作者：希赛教育软考学院　来源：希赛网　2014年05月21日

考点分析

第8章　系统开发和运行维护

系统开发和运行维护是软件设计师考试的一个重要考点，同时也是软件设计师考试的核心。上午考试中，有分数为10分左右的系统开发和运行维护方面的知识，下午考试中有1道试题（数据流图设计）属于系统开发和运行维护知识，占15分。

8.1　考点分析

本节把历次考试中系统开发和运行维护的试题进行汇总，得出本章的考点，如表8-1所示。

表8-1 系统开发和运行维护试题知识点分布

考试时间	分数	考查知识点
10.11	8	瀑布模型（1）、需求分析（1）、软件设计（1）、软件测试（1）、项目管理工具（1）、数据字典（1）、数据流图（1）、CMM（1）
11.05	8	软件可移植性（1）、系统转换（2）、数据流图（2）、软件测试（2）、瀑布模型（1）
11.11	11	开发模型（1）、测试计划（1）、风险（1）、代码评审（1）、甘特图（1）、软件评审（1）、软件测试（2）、操作手册（1）、开发进度（1）、软件维护（1）
12.05	10	项目管理工具（1）、耦合（2）、数据流图（1）、内聚（1）、项目风险（1）、软件质量（1）、技术评审（1）、软件测试（2）
12.11	11	开发模型（2）、CMM（1）、成本估算（1）、RUP（1）、可移植性（1）、可靠性（1）、用户手册（1）、软件维护（1）、路径覆盖（1）、确认测试（1）
13.05	13	数据流图（1）、原型法（1）、CVS（1）、开发过程（1）、风险分析（1）、程序注释（1）、软件质量（2）、CMM（1）、代码行（1）、判定覆盖（1）、等价类划分（1）、RUP（1）
13.11	7	甘特图（1）、PERT图（1）、开发工具（1）、内聚和耦合（1）、语句覆盖（1）、测试阶段（2）
14.05	11	开发方法（1）、甘特图（1）、风险分析（2）、极限编程（1）、CMM（1）、编程风格（1）、软件维护（1）、开发文档（1）、软件测试分类（2）

根据表8-1,我们可以得出系统开发和运行维护的考点主要有：

（1）系统开发模型：包括瀑布模型、RUP、原型法。

（2）需求分析：包括数据字典、数据流图、系统转换。

（3）软件设计：包括耦合和内聚的概念、软件设计的过程、结构化设计的思想等。

（4）程序编写：包括程序编写的风格、工具等。

（5）软件测试：包括测试阶段、测试类型和测试用例设计。

（6）软件维护：主要考查维护的类型。

（7）项目管理：包括项目管理工具、质量管理、风险管理、CMM、软件文档等。

对这些知识点进行归类，然后按照重要程度进行分类，如表8-2所示。其中的五角星号（*）代表知识点的重要程度，星号越多，表示越重要。

表8-2 系统开发和运行维护各知识点重要程度

知识点	10.11	11.05	11.11	12.05	12.11	13.05	13.11	14.05	合计	比例	重要程度
系统开发模型	1	1		1	3	1		2	9	13.24%	★★
需求分析	3	4				1			8	11.76%	★★
软件设计	1						1		2	2.94%	★
程序编写						1	1	1	3	4.41%	★
软件测试	1	2		3	2	2	3	2	15	22.06%	★★★
软件维护				1	1			1	3	4.41%	★
项目管理	2	1		6	5	7	2	5	28	41.18%	★★★★★

在本章的后续内容中，我们将对这些知识点进行逐个讲解。

系统开发模型

8.2 系统开发模型

在开发模型知识点中，我们要掌握软件生命周期的概念、各种开发模型的特点和应用场合。主要的开发模型有瀑布模型、增量模型、螺旋模型、喷泉模型、智能模型、V模型、RAD模型、CBSD模型、原型方法、XP方法、RUP方法等。

开发生命周期模型

8.2.1 开发生命周期模型

系统开发的生命周期是指一个系统历经计划、分析、设计、编程、测试、维护直至淘汰的整个过程。生命周期阶段的划分通常可以采用以下三种方法：

（1）Boehm划分法：计划（问题定义、可行性研究）、开发（需求分析、总体设计、详细设计、编码、测试）、运行（维护）三大阶段。

（2）国标（GB8566-1988）划分法：可行性研究与计划、需求分析、概念设计、详细设计、实现、组装测试、确认测试、使用和维护。并在《GB/T8566-1995信息技术--软件生存期过程》中定义了获取过程、供应过程、开发过程、运行过程、维护过程、管理过程、支持过程7个部分。

（3）RUP划分法：分为初始、细化、构造、移交4个主要阶段。

下面，我们对一些主要的开发模型和方法进行简单的介绍。

1.瀑布模型

瀑布模型也称为生命周期法，是生命周期法中最常用的开发模型，它把软件开发的过程分为软件计划、需求分析、软件设计、程序编码、软件测试和运行维护6个阶段，规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。采用瀑布模型的软件过程如图8-1所示。

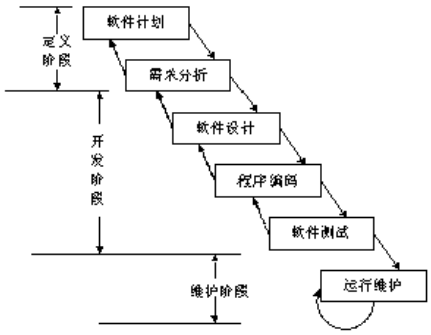


图8-1 软件生命周期的瀑布模型

（1）软件计划（问题的定义及规划）：主要确定软件的开发目标及其可行性。

（2）需求分析：在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段是一个很重要的阶段，这一阶段做得好，将为整个软件开发项目的成功打下良好的基础。

（3）软件设计：主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等。软件设计一般分为总体设计（概要设计）和详细设计。

（4）程序编码：将软件设计的结果转换成计算机可运行的程序代码。在程序编写中必须制定统一、符合标准的编写规范，以保证程序的可读性，易维护性，提高程序的运行效率。

（5）软件测试：在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。

（6）软件维护：软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件不能继续适应用户的要求，要延续软件的使用寿命，就必须对软件进行维护。

瀑布模型是最早出现的软件开发模型，在软件工程中占有重要的地位，它提供了软件开发的基本框架。瀑布模型的本质是"一次通过",即每个活动只做一次，最后得到软件产品，也称做"线性顺序模型"或者"传统生命周期".其过程是从上一项活动接收该项活动的工作对象作为输入，利用这一输入实施该项活动应完成的内容，给出该项活动的工作成果，作为输出传给下一项活动；对该项活动实施的工作进行评审，若其工作得到确认，则继续下一项活动，否则返回前项，甚至更前项的活动进行返工。

瀑布模型有利于大型软件开发过程中人员的组织与管理，有利于软件开发方法和工具的研究与使用，从而提高了大型软件项目开发的质量和效率。然而软件开发的实践表明，上述各项活动之间并非完全是自上而下的，而是呈线性图示，因此，瀑布模型存在严重的缺陷。

（1）由于开发模型呈线性，所以当开发成果尚未经过测试时，用户无法看到软件的效果。这样，软件与用户见面的时间间隔较长，也增加了一定的风险。

（2）在软件开发前期未发现的错误传到后面的开发活动中时，可能会扩散，进而可能导致整个软件项目开发失败。

（3）在软件需求分析阶段，完全确定用户的所有需求是比较困难的，甚至可以说是不太可能的。

2.变换模型

变换模型（演化模型）是在快速开发一个原型的基础上，根据用户在调用原型的过程中提出的

反馈意见和建议，对原型进行改进，获得原型的新版本，重复这一过程，直到演化成最终的软件产品。

3.螺旋模型

螺旋模型将瀑布模型和变换模型相结合，它综合了两者的优点，并增加了风险分析。它以原型为基础，沿着螺线自内向外旋转，每旋转一圈都要经过制订计划、风险分析、实施工程、客户评价等活动，并开发原型的一个新版本。经过若干次螺旋上升的过程，得到最终的系统，如图8-2所示。

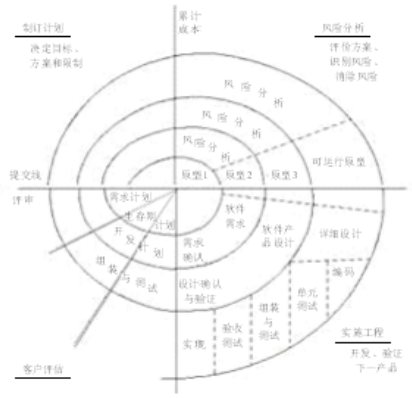


图8-2 螺旋模型

4.喷泉模型

喷泉模型对软件复用和生命周期中多项开发活动的集成提供了支持，主要支持面向对象的开发方法。"喷泉"一词本身体现了迭代和无间隙特性。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系统。所谓无间隙是指在开发活动中，分析、设计和编码之间不存在明显的边界，如图8-3所示。

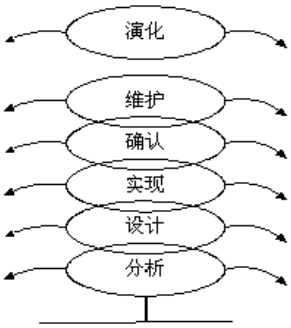


图8-3 喷泉模型

5、智能模型

智能模型是基于知识的软件开发模型，它综合了上述若干模型，并把专家系统结合在一起。该模型应用基于规则的系统，采用归约和推理机制，帮助软件人员完成开发工作，并使维护在系统规格说明一级进行。

6.V模型

在开发模型中，测试常常作为亡羊补牢的事后行为，但也有以测试为中心的开发模型，那就是V模型。V模型只得到软件业内比较模糊的认可。V模型宣称测试并不是一个事后弥补行为，而是一个同开发过程同样重要的过程，如图8-4所示。

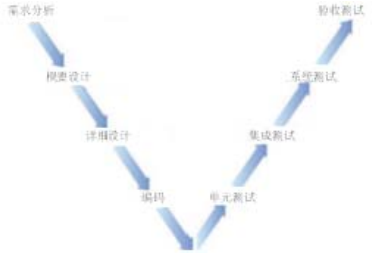


图8-4 V模型示意图

V模型描述了一些不同的测试级别，并说明了这些级别所对应的生命周期中不同的阶段。在图8-4中，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。请注意在不同的组织中，对测试阶段的命名可能有所不同。

V模型的价值在于它非常明确地表明了测试过程中存在的不同级别，并且清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系：

（1）单元测试的主要目的是针对编码过程中可能存在的各种错误。例如：用户输入验证过程中的边界值的错误。

（2）集成测试的主要目的是针对详细设计中可能存在的问题，尤其是检查各单元与其他程序部分之间的接口上可能存在的错误。

（3）系统测试主要针对概要设计，检查系统作为一个整体是否有效地得到运行。例如：在产品设置中是否达到了预期的高性能。

（4）验收测试通常由业务专家或用户进行，以确认产品能真正符合用户业务上的需要。

7.增量模型

增量模型融合了瀑布模型的基本成分（重复的应用）和原型实现的迭代特征。增量模型采用随着时间的进展而交错的线性序列，每一个线性序列产生软件的一个可发布的"增量".当使用增量模型时，第一个增量往往是核心的产品，也就是说第一个增量实现了基本的需求，但很多补充的特征还没有发布。客户对每一个增量的使用和评估，都作为下一个增量发布的新特征和功能。这个过程在每一个增量发布后不断重复，直到产生最终的完善产品。增量模型强调每一个增量均发布一个可操作的产品。

增量模型像原型实现模型和其他演化方法一样，本质上是迭代的。但与原型实现不同的是增量模型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，但它们确实提供了为用户服务的功能，并且提供了给用户评估的平台。增量模型的特点是引进了增量包的概念，无须等到所有需求都出来，只要某个需求的增量包出来即可进行开发。虽然某个增量包可能还需要进一步适应客户的需求，还需要更改，但只要这个增量包足够小，其影响对整个项目来说是可以承受的。

采用增量模型的优点是人员分配灵活，刚开始不用投入大量人力资源，如果核心产品很受欢迎，则可以增加人力实现下一个增量；当配备的人员不能在设定的期限内完成产品时，它提供了一种先推出核心产品的途径，这样就可以先发布部分功能给客户，对客户起到镇静剂的作用。此外，增量能够有计划地管理技术风险。增量模型的缺点是如果增量包之间存在相交的情况且不能很好地处理，就必须做全盘的系统分析。增量模型将功能细化、分别开发的方法适用于需求经常改变的软件开发过程。

8.RAD模型

快速应用开发（Rapid Application Development,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8-5所示。

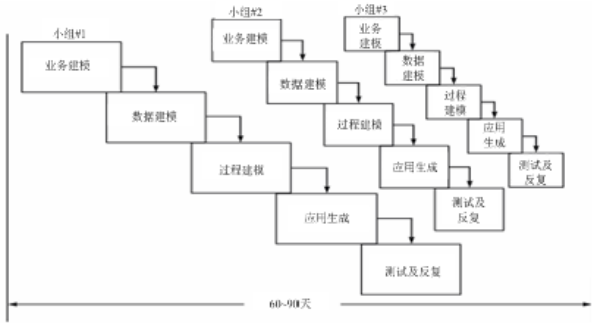


图8-5 采用RAD模型的开发过程

RAD模型各个活动期所要完成的任务如下。

（1）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。

（2）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。

（3）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。

（4）应用程序生成：利用第四代语言（4GL）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。

（5）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：

（1）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。

（2）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。

（3）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.基于构件的模型

构件（Component,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 Development,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8-6所示。

