**第1章　软件工程学概述**

**1.1　复习笔记**

一、软件危机

1．软件危机的介绍

（1）概念

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。这些问题绝不仅仅是不能运行的软件才具有的，实际上，几乎所有的软件都不同程度地存在这些问题。软件危机包含两方面的问题：

①如何开发软件，以满足对软件日益增长的需求；

②如何维护数量不断膨胀的已有软件。

（2）典型表现

①对软件开发成本和进度的估计不准确。实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

②用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就匆忙着手编写程序。软件开发人员和用户之间的信息交流往往很不充分，“闭门造车”必然导致最终的产品不符合用户的实际需要。

③软件产品的质量往往靠不住。软件可靠性和质量保证的确切的定量概念刚刚出现不久，软件质量保证技术（审查、复审、程序正确性证明和测试）还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

④软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

⑤软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。

⑥软件成本在计算机系统总成本中所占的比例逐年上升。由于微电子学技术的进步和生产自动化程度的不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。

⑦软件开发跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象使人类不能充分利用现代计算机硬件提供的巨大潜力。

2．产生软件危机的原因

（1）客观原因

①软件是计算机系统中的逻辑部件而不是物理部件。

②软件规模庞大，而且程序复杂性将随着程序规模的增加而呈指数上升。

（2）主观原因

①对软件开发和维护有不少糊涂观念，采用了错误的方法和技术。

②存在与软件开发和维护有关的许多错误认识和做法。

③对用户要求没有完整准确的认识就匆忙着手编写程序。

④一个软件从定义、开发、使用和维护，直到最终被废弃，要经历一个漫长的时期。

⑤一个软件产品必须由一个完整的配置组成，主要包括程序、文档和数据等成分。

⑥在软件开发的不同阶段进行修改需要付出的代价是很不相同的。

⑦轻视维护是一个最大的错误。

3．消除软件危机的途径

（1）应该对计算机软件有一个正确的认识。应该彻底消除在计算机系统早期发展阶段形成的“软件就是程序”的错误观念。软件是程序、数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列；数据是使程序能够适当地处理信息的数据结构；文档是开发、使用和维护程序所需要的图文资料。

（2）应该充分认识到软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法，特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。

（3）应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索更好更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法。

（4）应该开发和使用更好的软件工具。在软件开发的每个阶段都有许多繁琐重复的工作需要做，在适当的软件工具辅助下，开发人员可以把这类工作做得既快又好。

二、软件工程

1．软件工程的介绍

（1）概念

软件工程是是从管理和技术两方面研究如何更好地开发和维护计算机软件的一门新兴学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。

（2）产生

为了更有效地开发与维护软件，软件工作者在20世纪60年代后期开始认真研究消除软件危机的途径，从而逐渐形成了一门新兴的工程学科—计算机软件工程学（通常简称为“软件工程”）。

（3）本质特征

①软件工程关注于大型程序的构造

把一个人在较短时间内写出的程序称为小型程序，而把多人合作用时半年以上才写出的程序称为大型程序。传统的程序设计技术和工具是支持小型程序设计的，不能简单地把这些技术和工具用于开发大型程序。

②软件工程的中心课题是控制复杂性

软件所解决的问题通常十分复杂，不得不把问题分解，使得分解出的每个部分是可理解的，而且各部分之间保持简单的通信关系。用这种方法并不能降低问题的整体复杂性，但是却可使它变成可以管理的。

注意：许多软件的复杂性主要不是由问题的内在复杂性造成的，而是由必须处理的大量细节造成的。

③软件经常变化

绝大多数软件都模拟了现实世界的某一部分。现实世界在不断变化，软件为了不被很快淘汰，必须随着所模拟的现实世界一起变化。因此，在软件系统交付使用后仍然需要耗费成本，而且在开发过程中必须考虑软件将来可能发生的变化。

④开发软件的效率非常重要

社会对新应用系统的需求超过了人力资源所能提供的限度，软件供不应求的现象日益严重。软件工程的一个重要课题就是，寻求开发与维护软件的更好更有效的方法和工具。

⑤和谐地合作是开发软件的关键

软件处理的问题十分庞大，必须多人协同工作才能解决这类问题。为了有效地合作，必须明确地规定每个人的责任和相互通信的方法，每个人还必须严格地按规定行事，纪律是成功地完成软件开发项目的一个关键。

⑥软件必须有效地支持它的用户

开发软件的目的是支持用户的工作。软件提供的功能应该能有效地协助用户完成他们的工作。有效地支持用户意味着必须仔细地研究用户，以确定适当的功能需求、可用性要求及其他质量要求。还意味着软件开发不仅应该提交软件产品，而且应该写出用户手册和培训材料。

⑦软件工程领域由具有一种文化背景的人替具有另一种文化背景的人创造产品

软件工程师是软件设计、软件体系结构、测试或统一建模语言等方面的专家，但他们不仅缺乏应用领域和文化领域的实际知识，还缺乏该领域的文化知识，这是软件开发项目出现问题的常见原因。

2．软件工程的基本原理

（1）七条基本原理

①用分阶段的生命周期计划严格管理

在软件开发与维护的漫长的生命周期中，需要完成许多性质各异的工作。应该把软件生命周期划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件的开发与维护工作进行管理。不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划。

②坚持进行阶段评审

由于大部分错误都是在编码之前造成的，并且错误发现与改正的越晚，所需付出的代价也越高，所以软件的质量保证工作不能等到编码阶段结束之后再进行。因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的错误，是一条必须遵循的重要原则。

③实行严格的产品控制

当改变需求时，为了保持软件各个配置成分的一致性，必须实行严格的产品控制，其中主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分。

基准配置管理也称为变动控制，主要包括以下两个方面。

a．一切有关修改软件的建议，特别是涉及对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。

b．绝对不能谁想修改软件（包括尚在开发过程中的软件），就随意进行修改。

④采用现代程序设计技术

采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量。

⑤结果应能清楚地审查

软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员的工作进展情况可见性差，难以准确度量。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查。

⑥开发小组的人员应该少而精

软件开发小组的组成人员的素质应该好，而人数则不宜过多。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。

⑦承认不断改进软件工程实践的必要性

为了保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。例如，收集进度和资源耗费数据，收集出错类型和问题报告数据等。

（2）意义

这七条原理是确保软件产品质量和开发效率的原理的最小集合。这七条原理是互相独立的，而且是缺一不可的最小集合，然而这七条原理又是相当完备的。

3．软件工程方法学

（1）概念

软件工程包括技术和管理两方面的内容，是技术与管理紧密结合所形成的工程学科。所谓管理就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。通常把软件生命周期全过程中使用的一整套技术方法的集合称为方法学，也称为范型。目前使用得最广泛的软件工程方法学，分别是传统方法学和面向对象方法学。

（2）三要素

软件工程方法学包括3个要素：方法、工具和过程。

①方法：是完成软件开发的各项任务的技术方法，回答“怎样做”的问题。

②工具：是为运用方法而提供的自动的或半自动的软件工程支撑环境。

③过程：是为了获得高质量的软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。

（3）传统方法学（生命周期方法学或结构化范型）

①定义

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，并使用适当的软件工具或软件工程环境来支持结构化技术的运用。

②开发步骤

开发软件时，传统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。具体如下。

a．从对问题的抽象逻辑分析开始，一个阶段一个阶段地顺序进行开发。

b．前一个阶段任务的完成是开始进行后一个阶段工作的前提和基础，而后一阶段任务的完成通常是使前一阶段提出的解法更进一步具体化，加了更多的实现细节。

c．每一个阶段的开始和结束都有严格标准，对于任何两个相邻的阶段而言，前一阶段的结束标准就是后一阶段的开始标准。

d．在每一个阶段结束之前都必须进行正式严格的技术审查和管理复审，从技术和管理两个方面对这个阶段的开发成果进行检查，通过之后这个阶段才算结束；如果没通过检查，则必须进行必要的返工，而且返工后还要再经过审查。审查的一条主要标准就是每个阶段都应该交出和所开发的软件完全一致的高质量的文档资料，从而保证在软件开发工程结束时有一个完整准确的软件配置交付使用。

③优点

a．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工程的困难程度；

b．在软件生命周期的每个阶段都采用科学的管理技术和良好的技术方法，且在每个阶段结束之前都进行严格的审查，使得软件开发工程的全过程以一种有条不紊的方式进行，保证了软件的质量，特别是提高了软件的可维护性；

c．采用生命周期方法学可以大大提高软件开发的成功率，软件开发的生产率也能明显提高。

④缺点

a．当软件规模庞大，或者对软件的需求是模糊的或会随时间变化而变化的时候，传统方法学开发软件往往不成功。

b．结构化范型技术要么面向行为，要么面向数据，没有既面向数据又面向行为的结构化技术，而软件系统本质上是信息处理系统。离开了操作便无法更改数据，而脱离了数据的操作是毫无意义的。数据和对数据的处理原本是密切相关的，把数据和操作人为地分离成两个独立的部分，自然会增加软件开发与维护的难度。

（4）面向对象方法学

①定义

面向对象方法把数据和行为看成是同等重要的，它是一种以数据为主线，把数据和对数据的操作紧密地结合起来的方法。

②基本原则

面向对象方法学的出发点和基本原则，是尽量模拟人类习惯的思维方式，使开发软件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，从而使描述问题的问题空间与实现解法的解空间在结构上尽可能一致。

③要点

a．把对象作为融合了数据及在数据上的操作行为的统一的软件构件

面向对象程序是由对象组成的，程序中任何元素都是对象，复杂对象由比较简单的对象组合而成，用对象分解取代了传统方法的功能分解。

b．把所有对象都划分成类

每个类都定义了一组数据和一组操作，类是对具有相同数据和相同操作的一组相似对象的定义。数据用于表示对象的静态属性，是对象的状态信息，而施加于数据之上的操作用于实现对象的动态行为。

c．继承性

按照父类与子类的关系，把若干个相关类组成一个层次结构的系统。下层派生类自动拥有上层基类中定义的数据和操作，这种现象称为继承。

d．对象彼此间仅能通过发送消息互相联系

对象是数据处理的主体，必须向它发消息请求它执行它的某个操作以处理它的数据，不能从外界直接对它的数据进行处理，即对象的所有私有信息都被封装在该对象内，不能从外界直接访问，这就是通常所说的封装性。

④与传统方法学的对比

a．传统方法学强调自顶向下顺序地完成软件开发的各阶段任务。

b．用面向对象方法学开发软件的过程，是一个主动地多次反复迭代的演化过程。面向对象方法在概念和表示方法上的一致性，保证了在各项开发活动之间的平滑（即无缝）过渡。

⑤优点

a．降低了软件产品的复杂性；

b．提高了软件的可理解性；

c．简化了软件的开发和维护工作；

d．提高了软件的可重用性。

三、软件生命周期

软件生命周期由软件定义、软件开发和运行维护（也称为软件维护）3个时期组成，每个时期又进一步划分成若干个阶段。

1．组成

（1）软件定义

软件定义时期通常进一步划分成3个阶段，即问题定义、可行性研究和需求分析。这个时期的总任务是：

①确定软件开发工程必须完成的总目标；

②确定工程的可行性；

③导出实现工程目标应该采用的策略及系统必须完成的功能；

④估计完成该项工程需要的资源和成本，并且制定工程进度表。

（2）软件开发

通常由下述4个阶段组成：总体设计，详细设计，编码和单元测试，综合测试。其中前两个阶段又称为系统设计，后两个阶段又称为系统实现。

（3）运行和维护

软件运行和维护的主要任务是使软件持久地满足用户的需要。出现下述三种情况的时候需要进行维护：

①当软件在使用过程中发现错误时应该加以改正；

②当环境改变时应该修改软件以适应新的环境；

③当用户有新要求时应该及时改进软件以满足用户的新需要。

2．软件生命周期各阶段的基本任务。

（1）问题定义

问题定义阶段必须回答的关键问题是：“要解决的问题是什么？”。通过对客户的访问调查，系统分析员扼要地写出关于问题性质、工程目标和工程规模的书面报告，经过讨论和必要的修改之后这份报告应该得到客户的确认。

（2）可行性研究

可行性研究阶段要回答的关键问题是：“对于上一个阶段所确定的问题有行得通的解决办法吗？”为了回答这个问题，系统分析员需要进行一次大大压缩和简化了的系统分析和设计过程。可行性研究阶段的任务是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。可行性研究的结果是客户作出是否继续进行这项工程的决定的重要依据，只有投资可能取得较大效益的那些工程项目才值得继续进行下去。

（3）需求分析

需求分析阶段的任务是准确地确定“为了解决这个问题，目标系统必须做什么”，主要是确定目标系统必须具备哪些功能。系统分析员必须和用户密切配合，充分交流信息，以得出经过用户确认的系统逻辑模型。用数据流图、数据字典和简要的算法表示系统的逻辑模型。这个阶段的一项重要任务，是用正式文档准确地记录对目标系统的需求，这份文档通常称为软件需求规格说明书（SRS）。

（4）总体设计（概要设计）

总体设计阶段必须回答的关键问题是：“应该怎样实现目标系统?”主要是设计出实现目标系统的几种可能的方案。软件工程师应该用适当的表达工具描述每种方案，分析每种方案，推荐一个最佳方案并制定出详细计划。另一项主要任务就是设计程序的体系结构，即确定程序由哪些模块组成以及模块间的关系。

（5）详细设计（模块设计）

详细设计阶段的任务就是把设计方案具体化，也就是回答：“应该怎样具体地实现这个系统呢？”这个阶段的任务是设计出程序的详细规格说明。在这个阶段将详细地设计每个模块，确定实现模块功能所需要的具体算法和数据结构。

（6）编码和单元测试

编码和单元测试阶段的关键任务是写出正确的、容易理解的、容易维护的程序模块。程序员把详细设计的结果翻译成用选定的高级编程语言书写的程序，编写出的每一个模块，并对编写好的各个模块进行测试。

（7）综合测试

综合测试阶段的关键任务是通过各种类型的测试使软件达到预定的要求。最基本的测试是集成测试和验收测试。

a．集成测试是根据设计的软件结构，把经过单元测试检验的模块按某种选定的策略装配起来，在装配过程中对程序进行必要的测试。

b．验收测试则是按照软件需求规格说明书的规定，由用户对目标系统进行验收。

通过对软件测试结果的分析可以预测软件的可靠性；根据对软件可靠性的要求，也可以决定测试和调试过程什么时候可以结束。应该用正式的文档资料把测试计划、详细测试方案以及实际测试结果保存下来，作为软件配置的一个组成部分。

（8）软件维护

维护阶段的关键任务是通过各种必要的维护活动使系统持久地满足用户的需要。实质上维护过程是经历了一次压缩和简化了的软件定义和开发的全过程。每一项维护活动都应该准确地记录下来，作为正式的文档资料加以保存。

通常有四类维护活动：

①改正性维护，即诊断和改正在使用过程中发现的软件错误；

②适应性维护，即修改软件以适应环境的变化；

③完善性维护，即根据用户的要求改进或扩充软件使它更完善；

④预防性维护，即修改软件，为将来的维护活动预先做准备。

四、软件过程

1．概念

（1）定义

软件过程是为了获得高质量软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。软件过程描述为了开发出客户需要的软件，什么人（who）、在什么时候（when）、做什么事（what）以及怎样（how）做这些事以实现某一个特定的具体目标。在完成开发任务时必须进行一些开发活动，并且使用适当的资源，在过程结束时将把输入转化为输出。

（2）构成

科学、有效的软件过程应该定义一组适合于所承担的项目特点的任务集合。一个任务集合包括一组软件工程任务、里程碑和应该交付的产品。

（3）表示

使用生命周期模型简洁地描述软件过程。生命周期模型（过程模型）规定了把生命周期划分成哪些阶段及各个阶段的执行顺序。

2．典型软件过程模型

（1）瀑布模型

在20世纪80年代之前，瀑布模型一直是唯一被广泛采用的生命周期模型，现在它仍然是软件工程中应用得最广泛的过程模型。传统软件工程方法学的软件过程，基本上可以用瀑布模型来描述。

①传统的瀑布模型

a．定义

传统软件工程方法学的软件过程，基本上可以用瀑布模型来描述，如图1-1所示为传统的瀑布模型。

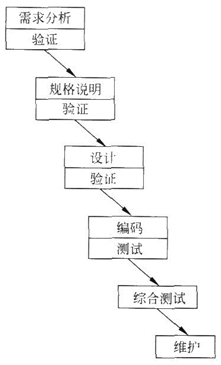


图1-1 传统的瀑布模型

b．特点

第一，阶段间具有顺序性和依赖性

必须等前一阶段的工作完成之后，才能开始后一阶段的工作；前一阶段的输出文档是后一阶段的输入文档，即只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。

第二，推迟实现的观点

瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。

第三，质量保证的观点

为了保证所开发的软件的质量，在瀑布模型的每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。完整、准确的合格文档不仅是软件开发时期各类人员之间相互通信的媒介，也是运行时期对软件进行维护的重要依据。越是早期阶段犯下的错误，暴露出来的时间就越晚，排除故障改正错误所需付出的代价也越高。因此，及时审查，是保证软件质量、降低软件成本的重要措施。每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

②实际的瀑布模型

a．定义

在设计阶段可能发现规格说明文档中的错误，而设计上的缺陷或错误可能在实现过程中显现出来，在综合测试阶段将发现需求分析、设计或编码阶段的许多错误。因此，实际的瀑布模型是带“反馈环”的，如图1-2所示（图中实线箭头表示开发过程，虚线箭头表示维护过程）。当在后面阶段发现前面阶段的错误时，需要沿图中左侧的反馈线返回前面的阶段，修正前面阶段的产品之后再回来继续完成后面阶段的任务。

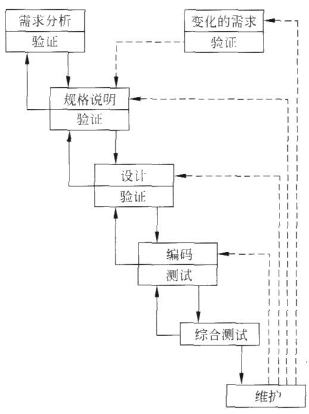


图1-2 实际的瀑布模型

b．优点

第一，可强迫开发人员采用规范的方法；

第二，严格地规定了每个阶段必须提交的文档；

第三，要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证；

第四，对文档的约束，使软件维护变得容易一些，且能降低软件预算。

c．缺点

“瀑布模型是由文档驱动的”是它的一个主要缺点。在可运行的软件产品交付给用户之前，用户只能通过文档来了解产品是什么样的。但是，仅仅通过写在纸上的静态的规格说明，很难全面正确地认识动态的软件产品。由于瀑布模型几乎完全依赖于书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要。

