作流应该具有非常少的工作量。如果不是这样，那么项目已陷入麻烦。

- 需求 —— 不适用；
- 分析 —— 不适用；
- 设计 —— 如果beta测试中发现设计问题，修改设计；
- 实现 —— 为用户场地裁减软件，修复在beta 测试中发现的问题；

• 测试 —— beta测试以及在用户场地终验。

2.9.12 移交 —— 里程碑：产品发布

这是最后的里程碑——beta测试、终验测试和缺陷修复已经完成，产品发布并被用户社区接受。该里程碑的满足条件列举在表2-4中。

表 2-4

满足的条件	提交的产品
已完成beta测试，已作出必要的修改，用户认可系统已经成功部署 用户社区积极使用该产品	软件产品
用户认可用户支持策略并且已经实施	用户支持计划 更新的用户手册

2.10 小结

• 软件工程过程（SEP）通过描述who在when做what而把用户需求转化成软件。
• 统一过程（UP）是从1967年开始一步步发展进化而来的。它是源于UML作者的成熟的开放的SEP。
• RUP 是UP的一个商业扩展。它完全兼容UP，但比UP更完整、更详细。
• UP（和RUP）必须通过添加内部的标准等为特定项目定制。
• UP是现代的 SEP，它是：
  —风险和用例（需求）驱动的；
  —以构架为中心的；
  —迭代和增量的。
• UP软件是在迭代中构造的：
  —每次迭代就像一个“迷你项目”它交付系统的某个部分；
  —迭代相互为基础，以便构造出最终的系统。
• 每次迭代都具有5种核心工作流：
  —需求 —— 捕获系统应该做什么；
  —分析 —— 精化和结构化需求；
  —设计 —— 在系统构架中实现需求（系统是如何工作的）；
  —实现 —— 构造软件；
  —测试 —— 验证实现是否如期望那样工作。
• UP具有四个阶段，每个阶段在一个主要里程碑处结束：
  —初始 —— 获得项目的基础：生命周期目标；
  —细化 —— 进化系统构架：生命周期构架；
  —构造 —— 构造软件：初始运作功能；
  —移交 —— 把软件部署到用户环境：产品发布。