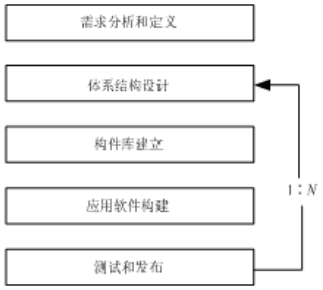


图8-6 采用基于构件的开发模型的软件过程

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。

10.原型方法

软件原型是所提出的新产品的部分实现，建立原型的主要目的是为了解决在产品开发的早期阶段的需求不确定的问题，其目的是明确并完善需求、探索设计选择方案、发展为最终的产品。

原型有很多种分类方法。从原型是否实现功能来分，软件原型可分为水平原型和垂直原型两种。水平原型也称为行为原型，用来探索预期系统的一些特定行为，并达到细化需求的目的。水平原型通常只是功能的导航，但并未真实实现功能。水平原型主要用在界面上。垂直原型也称为结构化原型，实现了一部分功能。垂直原型主要用在复杂的算法实现上。

从原型的最终结果来分，软件原型可分为抛弃型原型和演化型原型。抛弃型原型也称为探索型原型，是指达到预期目的后，原型本身被抛弃。抛弃型原型主要用在解决需求不确定性、二义性、不完整性、含糊性等。演化型原型为开发增量式产品提供基础，是螺旋模型的一部分，也是面向对象软件开发过程的一部分。演化型原型主要用在必须易于升级和优化的项目，适用于Web项目。

有些文献把原型分为实验型、探索型和演化型。探索型原型的目的是要弄清对目标系统的要求，确定所希望的特性，并探讨多种方案的可行性。实验型原型用于大规模开发和实现之前，考核方案是否合适，规格说明是否可靠。进化型原型的目的不在于改进规格说明，而是将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。

还有些文献也把原型分为抛弃式原型、演化式原型和递增式原型。

原型法适合于用户没有肯定其需求的明确内容的时候。它是先根据已给的和分析的需求，建立一个原始模型，这是一个可以修改的模型（在生命周期法中，需求分析成文档后一般不再多修改）。在软件开发的各个阶段都把有关信息相互反馈，直至模型的修改，使模型渐趋完善。在这个过程中，用户的参与和决策加强了，最终的结果是更适合用户的要求。这种原型技术又可分为三类：抛弃式、演化式和递增式。这种原型法成败的关键及效率的高低在于模型的建立及建模的速度。

11.XP方法

XP是一种轻量（敏捷）、高效、低风险、柔性、可预测、科学而且充满乐趣的软件开发方式。与其他方法论相比，其最大的不同在于：

（1）在更短的周期内，更早地提供具体、持续的反馈信息。

（2）迭代地进行计划编制，首先在最开始迅速生成一个总体计划，然后在整个项目开发过程中不断地发展它。

（3）依赖于自动测试程序来监控开发进度，并及早地捕获缺陷。

（4）依赖于口头交流、测试和源程序进行沟通。

（5）倡导持续的演化式的设计。

（6）依赖于开发团队内部的紧密协作。

（7）尽可能达到程序员短期利益和项目长期利益的平衡。

如图8-7所示，XP由价值观、原则、实践和行为四个部分组成，它们彼此相互依赖、关联，并通过行为贯穿于整个生命周期。

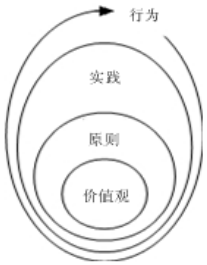


图8-7 XP组成示意图

XP的核心是其总结的4大价值观，即沟通、简单、反馈和勇气。它们是XP的基础，也是XP的灵魂。XP的5个原则是快速反馈、简单性假设、逐步修改、提倡更改和优质工作。而在XP方法论中，贯彻的是"小步快走"的开发原则，因此工作质量决不可打折扣，通常采用测试先行的编码方式来提供支持。

在XP中，集成了12个最佳实践：计划游戏、小型发布、隐喻、简单设计、测试先行、重构、结对编程、集体代码所有制、持续集成、每周工作40小时、现场客户、编码标准。

12.RUP方法

RUP（Rational 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UML.

与其他软件过程相比，RUP具有三个显著的特点：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四个顺序的阶段，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8-8所示。

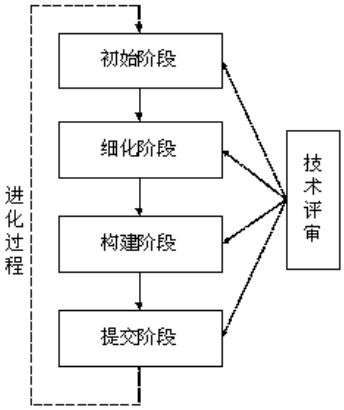


图8-8 基于RUP的软件过程

从图8-8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，每次经过这四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8-3所示。

表8-3 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。

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05月21日

系统开发方法论

8.2.2 系统开发方法论

系统的开发方法主要包括结构化分析与设计、面向数据结构的设计、面向对象分析与设计以及构件化方法四种。

1.结构化分析与设计

这种方法采用结构化技术来完成软件开发的各项任务。该方法把软件生命周期的全过程依次划分为若干阶段，然后顺序地完成每个阶段的任务，与瀑布模型有很好的结合度，是与其最相适应的

开发方法。

结构化方法的核心思想是"自顶向下，逐步分解".

2.面向数据结构的设计

数据的输入、存储都涉及不同的数据结构，面向数据结构设计方法的基本思想是根据数据结构导出程序结构。典型的面向数据结构的设计方法包括Jackson方法和Warnier方法。

Jackson方法的基本步骤：先建立系统的数据结构；接着以数据结构为基础，对应地建立程序结构；列出程序中要用到的各种基本操作，然后将操作分配到适当的模块中去。

面向数据结构的设计方法并没有明显地使用软件结构的概念，对于模块独立性原则也重视不足，因此并不适合于复杂的软件系统。

3.面向对象分析与设计

这种方法引入了"对象"的概念，将数据和方法封装在一起，提高了模块的聚合度，降低了耦合度，更大程度上支持软件复用。面向对象方法是现在最流行和最具有发展前景的软件开发方法，我们将在第9章进行详细讨论。

4.构件化开发

为了降低开发费用、提高生产率，以及在快速的技术演化面前提供受控的系统升级的开发方式，就催生了基于构件的软件开发（Component-Based Software Development,CBSD）。它通过有计划地集成现有的软件部分来进行软件开发。它可以有效地遏制复杂性、缩短发布时间、提高一致性，更有效地利用本领域的最佳方法、提高生产率、增加项目进度的可视性、支持并行和分布式的开发、减少维护费用。采用CBSD后，所有的软件解决方案将可以使用预建的构件和模板，像"搭积木"式地建造。

这种"积木"就是构件（组件），构件是一个功能相对独立的具有可重用价值的软件单元。在面向对象方法中，一个构件由一组对象构成，包含了一些协作类的集合，它们共同工作来提供一种系统功能。构件具有5个基本要素：规格说明、一个或多个实现、受约束的构件标准、包装方法和部署方法。

可重用性是指系统和（或）其组成部分能在其他系统中重复使用的程度。软件开发的全生命周期都有可重用的价值，包括项目的组织、软件需求、设计、文档、实现、测试方法和测试用例，都是可以被重复利用和借鉴的有效资源。可重用性体现在软件的各个层次，通用的、可复用性高的软件模块往往已经由操作系统或开发工具提供，如通用库、标准组件和标准模板库等，并不需要程序员重新开发。

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05月21日

需求分析

8.3 需求分析

本知识点主要考查需求分析的任务、数据流图、数据字典、系统转换等方面的知识。

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

需求分析概述

8.3.1 需求分析概述

需求分析所要做的工作是深入描述软件的功能和性能，确定软件设计的限制和软件同其他系统元素的接口细节，定义软件的其他有效性需求，细化软件要处理的数据域。用一句话概括就是需求分析主要是确定开发软件的功能、性能、数据、界面等要求。需求分析的实现步骤通常包括3个部分，分别是获取当前系统的物理模型，抽象出当前系统的逻辑模型，建立目标系统的逻辑模型。

1.需求分析的工作

具体来说，需求分析阶段的工作可以分成4个方面：

（1）问题识别：用于发现需求，描述需求，主要包括功能需求、性能需求、环境需求、可靠性需求、安全保密要求、用户界面需求、资源使用需求、软件成本消耗与开发进度需求、预先估计以后系统可能达到的目标。

（2）分析与综合：也就是对问题进行分析，然后在此基础上整合出解决方案。这个步骤经常是反复进行的，常用的方法有面向数据流的结构化分析方法，面向数据结构的Jackson方法，面向对象的分析方法，以及用于建立动态模型的状态迁移图和Petri网。

（3）编制需求分析的文档：也就是对已经确定的需求进行文档化描述，该文档通常称为软件需求说明书（需求规格说明书）。

（4）需求分析与评审：它是需求分析工作的最后一步，主要是对功能的正确性、完整性和清晰性，以及其他需求给予评价。

2.需求分析的原则

在软件需求分析的过程中，必须遵循以下原则：

（1）必须能够表达和理解问题的信息域和功能域。

（2）必须表示软件的行为（作为外部事件的结果）。

（3）必须划分描述信息、功能和行为的模型，从而使得可以以层次的方式揭示细节。

（4）分析过程应该从要素信息移向细节实现。

（5）必须按自顶向下、逐层分解的方式对问题进行分解和不断细化。

（6）要给出系统的逻辑视图和物理视图。

通过应用这些原则，系统分析员系统地处理某些问题。检查信息域以使得功能可以被更完整地理解，使用模型以使得可以以简洁的方式交流功能和行为的特征，应用划分以减少问题的复杂性。在这些处理过程中，软件的要素和视图实现对适当由处理需求带来的逻辑约束和由其他系统元素带来的物理约束是必需的。

3.需求的分类

什么是软件的需求呢？软件需求就是系统必须完成的事，以及必须具备的品质。具体来说，软件需求包括功能需求、非功能需求和设计约束3个方面的内容。

（1）功能需求：是指系统必须完成的那些事，即为了向它的用户提供有用的功能，产品必须执行的动作。

（2）非功能需求：是指产品必须具备的属性或品质，如可靠性、性能、响应时间、容错性、扩展性等。

（3）设计约束：也称为限制条件、补充规约，这通常是对解决方案的一些约束说明，例如必须采用国有自主知识版权的数据库系统、必须运行在UNIX操作系统之下等等。

除了这3种需求之外，还有业务需求、用户需求、系统需求这3个处于不同层面下的概念，充分理解这些模型才能够使你更加清晰地理清需求的脉络。

（1）业务需求：是指反映组织机构或客户对系统、产品高层次的目标要求，通常问题定义本身就是业务需求。

（2）用户需求：是指描述用户使用产品必须要完成什么任务、怎么完成的需求，通常是在问题定义的基础上进行用户访谈、调查，对用户使用的场景进行整理，从而建立从用户角度的需求。

（3）系统需求：是从系统的角度来说明软件的需求，它就包括了用特性说明的功能需求，质量属性以及其他非功能需求，还有设计约束。

我们经常围绕着"功能需求"来展开工作，而功能需求大部分都是从"系统需求"的角度来分析与理解的，也就是用"开发人员"的视角来理解需求的。但要想真正地得到完整的需求，仅戴上"开发人员"的眼镜是不够的，我们还需要"领域专家"的眼镜，我们要从更高的角度来理解需求，这就是"业务需求";同时我们还应该更好地深入用户，了解他们的使用场景，了解他们的所思所想，这就是"用户需求".这是一个理解层次的问题，并不仅仅是简单的概念。

4.需求工程

需求工程就是包括创建和维护系统需求文档所必需的一切活动的过程，也就是指需求开发和需求管理两大工作：

（1）需求开发：包括需求捕获、需求分析、编写规格说明书和需求验证4个阶段。在需求开发阶段需要确定产品所期望的用户类型、获取每种用户类型的需求、了解实际用户任务和目标，以及这些任务所支持的业务需求、分析源于用户的信息、对需求进行优先级分类、将所收集的需求编写成为软件规格说明书和需求分析模型、对需求进行评审等工作。

（2）需求管理：通常包括定义需求基线、处理需求变更、需求跟踪等方面的工作。

这两个方面是相辅相成的，需求开发是主线，是目标；需求管理是支持，是保障。换句话来说，需求开发是努力更清晰、明确地掌握客户对系统的需求；而需求管理则是对需求的变化进行管理的过程。

针对整个需求工程，通常有以下指导原则：

（1）在开始建立分析模型前先理解问题。人们通常总存在急于求成的倾向，甚至在问题被很好地理解前，这经常会导致产生一个解决错误问题的优美软件的诞生。

（2）开发原型，使得用户能够了解将如何发生人机交互。因为人们一般对软件质量的感觉经常基于对界面"友好性"的感觉，因此，强力推荐使用原型方法（以及相应产生的迭代）。

（3）记录每个需求的起源及原因。这是建立回溯到客户的可追踪性的第一步。

（4）使用多个需求视图。建立数据、功能和行为模型，为软件工程师提供三种不同的视图，这将减少忽视某些东西的可能性，并增加识别不一致性的可能性。

（5）给需求赋予优先级。过短的时限可能使每个软件需求得到实现的可能性减小，如果采用增

量模型，必须标识那些将在第一个增量中要交付的需求。

（6）努力删除含糊性。因为大多数需求以自然语言描述，存在含糊性的可能，正式的技术复审是发现并删除含糊性的一种方法。

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

数据流图

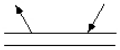
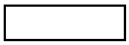
8.3.2 数据流图

数据流图简称DFD,是描述数据处理过程的一种图形工具。数据流图从数据传递和加工的角度，以图形的方式描述数据在系统流程中流动和处理的移动变换过程，反映数据的流向、自然的逻辑过程和必要的逻辑数据存储。

1.数据流图基本符号

数据流图采用4种基本的图形符号，见表8-4.

