第 3 章

软 件 过 程

本章学习目标

- 理解软件过程和过程模型的概念。
- 理解软件过程框架及其涉及的活动和任务。
- 掌握经典软件过程模型的特性。
- 具有选择和区分不同经典模型的初步能力。

本章向读者介绍软件工程中重要的概念之一——软件过程，它与软件开发技术相关联，但是侧重点不同。3.1 节介绍软件过程的概念，重点是软件过程的定义和软件过程模型的概念。3.2 节简述经典的软件过程模型，使读者具有软件过程模型选用的初步知识。3.3 节通过示例介绍软件开发中选择软件过程模型时需要考虑的主要因素。3.4 节为本章小结，概要总结了本章的主要内容和学习重点。3.5 节综合习题，用于巩固、扩展和检查本章所学知识。3.6 节提供实践指导，目的是通过实践活动加深读者对本章知识的理解。3.7 节推荐的读物供读者以不同的叙述方法了解相关的知识，还可以扩展知识，并获得新的理解。

3.1 软件过程的概念

成功地开发一个软件系统需要从事一系列复杂的工程活动，涉及技术和管理两方面，活动应有输入性和输出性的工作成果，需要合理安排活动的工序并提供支撑活动的资源。从软件工程发展历程看，早期没有软件过程的概念，相关的工程活动混乱到不堪回首，当产业界和学术界意识到必须解决工程活动的混乱后，过程概念被引进软件工程中，使得无序逐渐进步到有序，并建立起一套软件过程的方法和技术。

3.1.1 软件过程的概念和模型

软件过程(Software Process)是为建造高质量软件所需完成的任务的框架，即形成软件产品的一系列步骤，包括中间产品、资源、角色及过程中采取的方法、工具等范畴。

软件过程的通用构成框架如图 3.1 所示，它涉及软件生存周期中所涉及的一系列软件开发过程。过程(Process)是活动(Activity)的集合；活动是任务(Task)的集合；任务的作用是加工输入然后产生输出；输出是过程的产出的结果，这个结果可以是模型、程序、数据、文档、报告、表格等。活动之间的关系可以是顺序的、并行的或者是按某一规则安排的。3.2 节将介绍经典的软件过程模型，这些经典模型之间的根本差异就是软件开发时活动安排不同，而这些经典模型的活动安排都是软件开发最佳实践的总结。

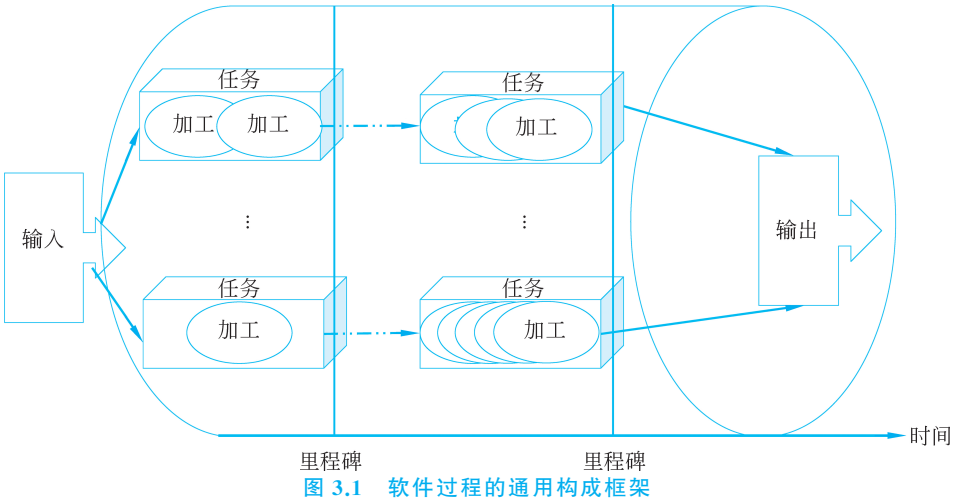


图 3.1　软件过程的通用构成框架

软件过程的基本目标是为了获得满足工程需求的软件，因此不仅要关注工程中的软件开发，还要关注涉及软件整个生命周期的工程支持和工程管理，包括需求获取、需求分析、软件设计、编程实现、软件测试、软件交付等工作任务。软件过程是软件项目管理的基础，它使得项目管理在此基础上运用技术和方法、规定产出的工作产品、设置里程碑、开展质量保证(SQA)、实施和控制变更等。

按照软件生命周期过程国际标准(ISO/IEC/IEEE 12207—2017 Systems and software engineering - Software life cycle processes)，软件过程可概括为 4 类：协商过程、组织项目启用过程、技术管理过程、技术过程，如表 3.1 所示，表的第 2 列为软件过程的 4 个大类，第 3 列为大类的细分过程。协商过程包含两个过程：采购过程、供应过程。组织项目启用过程包含 5 个过程：生命周期模型管理过程、基础设施管理过程、投资组合过程、人力资源管理过程、质量管理过程。技术管理过程包含 8 个过程：项目规划过程、项目评估和控制过程、决策管理过程、风险管理过程、配置管理过程、信息管理过程、测量过程、质量保证过程。技术过程包含 14 个过程：业务和任务分析过程、干系人需求和需求定义过程、系统/软件需求定义过程、架构定义过程、设计定义过程、系统分析过程、实现过程、集成过程、验证过程、转移过程、确认过程、运行过程、维护过程、处置过程。

一个软件在其生命周期中，从策划、开发、运维到消亡大体涵盖上述 4 大类共 29 个过程，过程的发生及过程之间的关系并没有固定的规范，在实际执行中过程之间可能有顺序关系，也可能有并行关系。但是，在软件工程实践中，为了避免软件生命周期中过程的混乱，形成了一些相对固定的、有典型意义的过程模型，为软件开发的规划提供了明确的路线图，3.2 节将介绍经典的软件过程模型。随着计算机技术的进步、社

会的发展，模型也在吐故纳新。

表 3.1 软件生命周期过程构成

软件生命周期过程	协商过程	采购过程
		供应过程
	组织项目启用过程	生命周期模型管理过程
		基础设施管理过程
		投资组合过程
		人力资源管理过程
		质量管理过程
	技术管理过程	项目规划过程
		项目评估和控制过程
		决策管理过程
		风险管理过程
		配置管理过程
		信息管理过程
		测量过程
		质量保证过程
	技术过程	业务和任务分析过程
		干系人需求和需求定义过程
		系统/软件需求定义过程
		架构定义过程
		设计定义过程
		系统分析过程
		实现过程
		集成过程
		验证过程
		转移过程
		确认过程
		运行过程
		维护过程
		处置过程

软件过程模型(Software Process Model)就是一种开发策略，是软件工程目标达成的保障。该策略为软件生命周期中的主要过程提供一套范型(Paradigm)，为工程进度的安排提供指导。过程模型就是对软件生命周期中基本活动的执行顺序给出建议性安排，基本活动来源于表 3.1 中的技术过程，主要包括软件需求、设计、实现、验证、交付。由于软件过程模型关注的软件过程及活动具有顺序性和依赖性，不能任意地安排。软件过程模型在安排活动时，采取的策略是基本活动的整体安排和拆分安排，从而形成模型的不同基本范型。模型中基本活动形成线性顺序、迭代、增量和并行的特性，如瀑布模型是典型的线性顺序范型，迭代模型是一种迭代范型，增量模型具有并行特性。不同模型适用的场景，通常与需求规模和清晰度有关，还与交付期要求和管理水平有关，以及与采用的技术和员工能力有关。

3.1.2 软件过程框架及活动

软件过程是指为实现交付和工件预定义的目标，项目团队对软件产品的开发与维

护的活动和基础设施，涉及工程和管理的技术和方法。一个软件过程的定义包含共性与特性两方面，特性是由不同的任务和不同的活动所决定的，特性不同过程就不同；共性是过程定义遵循的基本的、共同遵守的框架。如图 3.2 所示，过程框架就是共同方面，所有的软件过程都以此框架（模板）进行定义。特性是每个过程都有自己的目标、任务、交付物、质量保证点和里程碑。

