第 4 章：软件工程基础知识　　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件生命周期与软件开发模型

第4章　软件工程基础知识

根据考试大纲，要求考生掌握软件生命周期各阶段的任务、结构化分析和设计方法、面向对象的分析与设计、软件开发工具与环境的基础知识、软件质量保证的基础知识、软件过程改进与评估和软件项目管理基础知识等七个方面的知识。

4.1　软件生命周期与软件开发模型

本节将介绍软件生命周期与软件开发模型。

上一节　本书简介　下一节

第 4 章：软件工程基础知识　　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件危机与软件工程

4.1.1　软件危机与软件工程

软件工程是一门年轻的学科。"软件工程"这个概念最早是在1968年召开的一个当时被称为"软件危机"的会议上提出的。自1968年以来，我们在该领域已经取得了长足的进步。软件工程的发展已经极大地完善了我们的软件，使我们对软件开发活动也有了更深的理解。

1.软件危机

那么什么是软件危机呢？自从强大的第三代计算机硬件问世以后，许多原来难以实现的计算机应用成为现实，同时对软件系统的需求数量和复杂度要求变得更高。而当时的软件开发技术无法满足这一日益增长的需求，引发了软件危机。它主要表现为。

软件开发生产率提高的速度，远远跟不上计算机迅速普及的趋势。软件需求的增长得不到满足，软件产品"供不应求"的现象使人类无法充分利用现代计算机硬件提供的巨大潜力。

软件成本在计算机系统总成本中所占的比例逐年上升。

不能正确估计软件开发产品的成本和进度，致使实际开发成本高出预算很多，而且超出预期的开发时间要求。

软件开发人员和用户之间的信息交流往往很不充分，用户对"已完成的"软件系统不满意的现象经常发生。

软件产品的质量不易保证。

软件产品常常是不可维护的。

软件产品的重用性差，同样的软件多次重复开发。

软件通常没有适当的文档资料。

软件危机的产生一方面是由于软件开发本身的复杂性，另一方面是与当时的手工作坊式软件开

发模式有密切关系。

2.软件工程

开发一个具有一定规模和复杂性的软件系统和编写一个简单的程序大不一样。其间的差别，借用Booch的比喻，如同建造一座大厦和搭一个狗窝的差别。大型的、复杂的软件系统的开发是一项工程，必须按工程学的方法组织软件的生产与管理，必须经过计划、分析、设计、编程、测试、维护等一系列的软件生命周期阶段。这是人们从软件危机中获得的最重要的教益，这一认识促使了软件工程学的诞生。

软件工程学就是研究如何有效地组织和管理软件开发的工程学科。IEEE在1983年将软件工程定义为：软件工程是开发、运行、维护和修复软件的系统方法。

著名的软件工程专家Boehm于1983年提出了软件工程的7条基本原理：

用分阶段的生命周期计划严格管理；

坚持进行阶段评审；

实行严格的产品控制；

采用现代程序设计技术；

结果应能清楚地审查；

开发小组的人员应该少而精；

承认不断改进软件工程实践的必要性。

软件工程方法学包含三个要素，方法、工具和过程。方法是指完成软件开发的各项任务的技术方法；工具是指为运用方法而提供的软件工程支撑环境；过程是指为获得高质量的软件所需要完成的一系列任务的框架。

近30年来，影响力最大、使用最广泛的软件工程方法学是结构化方法学和面向对象的方法学。

上一节　本书简介　下一节

　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件生命周期

4.1.2 软件生命周期

软件生命周期（Software ife Cyce）是人们在研究软件开发过程时所发现的一种规律性的事实。如同人的一生要经历婴儿期、少年期、青年期、老年期直至死亡这样一个全过程一样，一个软件产品也要经历计划、分析、设计、编程、测试、维护直至被淘汰这样一个全过程。软件的这一全过程被称为软件生命周期。

目前，软件生命周期各阶段的划分尚不统一，有的分得粗些，有的分得细些，所包含的实际内容也不完全相同。

1970年，Boehm提出了如表4-1所示的软件生命周期模型。

表4-1 Boehm定义的软件生命周期模型

计划时期		开发时期					运行时期
问题定义	可行性研究	需求分析	总体设计	详细设计	编码	测试	维护

在1988年制订和公布的国家标准《GB8566-1988计算机软件开发规范》中将软件生命周期划分为如表4-2所示的8个阶段。

表4-2 GB8566定义的软件生命周期模型

可行性研究与计划	需求分析	概要设计	详细设计	实现	组装测试	确认测试	使用和维护

在20世纪90年代初有了软件工程过程的概念之后，于1995年制订和公布的国家标准《GB/T8566-1995信息技术–软件生存期过程》定义了软件生命周期的7个主要过程，如表4-3所示。

表4-3 GB/T8566定义的软件生命周期模型

管 理 过 程				
获取过程	供应过程	开发过程	运行过程	维护过程
支 持 过 程				

其中，"获取过程"和"供应过程"分别描述了软件的"获取者"（用户）和"供应者"（开发者）在开发之前的主要活动和任务。而"管理过程"和"支持过程"则贯穿于整个软件生命周期。

1995年国际标准化组织对软件生命周期过程做了调整，公布了新的国际标准《ISO/IEC 12207信息技术–软件生存期过程》。该标准全面、系统地阐述了软件生命周期的3组共17个过程活动和任务，如表4-4所示。

表4-4 ISO/IEC定义的软件生命周期模型

主要过程	获取过程、供应过程、开发过程、运行过程、维护过程
支持过程	文档编制过程、配置管理过程、质量保证过程、验证过程、确认过程、联合评审过程、审核过程、问题解决过程
组织过程	管理过程、基础设施过程、改进过程、培训过程

1999年，Rationa软件公司的3位软件工程大师Ivar Jacobson、Grady Booch和James Rumbaugh联合编写了一部划时代的著作《统一软件开发过程》（The Unified Software Deveopment Process），将他们多年研究所得的软件开发方法学融合在了一起。该书清楚地说明了支持整个软件生命周期的统一软件开发过程是一个用例驱动的、以架构为中心的、迭代与增量的开发过程。统一软件开发过程是在重复一系列组成软件生命周期的循环，每次循环都包括如下的四个阶段和五种工作流，分别如表4-5和表4-6所示。

表4-5 RUP定义的软件生命周期模型的四个阶段

初始阶段（inception phase）	捕捉用例，思考系统架构……
细化阶段（elaboration phase）	细化用例，设计系统架构……
构造阶段（construction phase）	程序设计，实现，α测试……
移交阶段（transition phase）	β测试……

表4-6 RUP定义的软件生命周期模型的五种工作流

需求工作流（requirements workflow）	捕捉需求，使开发导向正确的系统
分析工作流（analysis workflow）	生成一个有助于架构设计的需求描述
设计工作流（design workflow）	建立系统设计模型
实现工作流（implementation workflow）	实现系统
测试工作流（test workflow）	验证实现的结果

尽管软件生命周期中各阶段的划分标准不统一，名称也不一致，但主要包括了计划、分析、设计、编程、测试和维护等阶段。本书主要介绍分析、设计、测试和维护阶段的工作。

软件开发模型

4.1.3 软件开发模型

为了指导软件的开发，可以用不同的方式将软件生命周期中的所有开发活动组织起来，从而形成了不同的软件开发模型。常见的开发模型有瀑布模型（waterfa mode）、快速原型模型（rapid prototype mode）、演化模型（evoutionary mode）、增量模型（incrementa mode）、螺旋模型（spira mode）和喷泉模型（water fountain mode）等。

1.瀑布模型

瀑布模型严格遵循软件生命周期各阶段的固定顺序：计划、分析、设计、编程、测试和维护，上一阶段完成后才能进入到下一阶段，整个模型就像一个飞流直下的瀑布，如图4-1所示。

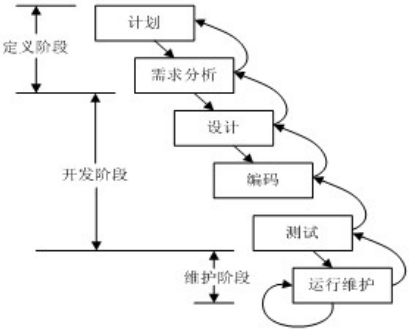


图4-1 瀑布模型示意图

瀑布模型有许多优点：可强迫开发人员采用规范的方法；严格规定了各阶段必须提交的文档；要求每个阶段结束后，都要进行严格的评审。

但瀑布模型过于理想化，而且缺乏灵活性，无法在开发过程中逐渐明确用户难以确切表达或一时难以想到的需求，直到软件开发完成之后才发现与用户需求有很大距离，此时必须付出高额的代价才能纠正这一偏差。

2.快速原型模型

快速原型是指快速建立起来的可以在计算机上运行的程序，它所完成的功能往往是最终软件产品功能的一个子集。快速原型模型的第一步是快速建立一个能反映用户主要需求的软件原型，让用户在计算机上使用它，通过实际操作了解目标系统的概貌。开发人员按照用户提出的意见快速地修改原型系统，然后再次请用户试用……，一旦用户认为这个

原型系统确实能够满足他们的需求，开发人员便可据此书写软件需求说明，并根据这份文档开发出可以满足用户真实需求的软件产品。

原型化方法基于这样一种客观事实：并非所有的需求在系统开发之前都能准确地说明和定义。因此，它不追求也不可能要求对需求的严格定义，而是采用了动态定义需求的方法。

具有广泛技能高水平的原型化人员是原型实施的重要保证。原型化人员应该是具有经验与才干、训练有素的专业人员。衡量原型化人员能力的重要标准是他是否能够从用户的模糊描述中快速获取实际的需求。

3.演化模型

演化模型也是一种原型化开发方法，但与快速原型模型略有不同。在快速原型模型中，原型的

用途是获知用户的真正需求，一旦需求确定了，原型即被抛弃。而演化模型的开发过程，则是从初始模型逐步演化为最终软件产品的渐进过程。也就是说，快速原型模型是一种"抛弃式"的原型化方法，而演化模型则是一种"渐进式"的原型化方法。

4.增量模型

增量模型是第三种原型化开发方法，但它既非"抛弃式"的，也非"渐进式"的，而是"递增式"的。增量模型把软件产品划分为一系列的增量构件，分别进行设计、编程、集成和测试。每个构件由多个相互作用的模块构成，并且能够完成特定的功能。如何将一个完整软件产品分解成增量构件，因软件产品特点和开发人员的习惯而异，但使用增量模型的软件体系结构必须是开放的，加入新构件的过程必须简单方便，新的增量构件不得破坏已经开发出来的产品。其示意图如图4-2所示。

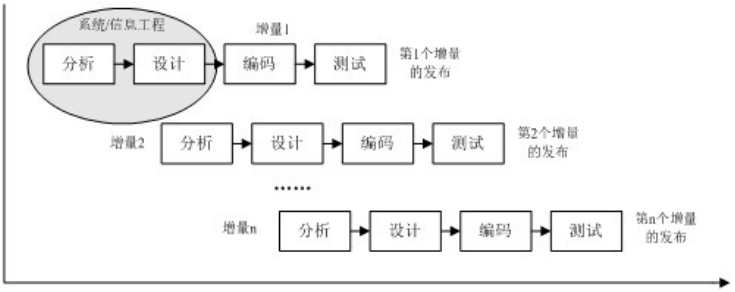


图4-2 增量模型示意图

5.螺旋模型

螺旋模型综合了瀑布模型和演化模型的优点，还增加了风险分析。螺旋模型包含了4个方面的活动：制订计划、风险分析、实施工程、客户评估。这4项活动恰好可以放在一个直角坐标系的4个象限，而开发过程恰好像一条螺旋线。采用螺旋模型时，软件开发沿着螺旋线自内向外旋转，每转一圈都要对风险进行识别和分析，并采取相应的对策。螺旋线第一圈的开始点可能是一个概念项目。从第二圈开始，一个新产品开发项目开始了，新产品的演化沿着螺旋线进行若干次迭代，一直运转到软件生命期结束。其示意图如图4-3所示。

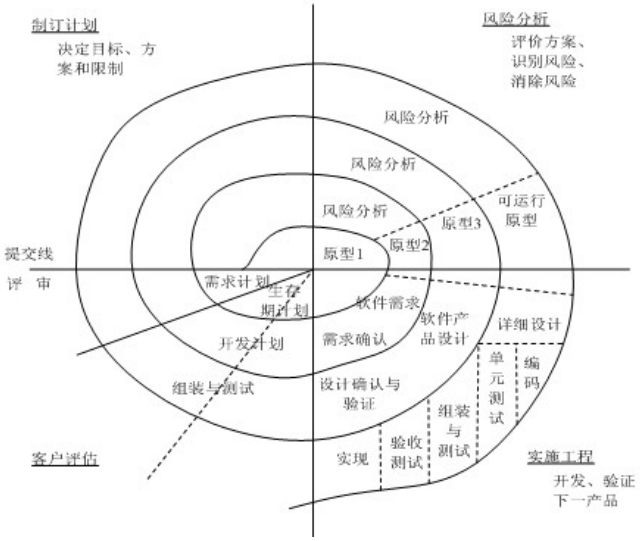


图4-3 螺旋模型示意图

6.喷泉模型

喷泉模型主要用于描述面向对象的开发过程。喷泉一词体现了面向对象开发过程的迭代和无间隙特征。迭代意味着模型中的开发活动常常需要多次重复，每次重复都会增加或明确一些目标系统的性质，但却不是对先前工作结果的本质性改动。无间隙是指在开发活动（如分析、设计、编程）之间不存在明显的边界，而是允许各开发活动交叉、迭代地进行。

7.基于构件的模型

构件（Component,也称为组件）是一个具有可重用价值的、功能相对独立的软件单元。基于构件的软件开发（Component Based Software Deveopment,CBSD）模型是利用模块化方法，将整个系统模块化，并在一定构件模型的支持下，复用构件库中的一个或多个软件构件，通过组合手段高效率、高质量地构造应用软件系统的过程。基于构件的开发模型融合了螺旋模型的许多特征，本质上是演化型的，开发过程是迭代的。基于构件的开发模型由软件的需求分析和定义、体系结构设计、构件库建立、应用软件构建、测试和发布5个阶段组成。采用基于构件的开发模型的软件过程如图4-4所示。

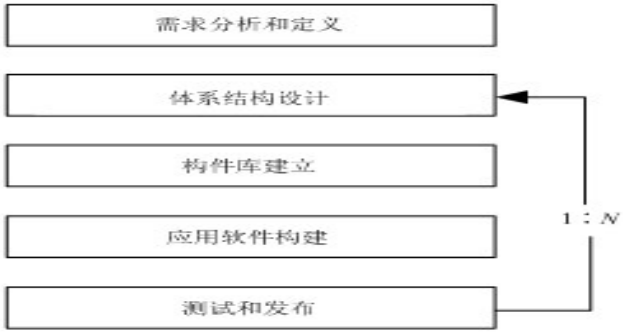


图4-4 采用基于构件的开发模型的软件过程

构件作为重要的软件技术和工具得到了极大的发展，这些新技术和工具有Microsoft的DCOM,Sun的EJB,OMG的CORBA等。基于构件的开发活动从标识候选构件开始，通过搜索已有构件库，确认所需要的构件是否已经存在，如果已经存在，就从构件库中提取出来复用；如果不存在，就采用面向对象方法开发它。在提取出来的构件通过语法和语义检查后，将这些构件通过胶合代码组装到一起实现系统，这个过程是迭代的。

基于构件的开发方法使得软件开发不再一切从头开始，开发的过程就是构件组装的过程，维护的过程就是构件升级、替换和扩充的过程，其优点是构件组装模型导致了软件的复用，提高了软件开发的效率；构件可由一方定义其规格说明，被另一方实现，然后供给第三方使用；构件组装模型允许多个项目同时开发，降低了费用，提高了可维护性，可实现分步提交软件产品。