表8-4 数据流图基本符号

符号	名称	说明
	加工	在圆中注明加工的名字与编号
	数据流	在箭头边给出数据流的名称与编号，注意不是控制流
	数据存储文件	文件名称为名词或名词性短语
	数据源点或汇点	在方框中注明数据源或汇点的名称

符号名称说明

加工在圆中注明加工的名字与编号

数据流在箭头边给出数据流的名称与编号，注意不是控制流

数据存储文件文件名称为名词或名词性短语

数据源点或汇点在方框中注明数据源或汇点的名称

（1）加工

用圆或椭圆描述，又称数据处理，表示输入数据在此进行变换产生输出数据，以数据结构或数据内容作为加工对象。加工的名字通常是一个动词短语，简明扼要地表明要完成的加工。

（2）数据流

用箭头描述，由一组固定的数据项组成，箭头方向表示数据的流向，作为数据在系统内的传输通道。它们大多是在加工之间传输加工数据的命名通道，也有在数据存储文件和加工之间的非命名数据通道。虽然这些数据流没有命名，但其连接的加工和文件的名称，以及流向可以确定其含义。

同一数据流图上不能有同名的数据流。如果有两个以上的数据流指向一个加工，或从一个加工中输出两个以上的数据流，这些数据流之间往往存在一定的关系。其具体的描述如图8-9所示，其中"*"表示相邻之间的数据流同时出现，"⊕"表示相邻之间的数据流只取其一。

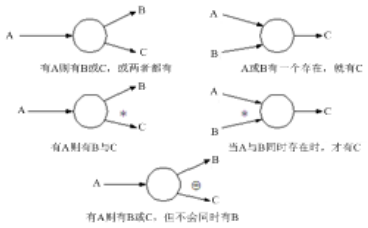


图8-9 数据流

一般把数据流图中的数据划分为变换流和事务流两种。变换流是指信息沿着输入通路进入系统，同时将信息的外部形式转换成内部形式，通过变换中心处理之后，再沿着输出通路转换成外部形式输出。事务流是指信息沿着输入通路到达一个事务中心，事务中心根据输入信息从若干个处理序列中选择一个路径执行。

（3）数据存储文件

用双杆描述，在数据流图中起保存数据的作用，又称数据存储或文件，可以是数据库文件或任何形式的数据组织。流向数据存储的数据流可以理解为写入文件或查询文件，从数据存储流出的数据流可以理解为从文件读数据或得到查询结果。

（4）数据源点或终点

用方框描述，表示数据流图中要处理数据的输入来源或处理结果要送往的地方，在图中仅作为一个符号，并不需要以任何软件的形式进行设计和实现，是系统外部环境中的实体，故称外部实体。它们作为系统与系统外部环境的接口界面，在实际的问题中可能是人员、组织、其他软硬件系统等。它们一般只出现在分层数据流的顶层图中。

2.数据流图设计要略

有时为了增加数据流图的清晰性，防止数据流的箭头线太长，减少交叉绘制数据流条数，一般在一张图上可以重复同名的数据源点、终点与数据存储文件。如某个外部实体既是数据源点又是数据汇点，可以在数据流图的不同的地方重复绘制。在绘制时应该注意以下要点：

（1）自外向内，自顶向下，逐层细化，完善求精。

（2）保持父图与子图的平衡。

为了表达较为复杂问题的数据处理过程，用一个数据流图往往不够。一般按问题的层次结构进行逐步分解，并以分层的数据流图反映这种结构关系。

根据层次关系一般将数据流图分为顶层数据流图、中间数据流图和底层数据流图，除顶层图外，其余分层数据流图从0开始编号。对任何一层数据流图来说，称它的上层数据流图为父图，在它的下一层的数据流图为子图。

顶层数据流图只含有一个加工，表示整个系统；输入数据流和输出数据流为系统的输入数据和输出数据，表明了系统的范围，以及与外部环境的数据交换关系。

底层数据流图是指其加工不能再分解的数据流图，其加工称为"原子加工".

中间数据流图是对父层数据流图中某个加工进行细化，而它的某个加工也可以再次细化，形成子图。中间层次的多少，一般视系统的复杂程度而定。

任何一个数据流子图必须与它上一层父图的某个加工对应，二者的输入数据流和输出数据流必须保持一致，即父图与子图的平衡。父图与子图的平衡是数据流图中的重要性质，保证了数据流图的一致性，便于分析人员阅读和理解。

在父图与子图平衡中，数据流的数目和名称可以完全相同；也可以在数目上不相等，但是可以借助数据字典中数据流描述，确定父图中的数据流是由子图中几个数据流合并而成的，即子图是对

父图中加工和数据流同时进行分解，因此也属于父图与子图的平衡，如图8-10所示。

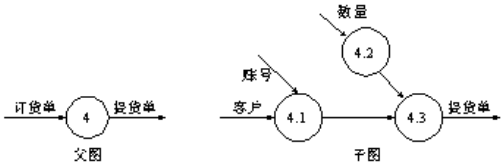


图8-10 父图与子图的平衡

（3）保持数据守恒。也就是说，一个加工所有输出数据流中的数据必须能从该加工的输入数据流中直接获得，或者是通过该加工能产生的数据。每个加工必须有输入数据流和输出数据流，反映此加工的数据来源和加工变换结果。一个加工的输出数据流只由它的输入数据流确定。数据流必须经过加工，即必须进入加工或从加工中流出。

（4）加工细节隐蔽。根据抽象原则，在画父图时，只需画出加工和加工之间的关系，而不必画出各个加工内部的细节。当某层数据流图中的数据存储不是父图中相应加工的外部接口，而只是本图中某些加工之间的数据接口时，那么这些数据存储为局部数据存储。

为了强调局部数据存储的隐蔽性，一般情况下，局部数据存储只有作为某些加工的数据接口或某个特定加工的输入和输出时，才画出来。即按照自顶向下的分析方法，某数据存储首次出现时只与一个加工有关，那么这个数据存储应该作为与之关联加工的局部数据存储，在该层数据流子图中不必画出，而在该加工的子图中画出，除非该加工为原子加工。

（5）简化加工间的关系。在数据流图中，加工间的数据流越少，各个加工就越相对独立，所以应尽量减少加工间输入输出数据流的数目。

（6）均匀分解。应该使一个数据流中的各个加工分解层次大致相同。

（7）适当地为数据流、加工、文件、源/宿命名，名字应反映该成分的实际意义，避免空洞的名字。

（8）忽略枝节。应集中精力于主要的数据流，而暂不考虑一些例外情况、出错处理等枝节性的问题。

（9）表现的是数据流而不是控制流。数据流图与传统的程序流程图不同，数据流图是从数据的角度来描述一个系统的，而流程图则是从对数据加工的角度来描述系统的。数据流图中的箭头是数据流，而流程图中的箭头则是控制流，它表达的是程序执行的次序。数据流图适合于宏观地分析一个组织的业务概况，而程序流程图只适合于描述系统中某个加工的执行细节。

每个加工必须既有输入数据流，又有输出数据流；在整套数据流图中，每个文件必须既有读文件的数据流又有写文件的数据流，但在某一张子图中可能只有读、没有写，或者只有写、没有读。

上一节　本书简介　下一节

数据字典

8.3.3 数据字典

数据字典是关于数据信息的集合，也就是对数据流图中包含的所有元素定义的集合。

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。

数据字典的任务就是对数据流图中出现的所有被命名的图形元素在数据字典中作为一个词条加以定义，使得每个图形元素的名称都有一个确切的解释。

数据字典描述的内容包括数据流、数据文件、加工逻辑、源（汇）点及数据元素等词条的描述。在数据流和数据文件词条的数据字典描述中包含一定的数据结构，对于数据结构常用的描述是定义式。表8-5给出了数据结构定义式可能出现的符号。

表8-5 数据结构定义式可能出现的符号

符号	含义	举例说明
=	被定义为	
+	与	$x=a+b$，表示 x 由 a 和 b 组成
[…，…]或[…|…]	或	$x=[a，b]$，$x=[a\|b]$，表示 x 由 a 或由 b 组成
{…}	重复	$x=\{a\}$，表示 x 由 0 个或多个 a 组成
(…)	可选	$x=(a)$，表示 a 可在 x 中出现，也可以不出现

在数据字典中有4种类型的条目。

1.数据项条目

数据项条目给出了某个数据单项的定义，通常为数据项值的类型、允许的取值范围等。

2.数据流条目

数据流条目给出某个数据流的定义，它通常是列出该数据流的各组成数据项。有些数据流的组成比较复杂，可以采用自顶向下分解的方式将它表示成更低层次的组合，一直分解到每个与项目有关的人都清楚其准确含义时为止。

由低的数据元素（或称分量）组成更复杂的数据的方式有以下几种：

（1）顺序：即以确定次序连接两个或多个分量。

（2）选择：即从两个或多个可能的元素中选取一个。

（3）重复：即把指定的分量重复零次或多次。

（4）可选：即一个分量是可有可无的（重复零次或多次）。

3.文件条目

文件条目给出某个文件的定义，通常也是列出其记录的组成数据项。此外，还可以指出文件的组织方式，如按单号递增次序排列等。

4.加工条目

加工条目是对数据流图中每一个不能再分解的基本加工的精确说明。

说明中应精确描述用户要求某个加工做什么，包括加工的激发条件、加工逻辑、优先级、执行频率和出错处理等。其中加工逻辑是最基本的部分，它描述了输入数据流、输入文件与输出数据流、输出文件之间的逻辑关系。数据字典的设计包括：数据流设计、数据元素字典设计、数据处理字典设计、数据结构字典设计和数据存储设计。这些设计涵盖了数据的采集、范围的确定等。在数据字典的每一个词条中应包含以下信息：

（1）名称：数据对象或控制项、数据存储或外部实体的名字。

（2）别名或编号。

（3）分类：数据对象？加工？数据流？数据文件？外部实体？控制项（事件/状态）？

（4）描述：描述内容或数据结构等。

（5）何处使用：使用该词条（数据或控制项）的加工。

对加工的描述是数据字典的组成内容之一，常用的加工描述方法有3种：结构化语言、判定树、判定表。

（1）结构化语言：介于自然语言和形式语言之间的一种半形式语言，是在自然语言基础之上加了一些限制，使用有限的词汇和有限的语句来描述加工逻辑。结构化语言是受结构化程序设计思想启发而扩展出来的。结构化程序设计只允三种基本结构。结构化语言也只允许三种基本语句，即简单的祈使语句、判断语句、循环语句。与程序设计语言的差别在于结构化语言没有严格的语法规定。与自然语言的不同在于它只有极其有限的词汇和语句。结构化语言使用三类词汇：祈使句中的动词、数据字典中定义的名词以及某些逻辑表达式中的保留字。

（2）判定树：若一个动作的执行不只是依赖一个条件，而是与多个条件有关，那么这项策略的表达就比较复杂。如果用结构化语言的判断语句，就有多重嵌套。层次一多，可读性就下降。用判定树来表示，可以更直观一些。

（3）判定表：一些条件较多、在每个条件下取值也较多的判定问题，可以用判定表表示。判定表能清晰地表达复杂的条件组合与应做动作之间的对应关系，判定表的优点是能够简洁，无二义性地描述所有的处理规则。但判定表表示的是静态逻辑，是在某种条件取值组合情况下可能的结果，它不能表达加工的顺序，也不能表达循环结构，因此判定表不能成为一种通用的设计工具。

这三种描述加工的方法各有千秋，除上面谈到的几个方面外，从直观性、可修改性等方面的比较，如表8-6所示。

表8-6 描述加工的方法比较

	结构化语言	判定树	判定表
直观性	一般	很好	一般
用户检查	不便	方便	不便
可修改性	好	一般	差
逻辑检查	好	一般	很好
机器可读性	很好	差	很好
机器可编程	一般	不好	很好

上一节　本书简介　下一节

系统转换策略

8.3.4 系统转换策略

为了明确旧系统完成的功能，需要对旧系统进行建模。对旧系统的建模，可使用类似数据流图、数据字典这样的工具进行高阶的业务建模。旧系统的分析结果可以是若干高阶的数据流图、功能点清单和性能清单等。其中描述了旧系统的主要工作过程、功能和问题。

1.比较新旧系统

通过比较新旧系统的逻辑模型，并参考新系统的目标和规模，可以确定新旧系统的主要差异。对新系统提出的要求至少包括：

（1）新系统必须完成旧系统的基本功能。

（2）新系统必须改正旧系统存在的问题，包括错误、缺陷。

（3）新系统应提供全新的功能和性能，并覆盖需求和分析阶段发现的软件需求。

2.复查问题

完成新系统设计后，复查在分析旧系统阶段发现的全部问题定义，并根据新系统的设计模型比较系统规模、功能改变、性能改进、与预期的开发目标之间的关系等，检查是否存在系统分析师对系统的误解或在新方案设计中没有涵盖的潜在问题。