（2）快速原型模型

①定义

快速原型是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。如图1-3所示（图中实线箭头表示开发过程，虚线箭头表示维护过程）。快速原型的本质是“快速”。UNIX Shell、超文本、第四代语言（4GL）都是广泛使用的快速原型语言。

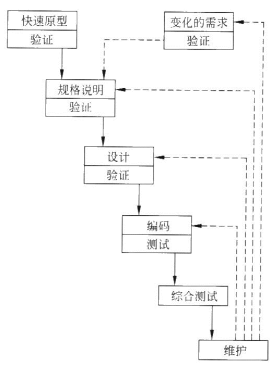


图1-3 快速原型模型

②原理

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原型系统，让用户通过实践来了解目标系统的概貌。通常，用户试用原型系统之后会提出许多修改意见，开发人员按照用户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做他们所需要的工作，开发人员便可据此书写规格说明文档，根据这份文档开发出的软件便可以满足用户的真实需求。

③优点

快速原型模型是不带反馈环的，这正是这种过程模型的主要优点，即软件产品的开发基本上是线性顺序进行的。能基本上做到线性顺序开发的主要原因如下：

a．原型系统已经通过与用户交互而得到验证，产生的规格说明文档正确地描述了用户需求。因此，在开发过程的后续阶段不会因为发现了规格说明文档的错误而进行较大的返工。

b．开发人员通过建立原型系统已经学到了许多东西。因此，在设计和编码阶段发生错误的可能性也比较小，减少了在后续阶段需要改正前面阶段所犯错误的可能性。

（3）增量模型

①原理

增量模型也称为渐增模型，如图1-4所示。使用增量模型开发软件时，把软件产品作为一系列的增量构件来设计、编码、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。使用增量模型时，第一个增量构件往往实现软件的基本需求，提供最核心的功能。把软件产品分解成增量构件时，应该使构件的规模适中。分解时唯一必须遵守的约束条件是：当把新构件集成到现有软件中时，所形成的产品必须是可测试的。



图1-4 增量模型

②原理

增量模型分批地逐步向用户提交产品，整个软件产品被分解成许多个增量构件，开发人员一个构件接一个构件地向用户提交产品。从第一个构件交付之日起，用户就能做一些有用的工作。

③优点

a．能在较短时间内向用户提交可完成部分工作的产品。

b．逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品，减少了一个全新的软件可能给客户组织带来的冲击。

④技术难点

a．在把每个新的增量构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。必须把软件的体系结构设计得便于按这种方式进行扩充，向现有产品中加入新构件的过程必须简单、方便，即软件体系结构必须是开放的。

b．增量模型本身是自相矛盾的。它一方面要求开发人员把软件看作一个整体，另一方面又要求开发人员把软件看作构件序列，每个构件本质上都独立于另一个构件。

⑤风险更大的增量模型

图1-5是一种风险更大的增量模型。用这种方式开发软件，不同的构件将并行地构建，因此有可能加快工程进度。但是，这种方法将有构件无法集成到一起的风险，除非密切地监控整个开发过程，否则整个工程可能毁于一旦。

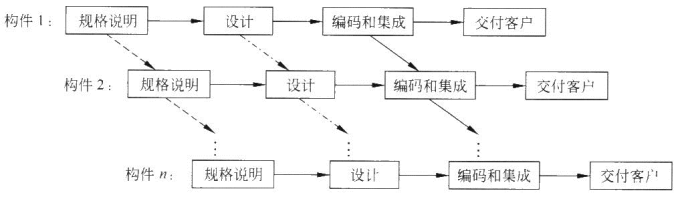


图1-5 风险更大的增量模型

（4）螺旋模型

①基本思想

螺旋模型的基本思想是，使用原型及其他方法来尽量降低风险。可以把它看作在每个阶段之前都增加了风险分析过程的快速原型模型，如图1-6所示。

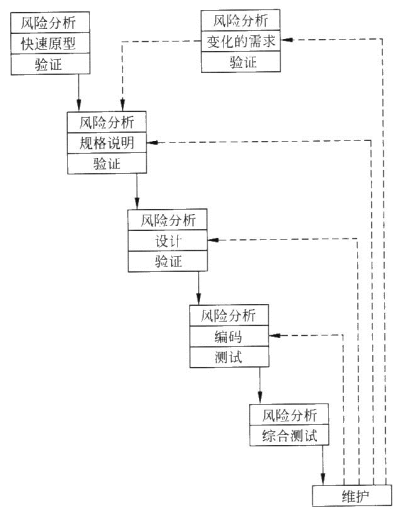


图1-6 简化的螺旋模型

②原理

完整的螺旋模型如图1-7所示。图中带箭头的点划线的长度代表当前累计的开发费用，螺旋线的角度值代表开发进度。螺旋线每个周期对应于一个开发阶段。每个阶段开始时的任务是，确定该阶段的目标、为完成这些目标选择方案及设定这些方案的约束条件。接下来的任务是，从风险角度分析上一步的工作结果，用建造原型的方法排除各种潜在的风险。如果风险不能排除，则停止开发工作或大幅度地削减项目规模。如果成功地排除了所有风险，则启动下一个开发步骤。最后是评价该阶段的工作成果并计划下一个阶段的工作。

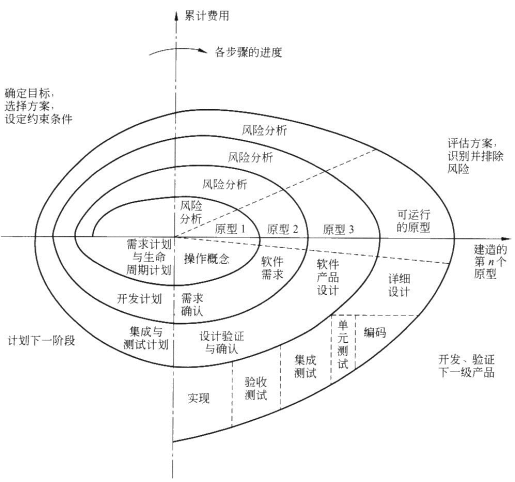


图1-7 完整的螺旋模型

③应用

螺旋模型主要适用于内部开发的大规模软件项目。如果进行风险分析的费用接近整个项目的经费预算，则风险分析是不可行的。事实上，项目越大，风险也越大，因此，进行风险分析的必要性也越大。此外，只有内部开发的项目，才能在风险过大时方便地中止项目。

④优点

a．对可选方案和约束条件的强调有利于已有软件的重用，有助于把软件质量作为软件开发的一个重要目标；

b．减少了过多测试（浪费资金）或测试不足（产品故障多）所带来的风险；

c．在螺旋模型中维护只是模型的另一个周期，在维护和开发之间并没有本质区别。

⑤缺点

螺旋模型是风险驱动的。因此，除非软件开发人员具有丰富的风险评估经验和这方面的专门知识，否则将出现真正的风险，即当项目实际上正在走向灾难时，开发人员可能还认为一切正常。

（5）喷泉模型

①面向对象方法学

这种方法在开发的早期阶段定义了一系列面向问题的对象，并且在整个开发过程中不断充实和扩充这些对象。分析阶段得到的对象模型也适用于设计阶段和实现阶段。整个开发过程都是“无缝”连接的，这就很容易实现各个开发步骤的多次反复迭代，达到认识的逐步深化。每次反复都会增加或明确一些目标系统的性质，但不是对先前工作结果的本质性改动，减少了不一致性，降低了出错的可能性。

②原理

图1-8所示的喷泉模型，是典型的面向对象的软件过程模型之一。“喷泉”这个词体现了面向对象软件开发过程迭代和无缝的特性。图中代表不同阶段的圆圈相互重叠，表示两个活动之间存在交迭；而面向对象方法在概念和表示方法上的一致性，保证了在各项开发活动之间的无缝过渡。图中在一个阶段内的向下箭头代表该阶段内的迭代（或求精）。图中较小的圆圈代表维护，圆圈较小象征着采用了面向对象范型之后维护时间缩短了。

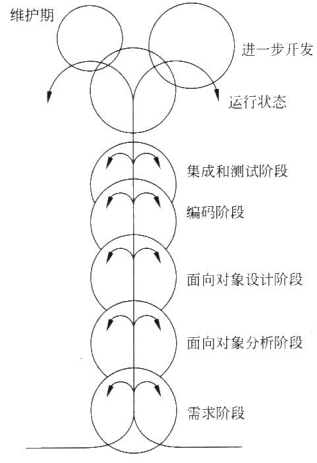


图1-8 喷泉模型

③要求

为避免使用喷泉模型开发软件时开发过程过分无序，应该把一个线性过程作为总目标。同时也应该记住，面向对象范型本身要求经常对开发活动进行迭代或求精。

（6）Rational统一过程

Rational统一过程（Rational Unified Process，RUP）是由Rational软件公司推出的一种完整而且完美的软件过程。

①最佳实践

RUP总结了经过多年商业化验证的6条最有效的软件开发经验，这些经验被称为“最佳实践”，具体如下。

a．迭代式开发

第一，产生原因

在大型复杂软件的开发过程中客户的需求会经常改变，因此需要有一种能够通过一系列细化、若干个渐近的反复过程而得出有效解决方案的迭代方法。

第二，优点

i．迭代式开发允许在每次迭代过程中需求都可以有变化，这种开发方法通过一系列细化来加深对问题的理解，因此能更容易地容纳需求的变更。

ii．采用迭代式开发方法，每个迭代过程以完成可执行版本结束，这不仅使最终用户可以不断地介入和提出反馈意见，而且开发人员也因随时有一个可交付的版本而提高了士气。

b．管理需求

在开发软件的过程中，客户需求将不断发生变化，因此，确定系统的需求是一个连续的过程。RUP描述了如何提取、组织系统的功能性需求和约束条件并把它们文档化。使用用例和脚本是捕获功能性需求的有效方法，RUP采用用例分析来捕获需求，并由它们驱动设计和实现。

c．使用基于构件的体系结构

所谓构件就是功能清晰的模块或子系统。系统可以由已经存在的、由第三方开发商提供的构件组成，因此构件使软件重用成为可能。RUP提供了使用现有的或新开发的构件定义体系结构的系统化方法，从而有助于降低软件开发的复杂性，提高软件重用率。

d．可视化建模

模型是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。RUP在开发过程中建立起软件系统的可视化模型，可以帮助人们提高管理软件复杂性的能力。

e．验证软件质量

某些软件不受用户欢迎的一个重要原因，是其质量低下,在软件投入运行后再去查找和修改出现的问题，比在开发的早期阶段就进行这项工作需要花费更多的人力和时间。在Rational统一过程中，软件质量评估不再是事后型的或由单独小组进行的孤立活动，而是内建在贯穿于整个开发过程的、由全体成员参与的所有活动中。

f．控制软件变更

在变更是不可避免的环境中，必须具有管理变更的能力，才能确保每个修改都是可接受的而且能被跟踪的。RUP描述了如何控制、跟踪和监控修改，以确保迭代开发的成功。

②RUP软件开发生命周期

RUP软件开发生命周期是一个二维的生命周期模型，如图1-9所示。图中纵轴代表核心工作流，横轴代表时间。

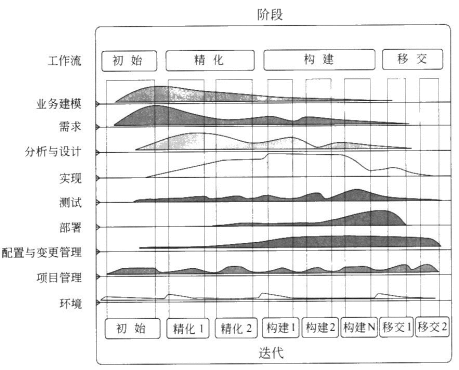


图1-9 RUP软件开发生命周期

③核心工作流

RUP中有九个核心工作流，前三个为核心过程工作流程，后三个为核心支持工作流程。

a．业务建模：了解使用目标系统的机构及其商业运作，评估系统对使用它的机构的影响。

b．需求：捕获客户的需求，并且使开发人员和用户达成对需求描述的共识。

c．分析与设计：把需求分析的结果转化成分析模型与设计模型。

d．实现：把设计模型转换成实现结果，形式化地定义代码结构，用构件实现类和对象，对开发出的构件进行单元测试；把不同实现人员开发出的模块集成为可执行的系统。。

e．测试：检查各个子系统的交互与集成，验证所有需求是否都被正确地实现了，识别、确认缺陷并确保在软件部署之前消除缺陷。

f．部署：成功地生成目标系统的可运行的版本，并把软件移交给最终用户。

g．配置与变更管理：跟踪并维护在软件开发过程中产生的所有制品的完整性和一致性。

h．项目管理：提供项目管理框架，为软件开发项目制定计划、人员配备、执行和监控等方面的实用准则，并为风险管理提供框架。

i．环境：向软件开发机构提供软件开发环境，包括过程管理和工具支持。

④工作阶段

RUP把软件生命周期划分成4个连续的阶段。每个阶段都有明确的目标，并且定义了用来评估是否达到这些目标的里程碑。每个阶段的目标通过一次或多次迭代来完成。在每个阶段结束之前都有一个里程碑评估该阶段的工作成果。如果未能通过评估，则决策者应该作出决定，要么中止该项目，要么重做该阶段的工作。这4个阶段的工作目标如下。

a．初始阶段：建立业务模型，定义最终产品视图，并且确定项目的范围。

b．精化阶段：设计并确定系统的体系结构，制定项目计划，确定资源需求。

c．构建阶段：开发出所有构件和应用程序，把它们集成为客户需要的产品，并且详尽地测试所有功能。

d．移交阶段：把开发出的产品提交给用户使用。

⑤RUP迭代式开发

a．定义

RUP强调采用迭代和渐增的方式来开发软件，整个项目开发过程由多个迭代过程组成。在每次迭代中只考虑系统的一部分需求，针对这部分需求进行分析、设计、实现、测试和部署等工作，每次迭代都是在系统已完成部分的基础上进行的，每次给系统增加些N的功能，如此循环往复地进行下去，直至完成最终项目。

b．特点

第一，每个阶段又进一步细分为一次或多次迭代过程。项目经理根据当前迭代所处的阶段以及上一次迭代的结果，对核心工作流程中的活动进行适当的裁剪，以完成一次具体的迭代过程。在每个生命周期中都一次次地轮流访问这些核心工作流程。

第二，在不同的迭代过程中是以不同的工作重点和强度对这些核心工作流程进行访问的。

⑥RUP适用性

目前，全球已经有上千家软件公司在使用Rational统一过程。这些公司分布在不同的应用领域，开发着或大或小的项目，这表明了RUP的多功能性和广泛适用性。

（7）敏捷过程

敏捷过程能够较好地适应商业竞争环境下对小型项目提出的有限资源和有限开发时间的约束。敏捷软件开发宣言如下：

①个体和交互胜过过程和工具

优秀的团队成员是软件开发项目获得成功的最重要因素；当然，不好的过程和工具也会使最优秀的团队成员无法发挥作用。团队成员的合作、沟通以及交互能力要比单纯的软件编程能力更重要。

②可以工作的软件胜过面面俱到的文档

软件开发的主要目标是向用户提供可以工作的软件而不是文档；但是，完全没有文档的软件也是一种灾难。开发人员应该把主要精力放在创建可工作的软件上面，仅当迫切需要并且具有重大意义时，才进行文档编制工作，而且所编制的内部文档应该尽量简明扼要、主题突出。

③客户合作胜过合同谈判

客户通常不可能做到一次性地把他们的需求完整准确地表述在合同中。能够满足客户不断变化的需求的切实可行的途径是，开发团队与客户密切协作，因此，能指导开发团队与客户协同工作的合同才是最好的合同。

④响应变化胜过遵循计划

软件开发过程中总会有变化，这是客观存在的现实。一个软件过程必须反映现实，因此，软件过程应该有足够的能力及时响应变化。

根据敏捷开发宣言提出的软件过程统称为敏捷过程，其中最重要的是极限编程。

（8）极限编程

①概念

极限编程（XP）是敏捷过程中最富盛名的一个，其名称中“极限”，二字的含义是指把好的开发实践运用到极致。目前，极限编程已经成为一种典型的开发方法，广泛应用于需求模糊且经常改变的场合。