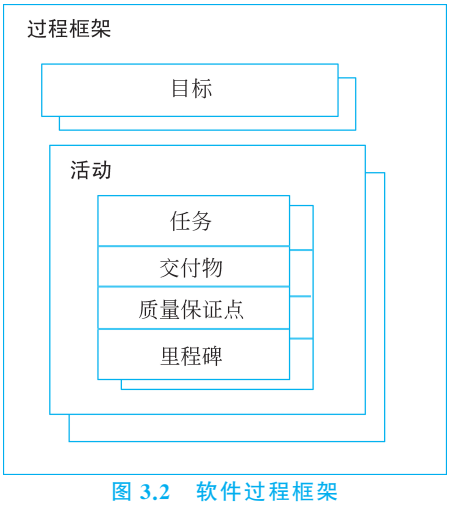


图 3.2　软件过程框架

活动（Activity），是要求角色执行的工作单元，角色通过执行一系列动作完成一个活动。活动主要实现宽泛的目标，例如，获取产品与产品组件需求，与应用领域、项目大小、结果复杂性等没有直接关系。

活动由 4 个基本单元构成：任务、交付物、质量保证点（要求）和里程碑。任务（Task）是基本的工作单位，通常作为一项工作指派给某人或某个团队，它能产生实际的成果，如建立产品需求。交付物是过程的产品物，是在指定的时间（里程碑）内提交的产品，并且要符合质量保证要求。质量保证点，是指在过程进行的某个时间点上要开展的质量保证活动，质量保证活动由保护性活动定义，如评审活动。里程碑是项目有关人员或管理人员负责的在预定时间将发生的事件，它用来标志过程执行时的工作进度，通常与工作产品提交时间和质量保证活动时间有关。

软件生命周期所需要开展的活动和需要完成的任务是多种多样的，没有一成不变的标准。其中，有核心的任务和为完成核心任务所需的基本活动，以及为做好核心任务而衍生的辅助或次要的任务和相关活动。要掌握软件过程的概念，首先要理解什么是软件生命周期中的核心任务和活动，以及这些任务和活动的目标和作用。过程中的活动是为了完成某一过程的一组密切相关的活动。以需求分析为例，其目的在于挖掘、分析并建立软件的需求规格，任务可细分为获取用户需求、开发产品的需求、确认需求。相关的活动包括领域理解、需求收集、需求检查、需求分类、需求优先级排序、需求描述、需求文档撰写、需求评审等。

生产一个软件的核心活动是编程，就是用程序设计语言描述软件，又称其为软件实现（Software Implementation），其他工作（活动）都是服务于编程的。但仅有编程活动无法保证开发出高质量的软件，还需要多项活动密切配合才能保证软件开发目标的实现，这就形成了基本的软件开发活动，包括需求获取（软件定义）、软件设计、代码编写、软件测试（有效性验证）、软件部署（交付）、软件维护（演化或进化）。有时将软件设计和代码编写合并称为软件构造（Software Construction）。

为了便于理解软件过程的概念，下面简要介绍基本软件开发活动。

（1）需求分析（Requirement Analysis）。为完成准确地回答“目标系统必须做什么”这一核心任务；另外一项重要任务是用正式文档准确地描述和定义目标系统的需求，这一份文档通常被称为需求规格说明书（Requirements Specification），是需求分析这一活动的主要产出物。

(2) 软件设计(Software Design)。基本任务是回答“怎样实现目标系统”,其中的主要任务是设计软件体系结构、数据存储方案,以及各功能模块或构件的实现方案,并产出软件设计规格说明书(Design Specification)。

(3) 代码编写(Coding)。任务明确且单一,就是将软件设计规格说明书的软件设计方案用程序代码实现(及相应的程序调试),因此又被称为实现(Implementation)。当然,除了产生程序代码外,当下许多软件开发还有一项任务就是生成软件需要的数据。

(4) 软件测试(Software Test)。软件测试是软件有效性验证最主要的技术手段,其任务是通过各种类型的测试验证软件是否达到预定的用户要求,即满足软件需求规格说明书。

(5) 软件交付(Software Delivery)。任务是将开发完成的软件或软件的一个新功能部署到生产系统中,供用户使用。

(6) 软件进化(Software Evolution)。主要任务是应对软件的变更,包括通过软件维护、变更和再工程等各种必要的手段使系统持久地满足用户的需求。

除了上述基本的活动外,还有一些保护性活动(Umbrella Activity),也被称为普适性活动,如配置管理、技术评审、风险管理、员工培训等。通常保护性活动与某一软件过程及这一个过程内的特定活动有关,但是又不局限于某个软件过程或其中某个活动。这类活动集自己可以构成软件过程。这些活动贯穿软件整个生命周期,以帮助软件开发团队管理和控制软件开发进度、质量、变更和风险管理等。以配置管理为例,几乎与所有软件开发活动有关,例如,需求变更需要配置管理过程支持,编码时产生的程序文件的提交和版本变更也需要配置管理过程支持。配置管理过程是一种支撑性的软件过程,其中的活动具有对其他软件工程活动的支撑作用。

典型的保护性活动如下。

(1) 项目跟踪和控制。对于计划驱动的项目,项目管理者根据计划来评估项目进度,并且采取必要的措施保证项目进度、成本和软件质量符合计划目标。

(2) 风险管理。应对可能影响项目成果或者产品质量的风险,降低风险的危害。

(3) 质量保证。确定和执行保证软件质量的活动。

(4) 技术评审。评估软件工程产品的质量,尽量在错误传播到下一个活动之前发现并清除错误。

(5) 软件度量。定义和收集过程、项目以及产品的度量,以帮助团队发布的软件能满足利益相关者的要求。度量往往与其他活动配合使用。

(6) 软件配置管理。在整个软件过程中管理变更及变更所带来的影响。

(7) 复用管理。定义工作产品复用的标准(包括软件构件),并且建立构件复用机制。

(8) 工作产品的准备和生产。包括生成产品(如建模、文档、日志、表格和列表等)所必需的活动。

理清纷杂的软件工程任务、活动和过程,可以先从通用过程框架入手,了解软件过程的各项任务和活动的内容和目标,再从通用过程模型来熟悉过程的定义,这样做可以从宏观再到微观全面地掌握软件过程的概念。过程框架定义了基本的若干框架活动,构成了实现完整的软件工程过程的基础,同时可以有选择地加入一些保护性活动。

通用软件工程过程框架包含以下 5 个活动。

(1) 沟通。在软件开发工作开始之前,与客户的沟通和协作极其重要,其目的在于理解利益相关方对项目的需求,并定义软件特性和功能。现代的软件开发中,开发者之间、开发者与用户之间的沟通所起的作用越来越重要,特别是开发者与用户的沟通是当前许多软件开发不可或缺的前提条件。

(2) 策划。策划就是为软件开发制订计划。计划驱动的软件开发依赖于明确的工作步骤,计划承担了定义和描述软件开发工作的任务和步骤,包括需要执行的技术任务、可能的风险、资源需求、工作产品和工作进度时间表等。

(3) 建模。模型能帮助软件开发人员理解和描述软件的体系结构,确立构件的特性和构件之间的关系。软件的开发过程就是模型不断演化的过程,需求模型、设计模型都是模型。软件开发人员利用模型来理解软件需求,并完成满足需求的软件设计。

(4) 构建。构建将设计模型建造为软件,是软件由模型实现为程序的过程,包括编码和测试,后者用于发现编码中的错误。

(5) 部署。软件交付给用户,在用户能够使用前,需要将软件的不同构件部署到不同计算机和设备上,然后用户才能对其进行评测,并确认软件是否满足其需求。

定义软件过程是一项艰巨的工作,涉及软件工程中的技术、技能、组织、管理,既要符合企业的商业目标和软件产品特性,还要满足用户的交付要求等。过程定义通常可以参照标准模型,如 ISO 9001、ISO 15504、CMMI 等国际标准或行业标准,过程定义超出本章范围,有兴趣的读者可访问相关的网站:①IOS 9001 和 ISO 15504 可访问国际标准化组织(International Organization for Standardization,ISO)官网;②CMMI 可访问国际信息系统审计协会(Information Systems Audit and Control Association,ISACA)官网。