缺点是由于采用自定义的组装结构标准，缺乏通用的组装结构标准，引入具有较大的风险；可重用性和软件高效性不易协调，需要精干的、有经验的分析人员和开发人员，一般的开发人员插不上手，客户的满意度低；过分依赖于构件，构件库的质量影响着产品质量。

8.快速应用开发模型（RAD）

快速应用开发（Rapid Appication Deveopment,RAD）模型是一个增量型的软件开发过程模型，强调极短的开发周期。RAD模型是瀑布模型的一个"高速"变种，通过大量使用可复用构件，采用基于构件的建造方法赢得快速开发。如果需求理解得好且约束了项目的范围，利用这种模型可以很快地创建出功能完善的"信息系统".其流程从业务建模开始，随后是数据建模、过程建模、应用生成、测试及反复。采用RAD模型的软件过程如图4-5所示。

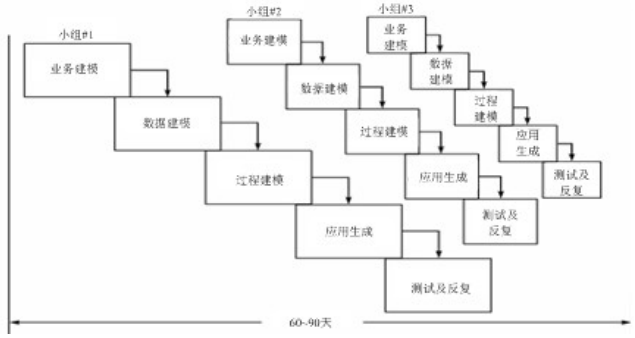


图4-5 采用RAD模型的开发过程

RAD模型各个活动期所要完成的任务如下。

业务建模：以什么信息驱动业务过程运作？要生成什么信息？谁生成它？信息流的去向是哪里？由谁处理？可以辅之以数据流图。

数据建模：为支持业务过程的数据流，找数据对象集合，定义数据对象属性，与其他数据对象的关系构成数据模型，可辅之以E-R图。

过程建模：使数据对象在信息流中完成各业务功能。创建过程以描述数据对象的增加、修改、删除、查找，即细化数据流图中的处理框。

应用程序生成：利用第四代语言（4G）写出处理程序，重用已有构件或创建新的可重用构件，利用环境提供的工具自动生成并构造出整个应用系统。

测试与交付，由于大量重用，一般只做总体测试，但新创建的构件还是要测试的。

与瀑布模型相比，RAD模型不采用传统的第三代程序设计语言来创建软件，而是采用基于构件的开发方法，复用已有的程序结构（如果可能的话）或使用可复用构件，或创建可复用的构件（如果需要的话）。在所有情况下，均使用自动化工具辅助软件创造。很显然，加在一个RAD模型项目上的时间约束需要"一个可伸缩的范围".如果一个业务能够被模块化使得其中每一个主要功能均可以在不到三个月的时间内完成，那么它就是RAD的一个候选者。每一个主要功能可由一个单独的RAD组来实现，最后再集成起来形成一个整体。

RAD模型通过大量使用可复用构件加快了开发速度，对信息系统的开发特别有效。但是像所有其他软件过程模型一样，RAD方法也有其缺陷：

并非所有应用都适合RAD.RAD模型对模块化要求比较高，如果有哪一项功能不能被模块化，那么建造RAD所需要的构件就会有问题；如果高性能是一个指标，且该指标必须通过调整接口使其适应系统构件才能赢得，RAD方法也有可能不能奏效。

开发者和客户必须在很短的时间完成一系列的需求分析，任何一方配合不当都会导致RAD项目失败。

RAD只能用于信息系统开发，不适合技术风险很高的情况。当一个新应用要采用很多新技术或当新软件要求与已有的计算机程序有较高的互操作性时，这种情况就会发生。

9.RUP方法

RUP（Rationa Unified Process）是一个统一的软件开发过程，是一个通用过程框架，可以应付种类广泛的软件系统、不同的应用领域、不同的组织类型、不同的性能水平和不同的项目规模。RUP是基于构件的，这意味着利用它开发的软件系统是由构件构成的，构件之间通过定义良好的接口相互联系。在准备软件系统所有蓝图的时候，RUP使用的是统一建模语言UM.

与其他软件过程相比，RUP具有3个显著的特点：用例驱动、以基本架构为中心、迭代和增量。

RUP中的软件过程在时间上被分解为4个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和

交付阶段。每个阶段结束时都要安排一次技术评审，以确定这个阶段的目标是否已经满足。如果评审结果令人满意，就可以允许项目进入下一个阶段。基于RUP的软件过程模型如图4-6所示。

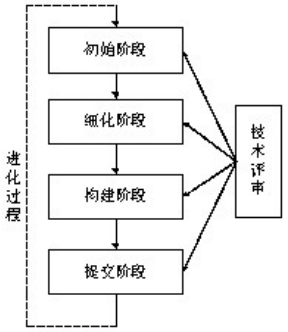


图4-6 基于RUP的软件过程

从图4-6中可以看出：基于RUP的软件过程是一个迭代过程。通过初始、细化、构建和提交四个阶段就是一个开发周期，每次经过这4个阶段就会产生一代软件。除非产品退役，否则通过重复同样的四个阶段，产品将演化为下一代产品，但每一次的侧重点都将放在不同的阶段上。这些随后的过程称为演化过程。

在进度和工作量方面，所有阶段都各不相同。尽管不同的项目有很大的不同，但一个中等规模项目的典型初始开发周期应该预先考虑到工作量和进度间的分配，如表4-7所示。

表4-7 RUP各阶段的工作量和进度分配

	初始阶段	细化阶段	构建阶段	提交阶段
工作量	5%	20%	65%	10%
进度	10%	30%	50%	10%

对于演进周期，初始和细化阶段就小得多了。能够自动完成某些构建工作的工具将会缓解此现象，并使得构建阶段比初始阶段和细化阶段的总和还要小很多。

RUP的工作流程分为两部分：核心工作流程与核心支持工作流程。核心工作流程（在项目中的流程）包括业务需求建模、分析设计、实施、测试、部署；核心支持工作流程（在组织中的流程）包括环境、项目管理、配置与变更管理。

统一过程4个阶段的核心任务，以及需要提交的文档和模型分别如下。

1）初始阶段

（1）核心任务

明确地说明项目规模。这涉及了解环境及最重要的需求和约束，以便于可以得出最终产品的验收标准。

计划和准备商业理由。评估风险管理、人员配备、项目计划和成本/进度/收益率折中的备选方案。

综合考虑备选构架，评估设计和自制/外购/复用方面的折中，从而估算出成本、进度和资源。此处的目标在于通过对一些概念的证实来证明可行性。该证明可采用可模拟需求的模型形式或用于探索被认为高风险区域的初始原型。先启阶段的原型设计工作应该限制在确信解决方案可行就可以了。该解决方案在精化和构建阶段实现。

准备项目的环境，评估项目和组织，选择工具，决定流程中要改进的部分。

（2）需要提交的文档和模型（见表4-8）

表4-8 初始阶段需要提交的文档和模型

核心文档及模型	里程碑状态
前景	已经对核心项目的需求、关键功能和主要约束进行了记录
商业理由	已经确定并得到了批准
风险列表	已经确定了最初的项目风险
软件开发计划	已经确定了最初阶段及其持续时间和目标。软件开发计划中的资源估算（特别是时间、人员和开发环境成本）必须与商业理由一致。资源估算可以涵盖整个项目直到交付所需的资源，也可以只包括进行精化阶段所需的资源。此时，整个项目所需的资源估算应该看做是大致的“粗略估计”。该估算在每个阶段和每次迭代中都会更新，并且随着每次迭代变得更加准确。根据项目的需要，可能在某种条件下完成了一个或多个附带的“计划”工件。此外，附带的“指南”工件通常也至少完成了“草稿”
迭代计划	第一个精化迭代的迭代计划已经完成并经过了复审
软件验收计划	完成复审并确定了基线；随着其他需求的发现，将对其在随后的迭代中进行改进
项目专用模板	已使用文档模板制作了文档工件
用例建模指南	确定了基线
工具	选择了支持项目的所有工具。安装了对先启阶段的工作必要的工具
词汇表	已经定义了重要的术语；完成了词汇表的复审
用例模型	已经确定了重要的主角和用例，只为最关键的用例简要说明了事件流
领域模型	已经对系统中使用的核心概念进行了记录和复审。在核心概念之间存在特定关系的情况下，已用作对词汇表的补充
原型	概念原型的一个或多个证据，以支持前景和商业理由，解决非常具体的风险

2）细化阶段

（1）核心任务

快速确定构架，确认构架并为构架建立基线。

根据此阶段获得的新信息改进前景，对推动构架和计划决策的最关键用例建立可靠的了解。

为构建阶段创建详细的迭代计划并为其建立基线。

改进开发案例，定位开发环境，包括流程和支持构建团队所需的工具和自动化支持。

改进构架并选择构件。评估潜在构件，充分了解自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求。

（2）需要提交的文档和模型（见表4-9）

表4-9 细化阶段需要提交的文档和模型

核心文档及模型	里程碑状态
原型	已经创建了一个或多个可执行构架原型，以探索关键功能和构架上的重要场景
风险列表	已经进行了更新和复审。新的风险可能是构架方面的，主要与处理非功能性需求有关
项目专用模板	已使用文档模板制作了文档工件

核心文档及模型	里程碑状态
工具	已经安装了用于支持精化阶段工作的工具
软件构架文档	编写完成并确定了基线，如果系统是分布式的或必须处理并行问题，则包括构架上重要用例的详细说明（用例视图）、关键机制和设计元素的标识（逻辑视图），以及（部署模型的）进程视图和部署视图的定义
设计模型（和所有组成部分）	制作完成并确定了基线。已经定义了构架方面重要场景的用例实现，并将所需行为分配给了适当的设计元素。已经确定了构件并充分理解了自制/外购/复用决策，以便有把握地确定构建阶段的成本和进度。集成了所选构架构件，并按主要场景进行了评估。通过这些活动得到的经验有可能导致重新设计构架、考虑替代设计或重新考虑需求
数据模型	制作完成并确定了基线。已经确定并复审了主要的数据模型元素（如重要实体、关系和表）
实施模型（以及所有组成工件，包括构件）	已经创建了最初结构，确定了主要构件并设计了原型
前景	已经根据此阶段获得的新信息进行了改进，对推动构架和计划决策的最关键用例建立了可靠的了解
软件开发计划	已经进行了更新和扩展，以便涵盖构建阶段和产品化阶段
指南，如设计指南和编程指南	使用指南对工作进行了支持

迭代计划	已经完成并复审了构建阶段的迭代计划
用例模型	用例模型（大约完成 80%）——已经在用例模型调查中确定了所有用例、确定了所有主角并编写了大部分用例说明（需求分析）
补充规约	已经对包括非功能性需求在内的补充需求进行了记录和复审
可选	里程碑状态
商业理由	如果构架调查不涵盖变更基本项目假设的问题，则已经对商业理由进行了更新
分析模型	可能作为正式工件进行了开发；进行了经常但不正式的维护，正演进为设计模型的早期版本
培训材料	用户手册与其他培训材料，根据用例进行了初步起草。如果系统具有复杂的用户界面，可能需要培训材料

（续表）

3）构建阶段

（1）核心任务

资源管理、控制和流程优化。

完成构件开发并根据已定义的评估标准进行测试。

根据前景的验收标准对产品发布版进行评估。

（2）需要提交的文档和模型（见表4-10）

表4-10 构建阶段需要提交的文档和模型

核心文档及模型	里程碑状态
“系统”	可执行系统本身随时可以进行“Beta”测试
部署计划	已开发最初版本，进行了复审并建立了基线
实施模型	对在精化阶段创建的模型进行了扩展；构建阶段末期完成所有构件的创建
核心文档及模型	里程碑状态
测试模型	为验证构建阶段所创建的可执行发布版而设计并开发的测试
培训材料	用户手册与其他培训材料，根据用例进行了初步起草。 如果系统具有复杂的用户界面，可能需要培训材料
迭代计划	已经完成并复审了产品化阶段的迭代计划
设计模型	已经用新设计元素进行了更新，这些设计元素是在完成所有需求期间确定的
项目专用模板	已使用文档模板制作了文档工件
工具	已经安装了用于支持构建阶段工作的工具
数据模型	已经用支持持续实施所需的所有元素（如表、索引、对象关系型映射等）进行了更新
可选	里程碑状态
补充规约	已经用构建阶段发现的新需求（如果有）进行了更新
用例模型	已经用构建阶段发现的新用例（如果有）进行了更新

4）产品化阶段（提交阶段）

（1）核心任务

执行部署计划。

对最终用户支持材料定稿。

在开发现场测试可交付产品。

制作产品发布版。

获得用户反馈。

基于反馈调整产品。

使最终用户可以使用产品。

（2）需要提交的文档和模型（见表4-11）

表4-11 提交阶段需要提交的文档和模型

核心文档及模型	里程碑状态
产品工作版本	已按照产品需求完成，客户应该可以使用最终产品
发布说明	完成
安装产品与模型	完成
培训材料	完成，以确保客户自己可以使用和维护产品
最终用户支持材料	完成，以确保客户自己可以使用和维护产品
可选	里程碑状态
测试模型	在客户想要进行现场测试的情况下，可以提供测试模型

10.XP方法

请参看第12章《软件新技术简介》的极限编程部分。

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

主要软件开发方法

4.2 主要软件开发方法

本节将介绍主要软件开发方法。

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

结构化分析和设计

4.2.1 结构化分析和设计

1977年出现的结构化方法学也称为生命周期方法学，它采用结构化技术（结构化分析、结构化设计和结构化实现）来完成软件开发的各项任务。这种方法学把软件生命周期的全过程依次划分为若干个阶段，然后顺序地完成每个阶段的任务。

结构化方法学具有以下特点。

阶段性。前一阶段工作完成以后，后一阶段工作才能开始，前一阶段的输出文档是后一阶段的输入文档。

推迟实施。将分析和设计阶段与实施分开，适当地推迟系统的具体程序实现。

文档管理。在每阶段就规定了要完成的文档资料，没有完成文档，就认为没有完成该阶段的任务。在每一阶段都要对已完成的文档进行复审，以便尽早发现问题，避免后期的返工。

把软件生命周期划分成若干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发过程的困难程度；在每个阶段结束之前都要从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发的全过程以一种有条不紊的方式进行。

结构化方法学曾经给软件产业带来巨大进步，在一定程度上缓解了软件危机。但结构化方法基于功能分析与功能分解，软件结构紧密依赖于系统功能，而在实际开发工作中，系统功能往往又是模糊易变的，功能的变化经常会引起软件结构的整体修改。

目前，面向对象方法已取代结构化方法成为软件方法学的主流。但对于一些功能需求非常明确而且不会轻易改变的软件系统，结构化方法仍然是比较有效的。

1.结构化分析

结构化分析（Structured Anaysis,SA）方法是一种面向数据流的需求分析方法。它的基本思想是自顶向下逐层分解，把一个大问题分解成若干个小问题，每个小问题再分解成若干个更小的问题。经过逐层分解，每个最低层的问题都是足够简单、容易解决的，于是复杂的问题也就迎刃而解了。

数据流图和数据字典是结构化分析的常见工具，软件需求说明书是需求分析阶段的最后成果。

1）数据流图

数据流图（Data Fow Diagram,DFD）用来描述数据流从输入到输出的变换流程。关于更多详细的内容参见"数据流图设计"的内容。

2）数据字典

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

数据流图和数据字典共同构成系统的逻辑模型。没有数据流图，数据字典难以发挥作用；没有数据字典，数据流图就不严格。只有把数据流图和对数据流图中每个元素的精确定义放在一起，才能共同构成系统的规格说明。关于更多详细的内容参见"数据设计"的内容。

