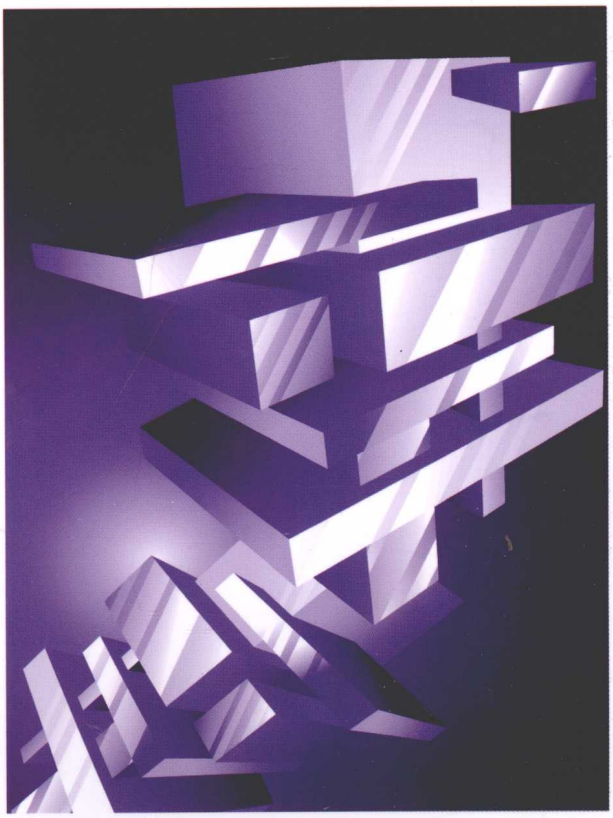
高等学校计算机应用规划教材

软件工程

- 软件危机与开发流程
- 需求工程
- 面向对象软件开发
- 软件设计
- 软件项目计划与管理
- 软件测试
- 软件质量管理与保证
- 软件配置管理
- 软件形式方法论
- 软件过程改进



李允中　著

Mc Graw Hill Education

清华大学出版社

高等学校计算机应用规划教材

软件工程

李允中 著

清华大学出版社

北京

内 容 简 介

本书由台湾软件工程领域领军人物李允中教授执笔，结合软件产业与当前教育，较为全面地介绍了软件工程的重要概念和专业知识，培养学生在理论及应用上的系统整合能力，从系统的角度来看待整个项目。全书共 10 章，内容涵盖软件危机及软件开发流程、需求工程、面向对象软件开发、软件设计、软件项目的计划与管理、软件的配置与管理、软件工程的数学理论基础、软件过程改进等。

本书内容丰富，讲解清晰、易懂，并提供真实的项目案例，帮助读者领悟真实的项目开发的困难和过程，从而意识到软件工程的好处。本书适合作为高等院校计算机专业的教材，也可供项目管理及开发人员参考。

李允中
軟體工程

图书在版编目(CIP)数据

软件工程/李允中 著. —北京：清华大学出版社，2010.9
(高等学校计算机应用规划教材)
ISBN 978-7-302-22845-5
Ⅰ. 软… Ⅱ. 李… Ⅲ. 软件工程—高等学校—教材 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字(2010)第 097329 号

责任编辑：王 军 李维杰
装帧设计：孔祥丰
责任校对：胡雁翎
责任印制：王秀菊
出版发行：清华大学出版社 **地 址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京鑫丰华彩印有限公司
装 订 者：三河市新茂装订有限公司
经 销：全国新华书店
开 本：185×260 **印 张**：16.25 **字 数**：375 千字
版 次：2010 年 9 月第 1 版 **印 次**：2010 年 9 月第1 次印刷
印 数：1～4000
定 价：35.00 元

产品编号：035067-01

作者简介

李允中教授(Dr. Jonathan Lee)，1993 年毕业于美国 Texas A&M 大学计算机科学系并获博士学位，同年受聘任教于中央大学信息工程系，目前任信息工程系教授，兼计算机中心主任。李教授曾于 1999～2002 年担任信息工程系主任，2003～2006 年担任软件研究中心主任。在大学任教期间，李教授积极献身于教学与人才培育，同时在国际学术界具有卓越的研究表现，获得多个学术奖项与荣誉称号。