3.控制规模

对旧系统的分析和建模，应控制在一个较小的开销规模上。因为新旧系统比较的目的主要是发现和复查问题，而不是得到旧系统的详细设计。

4.确定新旧系统的转换策略

根据新旧系统比较的结果，可以得到未来新系统建设完成后的转换策略。

（1）直接转换策略。新系统是完全重构的系统，可能采用了全新的技术平台和软件来构建，或者用户业务和使用方式发生了剧烈变化，对原有系统只能进行抛弃处理。采用这种策略的优点是新系统能够非常灵活地适应业务需要，功能齐全、结构合理、系统稳定、扩展性强，整个信息系统的利用率比较高。但也存在着一些问题，主要是新旧系统之间的转换代价比较大；由于需要一套比较完整的业务需求，开发新系统的周期比较长，一次性投资巨大，未经广泛使用并证明是成熟可靠的新技术平台通常具有一定的技术风险；另外旧系统通常积累下了大量的业务数据，必须将业务数据的录入、转换、检查以及在新系统中的重建作为重要的工作进行考虑，尽量减小在新旧系统转换的时候对用户现有业务的冲击；最后，考虑新旧系统转换还需要考虑诸如：维持新系统运行的日常开销、由于使用习惯改变带来的学习时间、培训人员的成本等因素。

（2）逐步转换策略。这种策略是部分继承原有系统，部分进行新系统的更新和开发的技术路线，每成熟一部分新系统软件，就更新一部分旧系统软件，最终采取渐进的方式，过渡到新的软件平台上来。这种策略的优点是新旧系统的转换震动比较小，用户容易接受，但也由于采用渐进式，导致新旧系统的转换周期加大，同时由于需求的变化，给新系统的稳定造成比较大的影响。在转换过程中，需要开发新旧系统之间的接口、还需要制定阶段性的转换目标和计划。

（3）并行转换策略。这种策略是在一定的阶段并行使用新旧系统。将同样的业务在新旧系统中都完成一次。然后在确认新系统已经可以稳定工作后，停止旧的系统和全面启用新系统。并行转换策略会较多增加用户的工作量，并且难以控制新旧系统中的数据变化，因此很少使用。

通常，对于现有信息系统比较稳定、能够适应自身业务发展需要的建设单位或新旧系统转换风险很大的系统（例如：订票系统或银行的中间业务系统），可以采用渐进方式；对于现有信息系统本身就存在问题，比如已经不能满足业务需要，存在安全、性能等方面问题的，应采用直接转换策略。

新旧系统比较的结果，可提供新旧系统转换策略选择上的基本建议。如果逐步转换策略可用，则可以由客户根据自身的具体情况决定采用那种策略；如果新旧系统差异很大，无法选择逐步转换策略，通常要在直接转换前，对新系统进行多次测试、排错、试运行和评估。

软件设计

8.4 软件设计

在软件设计阶段，主要考查结构化设计、模块内聚与耦合等概念。

上一节 本书简介 下一节

软件设计阶段

8.4.1 软件设计阶段

从工程管理角度，软件设计可分为概要设计和详细设计两个阶段。

1.概要设计

概要设计也称为高层设计，将软件需求转化为数据结构和软件的系统结构。例如：如果采用结构化设计，则将从宏观的角度将软件划分成各个组成模块，并确定模块的功能以及模块之间的调用关系。

概要设计主要是设计软件的结构，确定系统是由哪些模块组成的，以及每个模块之间的关系。它采用的是结构图（包括模块、调用、数据）来描述程序的结构，还可以使用层次图和HIPO（层次图加输入/处理/输出图）。整个过程主要包括：复查基本系统模型，复查并精化数据流图，确定数据流图的信息流类型（包括交换流和事务流），根据流类型分别实施变换分析或事务分析，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

详细设计也称为低层设计，将对结构表示进行细化，得到详细的数据结构与算法。同样的，如果采用结构化设计，则详细设计的任务就是为每个模块进行设计。

详细设计确定应该如何具体地实现所要求的系统，得出对目标系统的精确描述。它采用自顶向下、逐步求精的设计方式和单入口单出口的控制结构。经常使用的工具包括程序流程图、盒图、PAD图（问题分析图）、PDL（伪码）。

总的来说，在整个软件设计过程中，需完成以下工作任务：

（1）制定规范，作为设计的共同标准。

（2）完成软件系统结构的总体设计，将复杂系统按功能划分为模块的层次结构，然后确定模块的功能，以及模块间的调用关系、模块间的组成关系。

（3）设计处理方式，包括算法、性能、周转时间、响应时间、吞吐量、精度等。

（4）设计数据结构。

（5）可靠性设计。

（6）编写设计文档，包括概要设计说明书、详细设计说明书、数据库设计说明书、用户手册、初步的测试计划等。

（7）设计评审，主要是对设计文档进行评审。

在设计阶段，必须根据要解决的问题，做出设计的选择。例如：对于半结构化决策问题就适合于交互式计算机软件来解决。

上一节　本书简介　下一节

软件设计活动

8.4.2 软件设计活动

软件设计包括4个既独立又相互联系的活动：数据设计、体系结构设计、接口设计（界面设计）和过程设计。这4个活动完成以后就得到了全面的软件设计模型。设计方法也是以后实现设计模型的蓝图软件工程活动的基础。

数据设计是实施软件工程中的4个设计活动的第一个。由于数据结构对程序结构和过程复杂性都有影响，数据结构对软件质量的影响是很深远的。好的数据设计将改善程序结构和模块划分，降低过程复杂性。数据设计将分析时创建的信息域模型变换成实现软件所需的数据结构。在实体-关系图（E-R图）中定义的数据对象和关系以及数据字典中描述的详细数据内容为数据设计活动奠定了基础。

体系结构设计的主要目标是开发一个模块化的程序结构，并表示出模块间的控制关系。此外，体系结构设计将程序结构和数据结构相结合，为数据在程序中的流动定义了接口。

接口设计描述了软件内部、软件和协作系统之间以及软件域人（用户）之间如何通信。一个接口意味着信息流（如数据和/或控制流），因此，数据和控制流图提供了接口设计所需的信息。接口设计要实现的内容包括一般交互、信息显示和数据输入。接口设计主要包括三个方面：

（1）设计软件模块间的接口。

（2）设计模块和其他非人的信息生产者和消费者（比如外部实体）的接口。

（3）设计人（用户）和计算机间的接口（通常简称为"人机接口"或"人机界面"）。

过程设计应该在数据设计、体系结构设计和接口设计完成之后进行。所有的程序都可以建立在一组已有的逻辑构成元素上，这一组逻辑构成元素强调了"对功能域的维护",其中每一个逻辑构成元素有可预测的逻辑结构，从顶端进入，从底端退出，读者可以很容易地理解过程流。

上一节　本书简介　下一节

结构化设计

8.4.3 结构化设计

结构化设计包括体系结构设计、接口设计、数据设计和过程设计等任务。它是一种面向数据流

的设计方法，是以结构化分析阶段所产生的成果为基础，进一步自顶而下、逐步求精和模块化的过程。

在结构化方法中，模块化是一个很重要的概念，它将一个待开发的软件分解成为若干个小的简单部分--模块，每个模块可以独立地开发、测试。这是一种复杂问题的"分而治之"原则，其目的是使程序的结构清晰、易于测试与修改。

具体来说，模块是指执行某一特定任务的数据结构和程序代码。通常将模块的接口和功能定义为其外部特性，将模块的局部数据和实现该模块的程序代码称为内部特性。而在模块设计时，最重要的原则就是实现信息隐蔽和模块独立。模块通常具有连续性，也就是意味着作用于系统的小变动将导致行为上的小变化，同时规模说明的小变动也将影响到一小部分模块。

1.抽象化

对软件进行模块设计的时候，可以有不同的抽象层次。在最高的抽象层次上，可以使问题所处环境的语言描述问题的解法。而在较低的抽象层次上，则采用过程化的方法。抽象化包括对过程的抽象、对数据的抽象和对控制的抽象。

（1）过程的抽象。在软件工程过程中，从系统定义到实现，每进展一步都可以看做是对软件解决方案的抽象化过程的一次细化。在从概要设计到详细设计的过程中，抽象化的层次逐次降低。当产生源程序时到达最低的抽象层次。

（2）数据抽象。数据抽象与过程抽象一样，允许设计人员在不同层次上描述数据对象的细节。

（3）控制抽象。控制抽象可以包含一个程序控制机制而无须规定其内部细节。

2.自顶向下，逐步细化

将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。最初的说明只是概念性地描述了系统的功能或信息，但并未提供有关功能的内部实现机制或有关信息的内部结构的任何信息。设计人员对初始说明仔细推敲，进行功能细化或信息细化，给出实现的细节，划分出若干成分。然后再对这些成分，施行同样的细化工作。随着细化工作的逐步展开，设计人员就能得到越来越多的细节。

3.信息隐蔽

信息隐蔽是开发整体程序结构时使用的法则，即将每个程序的成分隐蔽或封装在一个单一的设计模块中，并且尽可能少地暴露其内部的处理。通常我们将难的决策、可能修改的决策、数据结构的内部连接，以及对它所做的操作细节、内部特征码、与计算机硬件有关的细节等隐蔽起来。

通过信息隐蔽可以提高软件的可修改性、可测试性和可移植性，它也是现代软件设计的一个关键性原则。

4.模块独立

模块独立是指每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系最简单。保持模块的高度独立性，也是在设计时的一个很重要的原则。通常我们用耦合（模块之间联系的紧密程度）和内聚（模块内部各元素之间联系的紧密程度）两个标准来衡量，我们的目标是高内聚、低耦合。

模块的内聚类型通常可以分为7种，根据内聚度从高到低排序如表8-7所示。

表8-7 模块的内聚类型

内聚类型	描　　述
功能内聚	完成一个单一功能，各个部分协同工作，缺一不可
顺序内聚	处理元素相关，而且必须顺序执行
通信内聚	所有处理元素集中在一个数据结构的区域上
过程内聚	处理元素相关，而且必须按特定的次序执行
瞬时内聚	所包含的任务必须在同一时间间隔内执行（如初始化模块）
逻辑内聚	完成逻辑上相关的一组任务
偶然内聚	完成一组没有关系或松散关系的任务

与此相对应，模块的耦合类型通常也分为7种，根据耦合度从低到高排序如表8-8所示。

表8-8 模块的耦合类型

耦合类型	描　　述
非直接耦合	没有直接联系，互相不依赖对方
数据耦合	借助参数表传递简单数据
标记耦合	一个数据结构的一部分借助于模块接口被传递
控制耦合	模块间传递的信息中包含用于控制模块内部逻辑的信息
外部耦合	与软件以外的环境有关
公共耦合	多个模块引用同一个全局数据区
内容耦合	一个模块访问另一个模块的内部数据；一个模块不通过正常入口转到另一模块的内部；两个模块有一部分程序代码重叠；一个模块有多个入口

除了满足以上两大基本原则之外，通常在模块分解时还需要注意：保持模块的大小适中；尽可能减少调用的深度；直接调用该模块的个数应该尽量大，但调用其他模块的个数则不宜过大；保证模块是单入口、单出口的；模块的作用域应该在其之内；功能应该是可预测的。

上一节　本书简介　下一节

程序编写

8.5 程序编写

在进行程序编写时，既可以使用文本工具进行（例如使用Windows中的记事本），也可以使用专门的程序设计工具所提供的程序编写界面。对于程序编码而言，除了要符合软件设计的要求外，还要使程序具有良好的风格，易于阅读和理解。

上一节　本书简介　下一节

程序设计风格

8.5.1 程序设计风格

程序设计风格包括4个方面：源程序文档化、数据说明、语句结构和输入/输出方法，力图从编码原则的角度提高程序的可读性，改善程序质量。

1.源程序文档化

（1）符号名的命名。符号名即标识符，包括模块名、变量名、常量名、子程序名、数据区名、缓冲区名等。这些名字应能反映它所代表的实际东西，应有一定实际意义。应当选择精练的意义明确的名字，改善对程序功能的理解。必要时可使用缩写名字，但缩写规则要一致，并且要给每一个名字加注释。在一个程序中，一个变量只应用于一种用途。就是说，在同一个程序中一个变量不能身兼几种工作。

（2）程序的注释。正确的注释能够帮助读者理解程序，可为后续阶段进行测试和维护提供明确的指导。一些正规的程序文本中，注释行的数量占到整个源程序的1/3到1/2,甚至更多。注释可分为序言性注释和功能性注释。

序言性注释通常置于每个程序模块的开头部分，它应当给出程序的整体说明，对于理解程序本身具有引导作用。有关项目包括程序标题、有关本模块功能和目的的说明、主要算法、接口说明、有关数据描述、模块位置、开发简历等。

功能性注释嵌在源程序体中，用以描述其后的语句或程序段是在做什么工作，不要解释下面怎么做，因为解释怎么做常常是与程序本身重复的，并且对于阅读者理解程序没有什么帮助。书写功能性注释，要注意：

用于描述一段程序，而不是每一个语句。

用缩进和空行，使程序与注释容易区别。

注释要正确。

（3）视觉组织。利用空格、空行和移行，提高程序的可视化程度。恰当地利用空格，可以突出运算的优先性，避免发生运算的错误。自然的程序段之间可用空行隔开；对于选择语句和循环语句，把其中的程序段语句向右做阶梯式移行。这样可使程序的逻辑结构更加清晰，层次更加分明。