2.结构化设计

系统设计是软件生命周期的重要组成部分，主要包括体系结构设计、接口设计、数据设计和过程设计。

结构化设计（Structured Design,SD）方法是一种面向数据流的设计方法，它是以结构化分析阶段所产生的文档（包括数据流图、数据字典和软件需求说明书）为基础，自顶向下，逐步求精和模块化的过程。结构化设计通常可分为概要设计和详细设计。概要设计的任务是确定软件系统的结构，进行模块划分，确定每个模块的功能、接口及模块间的调用关系。详细设计的任务是为每个模块设计实现的细节。更多详细的内容参见"软件设计概述"的内容。

1）概要设计

经过需求分析阶段的工作，系统必须已经清楚了"做什么",概要设计的基本目的就是回答"概括地说，系统应该如何实现？"这个问题。概要设计的重要任务就是设计软件的结构，也就是要确定系统是由哪些模块组成的，以及这些模块相互间的关系。

SD方法采用结构图（structure chart）来描述程序的结构。构成程序结构图的主要成分有模块、调用和数据，结构图中的模块用矩形表示，在矩形框内可标上模块的名字。模块间如有箭头或直线相连，表明它们之间有调用关系。SD方法有时也使用层次图和HIPO图（层次图加输入/处理/输出图）。

整个概要设计过程主要包括：

第1步：复查基本系统模型。复查的目的是确保系统的输入数据和输出数据符合实际。

第2步：复查并精化数据流图。应该对需求分析阶段得到的数据流图认真复查，并且在必要时进行精化。不仅要确保数据流图给出了目标系统的正确的逻辑模型，而且应该使数据流图中每个处理都代表一个规模适中相对独立的子功能。

第3步：确定数据流图的信息流类型。数据流图中从系统的输入数据流到系统的输出数据流的一连串连续变换形成了一条信息流。信息流大体可分为两种类型：

变换流：信息沿着输入通道进入系统，然后通过变换中心（也称主加工）处理，再沿着输出通道离开系统。具有这一特性的信息流称为变换流。具有变换流型的数据流图可明显地分成输入、变换（主加工）、输出三大部分。

事务流：信息沿着输入通道到达一个事务中心，事务中心根据输入信息（即事务）的类型在若干个动作序列（称为活动流）中选择一个来执行，这种信息流称为事务流。事务流有明显的事务中心，各活动以事务中心为起点呈辐射状流出。

第4步：根据流类型分别实施变换分析或事务分析。变换分析是从变换流型的数据流图导出程序结构图。具体过程如下。

确定输入流和输出流的边界，从而孤立出变换中心。

完成第一级分解，设计模块结构的顶层和第一层。

完成第二级分解，也就是输入控制模块、变换控制模块和输出控制模块的分解，设计中、下层模块。

事务分析是从事务流型的数据流图导出程序结构图，具体过程如下：

确定事务中心和每条活动流的流特性。

将事务流型数据流图映射成高层的程序结构，分解出接收模块、发送模块（调度模块），以及发送模块所控制的下层所有的活动流模块。

进一步完成接受模块和每一个活动流模块的分解。

第5步：根据软件设计原则对得到的软件结构图进一步优化。

2）详细设计

概要设计已经确定了每个模块的功能和接口，详细设计的任务就是为每个模块设计其实现的细节。详细设计阶段的根本目标是确定应该怎样具体地实现所要求的系统，得出对目标系统的精确描述。

结构化程序设计（Structured Programming,SP）采用自顶向下逐步求精的设计方法和单入口单出口的控制结构。在设计一个模块的实现算法时先考虑整体后考虑局部，先抽象后具体，通过逐步细化，最后得到详细的实现算法。单入口单出口的控制结构使程序的静态结构和动态执行过程一致，具有良好的结构，增强了程序的可读性。

针对在程序中大量无节制地使用GOTO语句而导致程序结构混乱的现象，Dijkstra于1965年提出在程序语言中取消GOTO语句。1966年，Bohm和Jacopini证明了任何单入口、单出口、没有死循环的程序都能用3种基本的控制结构来构造，这3种基本的控制结构是：顺序结构、IF_THEN_ESE型分支结构（选择结构）和DO_WHIE型循环结构。如果程序设计中只允许使用这三种基本的控制结构，则称为经典的结构化程序设计；如果还允许使用DO_CASE型多分支结构和DO_UNTI型循环结构，则称为扩展的结构化程序设计；如果再加上允许使用EAVE（或BREAK）结构，则称为修正的结构化程序设计。

应用于详细设计的工具主要包括以下几种。

程序流程图：程序流程图又称为程序框图，它是历史最悠久的描述过程设计的方法，然而它也是用得最混乱的一种方法。程序流程图的主要优点是对控制流程的描绘很直观，便于初学者掌握。但由于程序流程图中用箭头代表控制流，经常诱使程序员不顾结构化程序设计的精神而随意转移控制，且不支持逐步求精方法，不易表现数据结构。程序流程图尽管有种种缺点，许多人建议停止使用它，但至今仍在广泛使用着。不过总的趋势是越来越多的人不再使用程序流程图了。

盒图（N–S图）：盒图是由Nassi和Shneiderman提出的一种符合结构化设计原则的图形描述工具，它仅含五种基本的控制结构：顺序结构、IF-THEN-ESE型分支结构、CASE型多分支结构、DO-WHIE和DO-UNTI型循环结构、子程序结构。盒图具有以下特点。

功能域（即一个特定控制结构的作用域）明确，可以从盒图上一眼就看出来。

由于没有箭头，不可能任意转移控制。

容易确定局部和全程数据的作用域。

容易表示嵌套关系，也可以表示模块的层次结构。

坚持使用盒图作为详细设计的工具，可以使程序员逐步养成用结构化的方式思考问题和解决问题的习惯。

PAD图：PAD图是问题分析图（probem anaysis diagram）的英文缩写，它用二维树型结构的图表示程序的控制流，比较容易翻译成机器代码。PAD图具有以下特点。

使用表示结构化控制结构的PAD符号所设计出来的程序必然是结构化程序。

PAD图所描绘的程序结构十分清晰。

用PAD图表现程序逻辑，易读、易懂、易记。

容易将PAD图转换成高级语言源程序，这种转换可用软件工具自动完成。

PAD图既可表示程序逻辑，也可用于描绘数据结构。

PAD图的符号支持自顶向下、逐步求精方法的使用。

PD:PD是程序设计语言（Program Design anguage）的英文缩写，也称伪码，是一种以文本方式表示数据和处理过程的设计工具。PD是一种非形式化语言，它对控制结构的描述是确定的，但控制结构内部的描述语法是不确定的，它可根据不同的应用领域和不同的设计层次灵活选用其描述方式，甚至可用自然语言描述。与程序语言（programming anguage）不同，PD程序是不可执行的，但它可以通过转换程序自动转换成某种高级程序语言的源程序。

常见的详细设计工具还包括判定树、判定表等。

上一节　本书简介　下一节

面向数据结构的设计

4.2.2 面向数据结构的设计

上面讲的结构化设计方法是面向数据流的，另外还有一种面向数据结构的设计方法。它根据输入/输出数据结构导出程序结构。

在许多应用领域中信息都有清楚的层次结构，输入数据、内部存储的信息（数据库或文件）及输出数据都可能有独特的结构。数据结构既影响程序的结构，又影响程序的处理，重复出现的数据通常由具有循环控制结构的程序来处理，选择数据（即可能出现也可能不出现的信息）要用带有分支控制的程序来处理。层次的数据组织通常和使用这些数据的程序的层次结构十分相似。面向数据结构设计方法的基本思想就是根据数据结构导出程序结构。

Jackson方法和Warnier方法是最著名的两种面向数据结构的设计方法。

Jackson方法的基本步骤是：建立系统的数据结构；以数据结构为基础，对应地建立程序结构；列出程序中要用到的各种基本操作，再将这些操作分配到程序结构适当的模块中。

Warnier方法不再详细介绍。

由于面向数据结构的设计方法并不明显地使用软件结构的概念，对于模块独立原则也没有给予应有的重视，因此并不适合于复杂的软件系统。

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

面向对象的分析与设计

4.2.4 面向对象的分析与设计

结构化分析和设计方法在一定程度上缓解了"软件危机".但随着人们对软件提出的要求越来越高，结构化方法已经无法承担快速高效开发复杂软件系统的重任。20世纪80年代逐渐成熟的面向对象方法学，使软件开发者对软件的分析、设计和编程等方面都有了全新的认识。由于"对象"概念的引入，将数据和方法封装在一起，提高了模块的聚合度，降低了模块的耦合度，更大程度上支持了软件重用，从而十分有效地降低了软件的复杂度，提高了软件开发的生产率。目前，面向对象方法学已成为软件开发者的第一选择。

1.面向对象方法学概述

究竟怎样才算真正的"面向对象"（Object-Oriented,OO）？Peter Coad和Edward Yourdon提出了下列等式：

面向对象 = 对象（Objects）

+ 类（Classes）

+ 继承（Inheritance）

+ 消息通信（Communication With Messages）

1）对象与封装

对象（Object）是系统中用来描述客观事物的一个实体，它是构成系统的一个基本单位。面向对象的软件系统是由对象组成的，复杂的对象由比较简单的对象组合而成。也就是说，面向对象方法学使用对象分解取代了传统方法的功能分解。

对象三要素对象标识、属性和服务。

对象标识（Object Identifier），也就是对象的名字，供系统内部唯一地识别对象。定义或使用对象时，均应指定对象标识。

属性（Attribute），也称状态（State）或数据（Data），用来描述对象的静态特征。在某些面向对象的程序设计语言中，属性通常被称为成员变量（Member Variable）或简称变量（Variable）。

服务（Service），也称操作（Operation）、行为（Behavior）或方法（Method）等，用来描述对象的动态特征。在某些面向对象的程序设计语言中，服务通常被称为成员函数（Member Function）或简称函数（Function）。

封装（Encapsulation）是对象的一个重要原则。它有两层含义：第一，对象是其全部属性和全部服务紧密结合而形成的一个不可分割的整体；第二，对象是一个不透明的黑盒子，表示对象状态的数据和实现操作的代码都被封装在黑盒子里面。使用一个对象的时候，只需知道它向外界提供的

接口形式，无须知道它的数据结构细节和实现操作的算法。从外面看不见，也就更不可能从外面直接修改对象的私有属性了。

2）类

类（Class）是对具有相同属性和服务的一个或一组对象的抽象定义。

类与对象是抽象描述与具体实例的关系，一个具体的对象被称做类的一个实例（instance）。

3）继承与多态性

继承（Inheritance）是面向对象方法学中的一个十分重要的概念，其定义是：特殊类（或称子类、派生类）的对象拥有其一般类（或称父类、基类）的全部属性与服务，称做特殊类对一般类的继承。在面向对象的方法学中，继承是提高软件开发效率的重要原因之一。

多态性（Polymorphism）是指一般类中定义的属性或服务被特殊类继承之后，可以具有不同的数据类型或表现出不同的行为。使用多态技术时，用户可以发送一个通用的消息，而实现的细节则由接受对象自行决定，这样同一消息就可以调用不同的方法。多态性不仅增加了面向对象软件系统的灵活性，进一步减少了信息冗余，而且显著提高了软件的可重用性和可扩充性。多态有多种不同的形式，其中参数多态和包含多态称为通用多态，过载多态和强制多态称为特定多态。

4）消息通信

消息（Message）就是向对象发出的服务请求，它应该含有下述信息：提供服务的对象标识、消息名、输入信息和回答信息。对象与传统的数据有本质区别，它不是被动地等待外界对它施加操作，相反，它是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。

消息通信（Communication With Messages）也是面向对象方法学中的一条重要原则，它与对象的封装原则密不可分。封装使对象成为一些各司其职、互不干扰的独立单位；消息通信则为它们提供了唯一合法的动态联系途径，使它们的行为能够互相配合，构成一个有机的系统。

只有同时使用对象、类、继承与消息通信，才是真正面向对象的方法。

5）面向对象方法学的优点

与人类习惯的思维方法一致：面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使软件开发的方法与过程尽可能接近人类认识世界解决问题的方法与过程，也就是使描述问题的"问题域"与解决问题的"解域"在结构上尽可能一致。

稳定性好：传统的软件开发方法基于功能分析与功能分解，软件结构紧密依赖于系统所要完成的功能，当功能需求发生变化时将引起软件结构的整体修改。而用户需求变化大部分是针对功能的，因此这样的系统是不稳定的。

面向对象的方法用对象模拟问题域中的实体，以对象为中心构造软件系统，系统的功能需求变化时并不会引起软件结构的整体变化。由于现实世界中的实体是相对稳定的，因此以对象为中心构造的软件系统也是比较稳定的。

可重用性好：面向对象方法学在利用可重用的软件成分构造新的软件系统时有很大的灵活性。继承机制与多态性使得子类不仅可以重用其父类的数据结构与程序代码，并且可以方便地修改和扩充，而这种修改并不影响对原有类的使用。

较易开发大型软件产品：用面向对象方法学开发软件时，构成软件系统的每个对象相对独立。因此，可以把一个大型软件产品分解成一系列相互独立的小产品来处理。这不仅降低了开发的技术难度，而且也使得对开发工作的管理变得容易多了。

可维护性好：面向对象的软件比较容易理解、容易修改、容易测试。

2.面向对象的分析

综观计算机软件发展史，许多新方法和新技术都是在编程领域首先兴起，进而发展到软件生命周期的前期阶段--分析与设计阶段。结构化方法经历了从"结构化编程"、"结构化设计"到"结构化分析"的发展历程，面向对象的方法也经历了从"面向对象的编程"（Object-Oriented Programming,OOP）、"面向对象的设计"（Object-Oriented Design,OOD）到"面向对象的分析"（Object-Oriented Analysis,OOA）的发展历程。1989年之后，面向对象方法的研究重点开始转向软件生命周期的分析阶段，并将OOA和OOD密切地联系在一起，出现了一大批面向对象的分析与设计（OOA&D）方法。截止到1994年，公开发表并具有一定影响的OOA&D方法已达50余种。

由于各种OOA方法所强调的重点与该方法的主要特色不同，因此所产生的OOA模型从整体形态、结构框架到具体内容都有较大的差异。

1）OMT方法简介

1991年，James Rumbaugh在《面向对象的建模与设计》（Object-Oriented Modeling and Design）一书中提出了面向对象分析与设计的OMT（Object Modeling Technique）方法。20世纪90年代中期，笔者曾使用OMT方法开发了"印典"、"书林"等排版系统。本书的OOA模型主要依据OMT方法，同时参考了Peter Coad和Edward Yourdon的OOA模型。

OMT方法的OOA模型包括对象模型、动态模型和功能模型。

对象模型表示静态的、结构化的系统的"数据"性质。它是对模拟客观世界实体的对象及对象彼此间的关系的映射，描述了系统的静态结构。通常用类图表示。

动态模型表示瞬时的、行为化的系统的"控制"性质，它规定了对象模型中的对象的合法变化序列。通常用状态图表示。

功能模型表示变化的系统的"功能"性质，它指明了系统应该"做什么",因此更直接地反映了用户对目标系统的需求。通常用数据流图表示。

OMT方法的三个模型，分别从三个不同侧面描述了所要开发的系统：功能模型指明了系统应该"做什么";动态模型明确了什么时候做（即在何种状态下接受了什么事件的触发）；对象模型则定义了做事情的实体。这三种模型相互补充、相互配合，三者之间具有下述关系：

动态模型展示了对象模型中每个对象的状态及它接受事件和改变状态时所执行的操作；而功能模型中的处理则对应于对象模型中的对象所提供的服务。

对象模型展示了动态模型中是谁改变了状态和经受了操作；而功能模型中的处理则可能产生动态模型中的事件。

对象模型展示了功能模型中的动作者、数据存储和流的结构；而动态模型则展示了功能模型中执行加工的顺序。

2）建立对象模型

Peter Coad和Edward Yourdon在1991年出版的《面向对象的分析》（Object-Oriented Analysis）一书中指出，复杂系统的对象模型通常由下述五个层次组成：类及对象层、结构层、主题层、属性层和服务层。上述五个层次对应着建立对象模型的五项主要活动：确定类与对象、确定结构与关联、划分主题、定义属性和定义服务。但这五项活动完全没必要顺序完成，也无须彻底完成一项活动之后再开始另外一项活动。

确定类与对象：类与对象是在问题域中客观存在的，系统分析的重要任务之一就是找出这些类

与对象。首先找出所有候选的类与对象，然后进行反复筛选，删除不正确或不必要的类与对象。

确定结构与关联：结构与关联反应了对象（或类）之间的关系，主要有以下几种：

一般–特殊结构（generalization-specialization structure），又称分类结构（classification structure），是由一组具有一般–特殊关系（继承关系）的类所组成的结构。一般–特殊关系（generalization-specialization relation）的表达式为：is a kind of.