在学术成就之外，李教授致力于提倡软件工程教育，成立跨院校的软件工程联盟，规划适合高等院校的软件工程课程设置并编纂课程教材，联合 100 多个系共同促进推广。李教授还特别关注软件产业的发展与地位提升，多年来在各领域竭力提倡软件工程跳出传统学术框架，结合产业与教育，试图为整个软件产业注入新生命并开创一条新路。

序

作者编写本书的目的在于提供完整的软件工程思想及范例，引导读者跳出以组件方式来看软件，而改以系统的方式来看待整个软件项目。

本书内容共包含 10 章。第 1 章介绍软件过程，就像算法一样，合适的过程可以提高系统开发的效率，不合适的过程则会延宕项目的进行。组织或开发团队必须依照自己的特性去设计自己的过程，本章所提仅是大架构上的过程模式。目前在全世界所推行的 CMMI 过程改进方法，即是通过定义合适的过程来改善软件开发的效率与产品的质量，让软件开发过程与企业文化相融合。

第 2 章认识需求工程。需求工程是软件开发的第一步，是后续设计、开发、实现的基础。如果需求定义上出了问题，即使有好的设计与实现能力，也终究会产出不是使用者期待的产品或系统。然而，要建立正确的需求并不容易，它需要来自各种不同观点的分析方法、沟通技巧与专业的领域知识。除此之外，需求的管理更是一门学问，许多项目因为没有妥善地管理需求的变化，导致系统不断地发散扩张而无法完成。本章的主题在于阐述需求撷取、建立与管理的方法和技巧。

第 3 章深入了解面向对象软件开发，面向对象软件开发方法的主要概念，是将真实世界以对象封装思想进行建模。其包含四个阶段：企业建模、面向对象分析、面向对象设计、面向对象实现。企业建模是针对用户需求加以分析，建立系统的用例模型与领域模型。在面向对象分析阶段，即针对每个用例进行分析，利用互动图描述对象之间信息交换，以完成用例所需求的功能。在面向对象设计阶段，则依据分析阶段的结果，再加以细部设计转换成具体的软件系统模型。当细部设计完成后，程序设计人员便可以建立的设计图为蓝本来实现面向对象的软件系统。

第 4 章阐述软件设计的基本概念，从软件设计概念、软件设计策略与方法、软件设计规格书撰写，以及进阶设计概念等方面来描述软件设计的相关知识，希望读者在学习软件设计的技术知识时，能够同时受到工程化思维方式的训练。主要内容包含架构设计、接口设计、数据结构设计以及算法设计。本章强调软件系统分析与系统设计的分工与对应，尤其着重于软件架构设计对于大型软件系统开发的重要性。

第 5 章主要探讨软件项目的计划与管理。如同软件系统开发，为了能更有效地执行软件项目，软件项目的计划与管理引进了过程(Process)的概念，软件项目管理过程配合软件项目管理计划，对于项目的执行将有莫大的帮助。软件项目的生命周期大致可分成初始(Initial)、计划(Planning)、执行(Execution)与结束(Closure)四个阶段。软件项目过程的工作，必须考虑项目是否应该进行、如何开始进行，以及有哪些项目应列入考虑，进而形成一份计划书，然后根据这份计划书进行项目的执行、管理与监控，直至该软件项目完成。项目管理工作执行的切实与否，对软件项目开发过程能否顺利进行具有相当重要的影响。

第 6 章谈软件测试。在软件开发过程中，可能因为人为疏失、沟通不畅而造成规格不符、设计错误或程序编写疏漏等问题，这些问题不仅延误软件开发进度，更有可能导致软件开发成本的增加，甚至严重影响软件的质量。为了找出因上述各种因素造成的软件错误、瑕疵与损害，需要软件测试的相关技术来保证软件系统的质量。

在对软件过程、开发与管理有了基本认识之后，第 7 章将软件质量的概念引了进来。在我们的生活中，有许多因软件质量管理缺失而导致的非预期事件发生。例如，与铁路、公路相关的售票系统，曾发生售票重号的情况，这样不仅对旅客购票造成很大的困扰，也使得企业的形象受损。由此可见质量的管控对软件项目的重要性。本章讲解软件质量概念、质量管理工作内容，并介绍提升软件质量的方法。

第 8 章介绍软件配置管理及其工作内容。在软件开发中，管理软件版本、建立软件基准与控制软件变更是不可或缺的。有太多的软件项目因为软件变更的管控不当而导致重大损失，甚至是整个项目的失败。若能真正落实配置识别、配置控制、配置状态报告与配置审核等配置管理工作，则变更所造成的影响与损失将可得到妥善控制。