②极限编程的有效实践

a．客户作为开发团队的成员

必须至少有一名客户代表在项目的整个开发周期中与开发人员在一起紧密地配合工作，客户代表负责确定需求、回答开发人员的问题并且设计功能验收测试方案。

b．使用用户素材

所谓用户素材就是正在进行的关于需求的谈话内容的助记符。根据用户素材可以合理地安排实现该项需求的时间。

c．短交付周期

每两周完成一次的迭代过程实现了用户的一些需求，交付出目标系统的一个可工作的版本。通过向有关的用户演示迭代生成的系统，获得他们的反馈意见。

d．验收测试

通过执行由客户指定的验收测试来捕获用户素材的细节。

e．结对编程

结对编程就是由两名开发人员在同一台计算机上共同编写解决同一个问题的程序代码，通常一个人编码，另一个人对代码进行审查与测试，以保证代码的正确性与可读性。结对编程是加强开发人员相互沟通与评审的一种方式。

f．测试驱动开发

极限编程强调“测试先行”。在编码之前应该首先设计好测试方案，然后再编程，直至所有测试都获得通过之后才可以结束工作。

g．集体所有

极限编程强调程序代码属于整个开发小组集体所有，小组每个成员都有更改代码的权利，每个成员都对全部代码的质量负责。

h．持续集成

极限编程主张在一天之内多次集成系统，而且随着需求的变更，应该不断地进行回归测试。

i．可持续的开发速度

开发人员以能够长期维持的速度努力工作。XP规定开发人员每周工作时间不超过40小时，连续加班不可以超过两周，以免降低生产率。

j．开放的工作空间

XP项目的全体参与者（开发人员、客户等）一起在一个开放的场所中工作，项目组成员在这个场所中自由地交流和讨论。

k．及时调整计划

计划应该是灵活的、循序渐进的。制定出项目计划之后，必须根据项目进展情况及时进行调整，没有一成不变的计划。

l．简单的设计

开发人员应该使设计与计划要在本次迭代过程中完成的用户素材完全匹配，设计时不需要考虑未来的用户素材。在一次次的迭代过程中，项目组成员不断变更系统设计，使之相对于正在实现的用户素材而言始终处于最优状态。

m．重构

所谓代码重构就是在不改变系统行为的前提下，重新调整和优化系统的内部结构，以降低复杂性、消除冗余、增加灵活性和提高性能。

注意：在开发过程中不要过分依赖重构，特别是不能轻视设计，对于大中型系统而言，如果推迟设计或者干脆不做设计，将造成一场灾难。

n．使用隐喻

可以将隐喻看作把整个系统联系在一起的全局视图，它描述系统如何运作，以及用何种方式把新功能加入到系统中。

③极限编程的整体开发过程

图1-10描述了极限编程的整体开发过程。具体开发过程如下：

a．项目组针对客户代表提出的“用户故事”进行讨论，提出隐喻，在此项活动中可能需要对体系结构提出相关技术难点的试探性解决方案；

b．项目组在隐喻和用户故事的基础上，根据客户设定的优先级制订交付计划；

c．开始多个迭代过程，在迭代期内产生的新用户故事不在本次迭代内解决，以保证本次开发过程不受干扰。开发出的新版本软件通过验收测试之后交付用户使用。

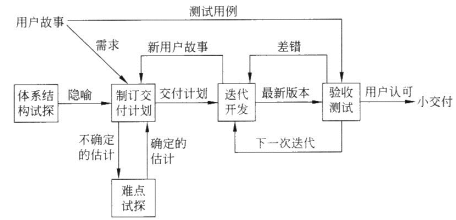


图1-10 XP项目的整体开发过程

④极限编程的迭代过程

a．项目组根据交付计划和项目速率，选择需要优先完成的用户故事或待消除的差错，将其分解成可在1～2天内完成的任务，制订出本次迭代计划；

b．通过每天举行一次的“站立会议”，解决遇到的问题，调整迭代计划；

c．进行代码共享式的开发工作。所开发出的新功能必须100%通过单元测试，并且立即进行集成，得到的新的可运行版本由客户代表进行验收测试。

d．开发人员与客户代表交流此次代码共享式编程的情况，讨论所发现的问题，提出新的用户故事，算出新的项目速率，并把相关的信息提交给站立会议。

图1-11描述了极限编程的迭代开发过程。

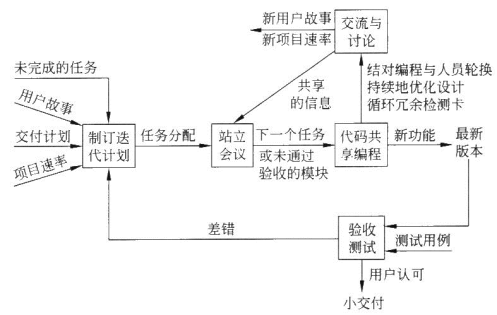


图1-11 XP迭代开发过程

⑤特点

a．极限编程为代表的敏捷过程，具有对变化和不确定性的更快速、更敏捷的反应特性；

b．在快速的同时仍然能够保持可持续的开发速度。

（9）微软过程

①微软过程遵循的基本准则

a．项目计划应该兼顾未来的不确定因素。

b．用有效的风险管理来减少不确定因素的影响。

c．经常生成并快速地测试软件的过渡版本，从而提高产品的稳定性和可预测性。

d．采用快速循环、递进的开发过程。

e．用创造性的工作来平衡产品特性和产品成本。

f．项目进度表应该具有较高稳定性和权威性。

g．使用小型项目组并发地完成开发工作。

h．在项目早期把软件配置项基线化，项目后期则冻结产品。

i．使用原型验证概念，对项目进行早期论证。

j．把零缺陷作为追求的目标。

k．里程碑评审会的目的是改进工作，切忌相互指责。

②微软软件生命周期

微软过程把软件生命周期划分成五个阶段，图1-12描绘了生命周期的阶段及每个阶段的主要里程碑。

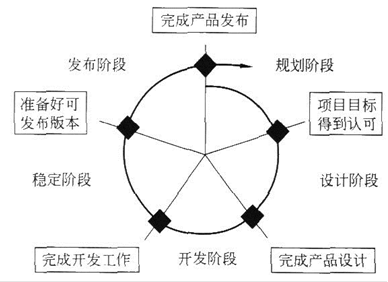


图1-12 微软软件生命周期阶段划分和主要里程碑

a．规划阶段

这个阶段的主要任务是，根据从市场上获得的用户情况和客户需求等信息，在调查、统计和分析的基础上，完成下述工作：

第一，确定产品目标。

第二，获取竞争对手的信息。

第三，完成对客户和市场的调研分析。

第四，确定新版本产品应该具备的主要特性。

第六，确定相对于前一版本而言，新版本应该解决的问题和需要增加的功能。

b．设计阶段

当项目团队已经确定了70%以上的产品需求时，开发工作就可以进入设计阶段了，主要工作内容如下：

第一，根据产品目标编写系统的特性规格说明书。

第二，从系统高层开始着手进行系统设计。

第三，划分出系统中的子系统，给出各个子系统和各个构件的规格说明。

第四，根据产品特性规格说明书制订产品开发计划。

c．开发阶段

开发阶段的主要任务是，完成产品中所有构件的开发工作。一些开发工作可能会持续到稳定阶段，以便在那时对测试中发现的问题作出修改。

d．稳定阶段

稳定阶段的主要任务是对产品进行测试和调试，以确保已经正确地实现了整个解决方案，产品可以发布了。这个阶段测试的重点是，产品在真实环境下的使用和操作。

e．发布阶段

发布阶段的主要任务是发布产品或解决方案，稳定发布过程，并把项目移交到运营和支持人员手中，以获得最终用户对项目的认可。

③微软过程模型

a．定义

图1-13描绘了微软过程的生命周期模型。微软过程的每一个生命周期发布一个递进的软件版本，各个生命周期持续、快速地迭代循环。

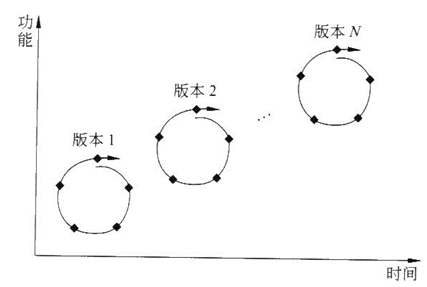


图1-13 微软过程的生命周期模型

b．特点

第一，微软过程综合了Rational统一过程和敏捷过程的许多优点，是对众多成功项目的开发经验的正确总结；

第二，微软过程也有某些不足之处，例如，对方法、工具和产品等方面的论述不如RUP和敏捷过程全面，人们对它的某些准则本身也有不同意见。在开发软件的实践中，应该把微软过程与RUP和敏捷过程结合起来，取长补短．针对不同项目的具体情况进行定制。

**1.2　课后习题详解**

1．什么是软件危机?它有哪些典型表现?为什么会出现软件危机?

答：（1）软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。软件危机包含两方面的问题：

①如何开发软件，以满足对软件日益增长的需求；

②如何维护数量不断膨胀的已有软件。

（2）软件危机的典型表现：

①对软件开发成本和进度的估计不准确；

②用户对“已完成的”软件系统不满意的现象经常发生；

③软件产品的质量往往靠不住；

④软件常常是不可维护的；

⑤软件通常没有适当的文档资料；

⑥软件成本在计算机系统总成本中所占的比例逐年上升；

⑦软件开发跟不上计算机应用迅速普及深入的趋势。

（3）软件危机出现的原因：

①软件是计算机系统中的逻辑部件而不是物理部件；

②软件显著特点是规模庞大；

③对软件开发和维护还有不少糊涂观念，采用了错误的方法和技术；

④存在与软件开发和维护有关的许多错误认识和做法；

⑤对用户要求没有完整准确的认识就匆忙着手编写程序；

⑥一个软件从定义、开发、使用和维护，直到最终被废弃，要经历一个漫长的时期；

⑦一个软件产品必须由一个完整的配置组成，要包括程序、文档和数据等成分；

⑧在软件开发的不同阶段进行修改需要付出的代价是很不相同的；

⑨轻视维护。

2．假设自己是一家软件公司的总工程师，当把图1-14给手下的软件工程师们观看，告诉他们及早发现并改正错误的重要性时，有人不同意这个观点，认为要求在错误进入软件之前就清除它们是不现实的，并举例说：“如果一个故障是编码错误造成的，那么，一个人怎么能在设计阶段清除它呢?”应该怎么反驳他?

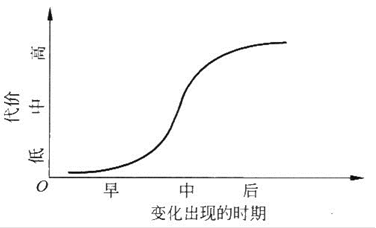


图1-14 引人同一变动付出的代价随时间变化的趋势

答：可以从以下两方面来反驳他：

（1）在软件开发的不同阶段进行修改需要付出的代价是很不相同的，在早期引入变动，涉及的面较少，因而代价也比较低；而在开发的中期，软件配置的许多成分已经完成，引入一个变动要对所有已完成的配置成分都做相应的修改，不仅工作量大，而且逻辑上也更复杂，因此付出的代价剧增。

（2）在软件“已经完成”时再引入变动，当然需要付出更高的代价。在后期引入一个变动比在早期引入相同变动所需付出的代价高2～3个数量级。

3．什么是软件工程?它有哪些本质特性?怎样用软件工程消除软件危机?

答：（1）软件工程定义：

软件工程是是从管理和技术两方面研究如何更好地开发和维护计算机软件的一门新兴学科。采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地开发出高质量的软件并有效地维护它，这就是软件工程。

（2）软件工程本质特征：

①软件工程关注于大型程序的构造；

②软件工程的中心课题是控制复杂性；

③软件经常变化；

④开发软件的效率非常重要；

⑤和谐地合作是开发软件的关键；

⑥软件必须有效地支持它的用户；

⑦由具有一种文化背景的人替具有另一种文化背景的人创造产品。

（3）消除软件危机的方法：

①应该对计算机软件有一个正确的认识；

②应该充分认识到软件开发是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目；

③应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索更好更有效的技术和方法；

④应该开发和使用更好的软件工具。

4．简述结构化范型和面向对象范型的要点，并分析它们的优缺点。

答：（1）结构化范型

①要点

结构化范型属于传统方法学。传统的软件开发方法大部分采用瀑布模型。这种模型要求每一阶段都以前一阶段形成的文档为基础完成工作。每一阶段将要完成时，都要求开发人员进行验证或确认。

②优点

把软件生命周期划分成基干个阶段，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。

③缺点

当软件规模庞大时，或者对软件的需求是模糊的或会承受时间而变化时，开发出的软件往往不成功；而且维护起来仍然很困难。

（2）面向对象范型

①要点

a．软件中的任何元素都是对象，复杂的软件对象由简单的软件对象组合而成。

b．所有对象划分成各种对象类，每个对象都定义了一组数据和一组方法。

c．按照子类和父类的关系，把若干个对象类组成一个层次结构的系统。

d．对象彼此之间仅能通过传递消息互相联系。

②优点

a．按照人类习惯的思维方法，对软件开发过程所有阶段进行综合考虑。

b．软件生存期各阶段所使用的方法、技术具有高度的连续性。

c．软件开发各个阶段有机集成，有利于系统的稳定性。

5．根据历史数据可以进行如下的假设。对计算机存储容量的需求大致按下面公式描述的趋势逐年增加：

IMG_270

存储器的价格按下面公式描述的趋势逐年下降：

IMG_271

如果计算机字长为16位，则存储器价格下降的趋势为：

IMG_272在上列公式中y代表年份，M是存储容量（字数），P1和P2代表价格。基于上述假设可以比较计算机硬件和软件成本的变化趋势。要求计算：

（1）在1985年对计算机存储容量的需求估计是多少?如果字长为16位，这个存储器的价格是多少?

（2）假设在1985年一名程序员每天可开发出10条指令，程序员的平均工资是每月4000美元。如果一条指令为一个字长，计算使存储器装满程序所需用的成本。

（3）假设在1995年存储器字长为32位，一名程序员每天可开发出30条指令，程序员的月平均工资为6000美元，重复（1）、（2）题。

答：（1）在1985年对计算机存储容量的需求，估计是：

M=4080e0.28(1985-1960)=4080e7=4474263（字）

　 如果字长为16位，则这个存储器的价格是：

P=0.048\*0.72(1985-1974)\*4474263=5789（美元）

（2）如果一条指令的长度为一个字，则使存储器装满程序共需4474263条指令。

如果每月有20个工作日，则每人每月可开发200条指令。需要的工作量是：

4474263/200=22371（人月）

　 程序员的月平均工资是4000美元，开发出4474263条指令的成本是：

22371\*4000=8948400（美元）

（3）在1995年对存储容量的需求估计为：

M=4080e0.28(1995-1960)= 4080e9.8=73577679（字）

如果字长为32位，则这个存储器的价格是：

P=0.003\*32\*0.72(1995-1974)\*73577679=7127（美元）

如果一条指令的长度为一个字，则使存储器装满程序共需73577679条指令。

如果每月有20个工作日，则每人每月可开发600条指令。需要的工作量是：

73577679/600=122629（人月）

　 开发出成本是：

122629\*6000=735776790（美元）

6．什么是软件过程?它与软件工程方法学有何关系?

答：（1）软件过程是为了开发出高质量的软件产品所需完成的一系列任务的框架，它规定了完成各项任务的工作步骤。软件过程定义了运用技术方法的顺序、应该交付的文档资料、为保证软件质量和协调软件变化必须采取的管理措施，以及标志完成了相应开发活动的里程碑。

（2）软件过程是软件工程方法学的三个重要组成部分之一。软件工程方法学的三个要素为：方法、工具和过程。

7．什么是软件生命周期模型?试比较瀑布模型、快速原型模型、增量模型和螺旋模型的优缺点，说明每种模型的适用范围。

答：（1）瀑布模型

①优点

a．可强迫开发人员采用规范的方法；

b．严格地规定了每个阶段必须提交的文档；

c．要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证；

d．对文档的约束，使软件维护变得容易一些，且能降低软件预算。

②缺点

a．开发过程一般不能逆转，否则代价太大；

b．实际的项目开发很难严格按该模型进行；

c．客户往往很难清楚地给出所有的需求；

d．软件的实际情况必须到项目开发的后期客户才能看到。

③适用范围

a．用户的需求非常清楚全面，且在开发过程中没有或很少变化；

b．开发人员对软件的应用领域很熟悉；

c．用户的使用环境非常稳定；

d．开发工作对用户参与的要求很低。

（2）快速原型模型

①优点

a．可以得到比较良好的需求定义，容易适应需求的变化；

b．有利于开发与培训的同步；

c．开发费用低、开发周期短且对用户更友好。

②缺点

a．客户与开发者对原型理解不同；

b．准确的原型设计比较困难；

c．不利于开发人员的创新。

③适用范围

a．对所开发的领域比较熟悉而且有快速的原型开发工具；

b．项目招投标时，可以以原型模型作为软件的开发模型；

c．进行产品移植或升级时，或对已有产品原型进行客户化工作时。

（3）增量模型

①优点

a．能在较短时间内向用户提交可完成部分工作的产品；

b．逐步增加功能，减少了全新的软件可能给客户组织带来的冲击。

②缺点

a．并行开发构件有可能遇到不能集成的风险，软件必须具备开放式的体系结构；

b．增量模型的灵活性很容易退化为边做边改模型，从而是软件过程的控制失去整体性。

③适用范围

a．进行已有产品升级或新版本开发；

b．对完成期限严格要求的产品；

c．对所开发的领域比较熟悉而且已有原型系统。

（4）螺旋模型

①优点

a．对可选方案和约束条件的强调有利于已有软件的重用；

b．减少了过多测试或测试不足所带来的风险；

c．在螺旋模型中维护只是模型的另一个周期，在维护和开发之间并没有本质区别。

②缺点

a．需要丰富的风险评估经验和专门知识，如果未能够及时标识风险，会造成重大损失；

b．过多的迭代次数会增加开发成本，延迟提交时间。

③适用范围

适用于内部开发的大规模软件项目。

8．为什么说喷泉模型较好地体现了面向对象软件开发过程无缝和迭代的特性?

答：喷泉模型，是典型的面向对象的软件过程模型之一。“喷泉”这个词体现了面向对象软件开发过程迭代和无缝的特性。代表不同阶段的圆圈相互重叠，表示两个活动之间存在交迭；而面向对象方法在概念和表示方法上的一致性，保证了在各项开发活动之间的无缝过渡。在一个阶段内的向下箭头代表该阶段内的迭代（或求精）。

9．试讨论Rational统一过程的优缺点。

答：（1）优点

提高了团队生产力，在迭代的开发过程、需求管理、基于组件的体系结构、可视化软件建模、验证软件质量以及控制软件变更等方面、针对所有关键的开发活动为每个开发成员提供了必要的准则、模板和工具指导，并确保全体成员共享相同的知识基础、它具有简洁和清晰的过程结构，为开发过程提供了较大的通用性。

（2）缺点

RUP只是一个开发过程，并没有涵盖软件过程的全部内容，例如它缺少关于软件运行和支持等方面的内容，此外，它没有支持多项目的开发结构，在一定程度上降低了在开发组织内大范围实现重用的可能性。

10.Rational统一过程主要适用于何种项目?