3.2　经典的软件过程模型

经典的软件过程模型是多年软件工程最佳实践的总结,并在软件开发中得到广泛应用。本节介绍的经典软件过程模型有瀑布模型、V 模型、增量模型、迭代模型、原型模型、螺旋模型、演化模型、统一过程模型,这些模型的主要差别在于过程流的不同。

过程流描述了在执行顺序和时间上如何组织框架中的活动任务,执行顺序和时间的安排直接影响到里程碑、基线和交付物的设置。线性过程流,如图 3.3 所示,从沟通到部署顺序执行。此类过程流包括瀑布模型和 V 模型。迭代过程流,如图 3.4 所示,在执行下一活动前重复执行之前的一个或多个活动,包括迭代模型、原型模型、统一过程模型。演化过程流,如图 3.5 所示,采用循环的方式执行各个活动,每次循环都能产生更为完善的软件版本,如演化模型、螺旋模型、增量模型。并行过程流将一个或多个活动与其他活动并行执行,如增量模型。

图 3.3　线性过程流示意图

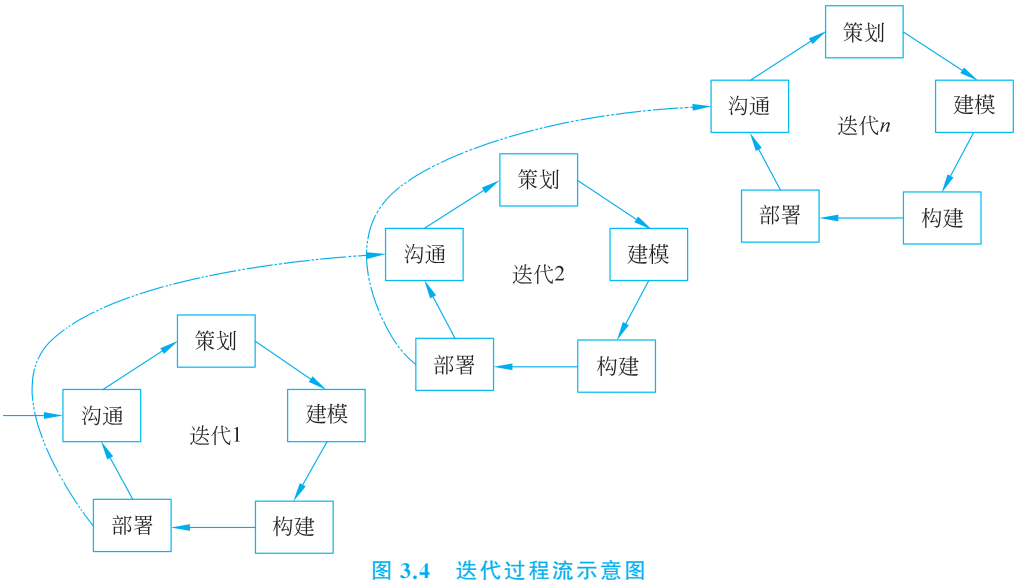


图 3.4 迭代过程流示意图

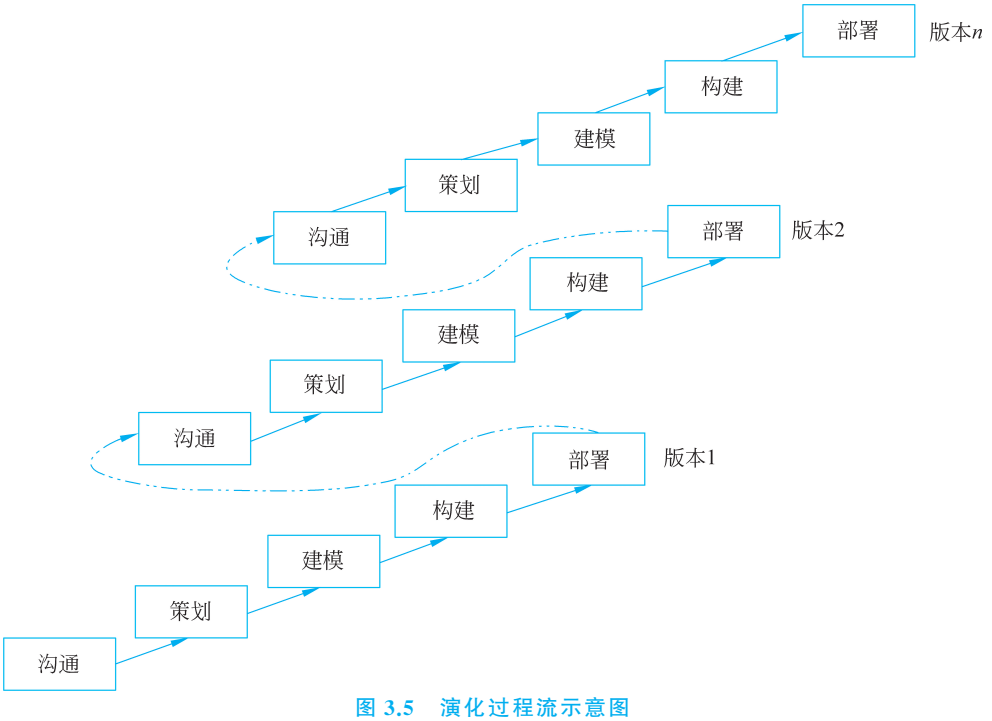


图 3.5 演化过程流示意图

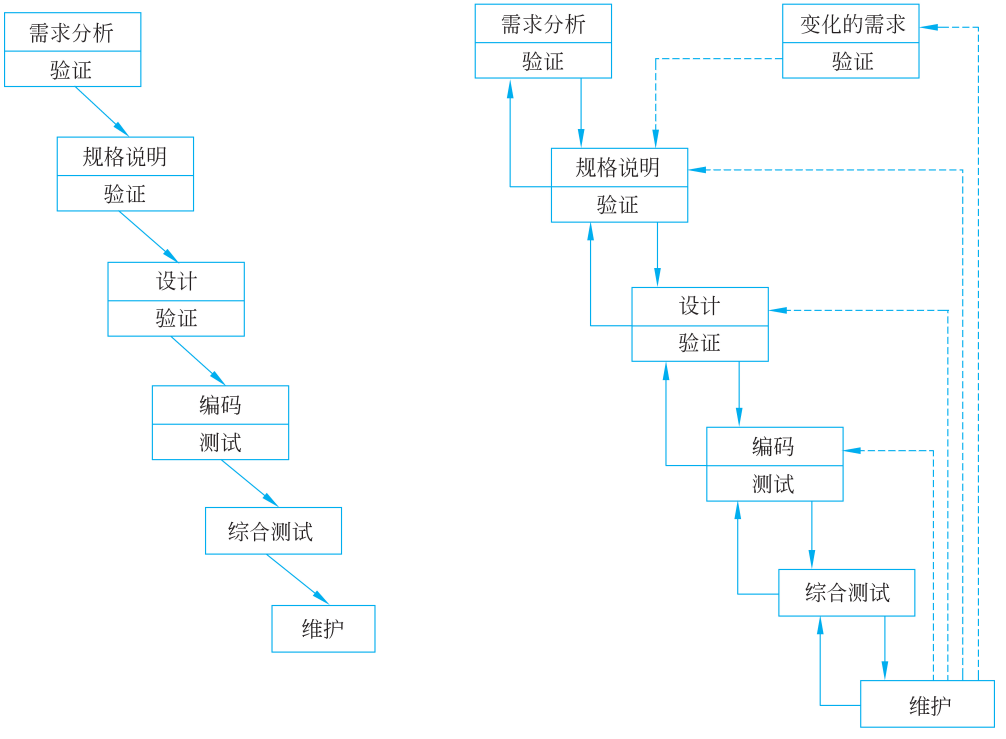
3.2.1 瀑布模型

瀑布模型，又称为经典生命周期模型，起源于 20 世纪 70 年代，是一种典型的线性过程流模型。其突出的特点是一个阶段活动完成后再开始下一个阶段活动。由于该模型从一个阶段到下一阶段形似瀑布，所以被称为“瀑布模型”。尽管当前瀑布模型的应用并不多，但是它仍然是最著名的软件过程模型。它是最早试图解决软件开发和维护这一复杂问题的，从时间维度对其进行分解和简化问题，将软件生命周期按时间依次划分为需求分析、软件设计、编程（编码）、软件测试、运行与维护等阶段（里程碑）。