第 9 章将数学理论与软件工程方法相结合，称为软件形式方法论。在许多传统的工程领域中，如电子、电机、土木、机械等，其所发展出来的任何方法论都有数学理论作为基础，数学分析在这些产品设计方法中，也成为开发过程中基本的一环。但在软件工程领域中并非如此普遍，即使软件工程的数学理论研究发展至今已有近三十年的历史，却也仍然没有被广泛地应用到实际的软件开发方法论中。本章将讨论以数学为基础的软件形式方法论在软件开发中的应用，同时还将探讨所面临的挑战。整体而言，软件形式方法是值得软件开发人员学习的主题，对于提供高质量的软件有相当大的帮助。本章的主题着重探讨软件工程与形式方法论的关系，以及如何通过形式方法的应用，提高所开发软件的质量与可靠性。

本书的第 10 章将带领读者认识对软件开发具有重要影响的软件过程改进方法，尤其针对美国卡内基梅隆大学软件工程研究所研发出来的能力成熟度整合模型集成(CMMI)进行了概要式的探讨，让读者可以了解以 CMMI 模型为基础的过程改进方法。基于本书前述章节的探讨分析，相信读者已经十分了解软件本身有着本质上的问题及困难，而软件工程的各种工程层面及管理层面的各项方法，便是为了能够逐步克服与减缓这些软件与生俱来的复杂性、易变性、隐藏性及一致性等问题，进而通过系统性的软件过程改进，让开发团队能循序渐进、阶段性地学习如何与软件共舞。

本书最后以一个软件开发的真实案例，引导读者体会真实的软件开发的困难与过程，了解需求管理是如何进行的，并希望通过此案例的安排，让读者能在理论与实际的对照中感受到软件工程及其相关的工程方法所带来的实质好处。至于各章节中的相关关键词汇，本书统一整理汇编成附录 B“词汇对照表”并附于书后，以方便初步接触软件工程的读者对照参考，帮助理解书中的基本概念与内容。

最后，我希望能对下列人士表达由衷的感谢。感谢郭忠义、薛念林、潘健一、赖联福、徐国勋、范姜永益、吴佳磷、李文廷、王跃强、邓焕友、李信杰、Carol Lai、Maggie Wu、Ariel Chou、Ariel Hsu、Stella Chang、Merry Chang 人等为完成本书所做的贡献。

李允中

目　录

第1章　软件危机与开发过程

软件产业发展至今已有数十个年头，其间经历了不断的技术创新和开发过程的改进，使得该项产业愈加重要，并成为日常生活中其他产业的重要运作核心。不论是金融、航空、制造、医疗，还是运输等行业，都实现了一定程度的信息化，并使用软件系统或产品来协助日常工作的运行。对于许多组织而言，由于软件错误而造成的计算机死机或系统崩溃，经常会造成难以弥补的损失。遗憾的是，这样的例子处处可见，并且不曾间断。

软件过程对于组织的重要性，就如同算法对子程序运行一般。合适的算法可以提高运行的效率，不合适的算法则不仅无法提高效率，而且会浪费组织资源的使用率。软件开发过程牵涉到的是更为复杂的人、事、物，而算法则是纯粹的机器代码执行。本章将介绍构成软件过程(Software Process)的基本活动，以及几种在软件与系统产业界常用的软件过程。

软件开发过程主要是描述开发软件系统所牵涉到的相关活动，以及如何循序渐进地执行这些活动。不同的系统、组织及开发，其管理工具所采用的流程都有可能不同。例如：有些系统适合采用按部就班的方式，从分析、设计、实现、测试到移交逐步地进行；有些系统则适合采用反复循环的方式，不断地重复执行分析、设计、实现、测试等活动。

本章共分 3 节：第 1 节分析软件危机与造成软件危机的原因，第 2 节介绍基本的软件开发活动，第 3 节探讨常见的软件开发模型。

本章的学习重点如下：

- 软件危机与造成软件危机的原因
- 软件开发过程的基本概念
- 瀑布模型
- 统一过程模型
- 极限编程模型

1.1　软 件 危 机

著名的研究机构 Standish Group 曾在 1995 年针对全美国 8 000 个软件项目做了一项调查，调查中发现超过 30%的项目被取消，并且项目的预算平均超出 189%。这些数字相当惊人，它如实地反映出那个时期软件项目的现况及问题。造成此种情况的主要原因经归纳分析有：1) 软件公司总是在不合理的期限(Unrealistic Deadline)压力下进行开发；2) 客户在项目结束前要求增加新功能，或是给予不明确的需求(Vague Requirements)；3) 软件本身非常复