答：Rational统一过程（RUP）强调采用迭代和渐增的方式来开发软件，整个项目开发过程由多个迭代过程组成。在每次迭代中只考虑系统的一部分需求，针对这部分需求进行分析、设计、实现、测试和部署等工作，每次迭代都是在系统已完成部分的基础上进行的，每次给系统增加些N的功能，如此循环往复地进行下去，直至完成最终项目。它适用于大型的需求不断变化的复杂软件系统项目。

11．说明敏捷过程的适用范围。

答：敏捷过程具有对变化和不确定性的更快速、更敏捷的反应特性，而且在快速的同时仍然能够保持可持续的开发速度。较适用于开发可用资源及开发时间都有较苛刻约束的小型项目。

12．说明微软过程的适用范围。

答：微软过程的每一个生命周期发布一个递进的软件版本，各个生命周期持续、快速地迭代循环。它综合了Rational统一过程和敏捷过程的许多优点，是对众多成功项目的开发经验的正确总结；适用于商业环境下具有有限资源和有限开发时间约束的项目。

**第2章　可行性研究**

**2.1　复习笔记**

一、可行性研究介绍

1．可行性研究的目的

并非任何问题都有简单明显的解决办法，事实上，许多问题不可能在预定的系统规模或时间期限之内解决。如果问题没有可行的解，那么花费在这项工程上的任何时间、人力、软硬件资源和经费，都是无谓的浪费。可行性研究的目的，就是用最小的代价在尽可能短的时间内确定问题是否能够解决。

2．可行性研究的任务

（1）对以后的行动方针提出建议。如果问题没有可行的解，分析员应该建议停止这项开发工程，以避免时间、资源、人力和金钱的浪费；如果问题值得解，分析员应该推荐一个较好的解决方案，并且为工程制定一个初步的计划。

（2）必须分析几种主要的候选解法的利弊，从而判断原定的系统目标和规模是否现实，系统完成后所能带来的效益是否大到值得投资开发这个系统的程度。

（3）对每种可能的解决方案都应该仔细研究它的可行性，通常，至少从下述三个方面研究每种解决方案的可行性。

①技术可行性：使用现有的技术能否实现这个系统。

②经济可行性：这个系统的经济效益能否超过它的开发成本。

③操作可行性：这个系统的操作方式在该客户组织内是否行得通。

3．可行性研究过程

可行性研究的一般步骤为：

（1）复查系统规模和目标

分析员访问关键人员，仔细阅读和分析有关的材料，以便对问题定义阶段书写的关于规模和目标的报告书进一步复查确认，改正含糊或不确切的叙述，清晰地描述对目标系统的一切限制和约束。这个步骤的工作，实质上是为了确保分析员正在解决的问题确实是要求他解决的问题。

（2）研究目前正在使用的系统

仔细阅读分析现有系统的文档资料和使用手册，也要实地考察现有的系统。应该注意了解这个系统可以做什么，为什么这样做，还要了解使用这个系统的代价。

（3）导出新系统的高层逻辑模型

优秀的设计过程通常是从现有的物理系统出发，导出现有系统的逻辑模型，再参考现有系统的逻辑模型，设想目标系统的逻辑模型，最后根据目标系统的逻辑模型建造新的物理系统。

（4）进一步定义问题

新系统的逻辑模型实质上表达了分析员对新系统必须做什么的看法。用户是否也有同样的看法呢？分析员应该和用户一起再次复查问题定义、工程规模和目标，这次复查应该把数据流图和数据字典作为讨论的基础。

（5）导出和评价供选择的解法

分析员应该从他建议的系统逻辑模型出发，导出若干个较高层次的（较抽象的）物理解法供比较和选择。导出供选择的解法的最简单的途径，是从技术角度出发考虑解决问题的不同方案。

（6）推荐行动方针

根据可行性研究结果应该决定的一个关键性问题是：是否继续进行这项开发工程？分析员必须清楚地表明他对这个关键性决定的建议。

（7）草拟开发计划

分析员应该为所推荐的方案草拟一份开发计划，除了制定工程进度表之外还应该估计对各类开发人员（例如，系统分析员、程序员）和各种资源（计算机硬件、软件工具等）的需要情况，应该指明什么时候使用以及使用多长时间。此外还应该估计系统生命周期每个阶段的成本。最后应该给出下一个阶段（需求分析）的详细进度表和成本估计。

（8）书写文档提交审查。

分析员应该把上述可行性研究各个步骤的工作结果写成清晰的文档，请用户、客户组织的负责人及评审组审查，以决定是否继续这项工程及是否接受分析员推荐的方案。

二、系统流程图

1．定义

系统流程图是概括地描绘物理系统的传统工具。

2．基本思想

系统流程图的基本思想是用图形符号以黑盒子形式描绘组成系统的每个部件（程序、文档、数据库、人工过程等）。系统流程图表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程，因此尽管系统流程图的某些符号和程序流程图的符号形式相同，但是它却是物理数据流图而不是程序流程图。

3．符号

（1）以概括的方式抽象地描绘实际系统时，使用图2-1中列出的基本符号就足够了。

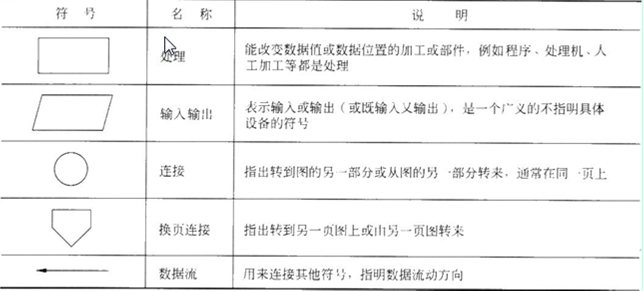


图2-1 基本符号

（2）需要更具体地描绘一个物理系统时，还需要使用图2-2中列出的系统符号。



图2-2 系统符号

4．实例分析

例1：

某装配厂有一座存放零件的仓库，仓库中现有的各种零件的数量以及每种零件的库存量临界值等数据记录在库存清单主文件中。当仓库中零件数量有变化时，应该及时修改库存清单主文件，如果哪种零件的库存量少于它的库存量临界值，则应该报告给采购部、门以便订货，规定每天向采购部门送一次订货报告。该装配厂使用一台小型计算机处理更新库存清单主文件和产生订货报告的任务。零件库存量的每一次变化称为一个事务，由放在仓库中的CRT终端输入到计算机中；系统中的库存清单程序对事务进行处理，更新存储在磁盘上的库存清单主文件，并且把必要的订货信息写在磁带上。最后，每天由报告生成程序读一次磁带，并且打印出订货报告。

解答与分析：

图2-3的系统流程图描绘了上述系统的概貌。图中每个符号用黑盒子形式定义了组成系统的一个部件，然而并没有指明每个部件的具体工作过程；图中的箭头确定了信息通过系统的逻辑路径（信息流动路径）。

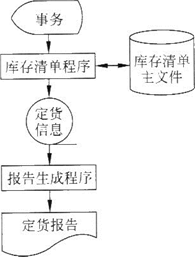


图2-3 库存清单系统的系统流程图

5．分层

面对复杂的系统时，一个较好的方法是分层次地描绘这个系统。步骤如下：

（1）用一张高层次的系统流程图描绘系统总体概貌，表明系统的关键功能；

（2）然后分别把每个关键功能扩展到适当的详细程度，画在单独的一页纸上。

使用分层次的描绘方法便于阅读者按从抽象到具体的过程逐步深入地了解一个复杂的系统。

三、数据流图

1．概念

（1）定义

数据流图（DFD）是一种图形化技术。它描绘信息流和数据从输入移动到输出的过程中所经受的变换。

（2）特点

①数据流图中没有具体的物理部件，只是描绘数据在软件中流动和被处理的逻辑过程。

②数据流图是系统逻辑功能的图形表示，是分析员与用户之间极好的通信工具。

③设计时只需考虑系统必须完成的基本逻辑功能，不考虑怎样具体地实现这些功能。

2．符号

（1）表示

如图2-4（a）所示，数据流图有4种基本符号，如图2-4（a）所示，它们代表的基本含义如下。

a．正方形（或立方体）：表示数据的源点或终点。

b．圆角矩形（或圆形:代表变换数据的处理。

c．开口矩形（或两条平行横线）：代表数据存储。

d．箭头：表示数据流，即特定数据的流动方向。

注意：数据流与程序流程图中用箭头表示的控制流有本质不同，千万不要混淆。

除了4种基本符号之外，有时也使用几种附加符号，图2-4（b）给出了这些附加符号的含义。其中星号（\*）表示数据流之间是“与”关系（同时存在）；加号（＋）表示“或”关系；＋号表示只能从中选一个（互斥的关系）。

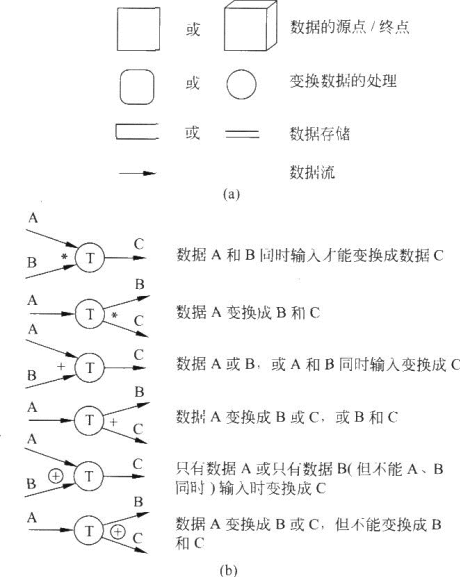


图2-4 数据流图的符号

（2）注意事项

①在数据流图中应该描绘所有可能的数据流向，而不应该描绘出现某个数据流的条件。

②一个处理框可以代表一系列程序、单个程序或者程序的一个模块。

③一个数据存储可以表示一个文件、文件的一部分、数据库的元素或记录的一部分等。

④数据存储是处于静止状态的数据，数据流是处于运动中的数据。

⑤通常在数据流图中忽略出错处理。

⑥表示数据的源点和终点相同的方法是再重复画一个同样的符号表示数据的终点。

⑦代表同一事物的符号出现在n个地方，在这个符号的角上画（n-1）条短斜线做标记。

3．实例分析

例2：

假设一家工厂的采购部每天需要一张订货报表，报表按零件编号排序，表中列出所有需要再次订货的零件。对于每个需要再次订货的零件应该列出下述数据：零件编号，零件名称，订货数量，目前价格，主要供应者，次要供应者。零件入库或出库称为事务，通过放在仓库中的CRT终端把事务报告给订货系统。当某种零件的库存数量少于库存量临界值时就应该再次订货。画出上述订货系统的数据流图。

解答与分析：

（1）第一步可以从问题描述中提取数据流图的4种成分：

①考虑数据的源点和终点

从上面对系统的描述可以知道“采购部每天需要一张订货报表”，“通过放在仓库中的CRT终端把事务报告给订货系统”，所以采购员是数据终点，而仓库管理员是数据源点。

②考虑处理

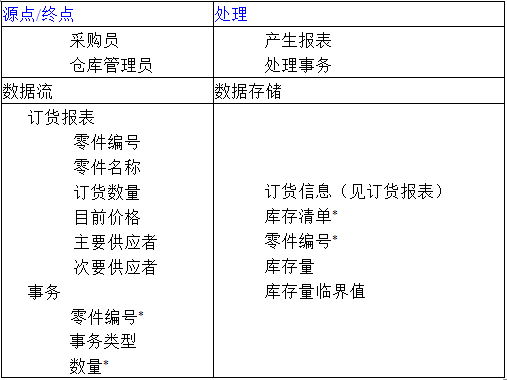
显然他们还没有“采购部需要报表”这种报表，因此必须有一个用于产生报表的处理。事务的后果是改变零件库存量，然而任何改变数据的操作都是处理，因此对事务进行的加工是另一个处理。

③考虑数据流和数据存储

系统把订货报表送给采购部，因此订货报表是一个数据流；事务需要从仓库送到系统中，显然事务是另一个数据流。产生报表和处理事务这两个处理在时间上明显不匹配，每当有一个事务发生时立即处理它，然而每天只产生一次订货报表。因此，用来产生订货报表的数据必须存放一段时间，也就是应该有一个数据存储。

表2-1总结了上面分析的结果，其中加星号标记的是在问题描述中隐含的成分。

表2-1 组成数据流图的元素可以从描述问题的信息中提取



（2）画数据流图

①画出基本系统模型

任何系统的基本模型都由若干个数据源点／终点以及一个处理组成，这个处理就代表了系统对数据加工变换的基本功能。可以画出图2-5这样的基本系统模型。



图2-5 订货系统的基本系统模型

②细化基本系统模型，描绘系统的主要功能

从表2-1可知，“产生报表”和“处理事务”是系统必须完成的两个主要功能，它们将代替图2-5中的“订货系统”（图2-6）。细化后的数据流图中还增加了两个数据存储：处理事务需要“库存清单”数据；产生报表和处理事务需要存储“订货信息”。除了表2-1中列出的两个数据流之外还有另外两个数据流，因为从一个数据存储中取出来的或放进去的数据通常和原来存储的数据相同。在图2-6中给处理和数据存储都加编号，这样做的目的是便于引用和追踪。

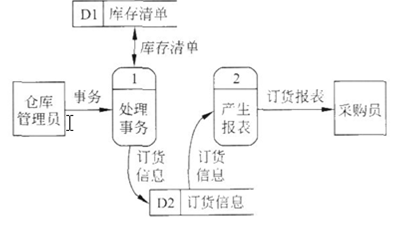


图2-6 订货系统的功能级数据流图

③进一步细化功能级数据流图中描绘的系统主要功能

考虑通过系统的逻辑数据流：当发生一个事务时必须首先接收它；随后按照事务的内容修改库存清单；最后如果更新后的库存量少于库存量临界值时，则应该再次订货，也就是需要处理订货信息。因此，把“处理事务”这个功能分解为：“接收事务”、“更新库存清单”和“处理订货”（图2-7）。

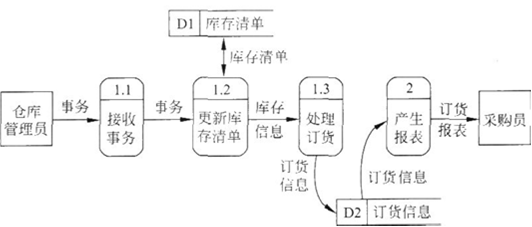


图2-7 把处理事务的功能进一步分解后的数据流图

④考虑是否要继续分解和细化

当进一步分解将涉及如何具体地实现一个功能时就不应该再分解了。订货报表中需要的数据在存储的订货信息中全都有，产生报表只不过是按一定顺序排列这些信息，再按一定格式打印出来。然而这些考虑纯属具体实现的细节，不应该在数据流图中表现。对“接收事务”或“更新库存清单”等功能也没有必要进一步细化。

⑤检查

a．对数据流图分层细化时必须保持信息连续性；

b．注意对处理进行编号的方法。

4．命名规则

（1）为数据流（或数据存储）命名

①名字应代表整个数据流（或数据存储）的内容，而不是只反映它的某些成分。

②不要使用空洞的、缺乏具体含义的名字（如“数据”、“信息”、“输入”之类）。

③如果在为某个数据流（或数据存储）起名字时遇到了困难，则很可能是因为对数据流图分解不恰当造成的，应该试试重新分解，看是否能克服这个困难。

（2）为处理命名

①通常先为数据流命名，然后再为与之相关联的处理命名。

②名字应该反映整个处理的功能，而不是它的一部分功能。

③名字最好由一个具体的及物动词加上一个具体的宾语组成。应该尽量避免使用“加工”、“处理”等空洞笼统的动词作名字。

④通常名字中仅包括一个动词，如果必须用两个动词才能描述整个处理的功能，则把这个处理再分解成两个处理可能更恰当些。

⑤如果在为某个处理命名时遇到困难，则很可能是发现了分解不当的迹象，应考虑重新分解。

（3）为数据源点／终点命名

数据源点／终点并不需要在开发目标系统的过程中设计和实现，它并不属于数据流图的核心内容，只不过是目标系统的外围环境部分，因此为数据源点／终点命名时通常采用它们在问题域中习惯使用的名字。

5．用途

（1）目的

①作为交流信息的工具（基本目的）

分析员把他对现有系统的认识或对目标系统的设想用数据流图描绘出来，供有关人员审查确认。

②作为分析和设计的工具

分析员在研究现有的系统时常用系统流程图表达他对这个系统的认识，这种描绘方法形象具体，比较容易验证它的正确性。

（2）自动化边界

以图中不同处理的定时要求为指南，能够在数据流图上画出许多组自动化边界，每组自动化边界可能意味着一个不同的物理系统，因此可以根据系统的逻辑模型考虑系统的物理实现。例如：

①考虑图2-7，事务随时可能发生，故处理1.1（“接收事务”）必须是联机的；采购员每天需要一次订货报表，因此处理2（“产生报表”）应该以批量方式进行。可以联机地接收事务并放入队列中，然而更新库存清单、处理订货和产生报表以批量方式进行（图2-8）。

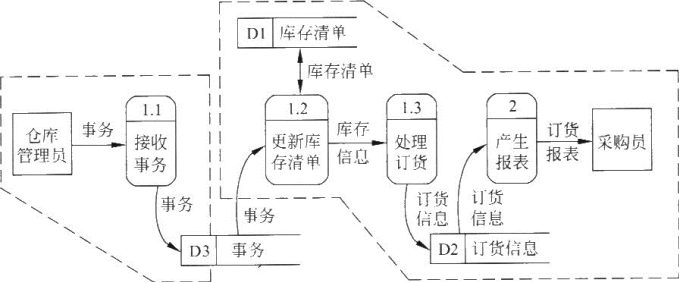


图2-8 这种划分自动化边界的方法暗示以批量方式更新库存清单

②改变自动化边界，把处理1.1，1.2和1.3放在同一边界内（图2-9），这个系统将联机地接收事务、更新库存清单和处理订货及输出订货信息；处理2将以批量方式产生订货报表。

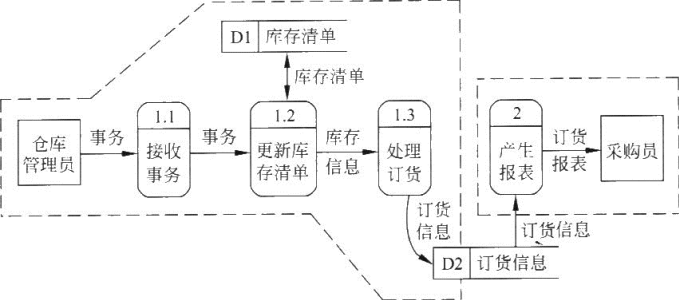


图2-9 另一种划分自动化边界的方法建议以联机方式更新库存清单

四、数据字典

1．概念

（1）定义

数据字典是关于数据的信息的集合，是对数据流图中包含的所有元素的定义的集合。它的作用是在软件分析和设计的过程中给人提供关于数据的描述信息。

（2）意义

数据流图和数据字典共同构成系统的逻辑模型，没有数据字典，数据流图就不严格，然而没有数据流图，数据字典也难于发挥作用。只有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。

2．内容

（1）一般来说，数据字典应该由对下列4类元素的定义组成。

①数据流；

②数据流分量，即数据元素；

③数据存储；

④处理。

（2）数据元素的别名就是该元素的其他等价的名字，出现别名主要是以下3个原因引起的。

①对于同样的数据，不同的用户使用了不同的名字。

②一个分析员在不同时期对同一个数据使用了不同的名字。

③两个分析员分别分析同一个数据流时，使用了不同的名字。

注意：虽然应该尽量减少出现别名，但是不可能完全消除别名。

3．定义数据的方法

定义绝大多数复杂事物的方法，都是用被定义的事物的成分的某种组合表示这个事物，这些组成成分又由更低层的成分的组合来定义。

（1）数据元素组成数据的方式

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（重复零次或一次）。

（2）数据元素组成数据的符号表示

①＝意思是等价于（或定义为）；

②＋意思是和（即连接两个分量）；

③[]意思是或（即从方括弧内列出的若干个分量中选择一个）；

④｜意思是隔开供选择的分量；

⑤{}意思是重复（即重复花括弧内的分量）；

⑥（）意思是可选（即圆括弧里的分量可有可无）。

⑦使用上限和下限进一步注释表示重复的花括弧，例如：51{A}和1{A}5含义相同。

4．数据字典的用途

（1）作为分析阶段的工具（最重要）。

在数据字典中建立的一组严密一致的定义很有助于改进分析员和用户之间的通信，可以消除许多可能的误解。对数据的这一系列严密一致的定义有助于改进在不同的开发人员或不同的开发小组之间的通信。如果要求所有开发人员都根据公共的数据字典描述数据和设计模块，则能避免许多麻烦的接口问题。