瀑布模型的重要贡献是揭示了软件开发的一些特性。其一是指明需求规格、软件设计、编码与测试存在阶段间的顺序性和过程输入输出间的依赖性；其二是推迟实现，即用成本较低的文档建模活动来建立软件的模型，再用成本较高的编码活动实现软件。瀑布模型提出的需求分析-设计-编码-测试的工作流程成为其他软件过程模型的基础工作流程。推迟实现也是各种工程实践中广泛采用的策略。

瀑布模型最重要的特性是基于确定性前提，即用户需求、软件运行环境等都是可以事先精确预知的。然而，大量软件工程实践表明，这一前提在大多数工程项目中很难满足，特别是在具有长生命周期、软件规模巨大以及新兴的软件应用领域的软件系统开发中，不确定性已经成为常态。当确定性前提能够得到满足时，瀑布模型就是效果最好的软件过程模型。

如图 3.6(a)所示的瀑布模型清楚地区分逻辑分析(需求分析和规格说明)与物理设计(设计)，并尽可能地推迟程序的物理实现(编码)。每个阶段都必须完成规定的产出物，并经过质量评审(验证)，交出的产出物不合格就是没有完成该阶段的任务，要返工，也就意味不能进入下一阶段。每个阶段结束前的验证，可以尽早发现问题，改正错误，这是瀑布模型的一大优点，作用是尽量不将软件的缺陷传递到下一阶段，效果是减少下一阶段的返工，达到降低开发成本和缩短工期的目的。



(a) 瀑布模型工作步骤　　(b) 瀑布模型的返工和变更

图 3.6　瀑布模型

瀑布模型也不是绝对不允许返工，图 3.6(b)中下一阶段返回上一阶段的箭头表示返工，当发现上阶段的工作成果有问题时，可以返回上一阶段，进行修正工作。图 3.6(b)中的虚线表明在软件交付后，仍然可以返工，修改前序的工作成果。瀑布模型的返工，由于阶段之间的依赖关系往往意味着较大的工作量和较长的工时。如果返工工作量大

时，瀑布模型难以应对，意味着瀑布模型不适用。

瀑布模型的突出优点是过程可见，容易管理，项目管理者能够根据项目计划监控项目过程，减少项目的不确定性。它的缺点也显而易见，即客户通常难以清楚地描述所有的需求，对需求变更响应慢，难以及时发现系统的问题。工作任务和工程活动为线性组织方式，任务之间存在依赖性，开发团队难以并行工作，软件交付期难以缩短。确定性前提要求，导致它适用范围窄。

3.2.2 V 模型

V 模型，如图 3.7 所示，因整个开发过程构成一个字母“V”形而得名。编码构成了 V 的顶点，V 的左侧是需求分析和设计，V 的右侧是测试和运维，表达了左右两侧的关联关系。该模型是瀑布模型的一个变体，左侧大体遵照瀑布模型，在编码完成后反弹，逐阶段进行测试，强调在各个阶段进行不同的验证和确认，以提升软件质量。V 模型更明确地说明了隐藏在瀑布模型中的一些迭代和返工，V 的左右两侧关联意味着，如果在验证和确认期间发现问题，可以重新执行 V 左侧，在重新执行右侧的测试步骤之前修复和改进需求、设计和代码。V 模型的左右两侧的对应关系如下。

- 需求分析对应验收测试。
- 概要设计对应系统测试。
- 详细设计对应集成测试。
- 软件编码对应单元测试。

V 模型是瀑布模型的一种加强，提升软件质量以更多地消耗人力和时间为代价。相较于构造软件的工作，V 模型对测试工作更具有指导意义，因此，通常它也被认为是软件测试模型。V 模型的左侧，如同瀑布模型，其中的需求分析和软件设计是软件设计构造过程，但同时也伴随着质量保证活动——验证过程，通常是各种评审活动，属于静态的软件测试过程。按瀑布模型流程，在软件构造过程中，软件的程序要么不存在，要么存在但尚不完整，往往不能采用动态的软件测试方法进行测试。V 模型的右侧是对左侧结果的验证，是动态测试过程，即对需求分析和软件设计结果进行软件测试，以验证和确认软件是否满足用户需求。经过编码过程后，软件已经被构造成一个完整的系统，可以运行，此时也可以采用动态的软件测试方法进行测试。

从垂直方向看图 3.7，水平虚线上部，需求分析、验收测试和运行维护等工作主要是面向用户。水平虚线下部是技术工作，主要由软件工程师、测试工程师等技术人员完成。

从垂直方向看图 3.7，越往下面，白盒测试方法使用越多，中间部分是灰盒测试方法。在验收测试过程中，使用黑盒测试方法。

本书后续将有专门的章节讲解白盒测试和黑盒测试。

3.2.3 原型模型

20 世纪 80 年代中期，在许多软件开发的早期阶段，用户和开发者对系统的认识模糊，很难完全、准确地表达系统的需求，瀑布模型难以应用，原型模型应运而生。原型模型先借用已有软件系统作为“样品”，通过向用户提供“可视化”的原型获取用户的反馈，保证开发出的软件能够满足用户真正的需求。原型模型采用逐步求精的方法完

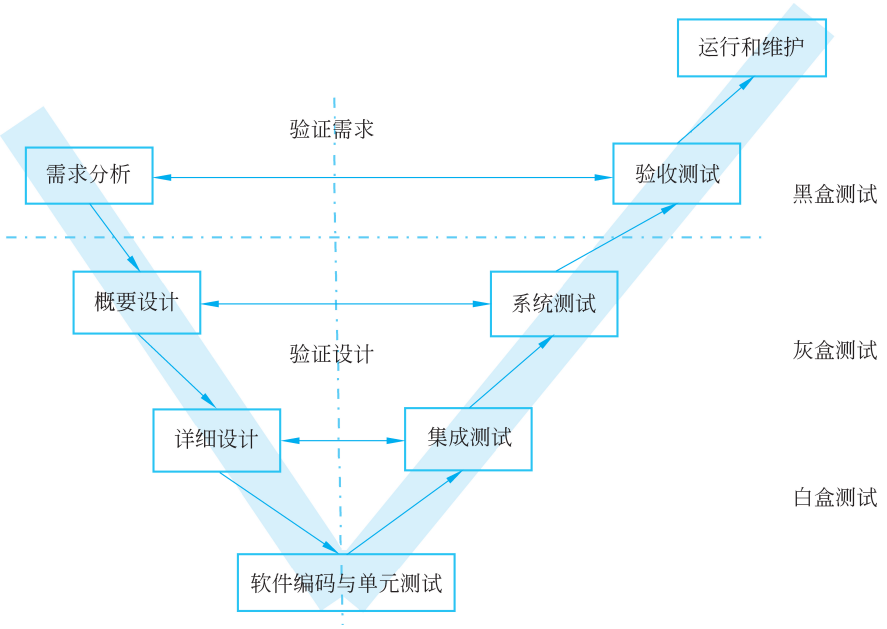


图 3.7　V 模型

善原型，使得原型能够“快速”开发，并取得用户的反馈，避免冗长的开发过程。原型模型与瀑布模型相比，原型开发过程短，可以快速地响应用户的反馈。

原型模型，是增量模型的另一种形式，是在开发真实系统之前，构造一个原型，在该原型的基础上，逐渐完成整个系统的开发工作。原型(Prototype)是指模拟某种产品的原始模型，是一个可实际运行、能够反复修改、需要不断完善的系统。软件开发中的原型是软件的一个早期可运行的版本，它反映了最终系统的重要特性。例如，开发一个打车软件，可以先设计开发基本功能，如用户注册与认证、乘客打车请求、乘客预约请求、出租车派单、司机接单/抢单等。原型系统推出后再逐步增加订单评价、定位导航、查看轨迹、沟通交流、在线支付、语音播报、实时监控等功能，还可以不断地修改或完善已有的功能。