整体–部分结构（whole-part structure），又称组装结构（composition structure），是由一组具有整体–部分关系（组成关系）的类所组成的结构。整体–部分关系（whole-part relation）的表达式为：has a.

实例关联（instance connection），即一个类的属性中含有另一个类的实例（对象），它反映了对象之间的静态联系。

消息关联（message connection），即一个对象在执行自己的服务时需要通过消息请求另一个对象为它完成某个服务，它反映了对象之间的动态联系。

划分主题：在开发大型、复杂软件系统的过程中，为了降低复杂程度，需要把系统划分成几个不同的主题。注意，应该按问题域而不是用功能分解方法来确定主题，应该按照使不同主题内的对象相互间依赖和交互最少的原则来确定主题。

定义属性：为了发现对象的属性，首先考虑借鉴以往的OOA结果，看看相同或相似的问题域是否有已开发的OOA模型，尽可能复用其中同类对象的属性定义。然后，按照问题域的实际情况，以系统责任为目标进行正确的抽象，从而找出每一对象应有的属性。

定义服务：发现和定义对象的服务，也应借鉴以往同类系统的OOA结果并尽可能加以复用。然后，研究问题域和系统责任以明确各个对象应该设立哪些服务，以及如何定义这些服务。

3）建立动态模型

建立动态模型的第一步，是编写典型交互行为的脚本。虽然脚本中不可能包括每个偶然事件，但至少必须保证不遗漏常见的交互行为。第二步，从脚本中提取出事件，确定触发每个事件的动作对象及接受事件的目标对象。第三步，排列事件发生的次序，确定每个对象可能有的状态及状态间的转换关系，并用状态图描绘它们。最后，比较各个对象的状态图，检查它们之间的一致性，确保事件之间的匹配。

4）建立功能模型

OMT方法中的功能模型实际上就是结构化方法中的数据流图。从这点看，OMT方法并不是"纯"面向对象的。这是OMT方法的一大缺陷。

1992年，Ivar Jacobson在《面向对象的软件工程--用例驱动的途径》（Object-Oriented Software Engineering,A Use Case Driven Approach）中首次提出了"用例"（use case）的概念。随后，有人提出以用例图取代数据流图进行需求分析和建立功能模型，这应该被看做是对OMT方法的重大改进。使用用例图建立起来的系统模型也被称为用例模型。

一个用例是可以被行为者感受到的、系统的一个完整的功能。一幅用例图包含的模型元素有系统、行为者、用例及用例之间的关系。用例模型描述的是外部行为者所理解的系统功能。

目前，"用例驱动"已成为软件开发过程的一条重要原则。

3.面向对象的设计

1）OOA与OOD的关系

与结构化方法不同，面向对象的方法并不强调分析与设计之间严格的阶段划分。OOA与OOD所

采用的概念、原则和表示法都是一致的，二者之间不存在鸿沟，不需要从分析文档到设计文档的转换，所以有些工作无论在分析时进行还是在设计时进行都不存在障碍。当然，OOA与OOD仍然有不同的分工和侧重点。

关于OOA与OOD的关系，目前有两种不同的观点。

一种观点是继续沿用传统的分工--分析着眼于系统"做什么",设计解决"怎么做"的问题。而Peter Coad和Edward Yourdon的OOA&D方法则采用了另外一种分工方式--分析阶段只考虑问题域和系统责任，建立一个独立于实现的OOA模型；设计阶段考虑与实现有关的因素，对OOA模型进行调整并补充与实现有关的部分，形成OOD模型。本书的OOD方法主要依据Coad/Yourdon的观点。

Coad/Yourdon的OOD模型包括以下四个部件：人机交互部件、问题域部件、任务管理部件、数据管理部件。与此对应的OOD过程也包括四项活动，设计人机交互部件、设计问题域部件、设计任务管理部件、设计数据管理部件。

2）设计问题域部件

通过OOA所得出的问题域精确模型，为设计问题域部件奠定了良好的基础。通常，OOD仅需从实现角度对问题域模型做一些补充和修改，主要是增添、合并或分解类与对象、属性及服务、调整继承关系等。

3）设计人机交互部件

在OOA过程中，已经对用户界面需求做了初步分析。在OOD过程中，则应该对系统的人机交互部件进行详细设计，以确定人机交互的细节，其中包括指定窗口和报表的形式、设计命令层次等项内容。

4）设计任务管理部件

主要是识别事件驱动任务，识别时钟驱动任务，识别优先任务，识别关键任务，识别协调任务，审查每个任务并定义每个任务。

5）设计数据管理部件

提供数据管理系统中存储和检索对象的基本结构，隔离具体的数据管理方案（如普通文件、关系数据库、面向对象数据库等）对其他部分的影响。

上一节　本书简介　下一节

软件测试与软件维护

4.3 软件测试与软件维护

本节将介绍软件测试与软件维护。

上一节　本书简介　下一节

软件测试

4.3.1 软件测试

软件测试是软件质量保证的主要手段之一，也是在将软件交付给客户之前所必须完成的步骤。目前，软件的正确性证明尚未得到根本的解决，软件测试仍是发现软件错误和缺陷的主要手段。

大量统计资料表明，目前软件测试所花费用已超过软件开发费用的30%.

1.软件测试基础

1）软件测试的目的

软件测试的目的就是在软件投入生产性运行之前，尽可能多地发现软件产品（主要是指程序）中的错误和缺陷。

为了发现程序中的错误，应竭力设计能暴露错误的测试用例。测试用例是由测试数据和预期结果构成的。一个好的测试用例是极有可能发现至今为止尚未发现的错误的测试用例。一次成功的测试是发现了至今为止尚未发现的错误的测试。

高效的测试是指用少量的测试用例，发现被测软件尽可能多的错误。

软件测试所追求的目标就是以尽可能少的时间和人力发现软件产品中尽可能多的错误。

2）软件测试准则

应该尽早地、不断地进行软件测试，把软件测试贯穿于开发过程的始终。

所有测试都应该能追溯到用户需求。从用户的角度看，最严重的错误是导致软件不能满足用户需求的那些错误。

应该从"小规模"测试开始，并逐步进行"大规模"测试。

应该远在测试之前就制定出测试计划。

根据Pareto原理，80%的错误可能出现在20%的程序模块中，测试成功的关键是怎样找出这20%的模块。

应该由独立的第三方从事测试工作。

对非法和非预期的输入数据也要像合法的和预期的输入数据一样编写测试用例。

检查软件是否做了应该做的事仅是成功的一半，另一半是看软件是否做了不该做的事。

在规划测试时不要设想程序中不会查出错误。

测试只能证明软件中有错误，不能证明软件中没有错误。

3）软件测试分类

从测试阶段划分，可分为单元测试、集成测试、确认测试。

从测试方法划分，可分为白盒测试、黑盒测试。

在实际应用中，一旦纠正了程序中的错误后，还应选择部分或全部原先已测试过的测试用例，对修改后的程序重新测试，这种测试称为回归测试。

2.单元测试

单元测试（Unit Testing），也称模块测试，通常可放在编程阶段，由程序员对自己编写的模块自行测试，检查模块是否实现了详细设计说明书中规定的功能和算法。单元测试主要发现编程和详细设计中产生的错误，单元测试计划应该在详细设计阶段制定。

单元测试期间着重从以下几个方面对模块进行测试：模块接口、局部数据结构、重要的执行通路、出错处理通路、边界条件等。

测试一个模块时需要为该模块编写一个驱动模块和若干个桩（stub）模块。驱动模块用来调用被测模块，它接收测试者提供的测试数据，并把这些数据传送给被测模块，然后从被测模块接收测试结果，并以某种可以看见的方式（例如显示或打印）将测试结果返回给测试者。桩模块用来模拟被测模块所调用的子模块，它接受被测模块的调用，检验调用参数，并以尽可能简单的操作模拟被调用的子程序模块功能，把结果送回被测模块。顶层模块测试时不需要驱动模块，底层模块测试时不需要桩模块。

模块的内聚程度高可以简化单元测试过程。如果每个模块只完成一种功能，则需要的测试方案数目将明显减少，模块中的错误也更容易预测和发现。

3.集成测试

集成测试（Integration Testing），也称组装测试，它是对由各模块组装而成的程序进行测试，主要目标是发现模块间的接口和通信问题。例如，数据穿过接口可能丢失；一个模块对另一个模块可能由于疏忽而造成有害影响；把子功能组合起来可能不产生预期的主功能；个别看来是可以接受的误差可能积累到不能接受的程度；全程数据结构可能有问题等。集成测试主要发现设计阶段产生的错误，集成测试计划应该在概要设计阶段制定。

集成的方式可分为非渐增式和渐增式。

非渐增式集成是先测试所有的模块，然后一下子把所有这些模块集成到一起，并把庞大的程序作为一个整体来测试。这种测试方法的出发点是可以"一步到位",但测试者面对众多的错误现象，往往难以分清哪些是"真正的"错误，哪些是由其他错误引起的"假性错误",诊断定位和改正错误也十分困难。非渐增式集成只适合一些非常小的软件。

渐增式集成是将单元测试和集成测试合并到一起，它根据模块结构图，按某种次序选一个尚未测试的模块，把它同已经测试好的模块组合在一起进行测试，每次增加一个模块，直到所有模块被集成在程序中。这种测试方法比较容易定位和改正错误，目前在进行集成测试时已普遍采用渐增式集成。

渐增式集成又可分为自顶向下集成和自底向上集成。自顶向下集成先测试上层模块，再测试下层模块。由于测试下层模块时它的上层模块已测试过，所以不必另外编写驱动模块。自底向上集成先测试下层模块，再测试上层模块。同样，由于测试上层模块时它的下层模块已测试过，所以不必另外编写桩模块。这两种集成方法各有利弊，一种方法的优点恰好对应于另一种方法的缺点，实际测试时可根据软件特点及进度安排灵活选用最适当的方法，也可将两种方法混合使用。

4.确认测试

确认测试（Va idation Testing）主要依据软件需求说明书检查软件的功能、性能及其他特征是否与用户的需求一致。确认测试计划应该在需求分析阶段制定。

软件配置复查是确认测试的另一项重要内容。复查的目的是保证软件配置的所有成分都已齐全，质量符合要求，文档与程序完全一致，具有完成软件维护所必须的细节。

如果一个软件是为某个客户定制的，最后还要由该客户来实施验收测试（Acceptance Testing），以便确认其所有需求是否都已得到满足。由于软件系统的复杂性，在实际工作中，验收测试可能会持续到用户实际使用该软件之后的相当长的一段时间。

如果一个软件是作为产品被许多客户使用的，不可能也没必要由每个客户进行验收测试。绝大多数软件开发商都使用被称为α（A pha）测试和β（Beta）测试的过程，来发现那些看起来只有最终用户才能发现的错误。

α测试由用户在开发者的场所进行，并且在开发者的指导下进行测试。开发者负责记录发现的错误和使用中遇到的问题。也就是说，α测试是在"受控的"环境中进行的。

β测试是在一个或多个用户的现场由该软件的最终用户实施的，开发者通常不在现场，用户负责记录发现的错误和使用中遇到的问题并把这些问题报告给开发者。也就是说，β测试是在"非受控的"环境中进行的。

经过确认测试之后的软件通常就可以交付使用了。

5.白盒测试

白盒测试，又称结构测试，主要用于单元测试阶段。它的前提是把程序看成装在一个透明的白盒子里，测试者完全知道程序的结构和处理算法。这种方法按照程序内部逻辑设计测试用例，检测程序中的主要执行通路是否都能按预定要求正确工作。

白盒测试常用的技术是逻辑覆盖，即考查用测试数据运行被测程序时对程序逻辑的覆盖程度。主要的覆盖标准有六种：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合条件覆盖和路径覆盖。

1）语句覆盖

语句覆盖是指选择足够多的测试用例，使得运行这些测试用例时，被测程序的每个语句至少执行一次。

很显然，语句覆盖是一种很弱的覆盖标准。

2）判定覆盖

判定覆盖又称分支覆盖，它的含义是，不仅每个语句至少执行一次，而且每个判定的每种可能的结果（分支）都至少执行一次。

判定覆盖比语句覆盖强，但对程序逻辑的覆盖程度仍然不高。

3）条件覆盖

条件覆盖的含义是，不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。

条件覆盖不一定包含判定覆盖，判定覆盖也不一定包含条件覆盖。

4）判定/条件覆盖

同时满足判定覆盖和条件覆盖的逻辑覆盖称为判定/条件覆盖。它的含义是，选取足够的测试用例，使得判定表达式中每个条件的所有可能结果至少出现一次，而且每个判定本身的所有可能结果也至少出现一次。

5）条件组合覆盖

条件组合覆盖的含义是，选取足够的测试用例，使得每个判定表达式中条件结果的所有可能组合至少出现一次。

显然，满足条件组合覆盖的测试用例，也一定满足判定/条件覆盖。因此，条件组合覆盖是上述5种覆盖标准中最强的一种。然而，条件组合覆盖还不能保证程序中所有可能的路径都至少经过一次。

6）路径覆盖

路径覆盖的含义是，选取足够的测试用例，使得程序的每条可能执行到的路径都至少经过一次（如果程序中有环路，则要求每条环路径至少经过一次）。

路径覆盖实际上考虑了程序中各种判定结果的所有可能组合，因此是一种较强的覆盖标准。但

路径覆盖并未考虑判定中的条件结果的组合，并不能代替条件覆盖和条件组合覆盖。

6.黑盒测试

黑盒测试，又称功能测试，主要用于集成测试和确认测试阶段。它把软件看做一个不透明的黑盒子，完全不考虑（或不了解）软件的内部结构和处理算法，它只检查软件功能是否能按照软件需求说明书的要求正常使用，软件是否能适当地接收输入数据并产生正确的输出信息，软件运行过程中能否保持外部信息（例如文件和数据库）的完整性等。

常用的黑盒测试技术包括等价类划分、边值分析、错误推测和因果图等。

1）等价类划分

在设计测试用例时，等价类划分是用得最多的一种黑盒测试方法。所谓等价类就是某个输入域的集合，对于一个等价类中的输入值来说，它们揭示程序中错误的作用是等效的。也就是说，如果等价类中的一个输入数据能检测出一个错误，那么等价类中的其他输入数据也能检测出同一个错误；反之，如果等价类中的一个输入数据不能检测出某个错误，那么等价类中的其他输入数据也不能检测出这一错误（除非这个等价类的某个子集还属于另一等价类）。