（2）数据字典中包含的每个数据元素的控制信息是很有价值的。

数据字典列出了使用一个给定的数据元素的所有程序（或模块），所以很容易估计改变一个数据将产生的影响，并且能对所有受影响的程序或模块做出相应的改变。

（3）数据字典是开发数据库的第一步，而且是很有价值的一步。

5．数据字典的实现

（1）开发大型软件系统是建议使用数据字典处理程序。

（2）在开发小型软件系统时建议采用卡片形式书写数据字典，如图2-10，给出了几个数据元素的数据字典卡片。

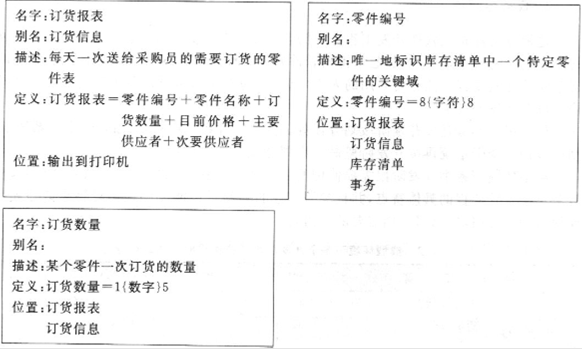


图2-10 数据字典卡片

五、成本／效益分析

1．目的

成本／效益分析的目的是从经济角度分析开发一个特定的新系统是否划算，从而帮助客户组织的负责人正确地做出是否投资于这项开发工程的决定。

2．成本估计的方法

软件开发成本主要表现为人力消耗（乘以平均工资则得到开发费用）。成本估计不是精确的科学，因此应该使用几种不同的估计技术以便相互校验。成本估计可以使用以下3种估算技术。

（1）代码行技术

代码行技术是比较简单的定量估算方法，它把开发每个软件功能的成本和实现这个功能需要用的源代码行数联系起来。根据经验和历史数据估计实现一个功能需要的源程序行数。当有以往开发类似工程的历史数据可供参考时，这个方法是非常有效的。

（2）任务分解技术

①方法步骤

a．把软件开发工程分解为若干个相对独立的任务；

b．分别估计每个单独的开发任务的成本，先估计完成该项任务需要用的人力（以人月为单位），再乘以每人每月的平均工资而得出每个任务的成本；

c．最后累加起来得出软件开发工程的总成本。

②划分规则

最常用的办法是按开发阶段划分任务。如果软件系统很复杂，由若干个子系统组成，则可以把每个子系统再按开发阶段进一步划分成更小的任务。

③典型环境

典型环境下各个开发阶段需要使用的人力的百分比大致如表2-2所示。

表2-2 典型环境下各个开发阶段需要使用的人力的百分比



（3）自动估计成本技术

采用自动估计成本的软件工具可以减轻人的劳动，并且使得估计的结果更客观。但是，

采用这种技术必须有长期搜集的大量历史数据为基础，并且需要有良好的数据库系统支持。

3．成本／效益分析的方法

（1）方法步骤

①估计开发成本、运行费用和新系统将带来的经济效益。

a．运行费用取决于系统的操作费用和维护费用。

b．系统的经济效益等于因使用新系统而增加的收入加上新系统可以节省的运行费用。

c．在进行成本／效益分析时一律假设生命周期为5年。

②比较新系统的开发成本和经济效益，以便从经济角度判断这个系统是否值得投资。但是，投资是现在进行的，效益是将来获得的，不能简单地比较成本和效益，应该考虑货币的时间价值。

（2）影响经济效益的因素

①货币的时间价值

通常用利率的形式表示货币的时间价值。假设年利率为i，如果现在存入P元，则n年后可以得到的钱数为：

F＝P（1＋i）n

这也就是P元钱在n年后的价值。反之，如果n年后能收入F元钱，那么这些钱的现在价值是：

P＝F／（1＋i）n

②投资回收期

通常用投资回收期衡量一项开发工程的价值。所谓投资回收期就是使累计的经济效益等于最初投资所需要的时间。投资回收期越短就能越快获得利润，那么这项工程也就越值得投资。投资回收期仅仅是一项经济指标，为了衡量一项开发工程的价值，还应该考虑其他经济指标。

③纯收入

工程的纯收入是在整个生命周期之内系统的累计经济效益（折合成现在值）与投资之差。如果纯收人为零，则工程的预期效益和在银行存款一样，但是开发一个系统要冒风险，因此这项工程可能是不值得投资的。如果纯收入小于零，那么这项工程显然不值得投资。

④投资回收率

计算投资回收率，用它衡量投资效益的大小，并且可以把它和年利率相比较，在衡量工程的经济效益时，它是最重要的参考数据。

IMG_285

其中，P是现在的投资额；Fi是第i年年底的效益（i＝1，2，…，n）；n是系统的使用寿命；j是投资回收率。

**2.2　课后习题详解**

1．在软件开发的早期阶段为什么要进行可行性研究？应该从哪些方面研究目标系统的可行性？

答：（1）进行可行性研究的目的

可行性研究的目的是，用最小的代价在尽可能短的时间内研究并确定客户提出的问题是否有行得通的解决办法。必须分析几种主要的候选解法的利弊，从而判断原定的系统目标和规模是否现实，系统完成后所能带来的效益是否大到值得投资开发这个系统的程度。

（2）从下述三个方面研究每种解决方案的可行性

①技术可行性：使用现有的技术能否实现这个系统。

②经济可行性：这个系统的经济效益能否超过它的开发成本。

③操作可行性：这个系统的操作方式在该客户组织内是否行得通。

2．为方便储户，某银行拟开发计算机储蓄系统。储户填写的存款单或取款单由业务员输入系统，如果是存款，系统记录存款人姓名、住址、存款类型、存款日期、利率等信息，并印出存款单给储户；如果是取款，系统计算利息并印出利息清单给储户。

写出问题定义并分析此系统的可行性。

答：（1）问题定义

①如果是存款，储户填写存款单，然后交给业务员键入系统，同时系统还要记录存款人姓名、住址（或电话号码）、身份证号码、存款类型、存款日期、利率等信息，完成后由系统打印存款单给储户。

②如果是取款，储户填写取款单，然后交给业务员，业务员把取款金额输入系统并要求储户输入密码以确认身份，核对密码正确无误后系统计算利息并印出利息清单给储户。

③为了满足储户的需求，该系统需要迅速的对用户的要求做出反馈，要对用户输入的信息作出最快的处理，所以就需要很大的主存容量，以及强大的数据库支持。由于是所面向的用户是广泛的储蓄用户群，所以需要系统强大的安全性能支持。

（2）可行性研究方法条件、假定和限制

①建议开发软件运行的最短寿命：5年。

②进行系统方案选择比较的期限：2个月。

③经费来源和使用限制：定制银行。

④硬件、软件、运行环境和开发环境的条件和限制：银行中心拥有大型机以及用来支持的数据库，各个银行网点都有安好的PC机，安装有Windows2000及以上的操作系统。

⑤建议开发软件投入使用的最迟时间：开发完成后试运行1个月。

（3）可行性研究方法

通过与银行熟练业务员进行深入讨论，制定详细用户调查问卷，真正了解用户以及银行业务员的实际需求，根据业务员提供的信息以及问题定义再综合调查问卷中用户提出的意见进行改进。最终确定项目需要解决的问题，并确定问题能不能被解决。决定可行性的主要因素：项目开发成本、所需设备置办成本、技术是否能满足需求、操作人员的熟练程度、资源有效性。

（4）处理流程和数据流程

①系统流程图

a．存款（如图2-11）

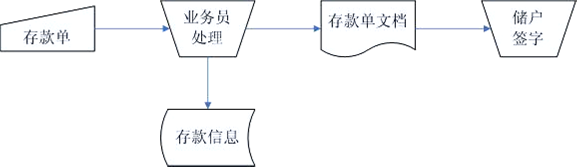


图2-11 存款系统流程图

b．取款（如图2-12）

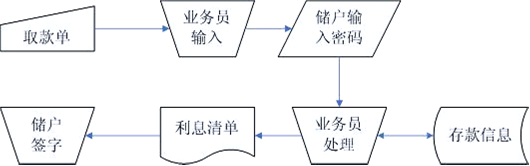


图2-12 取款系统流程图

②数据流图（如图2-13）

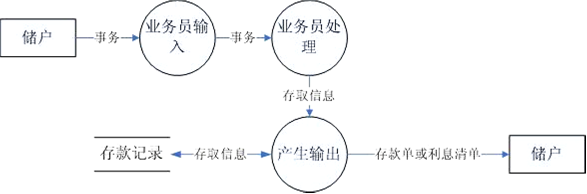


图2-13 银行业务数据流图

（5）分析影响因素

①工作负荷

当前大多数银行所使用的银行储蓄系统在办理业务时手续繁琐，人工业务操作过多，办理一个客户的业务就需耗费较长的时间，其他客户只有等待。这样太浪费时间，需要大量的业务员处理客户业务。也给银行工作人员增加了非常大的负担和额外的工作负荷；同时业务员增多，银行费用支出多，给银行公司的发展带来严重的制约和压力。

②费用支出

包括银行业务员及其他工作人员的工资，系统维护所需资金。

③人员

需要大量的业务员、客服人员，系统维护人员以及其他工作人员。

④设备

包括打印机、PC机、笔记本电脑。

⑤局限性

工作效率底下，不能符合大量群众的及时需求，给群众的生活带来不便。这是现有系统办理业务的流程中出现的一个严重问题。具体问题如下：

a．当前银行使用的银行储蓄系统在办理业务时仅仅靠手工操作，人工业务占据了全部，要求银行业务员办理业务时细心耐心，对数字敏感，算术水平高。严重影响了工作效率，且出错率高，储户办理事务时需要等待时间过长，需要大量的银行业务员才能满足日益增长的存取款要求。

b．用纸张记录保存用户存款记录查找繁琐，耗时久，不方便，且容易丢失。且人工记录不能保证万无一失，容易出现数据录入错误。

c．存款记录保密性差，任何一个业务员都可以随意更改查阅用户数据，用户的资料容易泄漏，带来安全隐患。

d．业务员服务的时间有限，不能24小时办理业务，没有应急处理。

e．所有客户的业务都由业务员手工办理，浪费人力资源。简单的业务经改进可由机器完成，完全不用人工参与，这样节约了人力成本，也提高效率。

f．对现有系统的改进维护只能依靠增多业务员数量，增多银行分行数目，提高业务员业务素质来实现。增多业务员数量会导致银行人员支出大大增多，同时需要多选址建设分行，或者扩大分行规模，费用代价巨大。业务员的业务素质需要培养，不能短期见成效，这又浪费资金和人力资源。对现用系统的改进型维护已经不能解决储户越来越多、储户时间越来越长的问题。

3．为方便旅客，某航空公司拟开发一个机票预订系统。旅行社把预订机票的旅客信息（姓名、性别、工作单位、身份证号码、旅行时间、旅行目的地等）输入进该系统，系统为旅客安排航班，印出取票通知和账单，旅客在飞机起飞的前一天凭取票通知和账单交款取票，系统校对无误即印出机票给旅客。

写出问题定义并分析此系统的可行性。

答：（1）问题定义

①目标：在一个月内建立一个高效率，无差错的航空公司机票预定系统。

②存在的主要问题：人工不易管理，手续繁琐。

（2）可行性分析

①经济可行性

a．成本估算

打印机一台（2000元）+开发费（3500元）=5500元

b．效益估算

该系统有很好的社会效益，提高了公司售票效率，方便旅客，售票方便、科学。

②技术可行性

经过调查分析，得到目前航空公司机票预定系统流程图如图2-14和2-15。

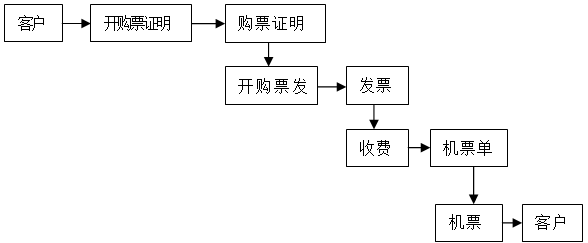


图2-14 订票系统流程图

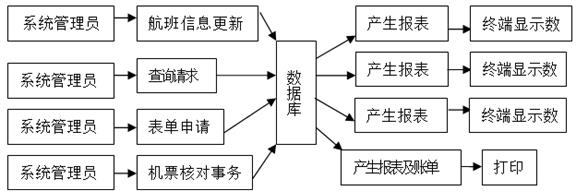


图2-15 订票系统流程图

③操作的可行性

比较以上两图看出，与人工系统相比，计算机保留了原有的主要工作流程，可以看出计算机系统是人工系统的优化，操作也不复杂，工作人员在短时间经过培训就可熟练掌握。

④结论

由于经济、技术、操作三方面的可行性分析都通过，因此此技术系开发航空公司机票预定系统是可行的。

4．目前住院病人主要由护士护理，这样做不仅需要大量护士，而且由于不能随时观察危重病人的病情变化，还可能会延误抢救时机。某医院打算开发一个以计算机为中心的患者监护系统，试写出问题定义，并且分析开发这个系统的可行性。

医院对患者监护系统的基本要求是随时接收每个病人的生理信号（脉搏、体温、血压、心电图等），定时记录病人情况以形成患者日志，当某个病人的生理信号超出医生规定的安全范围时向值班护士发出警告信息，此外，护士在需要时还可以要求系统印出某个指定病人的病情报告。

答：（1）问题定义

①本系统的数据源点是“病人”和“护士”，他们分别提供生理信号和要求病情报告的信息。从系统应该“定时记录病人情况以形成患者日志”这项要求可以想到，还应该有一个提供日期和时间信息的“时钟”作为数据源点。

②本系统的数据终点是接收警告信息和病情报告的护士。系统对病人生理信号的处理功能主要是“接收信号”、“分析信号”和“产生警告信息”。此外，系统还应该具有“定时取样生理信号”、“更新日志”和“产生病情报告”的功能。

（2）分析可行性

为了分析病人生理信号是否超出了医生规定的安全范围，应该存储“患者安全范围”信息。此外，定时记录病人生理信号所形成的“患者日志”显然也是一个数据存储。本系统的基本系统模型如图2-16所示，图2-17是本系统的功能级数据流图。

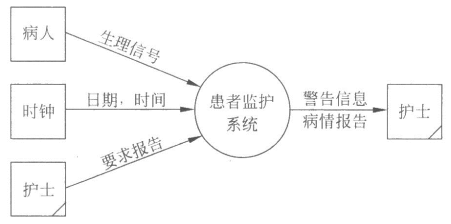


图2-16 患者监护系统的基本系统模型

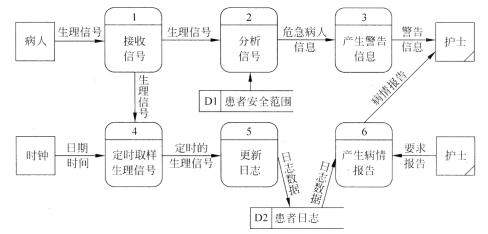


图2-17 患者监护系统的功能级数据流图

5．北京某高校可用的电话号码有以下几类：校内电话号码由4位数字组成，第1位数字不是0；校外电话又分为本市电话和外地电话两类，拨校外电话需先拨0，若是本市电话则再接着拨8位数字（第1位不是0），若是外地电话则拨3位区码再拨8位电话号码（第1位不是0）。

用定义数据的方法，定义上述的电话号码。

答：（1）电话号码=[校内电话号码|校外电话号码]

（2）校内电话号码=非零数字+3位数字

（3）校外电话号码=[本市号码|外地号码]

（4）本市号码=数字零+8位数字

（5）外地号码=数字零+3位数字+8位数字

（6）非零数字=[1|2|3|4|5|6|7|8|9]

（7）数字零=0

（8）3位数字=3{数字}3

（9）8位数字=非零数字+7位数字

（10）7位数字=7{数字}7

（11）数字=[0|1|2|3|4|5|6|7|8|9]

**第3章　需求分析**

**3.1　复习笔记**

一、需求分析的相关概念

1．定义

需求分析是软件定义时期的最后一个阶段，它的基本任务是准确地回答“系统必须做什么”这个问题，即对目标系统提出完整、准确、清晰、具体的要求。在需求分析阶段结束之前，系统分析员应该写出软件需求规格说明书，以书面形式准确地描述软件需求。

2．必要性

为了开发出真正满足用户需求的软件产品，首先必须知道用户的需求。对软件需求的深入理解是软件开发工作获得成功的前提条件，不论人们把设计和编码工作做得如何出色，不能真正满足用户需求的程序只会令用户失望，给开发者带来烦恼。

3．准则

需求分析和规格说明是一项十分艰巨复杂的工作。用户与分析员之间需要沟通的内容非常多，在双方交流信息的过程中很容易出现误解或遗漏，也可能存在二义性。因此，不仅在整个需求分析过程中应该采用行之有效的通信技术，集中精力细致工作，而且必须严格审查验证需求分析的结果。进行需求分析必须遵循以下4条准则。

（1）必须理解并描述问题的信息域，根据这条准则应该建立数据模型。

（2）必须定义软件应完成的功能，这条准则要求建立功能模型。

（3）必须描述作为外部事件结果的软件行为，这条准则要求建立行为模型。

（4）必须对描述信息、功能和行为的模型进行分解，用层次的方式展示细节。

二、需求分析的任务

1．确定对系统的综合要求

虽然功能需求是对软件系统的一项基本需求，但却并不是唯一的需求。通常对软件系统有下述几方面的综合要求。

（1）功能需求

功能需求指定系统必须提供的服务。通过需求分析应该划分出系统必须完成的所有功能。

（2）性能需求

性能需求指定系统必须满足的定时约束或容量约束，通常包括速度（响应时间）、信息量速率、主存容量、磁盘容量、安全性等方面的需求。

（3）可靠性和可用性需求

可靠性需求定量地指定系统的可靠性；可用性与可靠性密切相关，它量化了用户可以使用系统的程度。

（4）出错处理需求

出错处理需求说明系统对环境错误应该怎样响应。在某些情况下，“出错处理”指的是当应用系统发现它自己犯下一个错误时所采取的行动。但是，应该有选择地提出这类出错处理需求。人们的目的是开发出正确的系统，而不是用无休止的出错处理代码掩盖自己的错误。总之，对应用系统本身错误的检测应该仅限于系统的关键部分，而且应该尽可能少。