原型开发的步骤如图 3.8 所示。

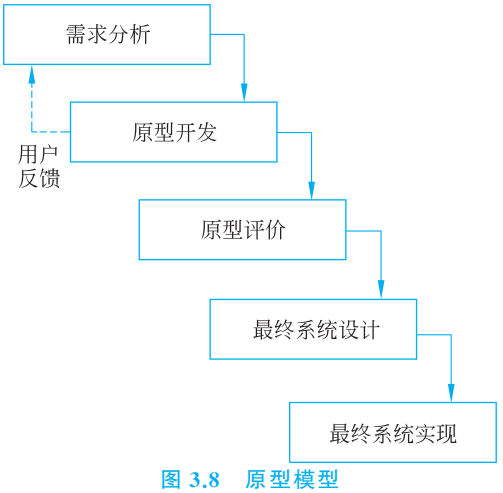


图 3.8　原型模型

首先，进行快速的需求分析，获取基本的需求。

其次，构造原型，先建立一个功能简单的原型系统；然后，进行原型评价，将原型系统展示给用户，在用户试用过程中，开发者收集用户的反馈。

最后，完善原型，通过对原型系统的反复评价、反复修改，从而逐步求精地确定各种需求的细节和变化，不断扩充完善系统的设计，最终得到较完整的软件系统。

在原型开发过程中，通过获取用户对原型的反馈来完善软件需求，经过“需求分析-原型开发”反复迭代，不断完善软件需求。

原型模型通过向用户提供原型获取用户的反馈，使开发出的软件能够真正反映用户的需求，因此，原型开发被认为是获取用户需求的一种方法，原型是为定义需求服务的。原型主要有以下三种类型。

(1) 探索型。开发原型的目的是要弄清用户对目标系统的要求，确定所期望的软件特性，并探讨多种实现方案的可行性。此类原型主要针对开发目标模糊，用户和开发者对项目都缺乏经验的情况。

(2) 实验型。原型并不是真正交付用户的软件，仅用于大规模开发和实际系统实现之前，验证方案是否合适，规格说明是否可靠。

(3) 进化型。原型是交付软件的一部分，开发原型的目的不仅是改进规格说明，而是将软件系统建造得易于变化，在改进原型的过程中，逐步将原型进化成最终系统。

探索型原型和实验型原型采用废弃策略，即先构造一个功能简单而且质量要求不高的模型系统，系统构造完成后，原型系统被废弃。废弃型策略用于验证概念，适用设计选型、发现更多的问题和可能的解决方法。进化型原型采用追加策略，先构造一个功能简单而且质量要求不高的模型系统，作为最终系统的核心。

原型模型特点突出，它的优点表现为降低软件开发初期需求不明带来的风险。它的缺点表现在增加了软件开发管理的难度；也不利于非功能需求的满足，如可维护性、性能和安全性等。

3.2.4 增量模型

增量模型(Incremental Model)，又称为渐增模型，也称为有计划的产品改进模型，它从一组给定的需求开始，通过构造一系列可执行的中间版本来实施开发活动。第一个版本(核心产品)纳入一部分需求，下一个版本纳入更多的需求，以此类推，直到系统完成。运用增量模型的软件开发过程是递增式的过程，是把待开发的软件模块化，将每个模块作为一个增量组件，从而分批次地分析、设计、编码、集成和测试这些增量组件。相对于瀑布模型而言，采用增量模型进行开发，开发人员不需要一次性地把整个软件产品提交给用户，而是分批次提交，交付的软件系统是集成一系列中间版本的成果。

增量模型是瀑布模型和原型模型的综合，在整体上按照瀑布模型的流程实施项目开发，即每个增量按瀑布模型的线性序列进行开发，每一个线性序列生产软件的一个可发布的“增量”，以方便对项目的管理。为加快软件的开发速度，缩短交付期，不同的增量可以并行推进，情形如图 3.9 所示，“增量 1”可以与“增量 2”在时间上存在重叠，也就是说，增量 1 还没有交付，增量 2 已经开始，当然，增量 1 与增量 2 也可以是串行的。增量模型是将软件系统按功能分解为若干增量构件，并以构件为单位逐个地创建

与交付，直到全部增量构件创建完毕，并都被集成到系统之中交付用户使用。

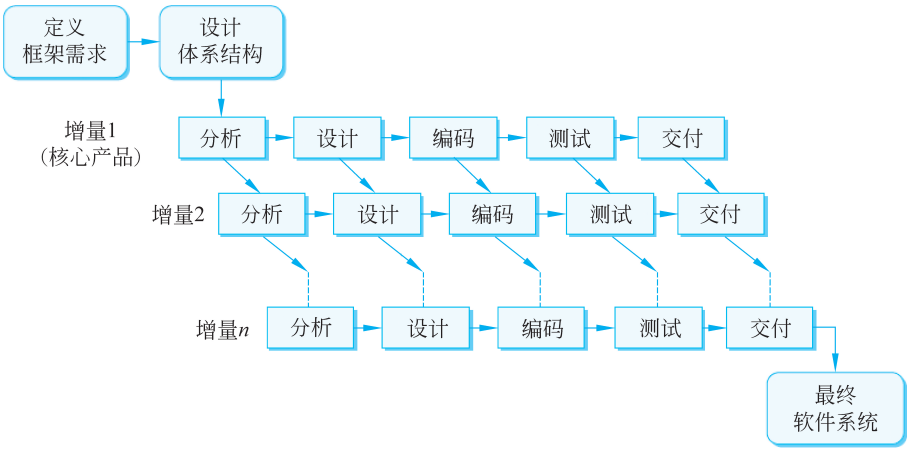


图 3.9　增量模型

增量模型适于不能提前制定出完整的问题解决方案的情况，采用摸着石头过河的方式，逐步逼近结果，本质上是迭代。同时，这种方法也体现了与时俱进的思想，可以较好地应对需求变更。增量模型分批地向用户提交产品，付出的代价是它的软件体系结构必须是开放的，适合软件的扩展并在较短期内交付产品。其强调每一个增量均发布一个可操作产品，早期的增量是最终产品的“可拆卸”版本，但提供了为用户服务的功能，并且为用户提供了评估的平台。把软件产品分解成增量构件时，唯一必须遵守的约束条件是，当把新构件集成到现有构件中时，所形成的产品必须是可测试的。

增量模型在项目管理和开发技术上都有较高的难度，使用时需要关注以下问题。

- 良好的可扩展性软件架构设计，是增量开发成功的基础。
- 由于一些模块必须在另一个模块之前完成，所以必须定义良好的接口。
- 与完整系统相比，增量方式对软件的评审难度更大，所以必须定义可行的过程。
- 要避免把难题往后推，首先完成的应该是高风险和重要的部分。
- 客户必须认识到总体成本不会更低。
- 分析阶段采用总体目标而不是完整的需求定义，可能不利于项目管理。
- 需要良好的计划和设计，管理必须注意动态分配工作，技术人员必须注意相关因素的变化。

增量模型的优缺点都很突出，相比于瀑布模型，主要优点有增量模型降低了应对需求变更的成本，能更经济、更容易、更快速地变更软件。由于更快地交付和部署，能够在较短时间内得到问题反馈，同时用户可以更早地使用软件并创造价值。其缺点也显而易见，缺乏整体规划，开发过程复杂，管理难度大，越往后变更越困难，导致功能堆砌，开发成本逐渐上升。

3.2.5　迭代模型

早在 20 世纪 50 年代末期，迭代模型（Iterative Model）就已出现在软件领域，那时软件工程还未诞生。最早的迭代模型被描述为分段模型（Stagewise Model），它的兴起得益于 RUP（统一过程模型）的推荐。瀑布模型是最早得到广泛应用的软件开发模

型，但是，瀑布模型难以有效应对软件规模越来越大、交付时间越来越短等情况。使用瀑布模型完成软件开发存在不可控的需求风险及交付周期压力。在某种程度上，迭代模型是瀑布模型的缩小与循环，每次迭代都完整地经历需求分析、设计、实施和测试工作流程，类似小型的瀑布式项目。只是迭代模型是重复相同的类似瀑布模型的开发过程，见图 3.10，每次的迭代都会产生一个可以发布的软件版本，这个软件版本是最终软件产品的一个子集。