2.数据说明

在编写程序时，需注意数据说明的风格。为了使程序中数据说明更易于理解和维护，必须注意以下几点。

（1）数据说明的次序应当规范化，使数据属性容易查找。

（2）当多个变量名用一个语句说明时，应当对这些变量按字母的顺序排列。

（3）如果设计了一个复杂的数据结构，应当使用注释来说明在程序实现时这个数据结构的固有特点。

3.语句结构

在设计阶段确定了软件的逻辑流结构，但构造单个语句则是编码阶段的任务。语句构造力求简单，直接，不能为了片面追求效率而使语句复杂化。

（1）在一行内只写一条语句，并且采取适当的移行格式，使程序的逻辑和功能变得更加明确。

（2）程序编写首先应当考虑清晰性，不要刻意追求技巧性，使程序编写得过于紧凑。

（3）程序编写得要简单，写清楚，直截了当地说明程序员的用意。

（4）除非对效率有特殊的要求，程序编写要做到清晰第一，效率第二。不要为了追求效率而丧失了清晰性。事实上，程序效率的提高主要应通过选择高效的算法来实现。

（5）首先要保证程序正确，然后才要求提高速度。反过来说，在使程序高速运行时，首先要保证它是正确的。

（6）让编译程序做简单的优化。

（7）尽可能使用库函数。

（8）避免使用临时变量而使可读性下降。

（9）尽量用公共过程或子程序去代替重复的功能代码段。

（10）用调用公共函数去代替重复使用的表达式。

（11）使用括号来清晰地表达算术表达式和逻辑表达式的运算顺序。

（12）避免不必要的转移。如果能保持程序的可读性，则不必用GOTO语句。

（13）尽量只采用三种基本的控制结构来编写程序。

（14）用逻辑表达式代替分支嵌套。

（15）避免使用空的ELSE语句和IF…THEN IF…的语句。

（16）避免使用ELSE GOTO和ELSE RETURN结构。

（17）使与判定相联系的动作尽可能地紧跟着判定。

（18）避免采用过于复杂的条件测试。

（19）尽量减少使用"否定"条件的条件语句。不要让读者绕弯子想。

（20）避免过多的循环嵌套和条件嵌套。

（21）不要使GOTO语句相互交叉。

（22）避免循环的多个出口。

（23）使用数组，以避免重复的控制序列。

（24）尽可能用通俗易懂的伪码来描述程序的流程，然后再翻译成必须使用的语言。

（25）数据结构要有利于程序的简化。

（26）要模块化，使模块功能尽可能单一化，模块间的耦合能够清晰可见。

（27）利用信息隐蔽，确保每一个模块的独立性。

（28）从数据出发去构造程序。

（29）不要修补不好的程序，要重新编写。也不要一味地追求代码的复用，要重新组织。

（30）对太大的程序，要分块编写、测试，然后再集成。

（31）对递归定义的数据结构尽量使用递归过程。

（32）注意计算机浮点数运算的特点，例如：浮点数运算 10.0*0.1 通常不等于1.0.

（33）不要单独进行浮点数的比较。用它们做比较，其结果常常发生异常情况。

（34）避免不恰当地追求程序效率，在改进效率前，要做出有关效率的定量估计。

（35）在程序中应有出错处理功能，一旦出现故障时不要让机器进行干预，导致停工。

4.输入和输出

输入和输出信息是与用户的使用直接相关的。输入和输出的方式和格式应当尽可能方便用户的使用。因此，在软件需求分析阶段和设计阶段，就应基本确定输入和输出的风格。系统能否被用户接受，有时就取决于输入和输出的风格。

不论批处理的输入/输出方式，还是交互式的输入/输出方式，在设计和程序编码时都应考虑下列原则：

（1）对所有的输入数据都进行检验，从而识别错误的输入，以保证每个数据的有效性。

（2）检查输入项的各种重要组合的合理性，必要时报告输入状态信息。

（3）使得输入的步骤和操作尽可能简单，并保持简单的输入格式。

（4）输入数据时，应允许使用自由格式输入。

（5）应允许默认值。

（6）输入一批数据时，最好使用输入结束标志，而不要由用户指定输入数据数目。

（7）在以交互式输入/输出方式进行输入时， 要在屏幕上使用提示符明确提示交互输入的请求，指明可使用选择项的种类和取值范围。同时，在数据输入的过程中和输入结束时，也要在屏幕上给出状态信息。

（8）当程序设计语言对输入/输出格式有严格要求时，应保持输入格式与输入语句的要求的一致性。

（9）给所有的输出加注解，并设计输出报表格式。

输入/输出风格还受到许多其他因素的影响。如输入/输出设备（例如：终端的类型，图形设备，数字化转换设备等）、用户的熟练程度，以及通信环境等。

Wasserman为"用户软件工程及交互系统的设计"提供了一组指导性原则，可供软件设计和编程参考：

（1）把计算机系统的内部特性隐蔽起来不让用户看到。

（2）有完备的输入出错检查和出错恢复措施，在程序执行过程中尽量排除由于用户的原因而造成程序出错的可能性。

（3）如果用户的请求有了结果，应随时通知用户。

（4）充分利用联机帮助手段，对于不熟练的用户，提供对话式服务，对于熟练的用户，提供较高级的系统服务，改善输入/输出的能力。

（5）使输入格式和操作要求与用户的技术水平相适应。对于不熟练的用户，充分利用菜单系统逐步引导用户操作；对于熟练的用户，允许绕过菜单，直接使用命令方式进行操作。

（6）按照输出设备的速度设计信息输出过程。

（7）区别不同类型的用户，分别进行设计和编码。

（8）保持始终如一的响应时间。

（9）在出现错误时应尽量减少用户的额外工作。

在交互式系统中，这些要求应成为软件需求的一部分，并通过设计和编码，在用户和系统之间建立良好的通信接口。

上一节　本书简介　下一节

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

程序效率

8.5.2 程序效率

程序的效率是指程序的执行速度及程序所需占用的内存的存储空间。讨论程序效率的几条准则为：

（1）效率是一个性能要求，应当在需求分析阶段给出。软件效率以需求为准，不应以人力所及为准。

（2）好的设计可以提高效率。

（3）程序的效率与程序的简单性相关。

一般说来，任何对效率无重要改善，且对程序的简单性、可读性和正确性不利的程序设计方法都是不可取的。

1.算法对效率的影响

源程序的效率与详细设计阶段确定的算法的效率直接有关。在详细设计翻译转换成源程序代码后，算法效率反映为程序的执行速度和存储容量的要求。

转换过程中的指导原则是：

（1）在编程序前，尽可能化简有关的算术表达式和逻辑表达式。

（2）仔细检查算法中的嵌套的循环，尽可能将某些语句或表达式移到循环外面。

（3）尽量避免使用多维数组。

（4）尽量避免使用指针和复杂的表。

（5）采用"快速"的算术运算。

（6）不要混淆数据类型，避免在表达式中出现类型混杂。

（7）尽量采用整数算术表达式和布尔表达式。

（8）选用等效的高效率算法。

许多编译程序具有"优化"功能，可以自动生成高效率的目标代码。它可剔除重复的表达式计算，采用循环求值法、快速的算术运算，以及采用一些能够提高目标代码运行效率的算法来提高效率。对于效率至上的应用来说，这样的编译程序是很有效的。

2.影响存储效率的因素

在大中型计算机系统中，存储限制不再是主要问题。在这种环境下，对内存采取基于操作系统的分页功能的虚拟存储管理，给软件提供了巨大的逻辑地址空间。这时，存储效率与操作系统的分页功能直接有关，并不是指要使所使用的存储空间达到最少。

采用结构化程序设计，将程序功能合理分块，使每个模块或一组密切相关模块的程序体积大小与每页的容量相匹配，可减少页面调度，减少内外存交换，提高存储效率。

在微型计算机系统中，存储容量对软件设计和编码的制约很大。因此要选择可生成较短目标代码且存储压缩性能优良的编译程序，有时需采用汇编程序。通过程序员富有创造性的努力，提高软件时间与空间效率。

提高存储效率的关键是程序的简单性。

上一节　本书简介　下一节

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

软件测试

8.6 软件测试

在软件测试阶段，重点考查软件测试的目的、测试的类型、测试的阶段等知识点。

 作者：希赛教育软考学院 来源：希赛网 2014年05月21日

测试的目的

8.6.1 测试的目的

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。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。

1983年，Bill Hetzel在"Complete Guide of Software Testing"一书中指出："测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量".Grenford J. Myers在"The Art of Software Testing"一书中指出：

（1）软件测试是为了发现错误而执行程序的过程。

（2）测试是为了证明程序有错，而不是证明程序无错误。

（3）一个好的测试用例是在于它能发现至今未发现的错误。

（4）一个成功的测试是发现了至今未发现的错误的测试。

这种观点可以提醒人们测试要以查找错误为中心，而不是为了演示软件的正确功能。但是仅凭字面意思理解这一观点可能会产生误导，认为发现错误是软件测试的唯一目的，查找不出错误的测试就是没有价值的，事实并非如此。

首先，测试并不仅仅是为了要找出错误。通过分析错误产生的原因和错误的分布特征，可以帮助项目管理者发现当前所采用的软件过程的缺陷，以便改进。同时，这种分析也能帮助我们设计出有针对性的检测方法，改善测试的有效性。

其次，没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。

因此，软件测试可以验证软件是否满足软件需求规格说明和软件设计所规定的功能、性能及其软件质量特性的要求，为软件质量的评价提供依据。我们要注意的是软件测试只是软件质量保证的手段之一，不能单凭测试来保证软件质量。

 作者：希赛教育软考学院 来源：希赛网 2014年05月21日

测试的类型

8.6.2 测试的类型

软件测试方法一般分为两大类，分别为动态测试和静态测试。

1.动态测试

动态测试指通过运行程序发现错误，分为黑盒测试法、白盒测试法和灰盒测试法。不管是哪一

种测试，都不能做到穷尽测试，只能选取少量最有代表性的输入数据，以期用较少的代价暴露出较多的程序错误。这些被选取出来的数据就是测试用例（一个完整的测试用例应该包括输入数据和期望的输出结果）。

（1）黑盒法

把被测试对象看成一个黑盒子，测试人员完全不考虑程序的内部结构和处理过程，只在软件的接口处进行测试，依据需求规格说明书，检查程序是否满足功能要求。常用的黑盒测试用例的设计方法有等价类划分、边值分析、错误猜测、因果图和功能图等。

等价类划分：将所有可能的输入数据，划分为等价的部分，然后从每个部分中选取少数有代表性的数据作为测试用例。等价类可以分为有效等价类（即合理、有意义的数据集合）、无效等价类（即不合理、无意义的数据集合）两种。而在选取测试用例时，应遵从"设计一个新的测试用例时，应尽可能多地覆盖尚未覆盖的有效等价类；但每次应仅覆盖一个尚未覆盖的无效等价类"的原则。

边界值分析：它是对等价类划分法的一个补充，即选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。

错误推测法：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

因果图：等价类划分、边界值分析都只考虑了输入条件，未考虑输入条件间的联系，而因果图则用来描述多种条件组合的测试用例，其最终生成的结果是判定表。它首先基于规格说明书分析原因（等价类）和结果（输出条件）；然后找出原因与结果之间的关系，画出因果图；在因果图上加上约束或限制条件；将其转换为判定表；根据判定表得出测试用例。

功能图：它是由状态迁移图和逻辑功能模型构建的，状态迁移图用于表示输入数据序列以及相应的输出数据；逻辑功能模型用于表示在状态中输入条件与输出条件之间的对应关系。测试用例则是由测试中经过的一系列状态和在每个状态中必须依靠输入/输出数据满足的一对条件组成的。

（2）白盒法

把测试对象看作一个打开的盒子，测试人员须了解程序的内部结构和处理过程，以检查处理过程的细节为基础，对程序中尽可能多的逻辑路径进行测试，检验内部控制结构和数据结构是否有错，实际的运行状态与预期的状态是否一致。 由于白盒测试是结构测试，所以被测对象基本上是源程序，以程序的内部逻辑为基础设计测试用例。常用的白盒测试用例设计方法有基本路径测试、循环覆盖测试、逻辑覆盖测试。

逻辑覆盖：以程序内部逻辑为基础的测试技术，如常用的语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、点覆盖、边覆盖、路径覆盖。

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。

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。

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。

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。

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五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。

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。

循环覆盖：单循环及嵌套循环。

基本路径法：在程序控制流程图的基础上，通过分析控制构造的环路复杂性导出基本路径集合。然后设计测试用例，保证这些路径至少通过一次。

（3）灰盒法

灰盒测试是一种介于白盒测试与黑盒测试之间的测试，它关注输出对于输入的正确性。同时也关注内部表现，但这种关注不像白盒测试那样详细且完整，而只是通过一些表征性的现象、事件及标志来判断程序内部的运行状态。

灰盒测试结合了白盒测试和黑盒测试的要素，考虑了用户端、特定的系统知识和操作环境，在系统组件的协同性环境中评价应用软件的设计。

2.静态测试

静态测试指被测试程序不在机器上运行，而是采用人工检测和计算机辅助静态分析的手段对程序进行检测。静态分析中进行人工测试的主要方法有桌前检查（Desk Checking）、代码审查和代码走查。经验表明：使用这种方法能够有效地发现30%~70%的逻辑设计和编码错误。

（1）桌前检查

由程序员自己检查自己编写的程序。程序员在程序通过编译之后，进行单元测试设计之前，对源程序代码进行分析、检验，并补充相关的文档，目的是发现程序中的错误。这种桌前检查，由于程序员熟悉自己的程序和自身的程序设计风格，可以节省很多的检查时间，但应避免主观片面性。