杂(Complex Structure)；4) 项目开发过程中具有许多不确定因素(Numerous Uncertainties)。

早期软件项目发生严重问题的情况比比皆是。例如[HRPL 1999]，1982 年，美国银行(Bank of America)想要进入信托领域，因此花了 18 个月深入地研究及分析信托软件系统，最后准备以 2 000 万美元的预算开发该系统，开发时间长达 9 个月，预计在 1984 年 12 月底前完成该系统。然而此系统直到 1987 年 3 月都未完成，并且已耗费 6 000 万美元，同时还失去了原先所规划的 6 亿美元的信托生意。最后，此系统因为不稳定而不得不放弃，只能将其信托账户转移出去。

类似上述的案例数不胜数。其他的如 1996 年发生的亚利安五号原型爆炸及波音 Delta III 火箭爆炸等事件，都是源于软件问题。而软件之所以会造成这些严重的问题，可以从 Fredrick Brooks 最经典的一篇论文[Bro 1987]*No Silver Bullet* 中得到部分解答。

Fredrick Brooks 在该文中认为，软件与生俱来的四种特性(Essential Difficulty)是造成直至今日仍然无法找到“银弹”的原因，同时也是造成软件危机的主要原因：

- 复杂性(Complexity)：软件与现实生活中接触到的任何事物相比，有一个很大的不同点，就是软件与生俱来的复杂性。当开发软件系统时，程序设计人员动辄必须编写小至数百行程序代码的软件组件、大至数万行程序代码的系统关键模块，因此，其复杂程度往往随着程序的大小及软件组件的个数，以非线性的方式、甚至是级数的方式递增。这往往导致一同开发的项目成员之间沟通困难程度提高、成本花费超出预算、交付时间延迟、系统进入非预期中的状态等而死机等，造成无法弥补的损失。
- 易变性(Changeability)：软件的易变性，更是所有软件设计师的梦魇。我们很少听说一栋大楼刚落成，屋主就要根据自己的意愿再大兴土木来变动外观或设计，因为众所周知，这样的调整必然要付出相当大的代价。而软件正好相反，软件的变动，相对于房子、车子、手机、电视等硬件的变动来说，更加容易且快速，因此，通常为了满足客户的需求，成功的软件系统从开发到完成、从产品交付到运营维护，随时都有可能要变更。
- 隐藏性(Invisibility)：软件本身是看不到、摸不着的。人们总是对复杂或隐藏的事物以几何或抽象的方式来具体描绘，以最大程度地帮助彼此进行沟通及思考。例如：对于房子的设计，设计师会借助设计图和模型来与客户进行沟通讨论；对于旅游行程景点间的复杂路径规划，可以通过地图来帮助我们理清方向并交换意见。而软件系统同样也可以通过抽象化的方式来表达其运作的数据流程图、控制流程图、系统架构图、系统状态图、数据结构图等，然而最关键的问题是，软件最终是看不见的。这个与生俱来的特性，常常会导致需求方面的误解，使错误的地方不容易被发现，从而在很大程度上会妨碍彼此间的沟通。
- 一致性(Conformity)：物理学家和数学家认为，在这浩瀚宇宙中万物能如此井然有序地运转，背后一定有一个亘古不变的规则或定律在主宰这一切的运行。然而在软件世界中，每位软件工程师都主导着各自的软件系统，创造属于他们自己的系统模块及组件。一旦在大型的协同作业环境下开发软件系统，接口与接口间、模

块与模块间、系统与系统间的衔接，便都存在一致性的问题需要解决，因此需要通过各种方式来转换或衔接不一致的地方，这都是避免不了的问题。

过去几十年来，为了解决软件开发与质量方面的问题，人们不断致力于各种软件开发技术的革新。例如：从早期的机器语言至 Fortran、Pascal、Cobol、C 等结构化的程序语言，再到 C++、Java 等面向对象的程序语言；或是从传统的功能模块设计演变至面向对象设计等。除了这些技术上的努力与突破外，人们同时意识到软件开发流程其实对软件开发流程及质量都具有举足轻重的影响，因此许多学者提出了不同的开发模型，试图解决软件开发过程中经常遇到的问题。