注意：上述错误不是由该应用系统本身造成的。

（5）接口需求

接口需求描述应用系统与它的环境通信的格式。常见的接口需求有：用户接口需求；硬件接口需求；软件接口需求；通信接口需求。

（6）约束

设计约束或实现约束描述在设计或实现应用系统时应遵守的限制条件。常见的约束有：精度；工具和语言约束；设计约束；应该使用的标准；应该使用的硬件平台。

（7）逆向需求

逆向需求说明软件系统不应该做什么。理论上有无限多个逆向需求，应选取能澄清真实需求且可消除发生误解的逆向需求。

（8）将来可能提出的要求

应该明确地列出那些虽然不属于当前系统开发范畴，但是将来很可能会提出来的要求。这样做的目的是，在设计过程中对系统将来可能的扩充和修改预做准备，以便一旦确实需要时能比较容易地进行这种扩充和修改。

2．分析系统的数据要求

（1）意义

任何一个软件系统本质上都是信息处理系统，系统必须处理的信息和系统应该产生的信息在很大程度上决定了系统的面貌。因此，必须分析系统的数据要求，这是软件需求分析的一个重要任务。

（2）工具

分析系统的数据要求通常采用建立数据模型的方法。复杂的数据由许多基本的数据元素组成，数据结构表示数据元素之间的逻辑关系。利用数据字典可以全面准确地定义数据，但是数据字典的缺点是不够形象直观。为了提高可理解性，常常利用图形工具辅助描绘数据结构。常用的图形工具有层次方框图和Warnier图。

（3）规范

软件系统经常使用各种长期保存的信息，为减少数据冗余，避免出现插入异常或删除异常，简化修改数据的过程，通常需要把数据结构规范化。

3．导出系统的逻辑模型

综合分析结果可以导出系统的详细的逻辑模型，通常用数据流图、实体联系图、状态转换图、数据字典和主要的处理算法描述这个逻辑模型。

4．修正系统开发计划

根据在分析过程中获得的对系统的更深入更具体的了解，可以比较准确地估计系统的成本和进度，修正以前制定的开发计划。

三、与用户沟通获取需求的方法

1．访谈

访谈是最早开始使用的获取用户需求的技术，也是迄今为止仍然广泛使用的需求分析技术。

（1）基本形式

①正式访谈

系统分析员将提出一些事先准备好的具体问题。

②非正式访谈

分析员将提出一些用户可以自由回答的开放性问题，鼓励被访问人员说出自己的想法。

（2）技术方法

①调查表技术

当需要调查大量人员的意见时，向被调查人分发调查表是一个十分有效的做法。经过仔细考虑写出的书面回答可能比被访者对问题的口头回答更准确。分析员仔细阅读收回的调查表，然后再有针对性地访问一些用户，以便向他们询问在分析调查表时发现的新问题。

②情景分析技术

a．概念

情景分析是对用户将来使用目标系统解决某个具体问题的方法和结果进行分析。系统分析员利用情景分析技术，往往能够获知用户的具体需求。

b．作用

第一，能在某种程度上演示目标系统的行为，从而便于用户理解，而且还可能进一步揭示出一些分析员目前还不知道的需求。

第二，使用这种技术能保证用户在需求分析过程中始终扮演一个积极主动的角色。需求分析的目标是获知用户的真实需求，而这一信息的唯一来源是用户，因此，让用户起积极主动的作用对需求分析工作获得成功是至关重要的。

2．面向数据流自顶向下求精

（1）定义

结构化分析方法是面向数据流自顶向下逐步求精进行需求分析的方法。通过可行性研究已经得出了目标系统的高层数据流图，需求分析的目标之一就是把数据流和数据存储定义到元素级。

（2）原理

①通常从数据流图的输出端着手分析。输出数据是由哪些元素组成的呢？通过调查访问不难搞清这个问题。每个输出数据元素又是从哪里来的呢？沿数据流图从输出端往输入端回溯，应该能够确定每个数据元素的来源，与此同时也就初步定义了有关的算法。

②为了得到某个数据元素需要用到数据流图中目前还没有的数据元素，或者得出这个数据元素需要用的算法，往往需要向用户和其他有关人员请教，他们的回答使分析员对目标系统的认识更深入更具体了，系统中更多的数据元素被划分出来了，更多的算法被搞清楚了。

③通常把分析过程中得到的有关数据元素的信息记录在数据字典中，把对算法的简明描述记录在IPO图中。通过分析而补充的数据流、数据存储和处理，应该添加到数据流图的适当位置上。

④请用户对分析过程中得出的结果仔细地复查，数据流图是帮助复查的极好工具。复查过程验证了已知的元素，补充了未知的元素，填补了文档中的空白。

⑤对经过细化后得到的新数据流图的分析追踪可能产生新的问题，这些问题的答案可能又在数据字典中增加一些新条目，并且可能导致新的或精化的算法描述。随着分析过程的进展，经过提问和解答的反复循环，最终得到对系统数据和功能要求的满意了解。图3-1概括了上述分析过程。

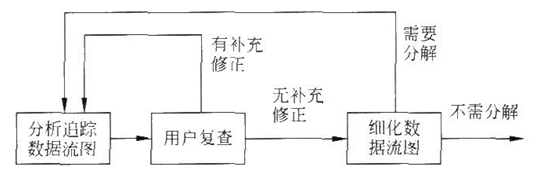


图3-1 面向数据流自顶向下求精过程

3．简易的应用规格说明技术

（1）定义

简易的应用规格说明技术是一种面向团队的需求收集法。这种方法提倡用户与开发者密切合作，共同标识问题，提出解决方案要素，商讨不同方案并指定基本需求。是信息系统领域使用的主流技术。

（2）应用过程

①进行初步的访谈并确定会议方案。

通过用户对基本问题的回答，初步确定待解决的问题的范围和解决方案。开发者和用户分别写出“产品需求”。选定会议的时间和地点，并选举一个负责主持会议的协调人。邀请开发者和用户双方组织的代表出席会议，并预先把写好的产品需求分发给每位与会者。

②进行会议准备

要求每位与会者提前认真审查产品需求，并且列出作为系统环境组成部分的对象、系统将产生的对象以及系统为了完成自己的功能将使用的对象。还要求每位与会者列出操作这些对象或与这些对象交互的服务。最后还应该列出约束条件和性能标准。

③开会讨论

a．讨论的第一个问题是，是否需要这个新产品，一旦确实需要这个新产品，每位与会者就应该把他们在会前准备好的列表展示出来供大家讨论。这个阶段严格禁止批评与争论。

b．共同创建一张组合列表。在组合列表中消去了冗余项，加入了在展示过程中产生的新想法，但并不删除任何实质性内容。由协调人主持讨论这些列表。组合列表将被缩短，加长或重新措辞。讨论的目标是，针对每个议题都创建出一张意见一致的列表。

c．一旦得出了意见一致的列表，就把与会者分成更小的小组，每个小组的工作目标是为每张列表中的项目制定小型规格说明。

d．每个小组都向全体与会者展示他们制定的小型规格说明。通过讨论可能会增加或删除一些内容，也可能进一步做些精化工作。

④会后总结并起草需求规格说明书

在完成了小型规格说明之后，每个与会者都制定出产品的一整套确认标准，并把自己制定的标准提交会议讨论，以创建出意见一致的确认标准。最后，由一名或多名与会者根据会议成果起草完整的软件需求规格说明书。

（3）优点

①开发者与用户不分彼此，齐心协力，密切合作；

②即时讨论并求精；有能导出规格说明的具体步骤。

4．快速建立软件原型

（1）定义

快速原型是快速建立起来的旨在演示目标系统主要功能的可运行的程序。构建原型的要点是，它应该实现用户看得见的功能，省略目标系统的“隐含”功能。快速建立软件原型是最准确、最有效、最强大的的需求分析技术。

（2）特性

①快速

快速原型的目的是尽快向用户提供一个可在计算机上运行的目标系统的模型，以便使用户和开发者在目标系统应该“做什么”这个问题上尽可能快地达成共识。因此，原型的某些缺陷是可以忽略的，只要这些缺陷不严重地损害原型的功能，不会使用户对产品的行为产生误解，就不必管它们。

②容易修改

如果原型的第一版不是用户所需要的，就必须根据用户的意见迅速地修改它，构建出原型的第二版，以更好地满足用户需求。在实际开发软件产品时，原型的“修改试用反馈”过程可能重复多遍，如果修改耗时过多，势必延误软件开发时间。

（3）使用的方法和工具。

①第四代技术

第四代技术包括众多数据库查询和报表语言、程序和应用系统生成器以及其他非常高级的非过程语言。第四代技术使得软件工程师能够快速地生成可执行的代码，因此，它们是较理想的快速原型工具。

②可重用的软件构件

使用一组已有的软件构件（组件）来装配原型。软件构件可以是数据结构（或数据库），或软件体系结构构件（程序），或过程构件（模块）。必须把软件构件设计成能在不知其内部工作细节的条件下重用。

注意：现有的软件可以被用作“新的或改进的”产品的原型。

③形式化规格说明和原型环境

在过去的20多年中，人们已经研究出许多形式化规格说明语言和工具，用于替代自然语言规格说明技术。交互式环境可以调用自动工具把基于形式语言的规格说明翻译成可执行的程序代码，用户能够使用可执行的原型代码去进一步精化形式化的规格说明。

四、分析建模与规格说明

1．分析建模

（1）模型

模型是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。模型由一组图形符号和组织这些符号的规则组成。

（2）建模过程

结构化分析实质上是一种创建模型的活动。为了开发出复杂的软件系统，系统分析员应该从不同角度抽象出目标系统的特性，使用精确的表示方法构造系统的模型，验证模型是否满足用户对目标系统的需求，并在设计过程中逐渐把和实现有关的细节加进模型中，直至最终用程序实现模型。

2．软件需求规格说明

软件需求规格说明书是需求分析阶段得出的最主要的文档。通常用自然语言完整、准确、具体地描述系统的数据要求、功能需求、性能需求、可靠性和可用性要求、出错处理需求、接口需求、约束、逆向需求以及将来可能提出的要求。自然语言的规格说明具有容易书写、容易理解的优点。

五、实体-联系图

1．数据模型的定义

为了把用户的数据要求清楚、准确地描述出来，通常建立一个概念性的数据模型（信息模型）。概念性数据模型是一种面向问题的数据模型，是按照用户的观点对数据建立的模型。它描述了从用户角度看到的数据，它反映了用户的现实环境，而且与在软件系统中的实现方法无关。

2．数据模型的构成

数据模型中包含3种相互关联的信息：数据对象、数据对象的属性及数据对象彼此间相互连接的关系。

（1）数据对象

①定义

数据对象是对软件必须理解的复合信息的抽象。复合信息是指具有一系列不同性质或属性的事物，仅有单个值的事物不是数据对象。

②特点

a．可以由一组属性来定义的实体都可以被认为是数据对象。

b．数据对象彼此间是有关联的。

c．数据对象只封装了数据而没有对施加于数据上的操作的引用，这也是数据对象与面向对象范型中的“类”或“对象”的显著区别。

（2）属性

属性定义了数据对象的性质。必须把一个或多个属性定义为“标识符”，即当希望找到数据对象的一个实例时，用标识符属性作为“关键字”（“键”）。应该根据对所要解决的问题的理解，来确定特定数据对象的一组合适的属性。

（3）联系

数据对象彼此之间相互连接的方式称为联系，也称为关系。联系也可能有属性。联系可分为以下3种类型。

①一对一联系（1：1）

一个数据对象只能对应一个数据对象，例如，一个部门有一个经理，而每个经理只在一个部门任职。

②一对多联系（1：N）

一个数据对象可以同时对应多个数据对象，例如，某校教师与课程之间存在一对多的联系“教”，即每位教师可以教多门课程，但是每门课程只能由一位教师来教。

③多对多联系（M：N）

两个数据对象之间的联系是多对多的。例如，学生与课程问的联系（“学”）是多对多的，即一个学生可以学多门课程，而每门课程可以有多个学生来学。

3．实体—联系图的符号

（1）定义

使用实体联系图来建立数据模型。可以把实体—联系图简称为E—R图，把用E—R图描绘的数据模型称为E—R模型。

E—R图中包含了实体（数据对象）、关系和属性3种基本成分，通常用矩形框代表实体，用连接相关实体的菱形框表示关系，用椭圆形或圆角矩形表示实体（或关系）的属性，并用直线把实体（或关系）与其属性连接起来。例如，图3-2是某学校教学管理的E—R图。

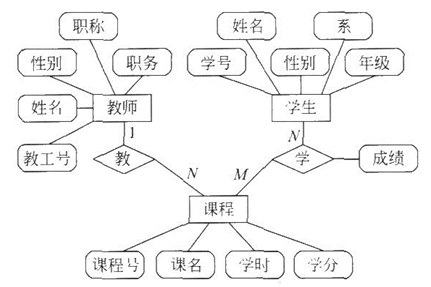


图3-2 某校教学管理E—R图

（2）优点

①E—R模型比较接近人的习惯思维方式；

②E—R模型使用简单的图形符号表达，便于用户理解。

六、数据规范化

（1）必要性

软件系统经常使用各种长期保存的信息，这些信息通常以一定方式组织并存储在数据库或文件中，为减少数据冗余，避免出现插入异常或删除异常，简化修改数据的过程，通常需要把数据结构规范化。

（2）范式特点

①通常用“范式”定义消除数据冗余的程度。第一范式数据冗余程度最大，第五范式数据冗余程度最小。

②范式级别越高，存储同样数据就要分解成更多张表，“存储自身”的过程也就越复杂。

③随着范式级别的提高，数据的存储结构与基于问题域的结构间的匹配程度也随之下降，故在需求变化时数据的稳定性较差。

④范式级别提高则需要访问的表增多，因此性能将下降。一般选用第三范式都比较恰当。

（3）各范式的定义。

①第一范式每个属性值都必须是原子值，即仅仅是一个简单值而不含内部结构。

②第二范式满足第一范式条件，而且每个非关键字属性都由整个关键字决定。

③第三范式符合第二范式的条件，每个非关键字属性都仅由关键字决定，而且一个非关键字属性不能仅仅是对另一个非关键字属性的进一步描述，即一个非关键字属性值不依赖于另一个非关键字属性值。

七、状态转换图

1．定义

状态转换图（状态图）通过描绘系统的状态及引起系统状态转换的事件，来表示系统的行为。状态图还提供了行为建模机制，指明了作为特定事件的结果系统将做哪些动作。

2．状态

（1）定义

状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。系统对事件的响应，既可以是做一个（或一系列）动作，也可以是仅仅改变系统本身的状态，还可以是既改变状态又做动作。

（2）分类

状态主要有：初态（初始状态）、终态（最终状态）和中间状态。在一张状态图中只能有一个初态，而终态则可以有0至多个。

（3）表示

状态图既可以表示系统循环运行过程，也可以表示系统单程生命期。

①描绘循环运行过程：通常并不关心循环是怎样启动的。

②描绘单程生命期：需要标明初始状态和最终状态。

3．事件

事件是在某个特定时刻发生的事情，它是对引起系统做动作或（和）从一个状态转换到另一个状态的外界事件的抽象。简而言之，事件就是引起系统做动作或（和）转换状态的控制信息。

4．状态图的符号

（1）符号的表示方法

①初态：用实心圆表示。

②终态：用一对同心圆（内圆为实心圆）表示。

③中间状态：用圆角矩形表示。可以用两条水平横线把它分成上、中、下3个部分。上面部分为状态的名称，这部分是必须有的；中间部分为状态变量的名字和值，下面部分是活动表。

（2）组成部分

图3-3给出了状态图中使用的主要组成部分和符号表示。

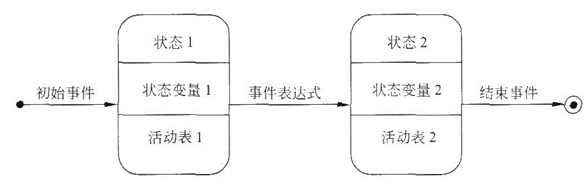


图3-3 状态图中使用的主要符号

①活动表

活动表的语法格式如下：

事件名（参数表）／动作表达式

其中，“事件名”可以是任何事件的名称。在活动表中经常使用下述3种标准事件：entry，exit和do。entry事件指定进入该状态的动作；exit事件指定退出该状态的动作；do事件则指定在该状态下的动作。需要时可以为事件指定参数表，活动表中的动作表达式描述应做的具体动作。

②状态转换

状态图中两个状态之间带箭头的连线称为状态转换，箭头指明了转换方向。状态变迁通常是由事件触发的，在这种情况下应在表示状态转换的箭头线上标出触发转换的事件表达式；如果在箭头线上未标明事件，则表示在源状态的内部活动执行完之后自动触发转换。

③事件表达式

事件表达式的语法如下：

事件说明[守卫条件]／动作表达式

其中，事件说明的语法为：事件名（参数表）。守卫条件是一个布尔表达式。如果同时使用事件说明和守卫条件，则当且仅当事件发生且布尔表达式为真时，状态转换才发生。动作表达式是一个过程表达式，当状态转换开始时执行该表达式。

5．实例分析

图3-4是电话系统的状态图。图中表明，没有人打电话时电话处于闲置状态；有人拿起听筒则进入拨号音状态，到达这个状态后，电话的行为是响起拨号音并计时；这时如果拿起听筒的人改变主意不想扣了，他把听筒放下（挂断），电话重又回到闲置状态；如果拿起听筒很长时间不拨号（超时），则进入超时状态。