如果一个等价类内的数据是符合（软件需求说明书）要求的、合理的数据，则称这个等价类为有效等价类。有效等价类主要用来检验软件是否实现了软件需求说明书中规定的功能。

如果一个等价类内的数据是不符合（软件需求说明书）要求的、不合理或非法的数据，则称这个等价类为无效等价类。无效等价类主要用来检验软件的容错性。

黑盒测试中，利用等价类划分方法设计测试用例的步骤如下。

根据软件的功能说明，对每一个输入条件确定若干个有效等价类和若干个无效等价类，并为每个有效等价类和无效等价类编号。

设计一个测试用例，使其覆盖尽可能多的尚未被覆盖的有效等价类。重复这一步，直至所有的有效等价类均被覆盖。

设计一个测试用例，使其覆盖一个尚未被覆盖的无效等价类。重复这一步，直至所有的无效等价类均被覆盖。

应当特别注意，无效等价类用来测试非正常的输入数据，因此每个无效等价类都有可能查出软件中的错误，所以要为每个无效等价类设计一个测试用例。

2）边值分析

经验表明，软件在处理边界情况时最容易出错。设计一些测试用例，使软件恰好运行在边界附近，暴露出软件错误的可能性会更大一些。

通常，每一个等价类的边界，都应该着重测试，选取的测试数据应该恰好等于、稍小于或稍大于边界值。

将等价类划分法和边值分析法结合使用，更有可能发现软件中的错误。

3）错误推测

使用等价类划分和边值分析技术，有助于设计出具有代表性的、容易暴露软件错误的测试方案。但是，不同类型不同特征的软件通常又有一些特殊的容易出错的地方。错误推测法主要依靠测试人员的经验和直觉，从各种可能的测试方案中选出一些最可能引起程序出错的方案。

4）因果图

因果图法是根据输入条件与输出结果之间的因果关系来设计测试用例的，它首先检查输入条件的各种组合情况，并找出输出结果对输入条件的依赖关系，然后为每种输出条件的组合设计测试用

例。

　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件维护

4.3.2 软件维护

软件维护就是在软件交付使用之后直至软件被淘汰的整个时期内为了改正错误或满足新的需求而修改软件的活动。

软件维护的代价是很大的，据1994年Software Engineering Encyclopedia记载，20世纪80年代末用于软件维护的花费约为整个软件生命周期总花费的75%,而且还在逐年上升。

1.软件维护类型

根据引起软件维护的原因，软件维护通常可分为以下四种类型：

1）改正性维护

改正性维护是指在使用过程中发现了隐蔽的错误后，为了诊断和改正这些隐蔽错误而修改软件的活动。

2）适应性维护

适应性维护是指为了适应变化了的环境而修改软件的活动。

3）完善性维护

完善性维护是指为了扩充或完善原有软件的功能或性能而修改软件的活动。

4）预防性维护

预防性维护是指为了提高软件的可维护性和可靠性、为未来的进一步改进打下基础而修改软件的活动。

2.软件的可维护性

软件的可维护性是指理解、改正、改动、改进软件的难易程度。根据Boehm质量模型，通常影响软件可维护性的因素有可理解性、可测试性和可修改性。

1）可理解性

可理解性是指维护人员理解软件的结构、接口、功能和内部过程的难易程度。

2）可测试性

可测试性是指测试和诊断软件错误的难易程度。

3）可修改性

可修改性是指修改软件的难易程度。

为了提高软件的可维护性，在软件生命周期的各个阶段都必须充分考虑维护问题。先进的软件工程方法是软件可维护的基础保证。

面向对象方法学的对象封闭机制、消息通信机制、继承机制和多态机制从根本上提高了软件的可理解性、可测试性和可修改性。

结构化设计的几条主要原则，如模块化、信息隐蔽、高内聚、低耦合等，对于提高软件的可理

解性、可测试性和可修改性也都有重要的作用。

另外，书写详细正确的文档、书写源文件的内部注解、使用良好的编程语言、具有良好的程序设计风格，也有助于提高软件的可理解性。使用先进的测试工具、保存以前的测试过程和测试用例，则有助于提高软件的可测试性。

3.软件维护管理

软件维护管理是指为保证维护质量、提高维护效率、控制维护成本而进行的维护过程管理，它要求对软件的每次"修改"均需经过申请、评估、批准、实施、验证等步骤。

软件维护管理的核心是维护评估和维护验证。维护评估的主要工作包括：判定维护申请的合理性与轻重缓急、确定维护的可行性与时间及费用、制定维护策略与维护计划等。维护验证主要审查修改后的软件是否实现了维护目标、软件文档是否也做了相应修改等。

上一节　本书简介　下一节

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件工具与软件开发环境

4.4 软件工具与软件开发环境

本节将介绍软件工具与软件开发环境。

上一节　本书简介　下一节

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

软件工具

4.4.1 软件工具

软件工具是指用于辅助软件开发、运行、维护、管理、支持等过程中的活动的软件，通常也称为CASE（Computer Aided Software Engineering,计算机辅助软件工程）工具。

软件开发工具种类繁多，很难有一种统一分类方法。由于大多数软件工具仅限于支持软件生命周期过程中的某些特定的活动，通常可按软件过程的活动分为软件开发工具、软件维护工具和软件管理工具等。

1.软件开发工具

1）需求分析工具

主要包括支持结构化方法的数据流图、数据字典和支持面向对象方法的类图、用例图和状态图等。

2）设计工具

主要包括概要设计阶段的模块结构图、层次图、HIPO图和详细设计阶段的程序流程图、盒图

（N–S图）、PAD图和过程设计语言（PD）等。

面向对象的设计工具一般与分析阶段使用的工具一致，可以通称分析设计工具。

3）编程工具

主要包括编辑程序、汇编程序、编译程序、构造程序（Buider）和调试程序等。

4）测试工具

包括静态分析工具、动态测试工具和测试数据自动生成工具等。

2.软件维护工具

1）版本控制工具

用来存储、更新、恢复和管理一个软件的多个版本。

2）文档分析工具

用来对软件开发过程中形成的文档进行分析，给出软件维护活动所需的维护信息。

3）开发信息库工具

用来维护软件项目的开发信息，包括对象、模块等。

4）逆向工程工具

在软件生命周期中，将某种形式表示的软件转换成更高抽象形式表示的软件活动称为逆向工程。逆向工程工具就是辅助软件人员进行这种逆向工程活动的软件工具，如反汇编工具、反编译工具等。

5）再工程工具

用来支持重构一个功能和性能更为完善的软件系统。目前的再工程工具主要集中在代码重构、程序结构重构和数据重构等方面。

3.软件管理工具

1）项目管理工具

用来辅助软件的项目管理活动（包括项目的计划、调度、通信、成本估算、资源分配及质量控制等）。

2）配置管理工具

用来辅助完成软件配置项的标识、版本控制、变化控制、审计和状态统计等基本任务，使各配置项的存取、修改和系统生成易于实现，从而简化审计过程、改进状态统计、减少错误、提高系统质量。

3）软件评价工具

用来辅助管理人员进行软件质量保证的有关活动。

上一节　本书简介　下一节

软件开发环境

4.4.2 软件开发环境

软件开发环境是指支持软件产品开发的软件系统。

集成型软件开发环境是一种把支持多种软件开发方法和开发模型、支持软件开发全过程的软件工具集成在一起的软件开发环境。这种环境通常应具有开放性和可剪裁性。开放性为将环境外的工具集成到环境中来提供方便；可剪裁性根据不同的应用或不同的用户需求进行剪裁，以形成特定的开发环境。

集成型开发环境通常可由工具集和环境集成机制两部分组成。环境集成机制主要有数据集成机制、控制集成机制和界面集成机制。

1）数据集成机制

数据集成机制提供统一的数据模式和数据接口规范，需要相互协作的工具通过这种统一的模式与规范交换数据。数据集成可以有不同的层次，如共享文件、共享数据结构和共享信息库等。

2）控制集成机制

控制集成机制支持各工具或各开发活动之间的通信、切换、调度和协同工作，并支持软件开发过程的描述、执行和转接。通常使用消息通信机制实现控制集成，工具间发送的消息统一由消息服务器进行管理。

3）界面集成机制

界面集成机制为统一的工具界面风格和统一的操作方式提供支持，使得环境中的工具具有相同的视觉效果和操作规则，减少用户为学习不同工具的使用所花费的开销。界面集成主要体现在相同或相似的窗口、菜单、工具条、快捷键、操作规则与命令语法等。

上一节　本书简介　下一节

软件质量保证

4.5 软件质量保证

本节将介绍软件质量保证。

上一节　本书简介　下一节

软件质量

4.5.1 软件质量

概括地说，软件质量就是软件与明确地和隐含地定义的需求相一致的程度。具体地说，软件质量是软件与明确叙述的功能和性能需求、文档中明确描述的开发标准，以及任何专业开发的软件产品都应该具有的隐含特征相一致的程度。

软件质量具有以下3个要点：

用户需求是衡量软件质量的基础，与需求不一致就无质量可言。

指定的开发标准定义了一组指导软件开发的准则。如果没有遵守这些准则，肯定会导致软件质量不高。

通常还有一些没有明确写进用户需求说明书但开发人员都应当了解的隐含需求（例如易理解性、易修改性等）。如果软件仅满足明确描述的需求，但不满足这些隐含的需求，那么软件的质量仍然是值得怀疑的。

计算机软件是一种复杂、抽象的逻辑实体，它所固有的一些特点包括：抽象性、复杂性、多样性、易变性、软件开发需求难于把握等。所有这些软件独具的特点都增加了软件开发的困难。

影响软件质量的因素主要包括：

人的因素；

软件需求；

质量问题可能出现在开发过程的各个环节上；

测试的局限性；

质量管理的困难；

质量管理未能给予足够的重视；

软件人员的传统习惯；

开发规范；

开发工具的支持不够。

上一节　本书简介　下一节

软件质量特性

4.5.2 软件质量特性

软件质量特性可用多种软件质量模型来描述。本书只介绍ISO/IEC 9126软件质量模型和Mc Call软件质量模型。

1.ISO/IEC 9126软件质量模型

国际标准化组织和国际电工委员会发布了关于软件质量的标准ISO/IEC 9126-1991.中国于1996年将其等同采用，成为国家标准《GB/T16260-1996软件产品评价、质量特性及其使用指南》。ISO/IEC 9126软件质量模型由3个层次组成：第一层是6个质量特性，第二层是21个质量子特性，第三层是度量指标。该模型的质量特性和质量子特性如表4-46所示。

表4-14 质量特性和质量子特性

质 量 特 性	质量子特性
功能性（functionality）	适宜性（suitability）
	准确性（accurateness）
	互用性（interoperability）
	依从性（compliance）
	安全性（security）
可靠性（reliability）	成熟性（maturity）
	容错性（fault tolerance）
	可恢复性（recoverability）

上一节　本书简介　下一节

软件质量保证

4.5.3 软件质量保证

软件质量保证是为保证软件系统充分满足用户要求的质量而进行的有计划、有组织的活动，其目的是生产高质量的软件。

软件质量保证的主要困难表现在以下几方面。

软件开发的管理人员往往更关心项目开发的成本与进度。因为成本和进度是显而易见的，而软件质量则难以度量。

如果软件开发的管理人员对于交付的软件含有多少隐患不必负什么责任，他们必定没有太高的热情去控制开发的质量，更不必说保证质量。

开发人员的习惯一旦形成便难以改变，他们的行为也难于控制。而高质量的软件产品，又主要取决于参与开发的人员。

复杂的软件项目需要许多技术人员和管理人员参与，对问题的不同认识和误解如不能及时消除，必然影响软件质量。

软件开发人员的频繁流动，特别是骨干开发人员的流失，也会使软件质量受到一定影响。

软件质量保证的主要手段如下。

开发初期制定质量保证计划，并在开发中坚持实行。

开发前选定或制定开发标准或开发规范，并遵照实施。

从选择分析设计方法和工具，形成高质量的分析模型和设计模型。

严格执行阶段评审，以便及时发现问题。

各个开发阶段的测试。

对软件的每次"变动"都要经过申请、评估、批准、实施、验证等步骤。

软件质量特性的度量化。

软件生存期的各阶段都要有完整的文档。

上一节　本书简介　下一节

 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

软件工程标准和软件文档

4.5.4 软件工程标准和软件文档

标准化是一门综合性学科，其工作内容极为广泛，可渗透到各个领域。标准化工作的特征包括横向综合性、政策性和统一性。本书只介绍软件工程标准。

1.软件工程标准

按制定软件工程标准的不同层次和适用范围，软件工程标准可分为五级：

国际标准：由国际联合机构制定和公布，提供各国参考的标准。例如以ISO起头的标准是国际标准化组织发布的标准，以IEC起头的标准是国际电工委员会标准。

国家标准：由政府或国家级的机构制定或批准，适用于全国范围的标准。例如以GB起头的标准是中华人民共和国标准，以FIPS起头的标准是美国国家标准局发布的标准。

行业标准：由行业机构、学术团体或国防机构制定，并适用于某个业务领域的标准。例如以GJB起头的标准是指中华人民共和国军用标准，以IEEE起头的标准是国际电气和电子工程师学会标准。

企业规范：一些大型企业或公司，由于软件工程工作的需要制定的适用于本部门的规范。

项目规范：由某一科研生产项目组织制定，且为该项任务专用的软件工程规范。

2.软件文档资料

国家标准局1988年1月批准并发布的《GB8567-1988计算机软件产品开发文件编制指南》规定在一项软件开发过程中应该产生14种文件：

A可行性研究报告

B项目开发计划

C软件需求说明书

D数据要求说明书

E概要设计说明书

F详细设计说明书

G数据库设计说明书

H用户手册

I操作手册

J模块开发卷宗

K测试计划

测试分析报告

M开发进度月报

N项目开发总结报告

其中管理人员主要使用的有项目开发计划、可行性研究报告、模块开发卷宗、开发进度月报和项目开发总结报告；开发人员主要使用的有项目开发计划、可行性研究报告、软件需求说明书、数据要求说明书、概要设计说明书、详细设计说明书、数据库设计说明书、测试计划和测试分析报告；维护人员主要使用的有设计说明书、测试分析报告和模块开发卷宗。

软件项目管理

4.6 软件项目管理

软件项目本身是复杂的，如果没有仔细地计划，复杂的项目是不可能成功的。一个计划良好的项目将受到有效的控制，进展明显，而参加该项目的人员都会得到支持以进行其工作。软件项目本身也具有风险性，如果没有有效的风险管理也是不能成功的。

通常软件工程项目的管理比其他工程项目的管理更困难，这是因为：

软件产品不可见。开发的进度及产品的质量是否符合要求不明显，比较难进行把握。

没有标准的软件过程。尽管近几年来"软件过程改进"领域有许多进步，但由于团队、人员的个性化因素，还不存在一个放之四海皆真理的标准化软件过程。

大型软件项目常常是一次性项目。由于这些项目都是"前无古人"的，因此缺乏可以借鉴的历史经验。

版权方授权希赛网发布，侵权必究

上一节　本书简介　下一节

第 4 章：软件工程基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月27日

软件项目管理的内容

4.6.1 软件项目管理的内容

软件工程项目管理的对象是软件工程项目，它涉及到整个软件开发过程。最核心的内容就是在成本、质量与进度之间做出平衡的取舍，如图4-8所示。

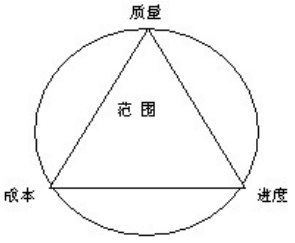


图4-8 项目管理铁三角

抽象地说，其实软件项目管理也十分简单，主要包括POIM四个方面：即Pan（计划）、Organize（组织）、Impement（实现）、Measurement（度量）。细化地说，主要包括以下一些主要的活动。

1.启动软件项目

通常，在启动软件项目之前，需要确定出项目的目标和范围，并根据这些东西来考虑解决方案，然后根据解决方案进行社会、经济、技术可行性分析，以确定是否启动项目。

然后再根据合理的、精确的成本估算，对其进行切实可行的任务分解，以及管理的进度安排，也就是完成最初的项目计划。

2.度量

在管理学中有一句名言："如果不能够用数字来描述它，那么说明你还没有了解它".在软件开发过程中也是一样，度量是帮助有效地定量管理的基础。度量的目的是为了把握软件工程过程的实际情况和它所生产的产品质量。