1.2　基本的软件开发活动

许多流程都有相似之处，因此可以将其总结为过程模型(Process Model)。目前常见的过程模型有瀑布模型(Waterfall Model)[ROY 1970]、统一过程(Unified Process，UP)模型[Kru 2000]、极限编程(Extreme Programming，XP)模型[Bec 2000]等。在介绍这些模型之前，先介绍一下构成开发过程的主要活动。

构成开发过程的主要活动有需求分析、设计、实现、测试与维护。

“需求分析”的主要内容是了解客户的需求、分析系统的可行性、分析需求的一致性及正确性等。

“设计”是将需求转换为系统的重要过程。设计包含架构设计、模块间的接口设计、数据库设计、算法设计与数据结构设计等。许多软件工程师常会认为，自己可以立即编写程序而不需要分析需求和撰写设计，因而忽略规划的重要性，直接进行程序编写。此种做法对于软件系统而言，可能会造成种种问题。举例而言，如果没有架构设计，就会缺乏整体性的思考，系统可能因此而无法满足接口需求以及非功能性的需求(例如性能、可维护性等)；此外，还可能会因为忽略事先的规划与分析而造成重复工作等。

“实现”指的是通过程序语言，将所设计的内容转化为可以执行的软件系统。“除错”是实现活动中不可避免的工作，主要是修改程序编写过程中产生的错误。除此之外，“单元测试”通常也会在实现阶段进行，目的是要确认单元程序代码的正确性。当程序有错误时，需要进行除错，将错误排除。

“测试”是对实现的程序代码模块进行检测，检验其功能是否正确、性能是否符合要求。一般而言，测试可以分为单元测试、集成测试、系统测试与验收测试：

- 单元测试：测试单元模块功能是否能正常运行。
- 集成测试：测试模块或子系统的接口集成是否能正常运行。
- 系统测试：测试系统的整体性能、安全性、稳定度等非功能性需求是否符合预期目标。
- 验收测试：测试系统的整体性能是否符合使用者的要求。

软件系统的特性之一就是需求会经常发生变动，许多系统每隔半年甚至是几个月就会改版。软件“维护”的目的是要确保已经发行的软件系统可以持续满足客户的需要。一般而言，维护可以有如下几种情况：修复错误、增加或变更功能，以及因为平台改变所做的调整。

1.3 软件开发模型

软件开发模型是抽象的软件开发模板，为组织提供定义软件开发的流程指引。本节介绍瀑布模型、统一过程模型与极限编程模型 3 种模型。

1.3.1 瀑布模型

瀑布式开发模型的概念是 Winston W. Royce 于 1970 年提出的，因为其描述各开发阶段的顺序性相当明确，所以称为瀑布模型。如图 1-1 所示，此模型的开发过程可以分为 5 个阶段：

- 需求定义。主要目的是了解顾客的需求或建立产品的功能需求，许多活动，例如需求收集、需求访谈、需求分析等都会在这个阶段进行。这个阶段典型的成果是软件需求规格书[IEEE Std 1984]。
- 系统设计。主要目的是依照上一阶段得到的软件需求规格书进行设计。系统设计包含架构设计与细部设计(例如接口设计、数据库设计等)。此阶段的主要成果为系统设计文档[IEEE Std 1993]。
- 系统实现。将上一阶段的系统设计文档落实为可以执行的软件程序代码，并进行单元测试。这个阶段的主要成果是程序代码与单元测试的结果。
- 系统集成与测试。系统集成是指依据系统设计文档的架构逐步集成各子系统或模块，并进行集成测试，以确保各子系统可以正确无误地集成。系统测试是针对整个系统进行整体性的测试，以确保其功能和性能都可以符合软件需求规格书的描述。此阶段的主要成果为系统测试报告[IEEE Std 1998]。