（2）代码审查

代码审查是由若干程序员和测试员组成一个会审小组，通过阅读、讨论和争议，对程序进行静态分析的过程。代码审查分两步：

第一步，小组负责人提前把设计规格说明书、控制流程图、程序文本及有关要求、规范等分发给小组成员，作为评审的依据。小组成员在充分阅读这些材料之后，进入审查的第二步。

第二步，召开程序审查会。在会上，首先由程序员逐句讲解程序的逻辑。在此过程中，程序员或其他小组成员可以提出问题，展开讨论，审查错误是否存在。实践表明，程序员在讲解过程中能发现许多原来自己没有发现的错误，而讨论和争议则促进了问题的暴露。

在会前，应当给会审小组每个成员准备一份常见错误的清单，把以往所有可能发生的常见错误罗列出来，供与会者对照检查，以提高会审的实效。这个常见错误清单也叫做检查表，它把程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，然后把它们制成表格，供在会审时使用。这种检查表类似于本章单元测试中给出的检查表。

（3）代码走查

代码走查与代码审查基本相同，其过程也分为两步。

第一步，把材料先发给走查小组每个成员，让他们认真研究程序，然后再开会。

第二步，开会的程序与代码审查不同，不是简单地读程序和对照错误检查表进行检查，而是让与会者"充当"计算机。即首先由测试组成员为被测程序准备一批有代表性的测试用例，提交给走查小组。走查小组开会，集体扮演计算机角色，让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹，供分析和讨论用。

值得说明的是使用静态测试的方法也可以实现白盒测试。例如：使用人工检查代码的方法来检查代码的逻辑问题，也属于白盒测试范畴。

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

测试的阶段

8.6.3 测试的阶段

为了保证系统的质量和可靠性，应力求在分析、设计等各个开发阶段结束前，对软件进行严格的技术评审。而软件测试是为了发现错误而执行程序的过程。

根据测试的目的、阶段的不同，可以把测试分为单元测试、集成测试、确认测试、系统测试等种类。

1.单元测试

单元测试又称为模块测试，是针对软件设计的最小单位（程序模块）进行正确性检验的测试工作。其目的在于检查每个程序单元能否正确实现详细设计说明中的模块功能、性能、接口和设计约束等要求，发现各模块内部可能存在的各种错误。单元测试需要从程序的内部结构出发设计测试用例，多个模块可以平行地独立进行单元测试。

单元测试根据详细设计说明书，包括模块接口测试、局部数据结构测试、路径测试、错误处理测试和边界测试，单元测试通常由开发人员自己负责。而由于通常程序模块不是单独存在的，因此常常要借助驱动模块（相当于用于测试模拟的主程序）和桩模块（子模块）完成。单元测试的计划通常是在软件详细设计阶段完成的。

2.集成测试

集成测试也称为组装测试、联合测试（对于子系统而言，则称为部件测试）。它主要是将已通过单元测试的模块集成在一起，主要测试模块之间的协作性。集成测试计划通常在软件概要设计阶段完成。

从组装策略而言，可以分为一次性组装和增量式组装，增量式组装又包括自顶向下、自底向上、混合式三种，其中混合式组装又称为三明治测试。

（1）自顶向下集成测试是一种构造程序结构的增量实现方法。模块集成的顺序是首先集成主控模块（主程序），然后按照控制层次结构向下进行集成。隶属于（和间接隶属于）主控模块的模块按照深度优先或者广度优先的方式集成到整个结构中去。

（2）自底向上集成测试是从原子模块（比如在程序结构的最低层的模块）开始来进行构造和测试的，跟自顶向下集成测试相反。

（3）三明治式测试是一种组合的折中测试策略，从"两头"往"中间"测试，其在程序结构的高层使用自顶向下策略，而在下面的较低层中使用自底向上策略，类似于"两片面包间夹馅的三明治"而得名。

软件集成的过程是一个持续的过程，会形成多个临时版本。在不断的集成过程中，功能集成的稳定性是真正的挑战。在每个版本提交时，都需要进行冒烟测试，即对程序主要功能进行验证。冒烟测试也称为版本验证测试或提交测试。

3.确认测试

确认测试也称为有效性测试，主要验证软件的功能、性能及其他特性是否与用户要求（需求）一致。确认测试计划通常在需求分析阶段完成。根据用户的参与程度，通常包括4种类型：

（1）内部确认测试：主要由软件开发组织内部按软件需求说明书进行测试。

（2）a测试（Alpha测试）：由用户在开发环境下进行测试。

（3）b测试（Beta测试）：由用户在实际使用环境下进行测试。

（4）验收测试：针对软件需求说明书，在交付前由用户为主进行的测试。

4.系统测试

如果项目不只包含软件，还有硬件和网络等，则要将软件与外部支持的硬件、外设、支持软件、数据等其他系统元素结合在一起，在实际运行环境下，对计算机系统的一系列集成与确认测试。一般地，系统测试的主要内容包括功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、安装与反安装测试等。系统测试计划通常是在系统分析阶段（需求分析阶段）完成的。

不管是哪个阶段的测试，一旦测试出问题，就要进行修改。修改之后，为了检查这种修改是否会引起其他错误，还要对这个问题进行测试，这种测试称为回归测试或退化测试。

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05月21日

性能测试

8.6.4 性能测试

性能测试是通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。负载测试和压力测试都属于性能测试，两者可以结合进行，统称为负载压力测试。通过负载测试，确定在各种工作负载下系统的性能，目标是测试当负载逐渐增加时，系统各项性能指标的变化情况。压力测试是通过确定一个系统的瓶颈或者不能接收的性能点，来获得系统能提供的最大服务级别的测试。

1.性能测试的目的

性能测试的目的是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件，最后起到优化系统的目的。具体来说，包括以下几个方面：

（1）评估系统的能力，测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助作出决策。

（2）识别体系中的弱点：受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系

的瓶颈或薄弱的地方。

（3）系统调优：重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能。

（4）检测软件中的问题：长时间的测试执行可导致程序发生由于内存泄露引起的失败，揭示程序中的隐含的问题或冲突。

（5）验证稳定性和可靠性：在一个生产负荷下执行测试一定的时间是评估系统稳定性和可靠性是否满足要求的唯一方法。

2.性能测试的类型

性能测试类型包括负载测试，强度测试，容量测试等。

（1）负载测试：负载测试是一种性能测试，指数据在超负荷环境中运行，程序是否能够承担。

（2）强度测试： 强度测试是一种性能测试，表明在系统资源特别低的情况下软件系统运行情况。

（3）容量测试：确定系统可处理同时在线的最大用户数。

3.负载压力测试

系统的负载压力测试（负载测试）是指系统在某种指定软件、硬件及网络环境下承受的流量，例如：并发用户数、持续运行时间、数据量等，其中并发用户数是负载压力的重要体现。系统在应用环境下主要承受并发访问用户数、无故障稳定运行时间、大数据量操作等负载压力。

负载压力测试的目的如下：

（1）在真实环境下检测系统性能，评估系统性能是否可以满足系统的性能设计要求。

（2）预见系统负载压力承受力，对系统的预期性能进行评估。

（3）进行系统瓶颈分析、优化系统。

在网络应用系统中，负载压力测试应重点关注客户端、网络、服务器（包括应用服务器和数据库服务器）的性能。应获取的关键测试指标如下：

（1）客户端：并发用户数、响应时间、交易通过率以及吞吐量等。

（2）网络：带宽利用率、网络负载、延迟以及网络传输和应用错误等。

（3）服务器：操作系统的CPU占用率、内存使用、硬盘I/O等；数据库服务器的会话执行情况、SQL执行情况、资源争用以及死锁等；应用服务器的并发连接数、请求响应时间等。

上一节　本书简介　下一节

第三方测试

8.6.5 第三方测试

第三方测试指独立于软件开发方和用户方的测试，组织的测试也称为"独立测试".软件质量工程强调开展独立验证和确认（IV&V）活动，是由在技术、管理和财务上与开发组织具有规定程序独立的组织执行验证和确认过程。软件第三方测试是相对独立的组织进行的软件测试，一般情况下是在模拟用户真实应用环境下，进行的软件确认测试。

第三方测试机构是一个中介的服务机构，它通过自己专业化的测试手段为客户提供有价值的服

务。但是这些服务不同于公司内部的测试，因为第三方测试机构的测试除了发现软件问题之外，还有科学公正地评价软件的职能，这就要求该机构要保持公正、廉洁、客观、科学且独立的态度。

第三方测试机构存在的价值主要是由软件公司、软件用户，以及国家的公正诉求所决定的。对于软件开发商来说，经过第三方测试机构的测试，不仅可以通过专业化的测试手段发现软件错误，帮助开发商提升软件的品质，而且可以对软件有一个客观且科学的评价，有助于开发商认清自己产品的定位。对于行业主管部门以及软件使用者来说，第三方测试机构可帮助选择合适且优秀的软件产品。而对于一些信息工程项目来说，在验收之前，经过第三方机构的严格测试，可以最大程度地避免信息行业的"豆腐渣"工程。此外，经过国家认可的第三方测试机构，还为国家软件产品的质量监督抽查提供独立公正的测试支持。

在选择第三方测试机构时，主要查看其资质、信息系统工程测评经验、测试环境、测试工具及测试工程师队伍的素质等。

上一节　本书简介　下一节

面向对象测试基础

8.6.6 面向对象测试基础

面向对象测试是与采用面向对象开发相对应的测试技术，它通常包括4个测试层次，从低到高排列分别是算法层、类层、模板层和系统层。

（1）算法层：测试类中定义的每个方法，基本上相当于传统软件测试中的单元测试。

（2）类层：测试封装在同一个类中的所有方法与属性之间的相互作用。在面向对象软件中类是基本模块，因此可以认为这是面向对象测试中所特有的模块（单元）测试。

（3）模板层：也称为主题层，测试一组协同工作的类或对象之间的相互作用。大体上相当于传统软件测试中的子系统测试，但是也有面向对象软件的特点（例如：对象之间通过发送消息相互作用）。

（4）系统层。把各个子系统组装成完整的面向对象软件系统，在组装过程中同时进行测试。

设计测试方案的传统技术，例如：逻辑覆盖、等价划分、边界值分析和错误推测等方法，仍然可以作为测试类中每个方法的主要技术。面向对象测试的主要目标，也是用尽可能低的测试成本和尽可能少的测试方案，发现尽可能多的错误。但是，面向对象程序中特有的封装、继承和多态等机制，也给面向对象测试带来一些新特点，增加了测试和调试的难度。

上一节　本书简介　下一节

软件维护

8.7 软件维护

在软件维护阶段，主要考查软件维护的分类、软件的可维护性等。

软件经过测试，交付给用户后，在使用和运行过程中可能在软件运行/维护阶段对软件产品进行的修改就是维护。

软件可维护性是指纠正软件系统出现的错误和缺陷，以及为满足新的要求进行修改、扩充和压缩的容易程度。目前广泛用来衡量程序可维护性的因素包括可理解性、可测试性、可修改性。

软件维护占整个软件生命周期的60%~80%,维护的类型主要有3种：

（1）改正性维护：为了识别和纠正软件错误、改正软件性能上的缺陷、排除实施中的误使用，应当进行的诊断和改正错误的过程就叫做改正性维护。

（2）适应性维护：在使用过程中，外部环境（新的硬、软件配置）、数据环境（数据库、数据格式、数据输入/输出方式、数据存储介质）可能发生变化。为使软件适应这种变化，而去修改软件的过程就叫做适应性维护。

（3）完善性维护：在软件的使用过程中，用户往往会对软件提出新的功能与性能要求。为了满足这些要求，需要修改或再开发软件，以扩充软件功能、增强软件性能、改进加工效率、提高软件的可维护性。这种情况下进行的维护活动叫做完善性维护。

在实际维护工作中，改正性维护大约占20%的工作量，适应性维护大约占25%的工作量，完善性维护大约占50%的工作量。除了这3类维护之外，还有一类维护活动，叫做预防性维护。这是为了提高软件的可维护性、可靠性等，为以后进一步改进软件打下良好基础。通常，预防性维护定义为："把今天的方法学用于昨天的系统以满足明天的需要".也就是说，采用先进的软件工程方法对需要维护的软件或软件中的某一部分（重新）进行设计、编制和测试。

在软件开发过程中，错误纠正成本在逐步放大。也就是说，错误发现得越早，纠正错误所花费的成本就会越低，反之则越高。例如：如果在软件设计阶段有个错误未被发现，而待编码阶段时才发现，这时，纠正这个设计错误比纠正源代码错误需要更大的成本。

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05月21日

项目管理

8.8 项目管理

在软件项目管理方面，主要考查软件可移植性、可维护性、技术评审、配置管理、质量保证、CMM、版本控制工具、软件复杂性、软件文档等。

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05月21日

软件项目估算

8.8.1 软件项目估算

项目估算的内容包括软件规模估算、软件工作量估算与成本估算三个方面。

1.估算策略

估算策略包括"自顶向下"和"自底向上"两种。

（1）自顶向下：以项目经理为核心，先根据用户、决策者的要求，确定一个时间期限，然后根据该期限进行分解，采用对号入座的方式将开发工作分配给具体的开发人员，以获得一个可以满足这个期限的估算。

（2）自底向上：由核心小组进行头脑风暴，完成工作任务分解，然后由开发人员进行合理估算，然后累计得到总的估算。

2.软件规模估算

也就是估算要完成的工作范围，常用的方法有LOC和FP估算法。

（1）LOC估算法：代码行估算法，将项目切分为一个个小模块，通过历史项目经验、开发人员经验，估算每个模块的代码行。

（2）FP估算法：FP是指功能点，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。它首先通过外部输入数（input）、外部输出数（output）、外部查询数（inquire）、内部逻辑文件数（file）、外部接口文件数（interface）等五个方面来衡量整个系统的信息处理规模（根据难度乘上系数，难度级别分为低、平均、高三级），然后再从数据通信、分布式处理、性能、配置项、事务率、在线数据、用户使用效率、在线更新、复杂处理、重用性、安装容易程度、操作容易程度、多个地点、修改容易程度等14个方面的复杂度，进行微调，每个方面都在0~0.05之间取值，最后累加出调节因子，再加上0.65得出技术复杂度。