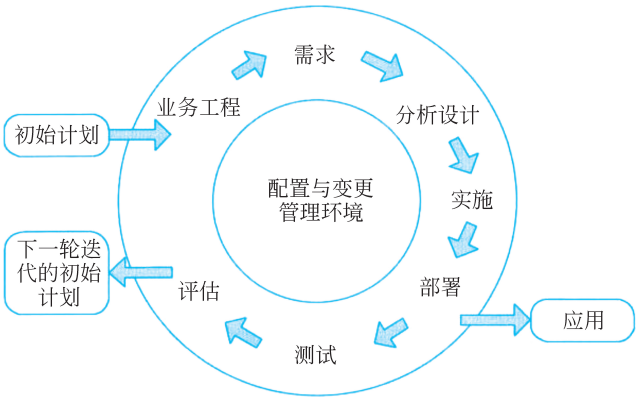


图 3.10 迭代模型

迭代策略与原型和增量策略不同，它是一种演进的思想，认为软件需求是可以逐渐逼近的，当用户需求不充分、需求变更较多，并且风险高时，可以通过迭代的演进方式达成目标。原型方法的核心思想是通过原型开发来获取用户的需求。增量方法是为了快速交付软件，而限制一次性开发庞大软件，通过增量累加逐步完成大型软件。而迭代有原型的思想，即每个迭代周期交付的软件可以是原型；也有增量的思想，每个迭代周期交付的软件都是上一个迭代周期的增量。

如图 3.10 所示的迭代模型示意图，形象地表明采用迭代模型进行软件开发就像一个不断向前转动的车轮，一步一步地逼近目标，车轮每转动一圈的过程是相同的，但车轮所经过的道路却不同，车子距离目标的距离也不同。车轮转动比作迭代周期，每个迭代周期都经历相同的软件开发活动，如图 3.10 中的业务工程、需求、分析设计、实施、部署、测试、评估等，每个迭代周期的结果通常是一个可运行的软件版本，甚至是可交付给用户的版本。当然，迭代模型也如同车子，也可能有后退的时候，这往往意味着有重要大需求变更，或者开发的软件版本有重大缺陷，需要回退重来。

迭代模型的优点是用户反馈时间短，每个迭代周期的工作成果可以快速地交付给用户，因此可以快速地获得用户使用效果的反馈；演进式推进可以降低软件项目的风险，在推进的过程中，可以结合上一个迭代周期的用户反馈来调整需求并变动部分功能/业务逻辑，再开始新一轮的迭代；阶段性的功能调整及快速质量反馈，使得开发人员清楚软件的功能定位和问题焦点，少走弯路，减少返工；当用户需求不能在项目一开始就做出完整和准确的定义时，迭代过程能够在后续的迭代周期中不断细化和完善，以更好地适应需求的变化。

采用迭代模型的项目，往往需求变更频繁，易导致软件体系结构频繁变动，对项目的管理、软件的质量控制都有非常高的要求，对软件开发团队的软件工程能力也有非常高的要求。

3.2.6　螺旋模型

螺旋模型(Spiral Model)在 1988 年提出。它是一种为应对大型、复杂、高风险软件开发项目,将原型模型的迭代和瀑布模型的系统性和严格监控结合起来的一种演进式过程模型。它具有风险驱动的特性,将开发活动与风险管理结合起来,提供了一种中止项目的止损机制。它采用迭代的方式逐步完善系统定义和推进系统的实现,以降低风险,即每次迭代都不是在原水平上进行,是对整个开发过程进行迭代,而不仅仅是对编码、测试进行迭代。

螺旋模型,如图 3.11 所示,采用迭代的策略,形似一圈套一圈的螺旋线,沿螺旋线自内向外,每旋转一圈就是一个迭代周期,每个迭代周期开发出一个新的软件版本(或新版本的原型)。从小规模开始,进行风险分析,制定风险控制计划,由此确定下一步是否还要继续项目,开始下一个螺旋的迭代。螺旋模型的一次迭代,包括制订计划、风险分析、实施工程、客户评估这 4 个阶段(如图 3.11 所示的 4 个象限)。

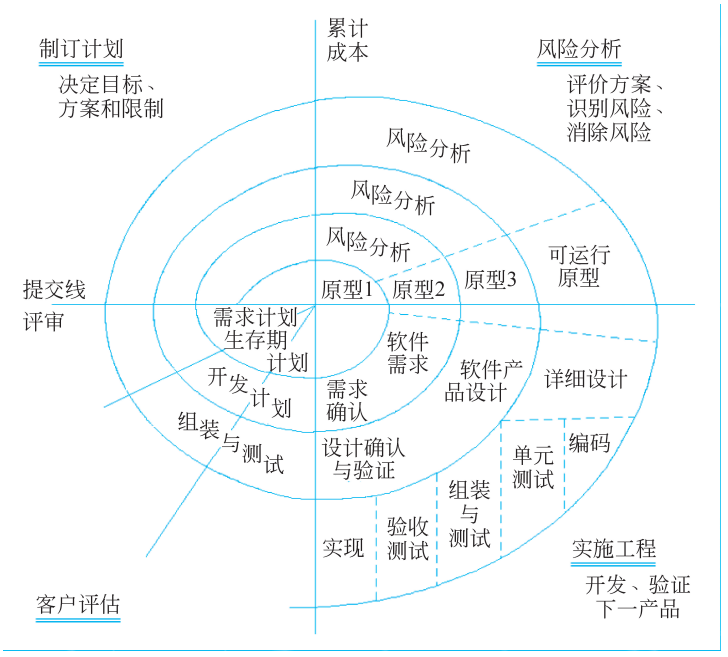


图 3.11　螺旋模型

第 1 象限:制订计划,该阶段要完成确定软件目标、选定实施方案、弄清项目开发的限制条件。

第 2 象限:风险分析,该阶段的工作有分析所选方案、识别和消除风险。

第 3 象限:实施工程,该阶段进行软件开发、验证工作。

第 4 象限:客户评估,该阶段的工作包括评价软件功能和性能,提出修正建议。

螺旋模型的突出优点体现在风险管理,能够有效降低项目不确定性带来的风险;一个迭代周期为一个关键的里程碑,确保利益相关方在一个迭代周期内都能支持项目推进。缺点为实践中很难说服客户以合同形式合作,并且需要依赖大量风险评估专家做风险评估工作。

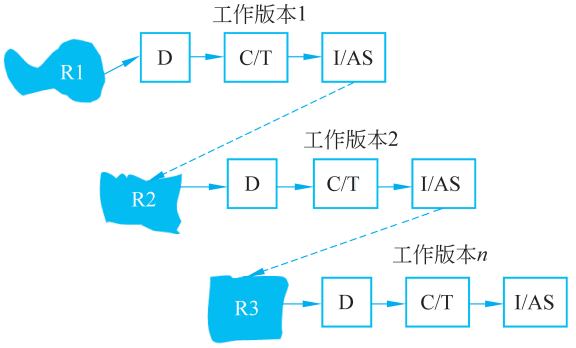
在开发模型中加入风险管理,表面上看是加强风险管控,但是实际上现代的软件

开发都是在现代项目管理管控下进行的软件开发，而现代项目管理都要求在里程碑处或有重要事件发生时进行风险分析。风险管理更偏向于管理人员承担的工作，不是软件开发人员擅长的工作。因此，螺旋模型的风险分析的作用有限，与其他模型相比优势并不是特别突出。

3.2.7 演化模型

演化模型(Evolutionary Model)利用演化的方法，渐进式地推进软件开发。在此模型指导作用下，软件经过一段时间的演化，逐步完善，达到满足用户需求的目的。此模型主要针对需求不够清晰，且常常变化，加之交付期要求紧迫的软件开发项目。它的基本思想是承认并未完全理解用户的需求，因而对于需求的理解将在后继演化(精化阶段)中不断完善。因此先提交一个有限的版本，该版本的目标只是在探索可行性，弄清软件的需求，细节部分可以在后续的开发中定义和完善。在整个软件的开发过程中采取分批循环演化开发的策略，每演化一次开发或完善一部分功能，演化的成果成为这个软件产品原型的新增功能。

演化模型的演进方式如图 3.12 所示，第一次演化(需求→设计→实现→测试→集成)→反馈→第二次演化(需求→设计→实现→测试→集成)→反馈→……实际上，这个模型可看作是重复执行了多个“瀑布模型”，每一次演化就是执行了一次瀑布模型。