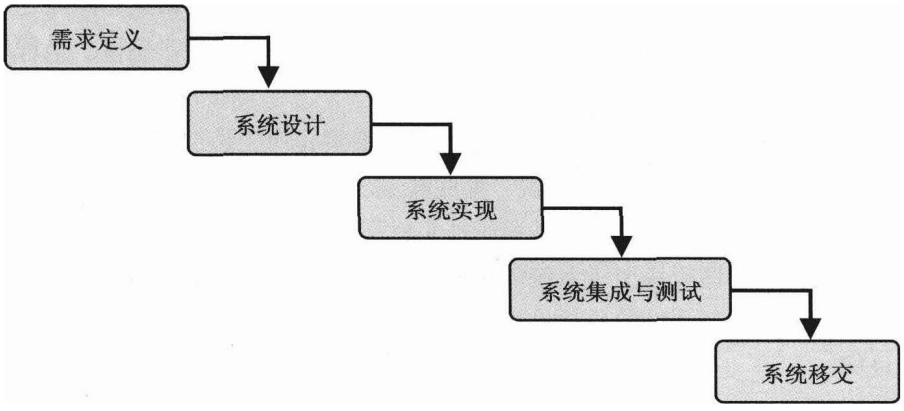


图 1-1　瀑布模型的开发过程

- 系统移交。当系统测试无误并进行移交后，此软件系统进入维护阶段。维护阶段通常很长，主要是进行错误修复以及对功能进行增强。

瀑布模型可以说是最常用的开发方式，因为此模型最能明确区分各个开发阶段。每个阶段结束前通常会产生明确的文件，当文件确认后，即代表该阶段结束并进入下一个阶段。瀑布式的开发方法建议系统遵循分析、设计、实现、测试等阶段逐步地完成系统创建，与“没有流程”的开发方式相比，它定义了几个重要的测试点，在每个测试点产生相关文件或模块供后续阶段参考。有了这些测试点，开发进度便不再如黑箱(Black Box)般隐秘不可知，从而有助于管理者确定整体进度。为了比较各种开发模型间的差别，我们假想一个为期一年的项目，其项目的进行采用瀑布式方法，则其可能的时间规划如表 1-1 所示。可以看出，每一个阶段完成后才能进入下一个阶段(请与表 1-2 和 1-3 做比较)。

表 1-1　采用瀑布式 6 开发模型时的时间规划

	1	2	3	4	5	6	7	8	9	10	11	12
需求定义	●	●										
系统设计			●	●	●							
系统实现						●	●	●				
系统集成与测试									●	●	●	
系统移交												●

瀑布式开发模型的优点有：1) 清楚的阶段区分和一般的工程方法(例如建筑、土木、化学工程等)一样有明确的流程阶段，使项目的控制更为容易；2) 明确的文件成果。使得合同的签订、技术或管理的审查更为容易，也有利于项目后期的维护。

然而，瀑布式开发模型也有其限制，它并非适用于所有的项目。它只有在每个测试点产生的文件都能完整表现该阶段所必须具备的信息时，才不会产生错误，也即用户需求必须经分析后完整且正确地描述在分析文件中，设计师据此建立完整的设计图，最后再根据设计图设计完整的程序。就实际而言，即使这中间的过程没有太大的错误，用户需求传达也难保正确。综合来说，瀑布式开发模型有以下问题：

- 需求提供者没有办法正确和完整地表达需求。
- 用户、分析师、设计师、程序员之间在传递信息上的误解。
- 用户的需求经常改变。

基于以上原因，瀑布式开发模型较适用于大型项目以及需求变更幅度不大的系统。

1.3.2　统一过程模型

有别于瀑布模型，统一过程(RUP)模型强调迭代(Iterative)、递增(Incremental)和改良(Evolutionary)[Kru 2000]。“迭代”表示系统分析、设计、实现、测试与集成是反复不断进行的；“递增”表示系统的需求是逐步增加的，并非一开始就必须全部收集完整；“改良”表示系统在开发过程中是不断改良的，而非仅在后期设置。

除了迭代、递增、改良外，统一过程模型的开发方式还包含另外两项主要特色：

1) 用例驱动(Use Case Driven)：强调用户的价值，从用户的观点思考用户想要的是什么、需要的又是什么，开发者在项目开发过程中将会不断地检查其设计是否符合用例模型。

(2) 以架构为中心(Architecture-Centric)：强调尽早建立一个以组件为基础的架构(Component-Based Architecture)。RUP 模型强调如何使用目前现有组件建立系统的架构。

统一过程模型的特色是以两个维(Dimension)来描述流程，如图 1-2 所示。在横轴方面，统一过程模型可以分为 4 个阶段：起始阶段、分析阶段、建构阶段与移交阶段。每个阶段都有其阶段性目的。各阶段的主要工作简述如下：