3.软件工作量估算

工作量的单位通常是人月，计算方法为规模/产能=工作量。

（1）IBM模型：是在60多个项目的基础上进行统计得出的静态模型。

（2）Putnum模型：是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。

（3）COCOMO模型：是最有代表性的方法。在该模型中使用了源指令条数（DSI）、开发工作量（MM）、开发进度（TDEV）三个基本量，它将项目分为组织型（相对较小、较简单的项目）、嵌入式（软硬件限制较多的项目）、半独立型（介于两者之间，规模和复杂性中等以上）。它包括基本（静态模型）、中间、详细三种不同的模型。

4.成本估算

得到工作量、人员需求、项目持续时间后，就可以进一步估算成本，通常包括人员成本、资源成本、其他开支等。

进度计划与监控

8.8.2 进度计划与监控

项目的进度安排与任何一个多重任务工作的进度安排基本差不多。项目的进度计划和工作的实际进展情况，通常表现为各项任务之间的进度依赖关系，因而通常使用图表的方式来说明。

1.甘特图

甘特图（Gantt图）使用水平线段表示任务的工作阶段，线段的起点和终点分别对应着任务的开工时间和完成时间，线段的长度表示完成任务所需的时间。而跟踪甘特图则是在甘特图的基础上，加上一个表示现在时间的纵线，可以直观地看出进度是否延误。甘特图的优点在于标明了各任务的计划进度和当前进度，能动态地反映项目进展；其缺点在于难以反映多个任务之间存在的复杂逻辑关系。

2.PERT技术和CPM方法

PERT（计划评审技术）和CPM（关键路径法）都是采用网络图来描述一个项目的任务网络的，通常使用两张图来定义网络图。一张图给出某一特定项目的所有任务，另一张图给出应按照什么次序来完成这些任务，给出各个任务之间的衔接。PERT技术和CPM方法都为项目计划人员提供了一些定量的工具：

（1）确定关键路径：即决定项目开发时间的任务链。

（2）应用统计模型：对每个单独的任务确定最可能的持续时间的估算值。

（3）计算边界时间：为具体的任务定义时间窗口。

3.评估项目进度

最常见的方法是挣值分析，它把实际进度和计划进度进行比较，发现项目是否拖期或超前。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。

上一节 本书简介 下一节

质量管理

8.8.3 质量管理

软件质量是指软件特性的综合，软件满足规定或潜在用户需求的能力。具体地说就是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与隐含特征相一致的程度。

软件质量保证是指为保证软件系统或软件产品充分满足用户要求的质量而进行的有计划、有组织的活动，这些活动贯穿于软件生产的各个阶段即整个生命周期。

影响软件质量的因素主要包括：人、软件需求、开发过程的各个环节、测试的局限性、质量管理的困难性、是否对质量管理予以重视、软件人员的传统习惯、开发规范和支持性的开发工具等方

面。

为了能够统一地描述软件质量特性，形成了许多质量特性标准，其中最常用的有国际通用的ISO/IEC 9126软件质量模型和Mc Call软件质量模型。

1.IEO/IEC 9126模型

该标准已于1996年被采纳为我国的国家标准《GB/T 16120-1996 软件产品评价、质量特性及其使用指南》，其包括以下6类21个质量特性如表8-9所示。

表8-9 GB/T1620-1996软件产品评价、质量特性及其使用指南

质量属性	子 特 性	说　　明
功能性	与功能及其指定的性质有关的一组软件质量	
	适合性	规定任务提供一组功能的能力，以及这组功能的适合程度
	准确性	系统满足需求规格说明和用户目标的程度
	互操作性	同其他指定系统的协同工作能力
	依从性	软件服从有关标准、约定、法规及类似规定的程度
	安全性	避免对程序及数据的非授权故意或意外访问的能力
可靠性	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	
	成熟性	由软件故障引起失效的频度
	容错性	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	在故障发生后重新建立性能水平，恢复数据的能力和时间
易使用性	与使用难易程度及规定或隐含用户对使用方式所做的评价相关的属性	
	易理解性	用户理解该软件系统的难易程度
	易学习性	用户学习使用该软件系统的难易程度
	易操作性	用户操作该软件系统的难易程度
效率	衡量在规定条件下软件的性能水平和所用资源量之间的关系的属性	
	时间特性	响应和处理时间及软件执行其功能时的吞吐量
	资源特性	软件执行其功能时，所使用的资源量及持续使用时间
可维护性	与软件维护的难易程度相关的一组软件属性	
	易分析性	诊断缺陷或失效原因、判定待修改程度的难易程度
	易更改性	修改、排错或适应环境变化的难易程度
	稳定性	修改造成难以预料的后果的风险程度
	易测试性	测试已修改软件的难易程度
可移植性	与从某一环境转移到另一环境的能力有关的属性	
	适应性	软件无需采用特殊处理就能够适应不同规定环境程度
	易安装性	在指定环境下安装软件的难易程度
	一致性	软件服从与可移植性有关的标准或约束的程度
	易替换性	软件在特定软件环境中用来替代指定的其他软件的可能性和难易程度

2.McCall质量模型

McCall质量模型体系如表8-10所示。

表8-10 McCall质量模型体系

类别	质量特性	含　　义
运行	正确性	程序能够满足规格说明和完成用户业务目标的程度
	可靠性	程序能够按要求的精确度实现其预期功能的程度
	效率	程序实现其功能所需要的计算资源量
运行	完整性	软件或数据不受未授权人控制的程度
	使用性	学习、操作程序、为其准备输入数据、解释其输出的工作量
修正	维护性	对运行的程序找到错误并排错的工作量
	测试性	为保证程序执行规定功能所需的测试工作量
	灵活性	修改运行的程序所需的工作量
转移	移植性	将程序从一种硬件配置和/或环境转移到另一硬件配置和/或环境所需工作量
	复用性	程序可被用于与其实现功能相关的其他应用问题的程度
	共运行性	让系统与另一系统协同运行所需的工作量

软件过程改进

8.8.4 软件过程改进

在软件过程改进方面，主要考查软件过程能力成熟度模型（Capability Maturity Model,CMM）和能力成熟度模型集成（Capability Maturity Model Integration,CMMI）。

1.CMM

CMM模型描述和分析了软件过程能力的发展程度，确立了一个软件过程成熟程度的分级标准：

（1）初始级：软件过程的特点是无秩序的，有时甚至是混乱的。软件过程定义几乎处于无章法和步骤可循的状态，软件产品所取得的成功往往依赖极个别人的努力和机遇。初始级的软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许，有些企业制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保证时，那么它仍然被视为初始级。

（2）可重复级：已经建立了基本的项目管理过程，可用于对成本、进度和功能特性进行跟踪。对类似的应用项目，有章可循并能重复以往所取得的成功。焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，一个可重复的过程则能逐渐演化和成熟。从管理角度可以看到一个按计划执行的且阶段可控的软件开发过程。

（3）已定义级：用于管理的和工程的软件过程均已文档化、标准化，并形成整个软件组织的标准软件过程。全部项目均采用与实际情况相吻合的、适当修改后的标准软件过程来进行操作。要求制定企业范围的工程化标准，而且无论是管理还是工程开发都需要一套文档化的标准，并将这些标准集成到企业软件开发标准过程中去。所有开发的项目需根据这个标准过程，剪裁出项目适宜的过程，并执行这些过程。过程的剪裁不是随意的，在使用前需经过企业有关人员的批准。

（4）已管理级：软件过程和产品质量有详细的度量标准。软件过程和产品质量得到了定量的认识和控制。已管理级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品的质量（包括工作产品和提交给用户的产品）需有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品，量化控制将使软件开发真正变成为一个工业生产活动。

（5）优化级：通过对来自过程、新概念和新技术等方面的各种有用信息的定量分析，能够不断地、持续地进行过程改进。如果一个企业达到了这一级，表明该企业能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到最佳。

在CMM中，每个成熟度等级（第一级除外）规定了不同的关键过程域，一个软件组织如果希望达到某一个成熟度级别，就必须完全满足关键过程域所规定的要求，即满足关键过程域的目标。每个级别对应的关键过程域（KPA）见表8-11.

表8-11 关键过程域的分类

过程分类 / 等级	管理方面	组织方面	工程方面
优化级		技术改进管理 过程改进管理	缺陷预防
可管理级	定量管理过程		软件质量管理
已定义级	集成（综合）软件管理 组间协调	组织过程焦点 组织过程定义 培训程序	软件产品工程 同级评审
可重复级	需求管理 软件项目计划 软件项目跟踪与监控 软件子合同管理 软件质量保证 软件配置管理		

2.CMMI

与CMM相比，CMMI涉及面更广，专业领域覆盖软件工程、系统工程、集成产品开发和系统采购。据美国国防部资料显示，运用CMMI模型管理的项目，不仅降低了项目的成本，而且提高了项目的质量与按期完成率。

CMMI可以看作是把各种CMM集成到一个系列的模型中，CMMI的基础源模型包括软件CMM 2.0版（草稿C）、EIA-731系统工程，以及集成化产品和过程开发IPD CMM （IPD） 0.98a版。CMMI也描述了5个不同的成熟度级别。

每一种CMMI模型都有两种表示法：阶段式和连续式。这是因为在CMMI的三个源模型中，CMM是"阶段式"模型，系统工程能力模型是"连续式"模型，而集成产品开发（IPD）CMM是一个混合模型，组合了阶段式和连续式两者的特点。两种表示法在以前的使用中各有优势，都有很多支持者，因此，CMMI产品开发群组在集成这三种模型时，为了避免由于淘汰其中任何一种表示法而失去对CMMI支持的风险，并没有选择单一的结构表示法，而是为每一个CMMI都推出了两种不同表示法的版本。

不同表示法的模型具有不同的结构。连续式表示法强调的是单个过程域的能力，从过程域的角度考察基线和度量结果的改善，其关键术语是"能力";而阶段式表示法强调的是组织的成熟度，从过程域集合的角度考察整个组织的过程成熟度阶段，其关键术语是"成熟度".

尽管两种表示法的模型在结构上有所不同，但CMMI产品开发群组仍然尽最大努力确保了两者在逻辑上的一致性，二者的需要构件和期望部件基本上都是一样的。过程域、目标在两种表示法中都一样，特定实践和共性实践在两种表示法中也不存在根本区别。因此，模型的两种表示法并不存在本质上的不同。组织在进行集成化过程改进时，可以从实用角度出发选择某一种偏爱的表示法，而不必从哲学角度考虑两种表示法之间的差异。

阶段式模型也把组织分为5个不同的级别：

（1）初始级：代表了以不可预测结果为特征的过程成熟度，过程处于无序状态，成功主要取决于团队的技能。

（2）已管理级：代表了以可重复项目执行为特征的过程成熟度。组织使用基本纪律进行需求管理、项目计划、项目监督和控制、供应商协议管理、产品和过程质量保证、配置管理，以及度量和分析。对于级别2而言，主要的过程焦点在于项目级的活动和实践。

（3）严格定义级：代表了以组织内改进项目执行为特征的过程成熟度。强调级别3的关键过程域的前后一致的、项目级的纪律，以建立组织级的活动和实践。

（4）定量管理级：代表了以改进组织性能为特征的过程成熟度。4级项目的历史结果可用来交替使用，在业务表现的竞争尺度（成本、质量、时间）方面的结果是可预测的。

（5）优化级：代表了以可快速进行重新配置的组织性能和定量、持续的过程改进为特征的过程成熟度。

CMMI的具体目标是：

（1）改进组织的过程，提高对产品开发和维护的管理能力。

（2）给出能支持将来集成其他科目CMM的公共框架。

（3）确保所开发的全部有关产品符合将要发布的关于软件过程改进的国际标准ISO/IEC15504对软件过程评估的要求。

使用在CMMI框架内开发的模型具有下列优点：

（1）过程改进能扩展到整个企业级。

（2）先前各模型之间的不一致和矛盾将得到解决。

（3）既有分级的模型表示，也有连续的模型表示，任你选用。

（4）原先单科目过程改进的工作可与其他科目的过程改进工作结合起来。

（5）基于CMMI的评估将与组织原先评估得分相协调，从而保护当前的投资，并与ISO/IEC15504评估结果相一致。

（6）节省费用，特别是当要运用多科目改进时，以及进行相关的培训和评估时。

（7）鼓励组织内各科目之间进行沟通和交流。

上一节　本书简介　下一节

配置管理

8.8.5 配置管理

软件配置管理（Software Configuration Management,SCM）是一种标识、组织和控制修改的技术。软件配置管理应用于整个软件工程过程。我们知道，在软件建立时变更是不可避免的，而变更加剧了项目中软件开发者之间的混乱。SCM活动的目标就是为了标识变更、控制变更、确保变更正确实现并向其他有关人员报告变更。从某种角度讲，SCM是一种标识、组织和控制修改的技术，目的是使错误降为最小并最有效地提高生产效率。