3.估算

制定项目计划是一个十分关键、重要的活动，但要想做出合理、有效的项目计划，就必须对需要的人力、项目的持续时间、成本做出相应的估算。

估算的基础是历史数据及经验模型。估算通常可以使用"自顶向下"和"自底向上"两种模型，在估算的过程中常见的经验模型包括FP（功能点）和COCOMO系列。而要想获得更加准确、符合项目组队实际情况的估算值的话，就必须建立完整的历史项目资料库。

4.风险分析

正如Tom Gib在其与软件工程管理相关的一本著作中说过："如果谁不主动攻击风险，那么它们就会主动地攻击谁。"项目中风险无处不在，但却有很多项目管理者对此熟视无睹，从而带来了许多巨大的损失。在系统开发过程中，对潜在的风险进行分析，记录下来，按其发生的可能性和影响性进行排序，并制订相应的解决方案和预防措施。

试想，如果风险按你的预料"如期而至"时，你从容不迫地按照预算设定的解决方案来解决，将会给团队多大的信心？

5.进度安排

每一个软件项目都应该制定一个进度安排，而制定进度计划时需要考虑的问题包括：预先如何对进度进行计划？工作如何到位？如何识别定义的任务？如何监控任务的完成？如何设立分隔任务的里程碑？

进度安排通常是建立在WBS（工作任务分解）的基础上，对时间、人员、设备等资源进行统一的分配，根据估算的工作量进行计划。最常见的进度计划的工具包括甘特图、PERT图等，而Microsoft Project则可以帮助使用者更好地应用这些工具进行进度的管理。

6.追踪和控制

正如拿破仑所说："没有一场战争是按计划完成的，但没有一场战争没有计划",这句话辩证地说明了计划与执行之间的关系。只有计划没有执行项目是不可能成功的，因此我们需要对计划进行有效的追踪和控制，根据实际的执行情况对其进行有效的调整。

上一节　本书简介　下一节

软件项目管理的三个阶段

4.6.2 软件项目管理的3个阶段

如果从项目的实施过程的工作性质进行划分的话，那么整个软件项目开发过程可以分成三个主

要阶段。

项目启动阶段：项目立项、项目初步需求调研/可行性分析、项目初步计划。

项目实施阶段：这个阶段的整个过程如图4-9所示。

图 4-9　项目实施阶段

也就是说，项目经理在项目实施过程中的工作是：制定计划（计划），按计划布置任务（组织、实现），并在过程中不断地对进度进行监控（度量），根据实际进度情况调整计划，并处理需求变更引起的项目计划变更。

项目关闭阶段：组织验收测试，组织项目回顾，存储项目数据。

上一节　本书简介　下一节

软件项目估算

4.6.3 软件项目估算

在计算技术发展的早期，软件的成本在整个计算机系统的总成本中占的只是一个较小的百分比，因此即使软件项目估算有很大的误差，对整个项目而言影响还是相对较小的。但随着信息化的深入普及与发展，软件在整个项目中的比重也越来越大，因此如果无法精确地进行软件项目的估算，将会给整个计算机系统的预算带来很大的麻烦。

软件项目的估算策略包括"自顶向下"和"自底向上"两种，而估算的内容主要包括：软件规模估算、软件工作量估算及成本估算三个方面。

1.两种软件估算策略

不同的估算策略能够应用于不同的场合，也将带来完全不同的效果。

1）自顶向下估算法

自顶向下的估算法，通常是由项目经理自身或者是以项目经理为主的一个核心小组（通常包括分析师、构架师、主程序员等主要的角色）来完成的。其工作的特点是：先根据用户、决策者的要求，确定一个时间期限。然后根据这个时间期限进行分解，将开发工作进行对号入座，以获得一个可以满足这个期限的估算。

这种方式是一种通常采用的方法，但其并不能够有效地解决项目估算的问题，经常容易使得估算值与实际值产生很大的差异。

2）自底向上估算法

自底向上估算法则采用了一种完全不同的策略。首先在核心小组内进行头脑风暴，完成工作任务分解，直到每一个任务块都小到能够得出合理的估算为止（通常是两周以内的工作量）。然后将每个任务根据项目成员的技能特点、兴趣特长进行分配，并要求其对此做出估算。最后将这些估算值合并在一起，得到总的项目估算。

这种方式通常能够得到较为客观的、可操作的估算结果，而且还能够使得项目组成员主动地参

与，并且通常能够对自己所做的承诺全力守信，从而为项目树立了一个良好的文化。但由于其通常得到的值要远比预期的值大，时间更久，因此许多项目不能够有效地使用它。

2.软件规模估算

软件规模也就是需要完成的工作范围是多少，这个估算结果是整个软件项目估算的基础。对于软件规模估算来说，最常见的是OC和FP估算法。

1）OC估算法

也就是估算软件的代码行数（ine Of Code），通常使用KOC（千代码行）为单位。这种估算的基础是将软件项目切分为一个个小模块，然后通过历史的项目经验数据，以及开发人员的经验，对其需要的OC数进行估算。

2）FP估算法

FP（功能点）是一种衡量工作量大小的单位，它的计算方法是：功能点=信息处理规模×技术复杂度。其中，技术复杂度=0.65+调节因子。

（1）信息处理规模

FP方法通过外部输入数、外部输出数、外部查询数、内部逻辑文件数、外部接口文件数五个方面来衡量整个软件系统的信息处理规模。计算的方法如表4-16所示。

表4-16 FP基本功能点计算公式

	低	平均	高	说明
Input	×3	×4	×6	Screen、From、etc.
Output	×4	×5	×7	Screen、Report、etc.
Inquire	×3	×4	×6	查询、处理请求
File	×7	×10	×15	内部所需的文件
Interface	×5	×7	×10	引用外部的数据

（2）技术复杂度

而技术复杂度是从数据通信复杂度、分布式处理复杂度、性能复杂度、配置项负载复杂度、事务率复杂度、在线数据项复杂度、用户使用效率复杂度、在线更新复杂度、复杂处理复杂度、重用性复杂度、安装容易程度复杂度、操作容易程度复杂度、多个地点复杂度、修改容易程度复杂度14个方面进行微调。每个方面都根据其复杂程度，在0~0.05之间取值。将这些值全部相加就可以得到调节因子，再加上0.65就可以得到技术复杂度。

3.软件工作量估算

当通过OC或FP估算获得了软件的规模之后，就可以进行工作量的估算了。工作量的单位通常是人月（对于大项目可以用人年，小项目则可以用人天）。从中不难得知，软件工作量的获得应该是：规模/产能=工作量。

例如，如果估算出项目需要5KOC,而每个人一个月的产能是1KOC的话，那很显然需要5个人月的时间来完成该项目。从中，我们也可以看出建立项目过程数据库，保存这些数字对于估算而言是十分重要的。

如果你没有足够的可类比的项目数据做参照，那么也可以借助一些经验模型来进行估算，从而获得软件的工作量和进度的值。常见的经验模型包括大名鼎鼎的COCOMO,以及相对较简单的IBM模型、普特南（Putnam）模型三种。

1）IBM模型

IBM模型是Waston和Feix在1977年发布的，是基于IBM联合系统分部负责的60个项目的数据进行总结而得到的。

$E=5.2\times L^{0.91}$ （E 为工作量，单位为人月；L 为源代码行数，单位为 KLOC）

$D=4.1\times L^{0.36}=13.47\times L^{0.35}$ （D 为项目持续时间，以月计）

$S=0.54\times E^{0.6}$ （S 是人员需求量，以人计）

$DOC=49\times L^{1.01}$ （DOC 是文档数量，单位为页）

从上面可以看出，IBM是一个静态的模型，而其取样只有60多个项目，因此有很大的局限性，有时需要根据具体的情况，对公式的参数做一定的修改。

2）普特南模型

这是普特南在1978年提出的一种动态多变量模型。它通过建立一个"资源需求曲线模型"来导出一系列等式，模型化资源特性。而这个曲线称为Rayeigh-Norden曲线。它所导出的"软件方程"如下所示：

$$L = Ck \bullet K^3 \bullet td^3$$

其中，td是开发持续的时间，以年计；K是软件开发与维护在内的整个生存期所花费的工作量，以人年计；是源代码行数，以OC计；Ck是技术状态常数，它因开发环境不同而不同。如果开发环境较差（没有系统开发方法，缺乏文档与评审，批处理方式），则通常取值为2 000;如果开发环境较好（有合适的系统开发方法，有充分的文档与评审，交互执行方式），则通常取值8 000;而如果开发环境较优（有自动开发工具和技术），则通常取值为11 000.

3）COCOMO模型

COCOMO模型是TRW公司开发的，是软件工作量估算的最有代表性的方法。在该模型中将使用到以下三个基本量：

DSI（源指令条数）：只包括有效的代码，不包括注释语句，如果一行有两个语句，则算一条指令。1KDSI=1024DSI.

MM（开发工作量）：单位为人月。定义1MM=19人天=152人时=1/12人年。

TDEV（开发进度），单位为月，由MM决定。

而COCOMO模型将项目分成为三种类型：

组织型：相对较小、较简单的项目。通常需求不是很苛刻，开发人员比较熟悉项目目标，掌握了充分的技能，硬件约束少，规模小于5万行。多数应用程序、操作系统、编译程序属于此种。

嵌入型：硬件、软件和操作限制较多，常与硬件紧密结合。大型的复杂事务处理程序、操作系统、控制系统、指挥系统属于此种。

半独立型：要求介于以上两者之间，规模和复杂性中等以上，最大可达30万行。大多数事务处理系统、新的操作系统及数据库管理系统、简单的指挥系统属于此种。

根据考虑的因素的多少，详细程度的不同，可以将COCOMO模型分为三种：基本COCOMO模型、中间COCOMO模型和详细COCOMO模型。

基本COCOMO模型：也是一种静态模型。

组织型：$MM=2.4(KDSI)^{1.05}$ $TDEV=2.5(MM)^{0.38}$

嵌入型：$MM=3.6(KDSI)^{1.20}$ $TDEV=2.5(MM)^{0.32}$

半独立型：$MM=3.0(KDSI)^{1.12}$ $TDEV=2.5(MM)^{0.35}$

中间COCOMO模型：先根据基本COCOMO模型计算出工作量（MM），然后再使用下列公式对其进行调整，最后算出开发所需时间（TDEV）。

$$MM = r\times\prod_{i=1}^{15} f \times(KDSI)^c$$

其中f_i是指15个影响软件工程量的因素，包括软件可靠性、数据库规模、产品复杂性、时间限

制、存储限制、机器、周转时间、分析员能力、工作经验、程序员能力、工作经验、语言使用经验、使用现代程序设计技术、使用软件工具及工期等。

详细COCOMO模型：在中间COCOMO模型的基础上，针对每一个影响因素按模块层、子系统层、系统层，做出三张工作量因素分级表，供不同层次的估算使用。

4.成本估算

当估算出工作量（也就是人月数），以及知道的人员需求量、项目的持续时间之后，就可以进一步进行成本估算。也就是将人员的工资及相关管理费用之和，去乘以其所需要的时间，就可以得到人员成本。

最后再加上相关的资源成本（如软硬件资源）、日常相关的其他开支成本等，就可以得到一个相对准确的成本估算值。

5.常用的估算辅助方法

在日常的估算活动中，还可以采用一些相关的辅助方法，最常见的包括Dephi及Standard-component两种，如图4-10所示。

1）Dephi法，专家判定技术

召集一组专家，每人发一份问题描述和估算表；

让他们一起讨论项目目标和假设；

匿名提交一份项目任务列表和估算；

由主持人将结果绘制成一系列估算结果表；

在每个专家的表中，除该专家的值标明，其他的均匿名；

专家讨论该结果，对自己估算进行修改；

经过2~3回合，得到一致的结果。

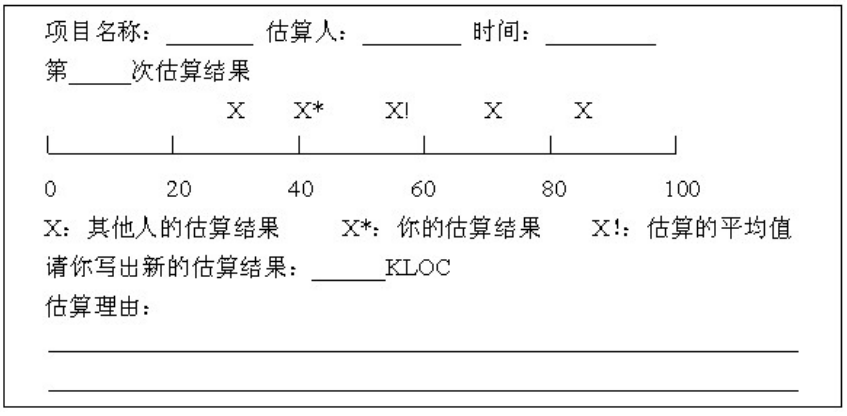


图4-10 专家估算表

2）Starndard-component方法

为了使得估算值更为准确，我们也可以借助以下公式来进行调整：

估计值=（乐观值+ 4×最可能值+悲观值）÷6

上一节 本书简介 下一节

4.6.4 软件项目组织与计划

从项目经理的角度来看，整个项目实施的过程，就像驾车开往某地一样。首先要根据目的地的距离（类比项目的需求范围）、车况（类比团队的人员结构、平均产能）、时间的要求（完成期限）来制订合理的行驶计划（类比软件的项目计划）。

接着，就按照既定的计划执行。

在这个过程中，每隔一段时间检查一下进度情况，也就是通过观察里程碑，来确定自己已经行驶过的路程（即已完成的部分），并与计划进行比较，然后根据实际的情况，调整计划。例如，如果全长300千米，预计3小时完成，而1小时后，你发现由于路上遇到了一些问题，只行驶了80千米，那么，若要按计划完成，则必须提高时速。这就是监控和计划修正的过程。

另外，如果在行驶的过程中，你得到了新的指示，目的地有了改变（类比项目的需求变更），那么也需要停下来，重新制订计划。

1.计划和执行

项目计划的优劣，直接影响到是否能够准确地进行进度的监控、人员的安排等各方面的事情。因此，可以说在某种意义上项目计划直接决定项目的成败。

那么项目计划应该如何来制订、修正呢？项目计划包括两个部分：项目组计划和个人项目计划。正如表4-17所示，在整个项目过程中，项目计划是一个逐步求精的过程。

表4-17 项目计划逐步求精过程

阶　段	子 阶 段	项 目 计 划	制 定 依 据	团 队 组 成
项目启动		初步项目计划（只需包括各主要阶段）	初步项目需求	项目经理 系统分析员
项目实施	需求分析	项目组计划（细化需求分析阶段）；	初步项目计划 初步项目需求	在原来基础上增加构架设计师
	系统分析设计	项目组计划（细化系统分析设计阶段）；系统分析员、构架设计师的个人项目计划	需求说明书 项目组计划	在原来基础上增加主要开发人员
	编码/单元测试	项目组计划（细化编码实现阶段）；开发人员及测试人员的个人项目计划	需求说明书 系统高层设计（体系结构）	相应的开发人员及测试人员到位
	集成/验收测试	项目组计划（细化集成/验收测试阶段，即测试计划）	需求说明书 系统高层设计	撤出部分开发人员

而这些计划的编制，应该包括两个重要的图表：人员职责矩阵和甘特图（也可以用PERT图），如表4-18和图4-11所示。

表4-18 人员职责矩阵

	人员 1	人员 2	人员…	人员 n
任务 1				
任务 2				
任务…				
任务 n				

图4-11 甘特图示例

2.进度监控与计划修正

在项目的实施阶段，需要不断地进行进度的监控，并且根据监控的结果修正项目计划，并将项目进度的实际情况通报市场部门相关人员，并且让客户了解实际的进展。

对于项目经理来说，不同的阶段中，进度监控工作有一些不同。在需求分析、系统分析与计划两个子阶段，可以将生成的文档等制品通过评审作为里程碑，跟踪完成的情况，还分析是否按进度进行。而在编码/单元测试阶段则可以基于每个开发小组（也可以是1个人）的微进度计划来做EVA分析。下面我们主要说明如何采用EVA分析法进行进度的监控。

1）EVA分析法简介

已获值分析（EVA）是最常用的项目进度监控方法。EVA是通过计算实际已花费在项目上的工作量，来预计该项目所需的成本和完成时间日期的一种方法。它依赖于"已获值"的测量，"已获值"使用了以下3种数据。

对已完成的工作部分，原来预算花费的成本（BCWP），即"完成了多少工作？";

对已完成的工作部分，实际花费的成本（ACWP），即"我们实际花费了多少成本？";

原计划到分析日期为止的总成本预算（BCWS），即"到该日期原来应该完成多少工作？".