- **起始阶段(Inception Phase)**：计划申请、风险评估、可行性分析、初步计划执行时间、资源估计及项目规划。
- **分析阶段(Elaboration Phase)**：分析需求、了解问题领域、建立系统架构。
- **建构阶段(Construction Phase)**：系统设计、系统实现、单元测试。
- **移交阶段(Transition Phase)**：移机测试、移机安装、文件制作。

特别要注意，分析阶段并非单纯只做分析，还包含分析、设计、实现测试及产生可执行的版本，主要差别在于分析阶段生成的系统要用于沟通以确定需求是否正确，而非系统的交付。当需求确定程度约达 90%后，即可进入建构阶段(后面阶段仍然可以修改新增需求)，此时仅完成约 20～30%的系统配置。分析阶段会进行多次迭代(Iteration)，每次迭代都会历经分析、设计、实现测试及生成可执行版本的过程。越重要、风险越高、与架构设计越有关系的需求，越会在前面的迭代中讨论、设计并实现。借助迭代的系统展示与架构检查，以确定设计者与客户对系统达成共识。

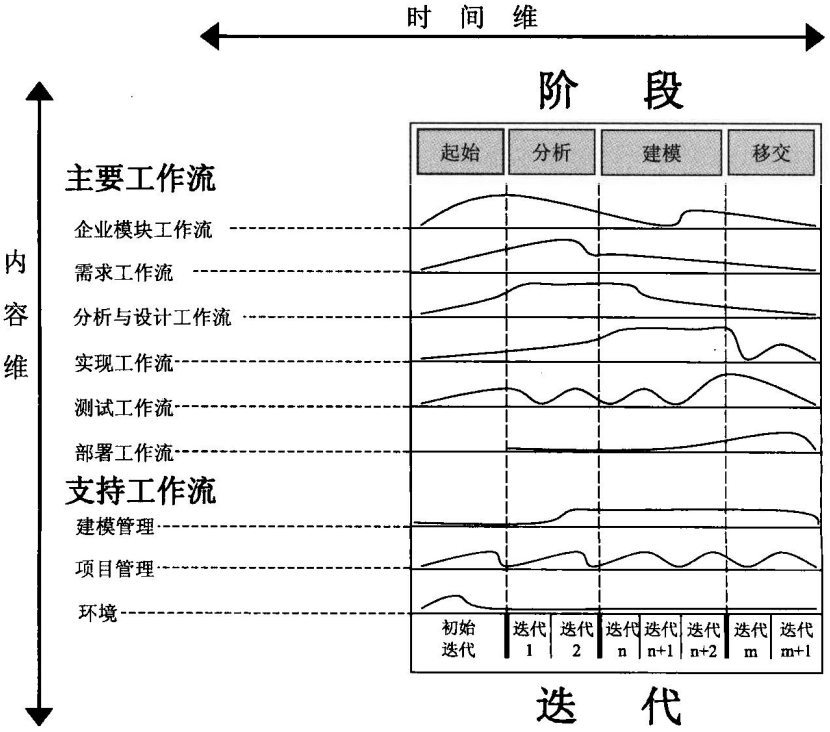


图 1-2　RUP 模型的生命周期[RSC 1998，Kru 2000]

同样，建构阶段也包含分析、设计、实现测试及生成可执行的版本，但其大部分的工作内容在于系统的设计、建构与测试。此阶段仍会新增或修改需求，但变动幅度通常不大。

在纵向方面，核心的流程工作可以分为 6 个工作流(Workflow)：

1) **企业模块工作流(Business Modeling Workflow)**。企业模块工作流的主要目的是通过此流程活动的进行，让系统开发端能深入了解企业内部的专业领域知识及相关专业用语，并为开发端、需求端与使用端建立沟通渠道。开发端会尝试着了解组织的愿望、企业信息化的动机，并分析可能遇到的困难。

2) **需求工作流(Requirements Workflow)**。需求工作流的主要目的是让系统开发端通过此流程活动的进行，来了解并确认该系统所需提供的功能。为获得明确且切实的需求，分析师必须与客户及未来该系统配置的相关使用人员进行沟通，然后整理并记录组织收集到的系统需求文件，并进一步与客户确认。