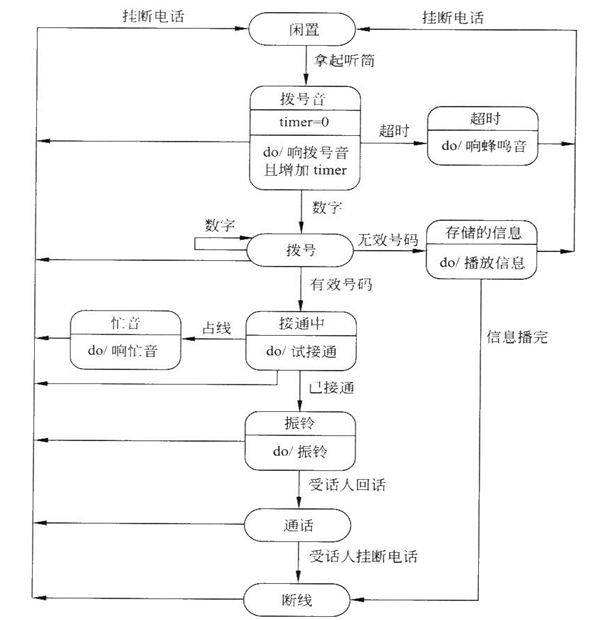


图3-4 电话系统的状态图

八、其他图形工具

1．层次方框图

（1）定义

层次方框图用树形结构的一系列多层次的矩形框描绘数据的层次结构。树形结构的顶层是一个单独的矩形框，它代表完整的数据结构，下面的各层矩形框代表这个数据的子集，最底层的各个框代表组成这个数据的实际数据元素。

（2）实例分析

例如，描绘一家计算机公司全部产品的数据结构可以用图3-5中的层次方框图表示。这家公司的产品由硬件、软件和服务3类产品组成，软件产品又分为系统软件和应用软件，系统软件又进一步分为操作系统、编译程序和软件工具等。

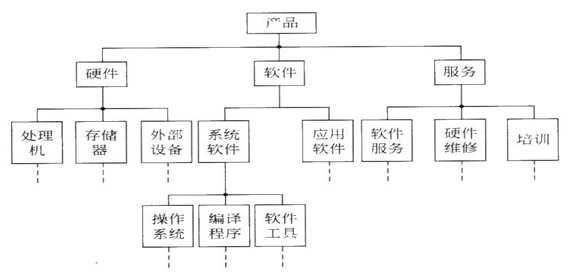


图3-5 层次方框图的一个例子

（3）特点

随着结构的精细化，层次方框图对数据结构也描绘得越来越详细，这种模式非常适合于需求分析阶段的需要。系统分析员从对顶层信息的分类开始，沿图中每条路径反复细化，直到确定了数据结构的全部细节时为止。

2.Warnier图

（1）定义

Warnier图是法国计算机科学家Warnier提出的表示信息层次结构的另外一种图像工具，它用树形结构描绘信息，可以表明信息的逻辑组织，即可以指出一类信息或一个信息元素是重复出现的，也可以表示特定信息在某一类信息中是有条件地出现的，它比层次方框图提供了更丰富的描绘手段。

（2）实例分析

图3-6是用Warnier图描绘一类软件产品的例子。图中花括号用来区分数据结构的层次，在一个花括号内的所有名字都属于同一类信息；异或符号（+）表明一类信息或一个数据元素在一定条件下才出现，而且在这个符号上、下方的两个名字所代表的数据只能出现一个；在一个名字下面（或右边）的圆括号中的数字指明了这个名字代表的信息类（或元素）在这个数据结构中重复出现的次数。

图中表示了一种软件产品要么是系统软件要么是应用软件。系统软件中有P1种操作系统，P2种编译程序，此外还有软件工具。软件工具是系统软件的一种，它又可以进一步细分为编辑程序、测试驱动程序和设计辅助工具，图中标出了每种软件工具的数量。



图3-6 Warnier图的一个例子

3.IPO图

（1）定义

IPO图是输入、处理、输出图的简称，它是由美国IBM公司发展完善起来的一种图形工具，能够方便地描绘输入数据、对数据的处理和输出数据之间的关系。

（2）用法

IPO图的基本形式是在左边的框中列出有关的输入数据，在中间的框内列出主要的处理，在右边的框内列出产生的输出数据。处理框中列出处理的次序暗示了执行的顺序。在IPO图中还用类似向量符号的粗大箭头清楚地指出数据通信的情况。图3-7是一个主文件更新的例子。

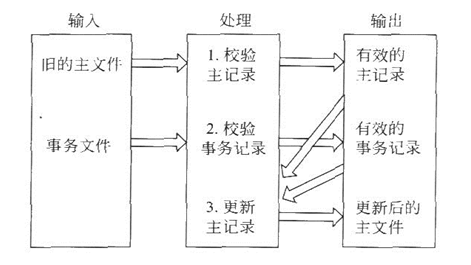


图3-7 IPO图的一个例子

（3）改进的IPO图

改进的IPO图（IPO表）中包含某些附加的信息。如图3-8所示，改进的IPO图中包含的附加信息主要有系统名称、图的作者，完成的日期，本图描述的模块的名字，模块在层次图中的编号，调用本模块的模块清单，本模块调用的模块的清单，注释，以及本模块使用的局部数据元素等。在需求分析阶段可以使用IPO图简略地描述系统的主要算法。

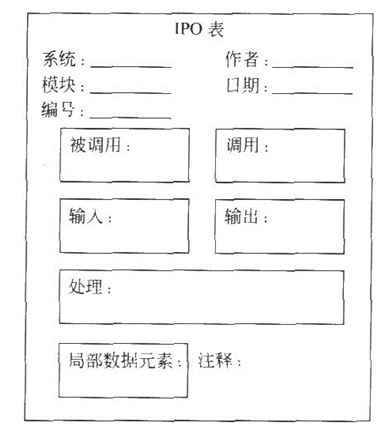


图3-8 改进的IPO图的形式

（4）优点

①IPO图使用的基本符号既少又简单，易于掌握。

②可以在软件设计阶段修正IPO图。

九、验证软件需求

1．验证软件需求的正确性

（1）验证需求正确性的目的

需求分析阶段的工作结果是开发软件系统的重要基础，大量统计数字表明，软件系统中15％的错误起源于错误的需求。为了提高软件质量，确保软件开发成功，降低软件开发成本，一旦对目标系统提出一组要求之后，必须严格验证这些需求的正确性。

（2）进行验证的四个方面

①一致性

所有需求必须是一致的，任何一条需求不能和其他需求互相矛盾。

②完整性

需求必须是完整的，规格说明书应该包括用户需要的每一个功能或性能。

③现实性

指定的需求应该是用现有的硬件技术和软件技术基本上可以实现的。

④有效性

必须证明需求是正确有效的，确实能解决用户面对的问题。

2．验证软件需求的方法

（1）验证需求的一致性

当软件需求规格说明书是用形式化的需求陈述语言书写的时候，可以用软件工具验证需求的一致性，从而能有效地保证软件需求的一致性。

（2）验证需求的现实性

为了验证需求的现实性，应该参照以往开发类似系统的经验，分折用现有的软、硬件技术实现目标系统的可能性。必要的时候应该采用仿真或性能模拟技术，辅助分析软件需求规格说明书的现实性。

（3）验证需求的完整性和有效性

只有目标系统的用户才真正知道软件需求规格说明书是否完整、准确地描述了他们的需求。因此，检验需求的完整性，特别是证明系统确实满足用户的实际需要，即需求的有效性，只有在用户的密切合作下才能完成。采用原型系统的方法来进行验证。用户通过试用原型系统，也能获得许多宝贵的经验，从而可以提出更符合实际的要求。在快速建立原型系统时，可以适当降低对接口、可靠性和程序质量的要求，以及省掉许多文档资料方面的工作，从而可以大大降低原型系统的开发成本。

3．用于需求分析的软件工具

（1）要求

为了更有效地保证软件需求的正确性，特别是为了保证需求的一致性，需要有适当的软件工具支持需求分析工作。这类软件工具应该满足下列要求。

①必须有形式化的语法（或表），因此可以用计算机自动处理使用这种语法说明的内容。

②使用这个软件工具能够导出详细的文档。

③必须提供分析（测试）规格说明书的不一致性和冗余性的手段，并且应该能够产生一组报告指明对完整性分析的结果。

④使用这个软件工具之后，应该能够改进通信状况。

（2）PSL/PSA系统

①定义

PSL是用来描述系统的形式语言，PSA是处理PSL描述的分析程序。用PSL描述的系统属性放在一个数据库中。一旦建立起数据库之后即可增加信息、删除信息或修改信息，并且保持信息的一致性。PSA对数据库进行处理以产生各种报告，测试不一致性或遗漏，并且生成文档资料。

②功能

a．描述任何应用领域的信息系统；

b．创建一个数据库保存对该信息系统的描述符；

c．对描述符施加增加、删除和更改等操作；

d．产生格式化的文档和关于规格说明书的各种分析报告。

③优点

a．改进了文档质量，能保证文档具有完整性、一致性和无二义性，从而可以减少管理和维护的费用；

b．数据存放在数据库中，便于增加、删除和更改。

**3.2　课后习题详解**

1．为什么要进行需求分析？通常对软件系统有哪些需求？

答：（1）需求分析的原因

为了开发真正满足用户需求的软件产品，需求分析是软件开发工作获得成功的前提条件，不能满足用户需求的程序只会令用户失望，给开发者带来烦恼。

（2）对软件系统的需求

功能需求、性能需求、可靠性和可用性需求、出错处理、借口需求、约束、逆向需求、将来可能提出的要求。

2．怎样与用户有效地沟通以获取用户的真实需求？

答：访谈是最早开始使用的获取用户需求的技术，也是目前广泛使用的需求分析技术，访谈有两种形式，分别是正式的和非正式的访谈。正式访谈时，系统分析员将提出一些事先准备好的具体问题。在非正式访谈中，分析员将提出一些用户可以自由回答的开放性问题，以鼓励被访问人员说出自己的想法。其中情景分析技术往往非常有效。

3．银行计算机储蓄系统的工作过程大致如下：储户填写的存款单或取款单由业务员输入系统，如果是存款则系统记录存款人姓名、住址（或电话号码）、身份证号码、存款类型、存款日期、到期日期、利率及密码（可选）等信息，并印出存单给储户；如果是取款而且存款时留有密码，则系统首先核对储户密码，若密码正确或存款时未留密码，则系统计算利息并印出利息清单给储户。

用数据流图描绘系统中的数据对象，并用实体联系图描绘系统中的数据对象。

答：（1）数据流图，如图3-9所示。

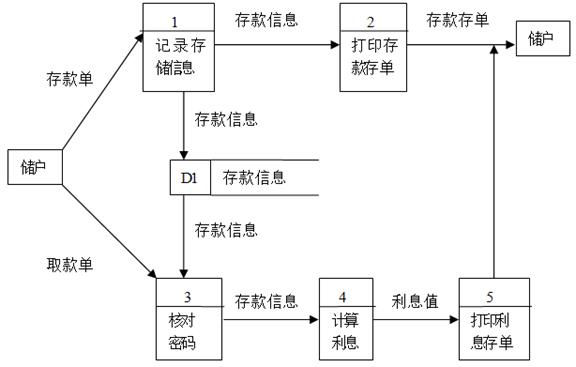


图3-9 银行计算机储蓄系统数据流图

（2）E-R模型

如图3-10所示。本题中共有两类实体，分别是“储户”和“储蓄所”，在它们之间存在“存取款”关系。因为一位储户可以在多家储蓄所存取款，一家储蓄所拥有多位储户，所以“存取款”是多对多（M：N）关系。

储户的属性主要有姓名、住址、电话号码和身份证号码，储蓄所的属性主要是名称、地址和电话号码，而数额、类型、到期日期、利率和密码则是关系类型存取款的属性。

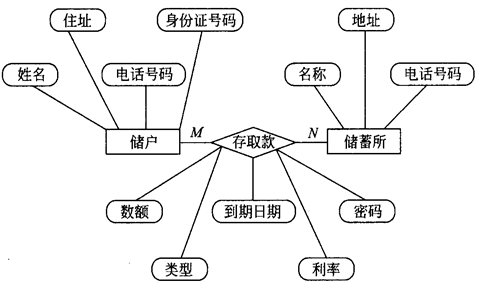


图3-10 银行计算机储蓄系E-R图

4．分析习题2第3题所述的机票预订系统。试用实体一联系图描绘本系统中的数据对象并用数据流图描绘本系统的功能。

答：（1）数据流图，如图3-11所示。

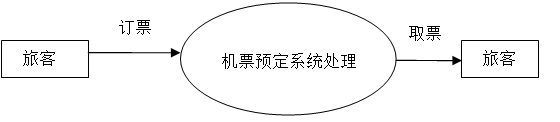


图3-11 购票系统数据流图

（2）E-R图，如图3-12所示。



图3-12 购票系统E—R图

5．分析习题2第4题所述的患者监护系统。试用实体联系图描绘本系统中的数据对象，画出本系统的顶层IPO图。

答：E—R图，如图3-13所示。

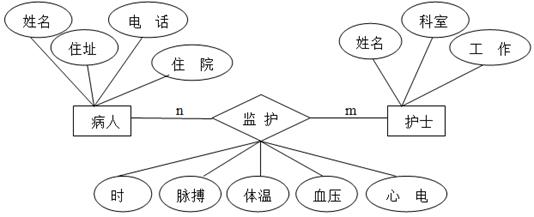


图3-13 患者监护系统ER图

6．复印机的工作过程大致如下：未接到复印命令时处于闲置状态，一旦接到复印命令则进入复印状态，完成一个复印命令规定的工作后又回到闲置状态，等待下一个复印命令；如果执行复印命令时发现没纸，则进入缺纸状态，发出警告，等待装纸，装满纸后进入闲置状态，准备接收复印命令；如果复印时发生卡纸故障，则进入卡纸状态，发出警告，等待维修人员来排除故障，故障排除后回到闲置状态。

试用状态转换图描绘复印机的行为。

答：状态转换图，如图3-14所示。

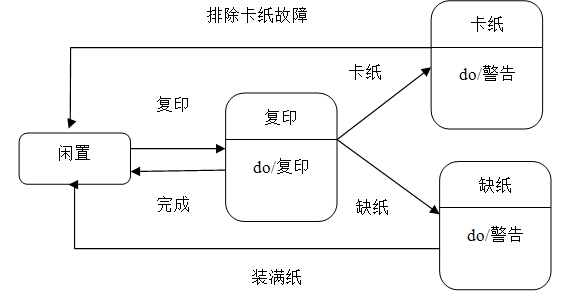


图3-14 复印机工作状态转换图

**第4章　形式化说明技术**

**4.1　复习笔记**

一、概述

按照形式化的程度，可以把软件工程使用的方法划分成非形式化、半形式化和形式化3类。用自然语言描述需求规格说明，是典型的非形式化方法。用数据流图或实体联系图建立模型，是典型的半形式化方法。所谓形式化方法，是描述系统性质的基于数学的技术，也就是说，如果一种方法有坚实的数学基础，那么它就是形式化的。

1．非形式化方法的缺点

（1）矛盾：一组相互冲突的陈述。

（2）二义性：指读者可以用不同方式理解的陈述。

（3）含糊性：指没有指明任何有用信息的笼统的陈述。

（4）不完整性：指没有指明具体功能的陈述。

（5）抽象层次混乱：指非抽象的陈述中混进了一些关于细节的低层次陈述。

2．形式化方法的优点

（1）能够简洁准确地描述物理现象、对象或动作的结果。准确到几乎没有二义性，而且可以用数学方法来验证，以发现存在的矛盾和不完整性，在这样的规格说明中完全没有含糊性。

（2）可以在不同的软件工程活动之间平滑地过渡。不仅功能规格说明，而且系统设计也可以用数学表达，当然，程序代码也是一种数学符号。

（3）提供了高层确认的手段。可以使用数学方法证明，设计符合规格说明，程序代码正确地实现了设计结果。

3．应用形式化方法的准则

为了更好地发挥形式化方法的长处，应用形式化方法应遵循以下几条准则：

（1）应该选用适当的表示方法

通常，一种规格说明技术只能用自然的方式说明某一类概念，如果用这种技术描述其不适于描述的概念，则不仅工作量大而且描述方式也很复杂。

（2）应该形式化，但不要过分形式化

形式化规格说明技术要求人们非常准确地描述事物，因此有助于防止含糊和误解。事实上，如果用形式化方法仔细说明系统中易出错的或关键的部分，则只用适中的工作量就能获得较大回报。

（3）应该估算成本

为了使用形式化方法，通常需要事先进行大量的培训。最好预先估算所需的成本并编入预算。

（4）应该有形式化方法顾问随时提供咨询

绝大多数软件工程师对形式化方法中使用的数学和逻辑并不很熟悉，而且没受过使用形式化方法的专业训练，因此，需要专家指导和培训。

（5）不应该放弃传统的开发方法

把形式化方法和结构化方法或面向对象方法集成起来是可能的，而且由于取长补短往往能获得很好的效果。

（6）应该建立详尽的文档

建议使用自然语言注释形式化的规格说明书，以帮助用户和维护人员理解系统。

（7）不应该放弃质量标准

形式化方法并不能保证软件的正确性，它们只不过是有助于开发出高质量软件的一种手段。除了使用形式化说明技术外，在系统开发过程中仍然必须一如既往地实施其他质量保证活动。

（8）不应该盲目依赖形式化方法

形式化方法并不能保证开发出的软件绝对正确，例如，无法用形式化方法证明从非形式化需求到形式化规格说明的转换是正确的，因此，必须用其他方法（例如评审、测试）来验证软件正确性。

（9）应该测试、测试再测试

形式化方法不仅不能保证软件系统绝对正确，也不能证明系统性能或其他质量指标符合需要，因此，软件测试的重要性并没有降低。

（10）应该重用

即使采用了形式化方法，软件重用仍然是降低软件成本和提高软件质量的唯一合理的方法。而且用形式化方法说明的软件构件具有清晰定义的功能和接口，使得它们有更好的可重用性。

二、有穷状态机

有穷状态机是表达规格说明的一种形式化方法。

1．概念

（1）引入

一个保险箱上装了一个复合锁，锁有3个位置，分别标记为1、2、3，转盘可向左（L）或向右（R）转动。在任意时刻转盘都有6种可能的运动，即1L、1R、2L、2R、3L和3R。保险箱的组合密码是1L、3R、2L转盘的任何其他运动都将引起报警。图4-1描绘了保险箱的状态转换情况。有一个初始态，即保险箱锁定状态。若输入为1L，则下一个状态为A，但是，若输入不是1L而是转盘的任何其他移动，则下一个状态为“报警”，报警是两个终态之一（另一个终态是“保险箱解锁”）。如果选择了转盘移动的正确组合，则保险箱状态转换的序列为从保险箱锁定到A再到B，最后到保险箱解锁，即另外一个终态。图4-1是一个有穷状态机的状态转换图。状态转换并不一定要用图形方式描述，表4-1的表格形式也可以表达同样的信息。除了两个终态之外，保险箱的其他状态将根据转盘的转动方式转换到下一个状态。