R：需求 D：设计 C/T：编码/测试 I/AS：安装和验收支持

图 3.12 演化模型

第一次演化，根据用户的基本需求，通过快速分析构造出该软件的一个初始可运行版本，这个初始的软件就是原型系统；第二次演化，根据用户在使用原型的过程中提出的意见和建议对原型进行改进，获得原型的新版本，为整个系统增加了一个可定义的、可管理的子集。重复这一过程，可直至软件生命周期终结。而随着软件变更的发生，其软件的结构逐渐变得更加复杂，需要额外的资源来保持和简化。

演化模型由于对待原型的策略不同可分为探索式演化模型和抛弃式演化模型，关于探索式原型和演化式原型可参见原型模型(3.2.3 节)。

演化模型为演化范型，与瀑布模型或者线性顺序范型模型的区别是，演化模型认为需求很难调研充分，现实环境中的软件必须进行变更，否则将逐渐失去在相应环境中的作用，所以很难一次性开发成功。因此演化模型提倡两次开发，第一次为试验开发，成果为试验性的原型产品，其目的只是在探索可行性，弄清软件需求；第二次是在第一次的基础上获得较为满意的软件产品。演化模型应用的前提是开发人员有能力

把软件需求分解为不同组，以便分批演化开发。这种分组不是随意的，而是根据功能的重要性及对总体目标的要求而做出的决策。有经验表明，每个开发循环周期时长以6～8 周为宜。

演化模型的优点表现为原型演化的开发方法可以短期内见到可运行软件，并尽早地进入软件测试以验证软件是否满足需求；快速交付给用户的版本，有助于引出高质量的软件需求，并获得快速的应用反馈，为下一次演化打下好的基础；演化范式为配置管理提供了清晰的里程碑，也为风险管理提供了风险分析的软件产品，为下一次演化的继续或取消提供了决策数据。

演化模型的缺点在于需要开发团队要有强大的过程能力，否则模型退化为一种原始的无计划的“试-错-改”模式；软件需求在初期如果不完整的话，势必影响软件的总体设计，这可能削弱软件设计的完整性，并因此影响软件的性能、可靠性和可维护性等。

3.2.8　统一过程模型

统一过程是由 Rational 软件公司创立的软件工程方法，通常被简称为 RUP (Rational Unified Process)，或者 UP(Unified Process)。UP 是一种以用例驱动、以体系结构为核心、迭代及增量的软件过程模型，由 UML 方法和工具支持，广泛应用于各类面向对象的软件开发项目。

1. RUP 迭代周期

RUP 是迭代和增量结合的过程，如图 3.13 所示，每个迭代周期分为 5 个阶段：起始、细化、构建、转移和发布。

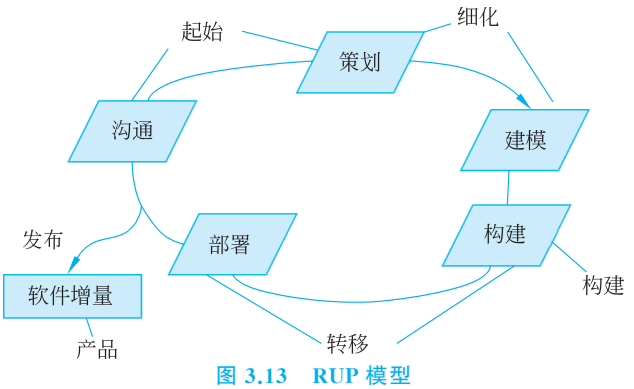


图 3.13　RUP 模型

1）起始阶段

起始阶段的任务是为系统建立用例模型，并确定系统的边界，为此必须识别所有与系统交互的外部实体，在较高层次上定义交互的特性。主要活动包括用户沟通和计划两个方面，强调定义和细化用例，并将其作为主要模型。起始阶段结束为第一个重要的里程碑，完成了系统目标、范围和外部交互特性的定义。

2）细化阶段

细化阶段的任务是分析问题，重点是建立需求分析模型和设计模型，包括类的定义和体系结构的表示，编制项目计划，淘汰项目中最高风险元素。为达到此目的，必须确定系统的范围、主要功能和非功能需求，建立系统的体系结构。同时为项目建立支持环境和准备工具。细化阶段结束为第二个重要的里程碑，要评估系统的目标和范

围、体系结构，以及主要风险的管理方法。

3）构建阶段

构建阶段的任务是将设计转换为实现（编码），并进行集成和测试。此阶段是一个软件实现的过程，其目标是将系统构建出来，工作重点是资源、成本、进度和质量的管理。构建阶段结束是第三个重要的里程碑，即初始功能里程碑，此里程碑的主要提交物为可以在测试环境中部署的软件版本，也常被称为 β 版。

4）转移阶段

转移阶段的任务是将产品交付给用户，由用户进行测试评价。重点是确保软件对最终用户是可用的。用户反馈应主要集中在软件的优化、设置、安装和可用性问题上，所有主要的结构问题应该已经在项目生命周期的早期阶段得到解决。在转移阶段的终点是第四个里程碑，即产品发布里程碑。此时，要确定目标是否实现，是否应该开始下一个迭代周期。

5）发布阶段

在发布阶段，已完成的所有人工制品即所有的程序和文档都已完成并交付。软件开发告一段落，后续可以启动新的迭代周期继续软件的增量开发，或者进入维护期。当然，也有一些软件，如提供信息服务为目的的软件系统，一直在前四阶段迭代循环直到软件的生命周期结束。

2. RUP 软件开发过程主要特征

RUP 是一种重量级过程，除过程模型外，还包括大量优秀的实践方法，如迭代式软件开发、需求管理、基于构件的构架应用、建立可视化的软件模型、软件质量验证、软件变更控制等。RUP 软件开发过程的主要特征是：用例驱动、以体系结构为中心、迭代和增量的软件过程。简单介绍如下。

1）用例驱动

用例驱动指通过用例获取软件系统的功能需求，由用例驱动需求分析之后的所有阶段的开发。用例（Use Case）是系统应对外界请求的描述，是通过用户的使用场景来获取需求的技术。在 RUP 中指利用 UML 用例图展示用户与系统交互的一种需求分析方法。有关用例的内容在本书后续章节将有详细的介绍。

2）以体系结构为中心

以体系结构为中心指在项目早期定义一个基础的软件体系结构，然后将它原型化并加以评估，最后进行精化。一个好的体系结构涉及功能和非功能两个方面，对定义一个易修改、易理解和允许重用的系统尤其重要。创建软件体系的一个重要工作是逻辑上把系统划分成子系统，UML 中的包图等可支持体系结构设计。

3）迭代和增量的软件过程

迭代和增量的软件过程指整个软件开发过程由多个迭代周期组成，如图 3.14 所示，在每次迭代中只考虑系统的一部分需求，针对这部分需求进行分析、设计、实现、测试和部署等工作，每次迭代都是在系统已完成部分的基础上进行的增量，每个增量系统都能够增加一些新的特性，如此循环往复地进行下去，直至完成最终项目。

RUP 是以面向对象方法为基础的方法学，在业务建模、需求分析、设计、实现、测试等各个过程中始终贯穿面向对象方法，因此，RUP 更适合面向对象类项目。RUP 是一种适应性软件过程，区别于瀑布模型类的预测性软件过程，它不假设从一开始就

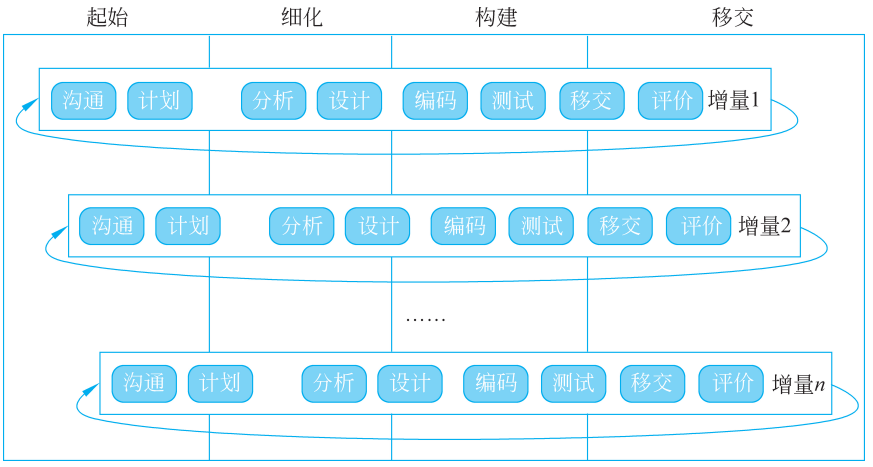


图 3.14　迭代和增量的软件过程

可以掌握软件开发的全过程，而是坚持以迭代方式推进软件开发，结合不断演进的项目状态和现实变化做出相应的调整，制订出新的计划。

RUP 作为一种重方法，定义了 9 个工作流，其中 6 个是针对软件开发的核心过程工作流，3 个是支持核心工作流；总结了经过多年商业化验证的 6 条最佳实践，这些内容本节不做介绍，读者可参看推荐阅读文献。