你可以根据需要每周或每月计算一次，在我的实践中是采用每周计算一次的方法。

从上述的3个基本数据中，可以得到四个确定进度情况的数据。

（1）进度偏差（SV）

计算公式：SV=BCWP–BCWS

如果SV等于0,说明项目完全按进度进行；如果SV小于0,说明已落后进度；如果SV大于0,说明超前于进度。

（2）进度效能指标（SPI）

计算公式：SPI=BCWP/BCWS

如果SPI为1,说明项目按进度表进行；如果SPI小于1,说明落后于进度；若SPI大于1,说明超前于进度。

（3）成本偏差（CV）

计算公式：CV=BCWP–ACWP

如果CV等于0,说明项目按预算进行；如果CV小于0,说明超出预算；若CV大于0,说明低于预算。

（4）成本状况指标（CPI）

计算公式：CPI=BCWP/ACWP

如果CV等于1,说明项目按预算进行；如果CV小于1,说明超出预算；若CV大于1,说明低于预算。

通过这些数据，我们还可以预测项目的进度与完成时间：

BAC:原来的总预算成本。

EAC:预计项目的最终成本，用于在过程中估计项目的最终成本，EAC=BAC/CPI.

SAC:预计项目的最终进度，用于在过程中估计项目的完成时间，SAC=进度/SPI（进度是指计划完成任务的时间）。

VAC:预计总成本偏差，VAC=BAC–EAC.

2）应用EVA分析法

下面，就将EVA分析法实际地应用到项目中去。例如，某个构件的开发人员小组制定的微进度计划如表4-19所示。

表4-19 某构件的微进度计划

任　务	工作量（人日）	估计完成（第几周）	开 发 人 员
详细设计	5	1	Pa
模块开发	21	3	Pa,Pb,Pc
单元测试	3	3	Pa,Pb,Pc
模块集成	4	3	Pa,Pb
总计	33		

根据这样的微进度计划，我们可以得到这个构件组的计划工作量（BCWS）：

第1周：BCWS=5.

第2周：模块开发是跨2~3周的工作，预计在本周将获取15人日的价值，累计后BCWS=15+5=20.

第3周：模块开发的延续，获取6个人日的价值，加上单元测试、模块集成，共有6+3+4=13个人日，累计BCWS=33.

当项目进行一周后，周五下午，项目经理和构件开发组成员共同进行已获值数据采集，统计出BCWP.如表4-20所示是进行了2周后的已获值分析情况。

表4-20 已获值分析

任　务	工作量（人日）	完成的百分数（%）	已 获 取
详细设计	5	100	5
模块开发	21	60%	12.6
单元测试	3	0	
模块集成	4	0	
总计	33		17.6(BCWP)

而ACWP则是已付出的人日。第1周投入1个人工作5天，计5人日；第2周则投入了3个人，工作5天，不过其中有1个人被抽调出1天，因此合计为5*3–1=14人日，合计已投入19人日，即ACWP=19.

通过上面的分析，得到：

BCWS=20

BCWP=17.6

ACWP=19

于是：

SV=BCWP–BCWS=17.6–20=–2.4（你落后于进度）

SPI=BCWP/BCWS=17.6/20=0.88（你以进度计划的88%效能在工作）

CV=BCWP–ACWP=17.6–19=–1.4（你已超出预算1.4人日）

CPI=BCWP/ACWP=17.6/19=0.926（你以超出预算约7.4%的状态在工作）

预测：

EAC=BAC/CPI=33/0.926=35.6人日

VAC=BAC–EAC=33–35.6=–2.6人日（你将超出预算2.6个人日）

SAC=3/0.88=3.4周（原计划3周，估计你将花费3.4周）

（3）修正项目计划

当你遇到这样的情况时，就需要问自己：为何比计划多花费了工作量？项目计划合理吗？理解任务吗？是否有分心的事？有什么未发现的障碍？然后对项目计划进行适当的调整，并与开发人员一同寻找解决方案，然后修正项目计划。

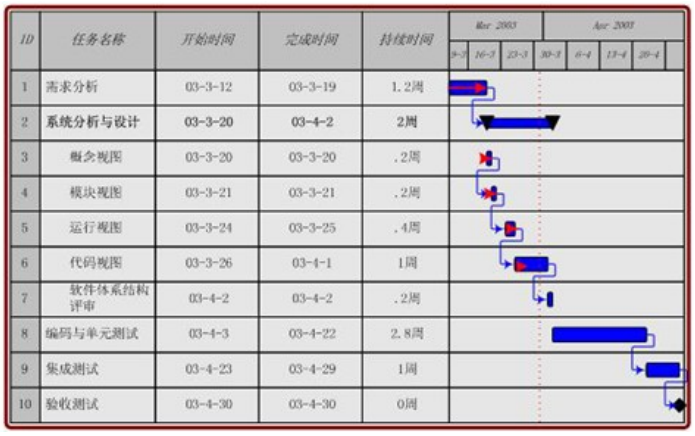
4）提交项目进度周报

项目经理根据监控的结果，编写项目进度周报，提交给市场部门的相关人员，并由市场部门相

关人员，将其以另一种形式反馈给客户。

5）更新甘特图

另外，项目经理可以根据项目的进度情况，更新跟踪甘特图。跟踪甘特图是在甘特图的基础上，加上以红色箭头一目了然地表示出进度的完成情况，如图4-12所示就是一个跟踪甘特图的实例。



ID	任务名称	开始时间	完成时间	持续时间
1	需求分析	03-3-12	03-3-19	1.2周
2	系统分析与设计	03-3-20	03-4-2	2周
3	概念视图	03-3-20	03-3-20	.2周
4	模块视图	03-3-21	03-3-21	.2周
5	运行视图	03-3-24	03-3-25	.4周
6	代码视图	03-3-26	03-4-1	1周
7	软件体系结构评审	03-4-2	03-4-2	.2周
8	编码与单元测试	03-4-3	03-4-22	2.8周
9	集成测试	03-4-23	03-4-29	1周
10	验收测试	03-4-30	03-4-30	0周

图4-12 跟踪甘特图示例

在图4-12中，红色虚线代表现在的时间，左边是过去，右边是将来。而红色箭头线则表示完成情况。从上图中，我们可以发现"代码视图"工作已经落后于进度了。在实际应用中，你可以使用Project来完成甘特图和跟踪甘特图的制作。

上一节 本书简介 下一节

第 4 章：软件工程基础知识 作者：希赛教育软考学院 来源：希赛网 2014年01月27日

风险管理

4.6.5 风险管理

我们为什么要在软件项目中进行风险管理呢？一是关心未来，风险是否会导致软件项目失败；二是关心变化，在用户需求、开发、技术、目标机器，以及所有其他与项目有关的实体中会发生什么变化；三是必须解决选择问题：应当采用什么方法和工具，应当配备多少人力，在质量上强调到什么程度才满足要求。

风险管理通常包括3个主要活动：风险识别、风险估计和风险驾驭。

1.风险识别

可以用不同的方法对风险进行分类。从宏观上来看，风险可以分为项目风险、技术风险和商业风险。项目风险识别潜在的预算、进度、个人、资源、用户和需求方面的问题。技术风险包括识别潜在的设计、实现、接口、检验和维护方面的问题。而商业风险则主要来源于市场。

风险识别的重要工作就是将潜在的风险找到，文档化。

2.风险估计

风险估计使用两种方法来估计每一种风险。一种方法是估计其发生的可能性；另一种方法是估计它可能带来的破坏性。然后根据这样的结果对其进行排列优先级，对于那种可能性大、破坏力也

大的风险就应该更加重视，拟定相应的解决方案才能够有效地防范。

3.风险驾驭

风险驾驭是指利用某种技术，如原型化、软件自动化、软件心理学、可靠性工程学，以及某些项目管理方法等设法避开或转移风险。

上一节　本书简介　下一节

例题分析

4.7 例题分析

例题1（2011年5月试题15）

包含8个成员的开发小组的沟通路径最多有 （15） 条。

（15）A.28　B.32　C.56　D.64

例题分析：

在知道小组成员后，求沟通路径可按公式n x （n-1）/2求解，那么题目告诉我们开发小组有8个成员，即存在的沟通路径为8 x （8-1）/2=28条。

例题答案：（15）A

例题2（2011年5月试题16）

模块A直接访问模块B的内部数据，则模块A和模块B的耦合类型为 （16）。

（16）A.数据耦合　B.标记耦合　C.公共耦合　D.内容耦合

例题分析：

本题主要考查耦合的基本内容。

耦合是指两个或两个以上的模块相互依赖于对方的一个量度。它可以分为非直接耦合、数据耦合、标记耦合、控制耦合、外部耦合、公共耦合及内容耦合等。

当一个模块直接修改或操作另一个模块的数据或者直接转入另一个模块时，就发生了内容耦合。所以本题的答案选D.

例题答案：（16）D

例题3（2011年5月试题17）

下列关于风险的叙述不正确的是：风险是指 （17）。

（17）A.可能发生的事件　B.一定会发生的事件

C.会带来损失的事件　D.可能对其进行干预，以减少损失的事件

例题分析：

本题主要我们对风险概念的理解。

目前，对风险大致有两种定义：一种定义强调了风险表现为不确定性；而另一种定义则强调风险表现为损失的不确定性。其中广义的定义是：风险表现为损失的不确定性，说明风险产生的结果可能带来损失、获利或是无损失也无获利。

从风险的定义我们不难看出，风险是可能发生的事件，而且是会带来损失的事件，人为对其干

预，可能会减少损失。

例题答案：（17）B

例题4（2011年5月试题18）

下列关于项目估算方法的叙述不正确的是 （18）。

（18）A.专家判断方法受到专家经验和主观性影响

B.启发式方法（如COCOMO模型）的参数难以确定

C.机器学习方法难以描述训练数据的特征和确定其相似性

D.结合上述三种方法可以得到精确的估算结果

例题分析：

项目估算的常用方法主要有专家判断法、启发式法和机器学习法等。

专家判断法是指向学有专长、见识广博并有相关经验的专家进行咨询、根据他们多年来的实践经验和判断能力对计划项目作出预测的方法。很显然，采用这种方法容易受到专家经验和主观性的影响。

启发式方法使用一套相对简单、通用、有启发性的规则进行估算的方法，它具有参数难以确定、精确度不高等特点。

机器学习方法是一种基于人工智能与神经网络技术的估算方法，它难以描述训练数据的特征和确定其相似性。

而无论采用哪种估算方法，估算得到的结果都是大概的，而不是精确的。

例题答案：（18）D

例题5（2011年5月试题19）

图4-13是一个软件项目的活动图，其中顶点表示项目里程碑，边表示包含的活动，边上的权重表示活动的持续时间，则里程碑 （19） 在关键路径上。

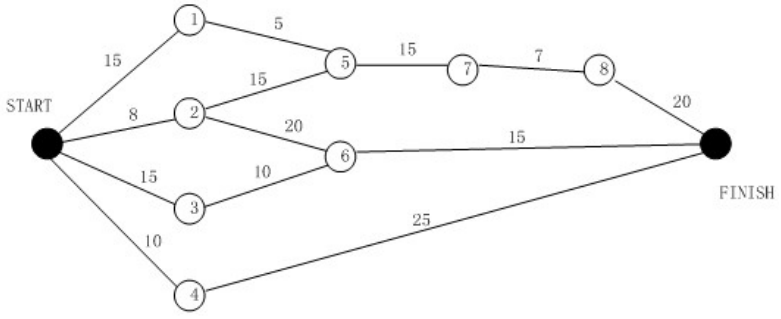


图4-13 项目的活动图

（19）A.1 B.2 C.3 D.4

例题分析：

本题主要考查关键路径求解的问题。

从开始顶点到结束顶点的最长路径为关健路径（临界路径），关键路径上的活动为关键活动。

在本题中找出的最长路径是Start->2->5->7->8->Finish,其长度为8+15+15+7+20=65,而其它任何路径的长度都比这条路径小，因此我们可以知道里程碑2在关键路径上。

例题答案：（19）B

例题6（2011年5月试题29）

为了有效地捕获系统需求，应采用 （29）。

（29）A.瀑布模型 B.V模型 C.原型模型 D.螺旋模型

例题分析：

瀑布模型严格遵循软件生命周期各阶段的固定顺序进行软件开发，其优点是可强迫开发人员采用规范的方法；严格规定了各阶段必须提交的文档；要求每个阶段结束后，都要进行严格的评审；而其缺点是过于理想化，缺乏灵活性，无法在开发过程中逐渐明确用户难以确切表达或一时难以想到的需求。

V模型是一种典型的测试模型，该模型强调开发过程中测试贯穿始终。

原型模型基于这样一种客观事实：并非所有的需求在系统开发之前都能准确地说明和定义。因此，它不追求也不可能要求对需求的严格定义，而是采用了动态定义需求的方法。原型模型适用于需求不够明确的项目，它能有效地捕获系统需求。

螺旋模型综合了瀑布模型和演化模型的优点，还增加了风险分析。采用螺旋模型时，软件开发沿着螺旋线自内向外旋转，每转一圈都要对风险进行识别和分析，并采取相应的对策。

例题答案：（29）C

例题7（2011年5月试题38）

下列关于一个类的静态成员的描述中，不正确的是 （38）。

（38）A.类的静态方法只能访问该类的静态数据成员

B.静态数据成员可被该类的所有方法访问

C.该类的对象共享其静态数据成员的值

D.该类的静态数据成员的值不可修改

例题分析：

类的静态成员与一般的类成员不同， 静态成员与对象的实例无关，只与类本身有关。它们一般用来实现类要封装的功能和数据，但不包括特定对象的功能和数据。静态成员和普通数据成员区别较大，体现在下面几点：

（1）普通数据成员属于类的一个具体的对象，只有对象被创建了，普通数据成员才会被分配内存。而静态数据成员属于整个类，即使没有任何对象创建，类的静态数据成员变量也存在。

（2）外部访问类的静态成员只能通过类名来访问。

（3）类的静态成员函数无法直接访问普通数据成员（可以通过类的指针等作为参数间接访问），而类的任何成员函数都可以访问类的静态数据成员。

（4）类的静态方法只能访问该类的静态数据成员。

另外，静态成员和类的普通成员一样，也具有public、protected、private3种访问级别，也可以具有返回值及被修改等性质。

例题答案：（38）D

例题8（2011年5月试题31）

软件产品的可靠性并不取决于（31）。

（31）A.潜在错误的数量　　B.潜在错误的位置

C.软件产品的使用方式　　D.软件产品的开发方式

例题分析：

软件产品的可靠性可描述为衡量在规定的一段时间内和规定条件下维护性能水平的一组软件质量，它主要体现在软件的成熟性、容错性和易恢复性方面。其中成熟性描述的是由软件故障引起软件失效的频度；容错性描述的是在错误或违反指定接口情况下维护指定性能水平的能力；易恢复性

描述的是在故障发生后重新建立性能水平，恢复数据的能力和时间。软件产品的可靠性不取决于软件产品的开发方式。

例题答案：（31）D

例题9（2011年5月试题32）

软件 （32） 是指一个系统在给定时间间隔内和给定条件下无失效运行的概率。

（32）A.可靠性　　B.可用性　　C.可维护性　　D.可伸缩性

例题分析：

软件的可靠性是指一个系统在给定时间间隔内和给定条件下无失效运行的概率。

软件的可用性是指软件在特定使用环境下为特定用户用于特定用途时所具有的有效性。

软件的可维护性是指与软件维护的难易程度相关的一组软件属性。

软件的可伸缩性是指是否可以通过运行更多的实例或者采用分布式处理来支持更多的用户。

例题答案：（32）A

例题10（2011年5月试题33）

高质量的文档所应具有的特性中，不包括 （33）。

（33）A.针对性，文档编制应考虑读者对象群

B.精确性，文档的行文应该十分确切，不能出现多义性的描述

C.完整性，任何文档都应当是完整的、独立的，应该自成体系

D.无重复性，同一软件系统的几个文档之间应该没有相同的内容，若确实存在相同内容，则可以用"见**文档**节"的方式引用

例题分析：

本题主要考查文档管理的相关内容。高质量的文档应具有针对性、精确性和完整性等特性。即文档编制应考虑读者对象群；文档的行文应该十分确切，不能出现多义性的描述；任何文档都应当是完整的、独立的，应该自成体系。

选项D描述的显然不符合高质量文档的要求。

例题答案：（33）D

例题11（2011年5月试题34）

在软件维护阶段，为软件的运行增加监控设施属于 （34） 维护。

（34）A.改正性　　B.适应性　　C.完善性　　D.预防性

例题分析：

根据引起软件维护的原因不同，软件维护通常可分为以下四种类型：

改正性维护。是指在使用过程中发现了隐蔽的错误后，为了诊断和改正这些隐蔽错误而修改软件的活动。该类维护一般占总维护工作量的25%.