3) **分析与设计工作流(Analysis & Design Workflow)**。分析与设计工作流的目的是系统开发人员将客户端认可的需求内容，转换成真正可以部署执行完整系统前的准备工作。从系统开发者的角度，对所要开发的系统加以分析及设计，产生各个设计模块以实现需求。所产生的设计模块将作为下一阶段实现的依据。

4) **实现工作流(Implementation Workflow)**。实现工作流的目的是将系统设计模块转换为可以执行的程序代码。通常包含对单元模块(例如某个函数或类别的单元)进行测试。由于 RUP 模型强调“迭代性”，系统实现活动在先前的迭代中就会进行，这样可及早发现可能的设计错误，某些不可行的技术方案也可以及早发现并适时调整。

5) **测试工作流(Test Workflow)**。测试工作流的主要目的在于确认每个单元可以正确地集成，特别是要确认单元之间的接口是兼容的，可以相互衔接沟通。测试也包含需求测试，检验实现的产品是否符合需求规格所要求的内容。由于 RUP 模型是采用迭代的方式，测试在先前的迭代中就会进行，因此具有可以及早发现缺陷(Defect)的优势。

6) **部署工作流(Deployment Workflow)**。此活动的目的是发行(Release)可以提供给用户安装部署并执行的版本。大部分的部署活动都会在移交阶段进行，但先前的迭代会进行部分的部署准备，例如准备软硬件设备、系统的资源配置等。

RUP 模型与瀑布模型最大的区别在于，它将阶段与工作流分开，每个阶段都会执行相同的工作流，而主要区别是执行量的多少。表 1-2 同样以一个一年的计划为例，假设该项目分 3 次迭代进行，则每 4 个月进行一次迭代。虽然每一次迭代都进行所有的工作流，但程度不同。例如，第一次迭代需要 1 个月的时间进行企业模块工作流，第二次迭代则需要 2 个星期(表 1-2 中的 2w 表示 2 个星期)，而第三次迭代仅需要 2 天的时间(表示为 2d)。

表 1-2　采用 RUP 开发模型时的时间规划

	1	2	3	4	5	6	7	8	9	10	11	12
迭代一/企业模块	●											
迭代一/需求收集		●										
迭代一/分析设计			●									

（续表）

	1	2	3	4	5	6	7	8	9	10	11	12
迭代一/实现				2w								
迭代一/测试				2w								
迭代一/部署				2d								
迭代二/企业模块					2w							
迭代二/需求收集					2w							
迭代二/分析设计						●						
迭代二/实现							●					
迭代二/测试								3w				
迭代二/部署								1w				
迭代三/企业模块									2d			
迭代三/需求收集									5w			
迭代三/分析设计									5w			
迭代三/实现									3w	●		
迭代三/测试											●	
迭代三/部署											2w	2w

1.3.3 极限编程模型

极限编程(XP)是 Kent Beck 于 1999 年提出的，目的是提倡更能“拥抱改变”(Embrace Changes)的敏捷开发方式(Agile Method)[Bec 2000]。极限编程的“极限”有着极端的含意，因为它提出了许多极端的做法。例如，极端地倡导多迭代开发方式、极端地要求顾客参与、极端地强调测试的重要性等。XP 的各项特性说明如下：

- 客户驻点(On-Site Customer)。顾客代表也是开发团队成员之一，在开发过程中必须全职参与开发团队的讨论。如此可以省去将需求文件化的时间并避免阅读需求文件可能产生的错误。随时沟通、快速回馈是 XP 的特性。
- 渐进式的规划(Incremental Planning)。顾客代表与开发团队一起确定需求。需求不是功能列表，而是一个个像故事般的故事卡(Story Card)。按照故事卡的轻重缓急和风险，快速制订出项目的范围。
- 频繁改版(Small Releases)。快速将简单的系统上线，并在极短时间内更换新版本。
- 简单设计(Simple Design)。任何时候，系统都应该尽可能设计得简单。XP 强调设计并不是一次就可以达到完美，通过简单的设计、测试与设计的改善，逐步修正设计，使系统可以符合用户需求与系统质量。一开始过于复杂的设计会使设计花费的时间过长，用户及架构师无法立即对系统产生回馈。
- 测试先行(Test-First Development)。先编写单元测试程序，以确保每个单元程序都正确。XP 十分强调回馈，而良好、正确的回馈需要好的测试。为了达到有效的测