RUP 的优点是全面兼顾阶段、风险、工作流、质量监控、项目管理等方面，具备迭代开发的灵活性、增量开发的响应速度、基于构件开发的成本优势、面向对象开发的严谨。其缺点是模型庞大、涉及多种软件工程的方法学、掌握难度较大。

3.3　软件过程模型的选用

选择软件过程模型实质上是选择软件开发的策略，是为软件开发的主要活动提供一套范型，使工程的进展有章可循。软件过程模型来自于软件工程长期发展的经验总结，经典的软件过程模型都经过了几十年的工程实践的检验。由于实际的工程项目千差万别，软件过程模型的选择并不是一件简单的事情。而且，许多工程实践中也不完全套用某个模型，通常会根据项目的特点综合运用模型。

选择模型的总目标是有利于项目的实施，降低软件开发的风险。需要从多个维度进行评估，评估维度一般包括需求规模和清晰度、用户参与程度、软件应用行业的特点、企业的商业目标、开发团队的架构设计能力、实现软件的技术要求、项目管理要求、软件交付期压力、开发团队的软件工程能力等。如果软件开发项目的不确定因素多，往往就要选择迭代范型，如迭代、演化和原型类的模型。如果需求明确，开发时间充裕，往往选择线性顺序范型，如瀑布、增量。而如果交付期压力大时，通常选择交付短且有并行开发特性的模型，如增量、迭代类的模型。表 3.2 列出了选择软件过程模型时需要考虑的主要因素，各个模型与项目特点的关系并不完全如表中所示，表中只是一般情况下选择模型时要考虑的模型特性，而且其中没有定量指标，只是一些定性描述，并不严格。比如软件复杂度的高、中、低划分并没有明确的标准，软件工程至今也没有给出可操作的量化标准体系。还有对开发团队的经验和技能要求也不明确，如瀑布模型对团队要求相对较低，更能容忍团队成员技术上的明显差异，但通常只要求有

一定数量的高技能成员。而迭代模型一般要求团队成员的技能水平普遍较高。

表 3.2 选择软件过程模型时需要考虑的主要因素

项目特点	瀑布	增量	原型	螺旋	演化	迭代	统一过程
行业特点	成熟领域	所有行业	新兴行业	所有行业	所有行业	新兴行业或研发型软件	所有行业
需求清晰度	清晰	较清晰	不清晰	不清晰	不清晰	不清晰	不清晰
特性要求(安全性、可靠性等)	高	高、中、低	中、低	高、中	中、低	中、低	中、低
软件复杂度	中、低	高、中	中、低	高、中	中、低	高、中、低	高、中、低
交付期压力	低	高	高、中	低	低	高、中	高、中
技术要求	成熟技术	少量新技术	新技术	新技术	成熟或新技术	新技术	成熟或新技术
项目风险	中、低	中、低	高、中	高	中、低	高、中	高、中
团队经验	不要求	不要求	有相似项目的经验	熟悉相似项目	熟悉相似项目	不要求	不要求
架构重构能力	成熟架构	成熟架构	架构重构	成熟架构	架构重构	架构重构	架构重构
管理要求	基线控制	版本规划	沟通和激励，原型验证	风险管理	过程控制	迭代周期控制	过程控制
用户参与度	低	低	必须参与	必须参与	中	高、中	高、中

3.3.1 软件过程模型选择示例

以开发一个面向公众和广大出租车司机的打车软件系统为例，选择一个适当的软件过程模型来开发这个打车软件系统。

1. 打车软件系统的特性描述

系统的目标是在乘客与出租车司机间搭建打车业务的交流平台，增强信息的透明度，提高预定出租车的准时率，降低出租车的空驶率，提高出租车的利用率。

系统的工作模式如图 3.15 所示，与打车软件系统交互的人或其他系统有乘客、司机、第三方支付公司和第三方地图提供商。乘客打车，司机驾驶出租车为打车乘客提供接送服务，第三方支付公司为打车系统提供打车费用的电子支票服务，第三方地图公司为打车系统提供打车导航和打车起点、终点定位服务。

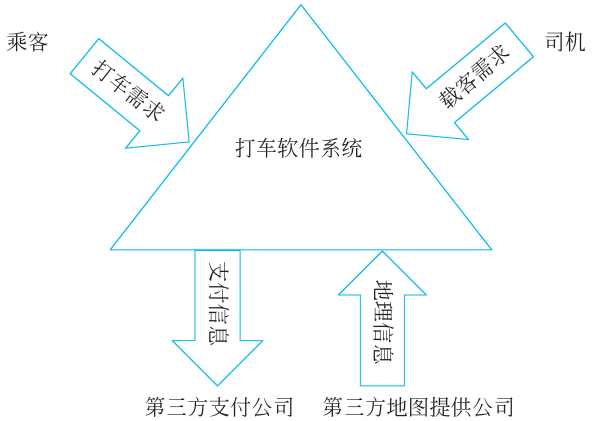


图 3.15 打车软件系统工作模式

2. 打车软件系统的基本功能

1）注册与验证

乘客和司机进行注册和验证操作，具体包括注册用户名、用户信息、手机号、车辆信息等相关信息，之后用手机发送短信，经过验证后即可成为系统用户。

2）乘客请求模块

现时请求：乘客通过此系统发出叫车请求，给出相应的乘车地点及目的地，当时即要求系统做出回应，给出可供选择的出租车。

3）预约请求

乘客通过此系统发出叫车请求，给出相应的乘车时间、乘车地点及目的地，要求系统在预定的时间做出回应，给出可供选择的出租车。

4）出租车司机抢单

在一定范围内的出租车司机都可以获得用户叫车请求，司机可抢单。

5）评价

评价打车乘客和司机的信用，评价司机的准确率和服务质量。

6）定位

提供打车乘客和出租车定位服务。

7）支付

司机提供电子账单，用户可以电子支付。

8）多种运行平台

系统可以运行在多种硬件和软件平台之上，包括手机等移动设备。

3. 开发打车系统选用增量模型

选择模型需要的一些评估因素在系统特性描述中并没有明确给出，这需要通过合理的假设加以解决，如系统开发时间、交付期限、开发团队的工程能力，以及打车软件系统是自研还是委托开发、企业的财务状况和企业商业目标等。

根据项目的特性、各个模型的特点，以及必要的假设，开发打车系统采用增量模型是一个好的选择，以下仅对选择增量模型的理由做简要说明。

1）系统需求明确

(1) 打车软件历经多年的发展，在国内外都有许多大型、成熟的系统在运行，这为新开发同类系统，提供了借鉴和样板。

(2) 针对乘客与出租车司机之间提供打车业务的交流平台的需求，通过系统特性描述以及现有同类系统作为样板，软件需求较为完整，具有明确的定义。

2）技术要求

(1) 实现打车软件系统的主要技术都为成熟技术，但用户规模可能很大。当并发用户数量多时，对软件性能要求很高。系统构架可以选择成熟技术，一般情况下，不需要重构系统构架。

(2) 有经验的团队几乎对所有软件开发项目都是大有益处的，团队软件工程能力强的话对软件质量和按时交付都是强有力的保障。

3）交付期压力

(1) 如果企业希望打车系统尽快面市，可以将系统功能划分为不同的层次，分阶段投入市场，每个阶段专注于特定功能，从而更快地完成阶段性的开发目标。