适应性维护。是指为了适应变化了的环境而修改软件的活动。该类维护一般占总维护工作量的20%.

完善性维护。是指为了扩充或完善原有软件的功能或性能而修改软件的活动。该类维护一般占总维护工作量的5%.

预防性维护。是指为了提高软件的可维护性和可靠性、为未来的进一步改进打下基础而修改软件的活动。该类维护一般占总维护工作量的50%.

而本题所描述的为软件的运行增加监控设施属于完善性维。

例题答案：（34）C

例题12（2011年5月试题35）

图4-14所示的逻辑流，最少需要 （35） 个测试用例可实现语句覆盖。

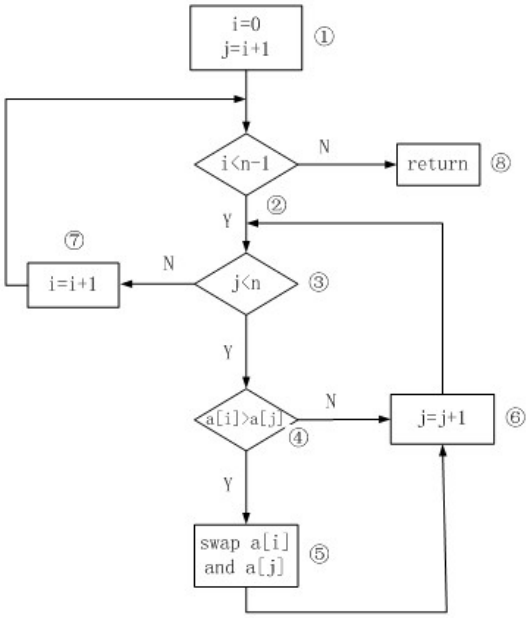


图4-14 逻辑流图

（35）A.1 B.2 C.3 D.5

例题分析：

语句覆盖是一种白盒测试，它是指选择足够多的测试用例，使得运行这些测试用例时，被测程序的每个语句至少执行一次。显然，语句覆盖是一种很弱的覆盖标准。

根据题目给出的逻辑图，程序的出口只有⑧，路径主要有①②⑧、①②③⑦②⑧、①②③④⑥③⑦②⑧及①②③④⑤⑥③⑦②⑧四条。那么很显然，路径①②③④⑤⑥③⑦②⑧覆盖了所有的语句，因此本题答案选A.

例题答案：（35）A

例题13（2011年5月试题36）

在改正当前故障的同时可能会引入新的故障，这时需要进行（36）。

（36）A.功能测试 B.性能测试 C.回归测试 D.验收测试

例题分析：

回归测试是指修改了当前故障后，重新进行测试以确认修改没有引入新的错误或导致其他的错误。因此本题答案选C.

例题答案：（36）C

例题14（2011年5月试题37）

面向对象分析的第一步是 （37）。

（37）A.定义服务 B.确定附加的系统约束

C.确定问题域 D.定义类和对象

例题分析：

面向对象分析的任务是了解问题域所涉及的对象、对象间的关系和操作，然后构造问题的对象模型。问题域是指一个包含现实世界事物与概念的领域，这些事物和概念与所设计的系统要解决的问题有关。因此面向对象分析的第一步是要确定问题域。

例题答案：（37）C

例题15（2011年5月试题39~41）

UML的设计视图包含了类、接口和协作，其中，设计视图的静态方面由 （39） 和 （40） 和表现，动态方面由交互图、 （41） 表现。

（39）A.类图　　B.状态图　　C.活动图　　D.序列图

（40）A.交互图　　B.对象图　　C.通信图　　D.定时图

（41）A.状态图和类图　　B.类图和活动图

C.对象图和状态图　　D.状态图和活动图

例题分析：

本题主要考查UML的设计视图。

UML 2.0包括14种图，分别列举如下：

（1）类图。描述一组类、接口、协作和它们之间的关系。在OO系统的建模中，最常见的图就是类图。类图给出了系统的静态设计视图，活动类的类图给出了系统的静态进程视图。

（2）对象图。描述一组对象及它们之间的关系。对象图描述了在类图中所建立的事物实例的静态快照。和类图一样，这些图给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的。

（3）构件图。描述一个封装的类和它的接口、端口，以及由内嵌的构件和连接件构成的内部结构。构件图用于表示系统的静态设计实现视图。对于由小的部件构建大的系统来说，构件图是很重要的。构件图是类图的变体。

（4）组合结构图。描述结构化类（例如，构件或类）的内部结构，包括结构化类与系统其余部分的交互点。组合结构图用于画出结构化类的内部内容。

（5）用例图。描述一组用例、参与者及它们之间的关系。用例图给出系统的静态用例视图。这些图在对系统的行为进行组织和建模时是非常重要的。

（6）顺序图。是一种交互图（interaction diagram），交互图展现了一种交互，它由一组对象或参与者以及它们之间可能发送的消息构成。交互图专注于系统的动态视图。顺序图是强调消息的时间次序的交互图。

（7）通信图。也是一种交互图，它强调收发消息的对象或参与者的结构组织。顺序图和通信图表达了类似的基本概念，但它们所强调的概念不同，顺序图强调的是时序，通信图强调的是对象之间的组织结构（关系）。在UML 1.X版本中，通信图称为协作图（collaboration diagram）。

（8）定时图。也是一种交互图，它强调消息跨越不同对象或参与者的实际时间，而不仅仅只是关心消息的相对顺序。

（9）状态图。描述一个状态机，它由状态、转移、事件和活动组成。状态图给出了对象的动态视图。它对于接口、类或协作的行为建模尤为重要，而且它强调事件导致的对象行为，这非常有助于对反应式系统建模。

（10）活动图。将进程或其他计算结构展示为计算内部一步步的控制流和数据流。活动图专注于系统的动态视图。它对系统的功能建模和业务流程建模特别重要，并强调对象间的控制流程。

（11）部署图。描述对运行时的处理节点及在其中生存的构件的配置。部署图给出了架构的静态部署视图，通常一个节点包含一个或多个部署图。

（12）制品图。描述计算机中一个系统的物理结构。制品包括文件、数据库和类似的物理比特

集合。制品图通常与部署图一起使用。制品也给出了它们实现的类和构件。

（13）包图。描述由模型本身分解而成的组织单元，以及它们之间的依赖关系。

（14）交互概览图。是活动图和顺序图的混合物。

其中类图、对象图、用例图、组件图及配置图为静态图，其它的为动态图。

例题答案：（39）A　　（40）B　　（41）D

例题16（2011年5月试题42）

UML中关联的多重度是指 （42） .

（42）A.一个类中被另一个类调用的方法个数

B.g一个类的某个方法被另一个类调用的次数

C.一个类的实例能够与另一个类的多少个实例相关联

D.两个类所具有的相同的方法和属性

例题分析：

在UML中，关联的多重度是指一个类的实例能够与另一个类的多少个实例相关联。它又称为重复度多重度表示为一个整数范围nm,整数n定义所连接的最少对象的数目，而m则为最多对象数（当不知道确切的最大数时，最大数用*号表示）。最常见的多重性有01、0*、11和1*.

例题答案：（42）C

例题17（2011年5月试题43）

在面向对象软件开发过程中，采用设计模式 （43）。

（43）A.以复用成功的设计

B.以保证程序的运行速度达到最优值

C.以减少设计过程创建的类的个数

D.允许在非面向对象程序设计语言中使用面向对象的概念

例题分析：

模式是一种问题的解决思路，它已经适用于一个实践环境，并且可以适用于其它环境。设计模式通常是对于某一类软件设计问题的可重用的解决方案，将设计模式引入软件设计和开发过程，其目的就在于要重用成功的软件开发经验。

例题答案：（43）A

例题18（2011年5月试题44~45）

设计模式 （44） 将抽象部分与其实现部分相分离，使它们都可以独立地变化。图4-15为该设计模式的类图，其中， （45） 用于定义实现部分的接口。

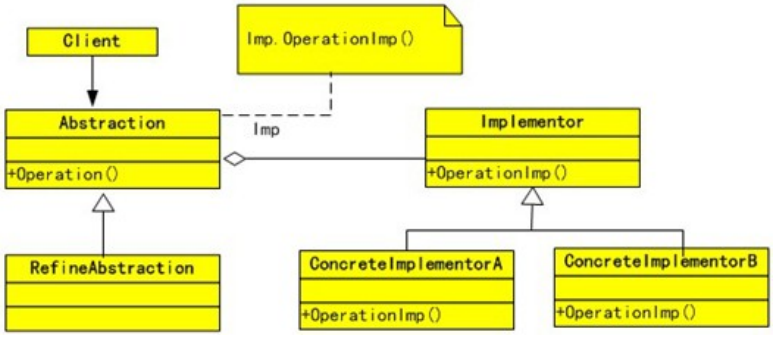


图4-15 本题设计模式的类图

（44）A.Bridge（桥接）　　B.Composite（组合）

C.Facade（外观）　　D.Singleton（单例）

（45）A.Abstraction　　B.ConcreteImplementorA

C.ConcreteImplementorB　　D.Implementor

例题分析：

本题主要考查常见的设计模式。

在本题中，根据题目给出的图，我们不难看出该图描述的是桥接模式，它的显著特征是它将抽象部分与实现部分分离，使它们可以相互独立地变化。我们不难从题目给出的图中看出，左边的是抽象类接口，而右边都是实现类接口，显然实现了分离。抽象类接口的下面是抽象的扩充，而实现类接口的下面是具体实现，因此他们可以相互独立地变化。其中：

Abstraction:抽象类定义抽象类的接口。维护一个Implementor（实现抽象类）的对象。

RefinedAbstraction:扩充的抽象类，扩充由Abstraction定义的接口。

Implementor:实现类接口，定义实现类的接口，这个接口不一定要与Abstraction的接口完全一致，事实上这两个接口可以完全不同，一般的讲Implementor接口仅仅给出基本操作，而Abstraction接口则会给出很多更复杂的操作。

ConcreteImplementor:具体实现类，实现Implementor定义的接口并且具体实现它。

例题答案：（44）A　　（45）D

上一节　　本书简介　　下一节

作者：希赛教育软考学院　来源：希赛网　2014年01月26日

数据库管理系统的功能和特征

第5章　数据库系统

在考试大纲中，有关数据库系统的知识点包括：

数据库管理系统的功能和特征；

数据库模型（概念模式、外模式、内模式）；

数据模型，ER图，第一范式、第二范式、第三范式；

数据操作（集合运算和关系运算）；

数据库语言（SQ）；

数据库的控制功能（并发控制、恢复、安全性、完整性）；

数据仓库和分布式数据库基础知识。

5.1　数据库管理系统的功能和特征

数据管理技术的发展大致经历了人工管理阶段（20世纪50年代中期前）、文件系统阶段（20世纪50年代后期到60年代中期）、数据库阶段（20世纪60年代末到70年代末）和高级数据库技术阶段（20世纪80年代初开始）。

数据库是长期存储在计算机内的、有组织的、可共享的数据的集合。

数据库管理系统（DBMS）是一种负责数据库的定义、建立、操作、管理和维护的软件系统。其目的是保证数据安全可靠，提高数据库应用的简明性和方便性。DBMS的工作机理是把用户对数据的操作转化为对系统存储文件的操作，有效地实现数据库三级之间的转化。数据库管理系统的主

要职能有：数据库的定义和建立、数据库的操作、数据库的控制、数据库的维护、故障恢复和数据通信。

数据库系统（DBS）是实现有组织地、动态地存储大量关联数据方便多用户访问的计算机软件、硬件和数据资源组成的系统。一个典型的数据库系统包括数据库、硬件、软件（应用程序）和数据库管理员（DBA）四个部分。根据计算机的系统结构，DBS可分成集中式、客户/服务器式、并行式和分布式4种。

与文件系统阶段相比，数据库技术的数据管理方式具有以下特点。

采用复杂的数据模型表示数据结构，数据冗余小，易扩充，实现了数据共享。

具有较高的数据和程序独立性，数据库的独立性有物理独立性和逻辑独立性。

数据库系统为用户提供了方便的用户接口。

数据库系统提供四个方面的数据控制功能，分别是并发控制、恢复、完整性和安全性。数据库中各个应用程序所使用的数据由数据库系统统一规定， 按照一定的数据模型组织和建立，由系统统一管理和集中控制。

增加了系统的灵活性。

高级数据库技术阶段的主要标志是分布式数据库系统和面向对象数据库系统的出现。

集中式系统的弱点是随着数据量的增加，系统相当庞大，操作复杂，开销大，而且因为数据集中存储，大量的通信都要通过主机，造成拥挤。分布式数据库系统的主要特点是数据在物理上分散存储，在逻辑上是统一的。分布式数据库系统的多数处理就地完成，各地的计算机由数据通信网络相联系。

面向对象数据库系统是面向对象的程序设计技术与数据库技术相结合的产物。面向对象数据库系统的主要特点是具有面向对象技术的封装性和继承性，提高了软件的可重用性。

从目前的数据库系统来看，主要存在以下缺点：

采用静态数据模型，数据类型和操作简单、固定，只能处理短寿命事务；

不能适应计算机辅助设计、计算机辅助软件工程、图像处理、超文本、多媒体等新的应用。

数据库的未来发展趋势如下：

分布式数据管理；

支持面向对象的数据模型；

体系结构适应功能扩展，能处理复杂数据类型和长寿命事务，能和以前数据库共存；

数据库技术与其他学科相结合（分布式数据库、并行数据库、多媒体数据库、Internet数据库、知识库、演绎数据库、主动数据库）。

上一节　本书简介　下一节

数据库模型

5.2 数据库模型

数据库系统的设计目标是允许用户逻辑地处理数据，而不必涉及这些数据在计算机中是怎样存