软件配置管理的4 项最基本的活动为配置项标识、变更控制、配置状态报告和配置审核。

1.配置项标识

软件过程的输出信息可以分为三个主要类别：

（1）计算机程序，包括源代码和可执行程序。

（2）描述计算机程序的文档，分别针对技术开发者、管理人员和用户。

（3）数据，包含在程序内部或外部。

这些信息包含了所有的在软件过程中产生的信息，总称为软件配置项（Software Configuration Item,SCI）。由此可见，配置项的识别是配置管理活动的基础，也是制定配置管理计划的重要内容。

在软件的开发过程中把所有需要加以控制的配置项分为基线配置项和非基线配置项两类。例

如：基线配置项可能包括所有的设计文档和源程序等，非基线配置项可能包括项目的各类计划和报告等。基线又称为里程碑，通常作为一个阶段完成的标志。

所有配置项都应按照相关规定统一编号，按照相应的模板生成，并在文档中的规定章节（部分）记录对象的标识信息。在引入软件配置管理工具进行管理后，这些配置项都应以一定的目录结构保存在配置库中。

在配置管理系统中，应建立下列3个库：

（1）开发库（动态系统）：存放开发过程中需要保留的各种信息，供开发人员个人专用。库中的信息可能有较为频繁的修改，只要开发库的使用者认为有必要，无须对其做任何限制。因为这通常不会影响到项目的其他部分。通常仅在项目开发组内设立，并由其负责维护。

（2）受控库（主库）：在信息系统开发的某个阶段工作结束时，将工作产品存入或将有关的信息存入。存入的信息包括计算机可读的以及人工可读的文档资料。通常以软件配置项为单位建立并维护。

（3）产品库（静态系统）：在开发的信息系统产品完成系统测试之后，作为最终产品存入库存内，等待交付用户或现场安装。可在系统、子系统级上设立并维护。

各类库中应存放哪些配置项，应根据所开发软件的实际情况决定。

2.变更控制

在对各个SCI做出了识别，并且利用工具对它们进行了版本管理之后，如何保证它们在复杂多变的开发过程中真正地处于受控的状态，并在任何情况下都能迅速地恢复到任一历史状态，这就成为了软件配置管理的另一重要任务。因此，变更控制就是通过结合人的规程和自动化工具，以提供一个变化控制的机制。

例如：CVS（Concurrent Versions System）是一种广泛应用的、开源的、透明于网络的版本控制系统。它只保存一份源码并记录所有对它的改动。当开发者需要文件的某个特定版本时，CVS会根据那些记录重建出需要的版本。

3.配置状态报告

配置状态报告就是根据配置项操作数据库中的记录来向管理者报告软件开发活动的进展情况。这样的报告应该是定期进行的，并尽量通过CASE（Computer Aided Software Engineering,计算机辅助软件工程）工具自动生成，用数据库中的客观数据来真实地反映各配置项的情况。

配置状态报告应根据报告着重反映当前基线配置项的状态，以作为对开发进度报告的参照。同时也能从中根据开发人员对配置项的操作记录来对开发团队的工作关系做一定的分析。

配置状态报告应该包括下列主要内容：

（1）配置库结构和相关说明。

（2）开发起始基线的构成。

（3）当前基线位置及状态。

（4）各基线配置项集成分支的情况。

（5）各私有开发分支类型的分布情况。

（6）关键元素的版本演进记录。

（7）其他应该报告的事项。

4.配置审核

配置审核的主要作用是作为变更控制的补充手段，来确保某一变更需求已被切实实现。在某些

情况下，它被作为正式的技术复审的一部分，但当软件配置管理是一个正式的活动时，该活动由SQA人员单独执行。

风险管理

8.8.6 风险管理

风险是指可能给项目的成功带来威胁或损失的情况，而风险管理是指在风险给项目带来损失之前，就指明、评估并对风险加以控制，使用工具和方法把项目风险限制在一个可接受的范围内。

1.风险的定义

Robert Charette在他关于风险分析和驾驭的书中对风险的概念给出定义。他所关心的有三个方面。

（1）关心未来：风险是否会导致软件项目失败？

（2）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及时工作和全面完成有关的实体中会发生什么样的变化？

（3）关心选择：应采用什么方法和工具，应配备多少人力，在质量上强调到什么程度才满足要求？

2.风险的类型

进行风险分析时，重要的是量化不确定性的程度和与每个风险相关的损失的程度。为了实现这点，必须考虑不同类型的风险。

（1）项目风险：指潜在的预算、进度、人力（工作人员及组织）、资源、客户和需求等方面的问题以及它们对软件项目的影响。例如：项目复杂性、规模和结构不确定性等都是项目风险。项目风险威胁到项目计划，也就是说，如果项目风险变成现实，有可能会拖延项目的进度，增加项目的成本。

（2）技术风险：指潜在的设计、实现、接口、验证和维护等方面的问题。此外，规约的二义性、技术的不确定性、陈旧的技术和"先进的"技术也是技术风险因素。技术风险威胁到要开发软件的质量及交付时间，如果技术风险变成现实，则开发工作可能变得很困难或根本不可能。

（3）商业风险：在信息系统项目中，商业风险威胁到要开发系统的生存能力。一般主要有5类商业风险：

开发了一个没有人真正需要的优秀产品或系统（市场风险）。

开发的产品不再符合公司的整体商业策略（策略风险）。

开发了一个销售部门不知道如何去卖的产品（销售风险）。

由于重点的转移或人员的变动而失去了高级管理层的支持（管理风险）。

没有得到预算或人力上的保证（预算风险）。

驾驭三个主要活动。

包括项目风险、技术风险和商业风险三种。

（2）风险评估：可分为定量评估和定性评估。通过对各种风险发生的可能性和破坏性这两个方面进行评估，并将它们按优先级进行排列。在进行软件工程分析时，项目管理人员要进行四种风险评估活动，包括建立表示风险概率的尺度，描述风险引起的后果，估计风险影响的大小，确定风险估计的正确性。

（3）风险驾驭：利用某种技术，如原型化、软件自动化、软件心理学、可靠性工程学等方法设法避开风险。

制定风险应对计划时有多种不同的策略，对于相同的风险，采用不同[illegible]会有不同的应对方法。通常可以把风险应对策略分为两种类型，即防范策略和响应策略。[illegible]指的是在风险发生前，项目组会采取一定的措施对风险进行控制的方法；而响应策略则[illegible]生后采取的相应措施以降低风险带来的损失。

1.风险防范策略

消极的风险防范策略是最常用的策略，其目的是降低风险发生的概率[illegible]险带来的损失。例如：规避策略、转移策略和减轻策略。

（1）规避策略指的是想方设法阻止风险的发生或消除风险发生的危害。避免策略如果成功则可以消除风险对项目的影响。例如：针对技术风险可以采取聘请技术专家、针对项目进度风险可以采取延长项目时间或缩减项目范围的办法。

（2）转移策略指的是将风险转嫁给其他的组织或个体，通过这种方式来降低风险发生后的损失。例如：在固定成本的项目中，进行需求签字确认，对于超出签字范围的需求变更需要客户增加费用。这种方式就是一种将需求风险转移的策略。经过转移的风险并没有消失，其发生的可能性也没有变化，但对于项目组而言，风险发生后的损失降低了。

（3）减轻策略。当风险很难避免或转移时，可以考虑采取减轻策略来降低风险发生的概率或减轻风险带来的损失。风险是一种不确定因素，可以通过前期的一些工作来降低风险发生的可能性；或者也可以通过一些准备来降低风险发生的损失。例如对于需求风险，如果认为需求变化可能很剧烈，那么可以考虑采用柔性设计的方法降低需求变更的代价。尤其对于IT项目而言，越早发现问题越容易解决，例如对于需求风险带来的问题，在设计阶段发现要好过编码阶段才发现。针对这种特点，也可以采用尽早暴露风险的方法降低发生风险的损失。

需要说明的是制定出的风险防范措施需要对应到项目进度表中，安排出专门的人员来执行一些工作来防范风险的发生。否则制定出风险防范措施也不会对项目有太大的意义。

2.制定风险响应策略

虽然我们采用了很多方法来防范风险的发生。但风险本身就是一种不确定因素，不可能在项目中完全消除。那么，我们还需要制定一些风险发生后的应急措施来解决风险带来的问题。例如：对于系统性能的风险，由于不清楚目前的系统架构是否能够满足用户的需求，可能在系统发布后出现系统性能不足的问题。对于这个风险，我们可以定义其风险响应策略为增加硬件资源以提高系统性能。

风险响应策略同风险防范策略不同，无论风险是否发生，风险防范策略都需要体现在项目计划中，在项目过程中需要有人来执行对应的防范策略；而风险响应策略是事件触发的，直到当风险发生后才会被执行，如果始终没有发生该风险，则始终不会被安排到项目活动中。

3.风险曝光度

风险曝光度（risk exposure）将表示实际损失的可能性与表示大量可能损失的信息结合到单一数字评估中，在形式最简单的定量风险分析中，风险曝光度可通过将风险可能性及影响相乘算出，即其计算公式为：

风险曝光度=风险损失×风险概率

例如：正在开发的软件项目可能存在一个未被发现的错误，这个错误出现的概率是0.5%,给公司造成的损失将是100万元，那么这个错误的风险曝光度是5000元。

第 8 章：系统开发和运行维护　作者：希赛教育软考学院　来源：希赛网　2014年05月21日

软件文档

8.8.7 软件文档

文档是信息系统产品的重要组成部分，对于开发人员、管理人员以及用户都是十分重要的辅助文件。定义清晰、维护及时的文档能够帮助开发人员理解需求、顺畅沟通；帮助管理人员了解进度、加强管理；帮助用户更好地使用和维护软件。因此，对于信息系统监理师而言，必须掌握系统文档编制的技能。

1.文档的分类

根据文档产生、使用的范围的不同，可以将其分为三大类：

（1）开发文档：为开发工作提供支持的各种文档，其读者群主要针对开发人员。其中主要包括需求规格说明书、数据要求规格说明书、概要设计说明书、详细设计说明书、项目开发计划等。

（2）管理文档：为项目的开发管理提供支持的各种文档，其读者群主要针对管理人员，其中主要包括可行性研究报告、项目开发计划、测试计划、技术报告、开发进度记录、项目开发总结报告等。

（3）用户文档：向用户传达各种与开发相关、与产品相关的信息，其读者群主要针对最终用户。其中主要包括用户手册、操作手册、维护修改建议书、软件需求说明书等。

2.文档产生的阶段

在编写文档时，可以采用自然语言直接编写，也可以采用形式化语言来编写，还可以借助各种图表提高可读性，例如UML、DFD、E-R图等。

一般来说，用瀑布模型进行系统开发的过程中，每个阶段产生的文档为：

（1）可性行研究阶段：可行性研究报告。

（2）项目计划阶段：项目开发计划。

（3）需求分析阶段：软件需求说明书，数据要求规格说明书、系统测试计划、确认测试计划、用户手册。

（4）概要设计阶段：概要设计说明书，集成测试计划。

（5）详细设计阶段：详细设计说明书，单元测试计划。

（6）编码与单元测试阶段：用户手册、操作手册。

（7）集成测试阶段：测试分析报告，项目开发总结。

（8）运行维护阶段：维护修改建议书。

3.文档的主要内容

（1）可行性研究报告：说明该软件项目的实现在技术上、经济上和社会因素上的可行性，评述为合理地达到开发目标可供选择的各种可能的实现方案，说明并论证所选定实施方案的理由。

（2）项目开发计划：为软件项目实施方案制定出的具体计划。它应包括各部分工作的负责人员、开发的进度、开发经费的概算、所需的硬件和软件资源等。项目开发计划应提供给管理部门，并作为开发阶段评审的基础。

（3）软件需求规格说明：对所开发软件的功能、性能、用户界面及运行环境等作出详细的说明。它是用户与开发人员双方对软件需求取得共同理解基础上达成的协议，也是实施开发工作的基础。

（4）数据要求规格说明：给出数据逻辑描述和数据采集的各项要求，为生成和维护系统的数据文件做好准备。

（5）概要设计规格说明：是概要设计工作阶段的成果。它说明系统的功能分配、模块划分、程序的总体结构、输入输出及接口设计、运行设计、数据结构设计和出错处理设计等，为详细设计奠定基础。

（6）详细设计规格说明：着重描述每个模块如何实现，包括实现算法、逻辑流程等。

（7）用户手册：详细描述软件的功能、性能和用户界面，使用户了解如何使用该软件。

（8）操作手册：为操作人员提供该软件各种运行情况的有关知识，特别是操作方法细节。

（9）测试计划：针对组装测试和确认测试，需要为组织测试制定计划。计划应包括测试的内容、进度、条件、人员、测试用例的选取原则、测试结果允许的偏差范围等。

（10）测试分析报告：测试工作完成以后，应当提交测试计划执行情况的说明。对测试结果加以分析，并提出测试的结论性意见。

（11）开发进度月报：该月报是软件人员按月向管理部门提交的项目进展情况的报告。报告应包括进度计划与实际执行情况的比较、阶段成果、遇到的问题和解决的办法以及下个月的打算等。

（12）项目开发总结报告： 软件项目开发完成之后，应当与项目实施计划对照，总结实际执行的情况，如进度、成果、资源利用、成本和投入的人力。此外，还需对开发工作作出评价，总结经验和教训。

（13）维护修改建议：软件产品投入运行之后，可能有修正、更改等问题，应当对存在的问题、修改的考虑以及修改的影响估计等做详细的描述，写成维护修改建议，提交审批。

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05月21日

考点分析

第9章　面向对象方法

面向对象方法是软件设计师级别考试的一个考试重点，上午考试的分数中它大约占12分，下午考试通常有一道面向对象设计的试题，占15分。