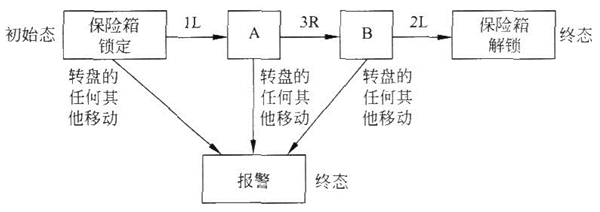


图4-1 保险箱的状态转换图

表4-1 保险箱的状态转换表



（2）构成

一个有穷状态机包括下述5个部分：状态集J、输入集K、由当前状态和当前输入确定下一个状态（次态）的转换函数T、初始态S和终态集F。一个有穷状态机可以表示为一个5元组（J，K，T，S，F）。

①状态集J：{保险箱锁定，A，B，保险箱解锁，报警}。

②输入集K：{1L，1R，2L，2R，3L，3R}。

③转换函数T：如表4-1所示。

④初始态S：保险箱锁定。

⑤终态集F：{保险箱解锁，报警}。

其中：J是一个有穷的非空状态集；K是一个有穷的非空输入集；T是一个从（J-F）×K到J的转换函数；S∈J，是一个初始状态；F⊆J，是终态集。

（3）状态转换

状态的每个转换都具有下面的形式：

当前状态[菜单]+事件[所选择的项]IMG_309下个状态

（4）扩展

对有穷状态机做一个扩展，即在前述的5元组中加入第6个组件—谓词集P，从而把有穷状态机扩展为一个6元组，其中每个谓词都是系统全局状态Y的函数。转换函数T现在是一个从（J-F）×K×P到J的函数。现在的转换规则形式如下：

当前状态[菜单]+事件[所选择的项]+谓词IMG_310下个状态

2．评价

（1）优点

①有穷状态机方法采用了一种简单的格式来描述规格说明：当前状态+事件+谓词IMG_311下个状态。这种形式的规格说明易于书写、易于验证，而且可以比较容易地把它转变成设计或程序代码。

②有穷状态机方法比数据流图技术更精确，而且易于理解。

（2）缺点

①在开发一个大系统时，三元组（即状态、事件、谓词）的数量会迅速增长。

②形式化的有穷状态机方法没有处理定时需求。

三、Petri网

1．概念

（1）功能

并发系统中遇到的一个主要问题是定时问题。定时问题通常是由不好的设计或有错误的实现引起的，而这样的设计或实现通常又是由不好的规格说明造成的。如果规格说明不恰当，则有导致不完善的设计或实现的危险。用于确定系统中隐含的定时问题的一种有效技术是Petri网，这种技术的一个很大的优点是它也可以用于设计中。

（2）构成

①一般构成

Petri网包含4种元素：一组位置P、一组转换T、输入函数I，以及输出函数O。图4-2举例说明了Petri网的组成。

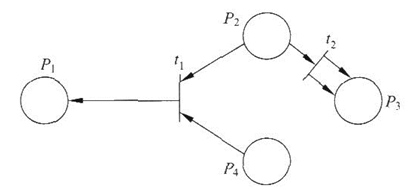


图4-2 Petri网的组成

a．一组位置P为{ P1，P2，P3，P4}，在图中用圆圈代表位置。

b．一组转换T为{ T1，T2}在图中用短直线表示转换。

c．两个用于转换的输入函数，用由位置指向转换的箭头表示，它们是：

I（t1）={ P2，P4}

I（t2）={ P2}

d．两个用于转换的输出函数，用由转换指向位置的箭头表示，它们是：

O（t1）={ P1}

O（t2）={ P3，P3}

注意：输出函数O（t1）中有两个P3，是因为有两个箭头由t2指向P3。

②形式化的Petri网

更形式化的Petri网结构是一个四元组：

C=（P，T，I，O）

其中：

a．P={P1，...Pn}是一个有穷位置集，n≥0。

b．T={t1，…，tm}是一个有穷转换集，m≥0，且T和P不相交。

c．I：T→P∞为输入函数，是由转换到位置无序单位组（bags）的映射。

d．O：T→P∞为输出函数，是由转换到位置无序单位组的映射。

（3）分配权标

①带标记的Petri网

在图4-3中有4个权标，其中一个在P1中，两个在P2中，P3中没有，还有一个在P4中。标记可以用向量（1，2，0，1）表示。由于P2和P4中有权标，因此t1启动（即被激发）。当每个输入位置所拥有的权标数大于等于从该位置到转换的线数时，就允许转换。当t1被激发时，P2和P4上各有一个权标被移出，而P1上则增加一个权标。Petri网中权标总数不是固定的，在这个例子中两个权标被移出，而P1上只能增加一个权标。

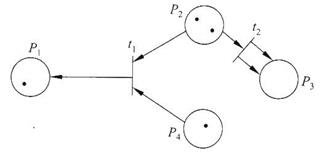


图4-3 带标记的Petri网

②Petri网转换

在图4-3中P2上有权标，因此t2也可以被激发。当t2被激发时，P2上将移走一个权标，而P3上新增加两个权标。Petri网具有非确定性，也就是说，如果数个转换都达到了激发条件，则其中任意一个都可以被激发。图4-3所示Petri网的标记为（1，2，0，1），t1和t2都可以被激发。假设t1被激发了，则结果如图4-4所示，标记为（2，1，0，0）。此时，只有t2可以被激发。如果t2也被激发了，则权标从P2中移出，两个新权标被放在P3上，结果如图4-5所示，标记为（2，0，2，0）。

更形式化地说，Petri网C=（P，T，I，O，M）中的标记M，是由一组位置P到一组非负整数的映射：M：P→{0，1，2．…}。

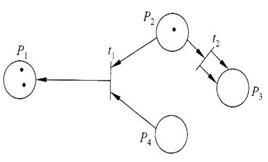


图4-4 t1被激发后的情况

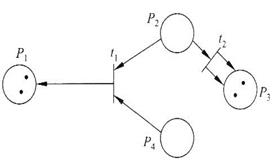


图4-5图 t2被激发后的情况

③加入禁止线。

如图4-6所示，禁止线是用一个小圆圈而不是用箭头标记的输入线。通常，当每个输入线上至少有一个权标，而禁止线上没有权标的时候，相应的转换才是允许的。在图4-6中，P3上有一个权标而P2上没有权标，因此转换t1可以被激发。

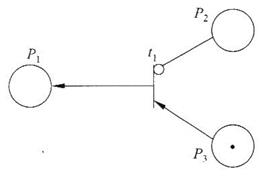


图4-6 含禁止线的Petri网

四、Z语言

1．简介

使用Z语言需要具备集合论、函数、数理逻辑等方面的知识。即使用户已经掌握了所需要的背景知识，Z语言也是相当难学的，因为它除了使用常用的集合论和数理逻辑符号之外，还使用一些特殊符号。用Z语言描述的、最简单的形式化规格说明含有下述四个部分：

（1）给定的集合

一个Z规格说明从一系列给定的初始化集合开始。所谓初始化集合就是不需要详细定义的集合，这种集合用带方括号的形式表示。

（2）状态定义

一个Z规格说明由若干个“格”组成，每个格含有一组变量说明和一系列限定变量取值范围的谓词。Z格S的格式如图4-7所示。

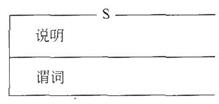


图4-7 Z格S的格式

（3）初始状态

抽象的初始状态是指系统第一次开启时的状态。

（4）操作

①当一个格被用在另一个格中时，要在它的前面加上三角形符号△作为前缀。问号“？”表示输入变量，而感叹号“!”代表输出变量。

②谓词部分，包含了一组调用操作的前置条件，以及操作完全结束后的后置条件。如果前置条件成立，则操作执行完成后可得到后置条件。但是，如果在前置条件不成立的情况下调用该操作，则不能得到指定的结果。

2．评价

目前，Z也许是应用得最广泛的形式化语言，尤其是在大型项目中Z语言的优势更加明显。Z语言获得成功的主要原因为：

（1）可以比较容易地发现用Z写的规格说明的错误，特别是在自己审查规格说明，及根据形式化的规格说明来审查设计与代码时，情况更是如此。

（2）要求十分精确地使用Z说明符写规格说明。减少了模糊性、不一致性和遗漏。

（3）Z是一种形式化语言，在需要时开发者可以严格地验证规格说明的正确性。

（4）只用比较短的时间就能够让开发人员学会编写Z规格说明。

（5）使用Z语言通过减少开发过程所需要的总时间来降低软件开发费用。

（6）用户可以依据Z规格说明用自然语言重写比直接用自然语言写出的非形式化规格说明更加清楚和正确。

**4.2　课后习题详解**

1．举例对比形式化方法和欠形式化方法的优缺点。

答：（1）欠形式化方法的缺点

①矛盾

矛盾是指一组相互冲突的陈述。

②二义性

二义性是指读者可以用不同方式理解的陈述。

③含糊性

例如，人们可能经常在文档中看到类似下面这样的需求：“系统界面应该是对用户友好的。”实际上，这样笼统的陈述并没有给出任何有用的信息。

④不完整性。

如果在规格说明书中对一个命令的功能没有更多的描述，那么，这个命令的细节是严重不完整的。

⑤抽象层次混乱。

抽象层次混乱是指在非常抽象的陈述中混进了一些关于细节的低层次陈述。这样的规格说明书使得读者很难了解系统的整体功能结构。

（2）形式化方法的优点

①能够简洁准确地描述物理现象、对象或动作的结果。

在理想情况下，分析员可以写出系统的数学规格说明，它准确到几乎没有二义性，而且可以用数学方法来验证，以发现存在的矛盾和不完整性，在这样的规格说明中完全没有含糊性。

②可以在不同的软件工程活动之间平滑地过渡。

③提供了高层确认的手段。

可以使用数学方法证明，设计符合规格说明，程序代码正确地实现了设计结果。

2．在什么情况下应该使用形式化说明技术？使用形式化说明技术时应遵守哪些准则？

答：（1）人们在理解用自然语言描述的规格说明时，容易产生二义性。为了克服非形式化方法的缺点，人们把数学引入软件工程。创造了基于数学的形式化说明技术。

（2）应用形式化方法的准则：

①应该选用适当的表示方法。

②应该形式化，但不要过分形式化。

③应该估算成本。

④应该有形式化方法顾问随时提供咨询。

⑤不应该放弃传统的开发方法。

⑥应该建立详尽的文档。

⑦不应该放弃质量标准。

⑧不应该盲目依赖形式化方法。

⑨应该测试、测试再测试。

⑩应该重用。

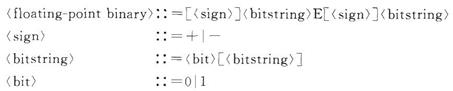
3．一个浮点二进制数的构成是：一个可选的符号（+或-），后跟一个或多个二进制位，再跟上一个字符E，再加上另一个可选符号（+或-）及一个或多个二进制位。例如，下列的字符串都是浮点二进制数：

110101E-101

-100111E11101

+1E0

更形式化地，浮点二进制数定义如下：



其中：

符号∷=表示定义为；

符号[...]表示可选项；

符号a | b表示“a或b。

假设有这样一个有穷状态机：以一串字符为输入，判断字符串中是否含有合法的浮点二进制数。试对这个有穷状态机进行规格说明。

答：（1）自然语言描述

改有穷状态机的初态是“等待字符串输入”。在初态若接收到字符“+”、字符“-”、或二进制位，则进入“输入尾数”状态；在初态若接收到其它字符，则进入终态“非浮点二进制数”。在“输入尾数”状态若接收到二进制位，则保持该状态不变；若接收到字符“E”，则进入“等待输入指数”状态；若接收到其它字符，则进入终态“非浮点二进制数”。在“等待输入指数”状态若接收到字符“+”、字符“-”、或二进制位，则进入“输入指数”状态；若接收其它字符，则进入终态“非浮点二进制数”。在“输入指数”状态若接收到二进制位，则保持该状态不变；若输入其他字符，则进入终态“非浮点二进制数”；若输入结束，则进入终态“浮点二进制数”。

（2）形式化图，如图4-8所示。

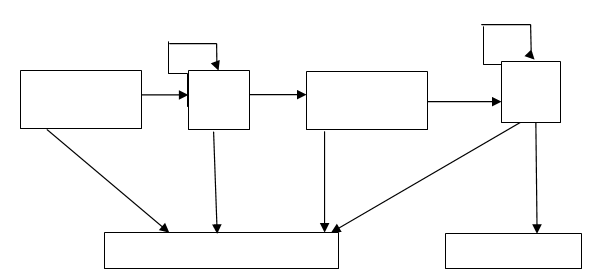


图4-8 有穷机形式化图

4．考虑下述的自动化图书馆流通系统：每本书都有一个条形码，每个借阅人都有一个带有条形码的卡片。当一个借阅人想借一本书时，图书管理员扫描书上的条形码和借阅人卡片上的条形码，然后在计算机终端上输入C；当归还一本书时，图书管理员将再做一次扫描，并输入R。图书管理员可以把一些书加到（+）图书集合中，也可以删除（-）它们。借阅人可以在终端上查找到某个作者所有的书（输入“A=”和作者名字），或具有指定标题的所有书籍（输入“T=”和标题），或属于特定主题范围内的所有图书（输入“S=”加主题范围）。最后，如果借阅人想借的书已被别人借走，图书管理员将给这本书设置一个预约，以便书归还时把书留给预约的借阅人（输入“H=”加书号）。

试用有穷状态机说明上述的图书流通系统。

答：图书馆流通系统的有穷状态机描述如下：

（1）图书状态的有穷状态机描述

状态机J：{书在图书馆S1，书被借出S2，书被预约S3}

输入集K：{书上条形码，借阅卡条形码，终端输入各种命令}

转换函数T：如图4-9所示。

初始态S：{书在图书馆S1，书被借出S2}

终态集F：{书被借出S2，书被预约S3}

IMG_320

图4-9 图书状态转换函数

(2)图书馆终端管理员模式的有穷状态机描述。

状态机J：{管理员设置状态，书入库，书出库（删除），预约}

输入集K：{终端输入管理员命令，书的各种状态（S1，S2，S3）}

转换函数T：如图4-10所示。

初始态S：{管理员设置状态}

终态集F：{书入库，书出库（删除），预约，}

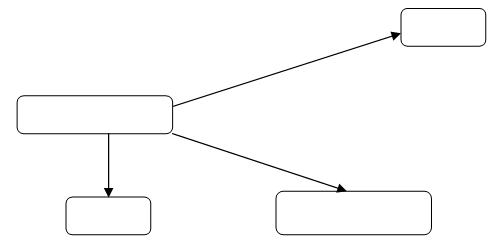


图4-10 图书馆终端管理员模式转换函数

（3）图书馆终端用户模式的有穷状态机描述

状态机J：{读者查询状态，查询结果}

输入集K：{终端输入用户查询命令，书的各种状态（S1，S2，S3）}

转换函数T：如图4-11所示

初始态S：{读者查询状态}

终态集F：{查询结果}

IMG_322

图4-11 图书馆终端用户模式转换函数

5．试用Petri网说明第4题所述图书馆中一本书的循环过程。在规格说明中应该包括操作H、C及R。

答：图书馆流通系统中一本书的巡回过程的Petri图如图4-12所示。其中P1表示书在图书馆，P2表示书在读者手上，P3书被预约。

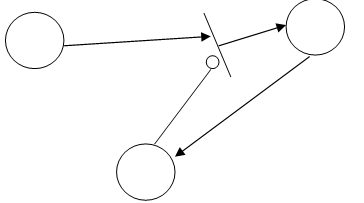


图4-12 图书馆流通系统Petri网

6．试用Z语言对第4题所述图书馆图书流通系统做一个完整的规格说明。

答：略。

**第5章　总体设计**

**5.1　复习笔记**

一、总体设计概念

1．定义

总体设计的基本目的就是回答“系统应该如何实现”这个问题，总体设计又称为概要设计或初步设计。

2．主要任务

（1）划分出组成系统的物理元素程序、文件、数据库、人工过程和文档等，但是每个物理元素仍然处于黑盒子级，这些黑盒子里的具体内容将在以后仔细设计。

（2）设计软件的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系。

3．步骤

①寻找实现目标系统的各种不同的方案，需求分析阶段得到的数据流图是设想各种可能方案的基础。

②分析员从这些供选择的方案中选取若干个合理的方案，为每个合理的方案都准备一份系统流程图，列出组成系统的所有物理元素，进行成本／效益分析，并且制定实现这个方案的进度计划。

③进行必要的数据库设计，确定测试要求并且制定测试计划。

4．必要性

可以站在全局高度上，花较少成本，从较抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低成本开发出较高质量的软件系统。

二、设计过程

总体设计过程通常由两个主要阶段组成：系统设计阶段，确定系统的具体实现方案；结构设计阶段，确定软件结构。典型的总体设计过程包括下述9个步骤。

1．设想供选择的方案

在总体设计阶段应该考虑各种可能的实现方案，并且力求从中选出最佳方案。在总体设计阶段开始时只有系统的逻辑模型，分析员有充分的自由分析比较不同的物理实现方案，一旦选出了最佳的方案，将能大大提高系统的性能／价格比需求分析阶段得出的数据流图是总体设计的极好的出发点。常用的方法是，设想把数据流图中的处理分组的各种可能的方法，抛弃在技术上行不通的分组方法，余下的分组方法代表可能的实现策略，并且可以启示供选择的物理系统。

2．选取合理的方案

应该从前一步得到的一系列供选择的方案中选取若干个合理的方案，通常至少选取低成本、中等成本和高成本的3种方案。

对每个合理的方案，都应该准备下列4份资料：

（1）系统流程图。

（2）组成系统的物理元素清单。

（3）成本/效益分析。

（4）实现这个系统的进度计划。

3．推荐最佳方案

应该综合分析对比各种合理方案的利弊，推荐一个最佳的方案，并且为推荐的方案制定详细的实现计划。用户和有关的技术专家应该认真审查分析员所推荐的最佳系统，如果该系统确实符合用户的需要，并且是在现有条件下完全能够实现的，则应该提请使用部门负责人进一步审批。在使用部门的负责人也接受了分析员所推荐的方案之后，将进入总体设计过程的下一个重要阶段—结构设计。

4．功能分解

为确定软件结构，需要从实现角度把复杂的功能进一步分解。需要结合算法描述仔细分析数据流图中的每个处理，如果一个处理的功能过分复杂，必须把它的功能适当地分解成一系列比较简单的功能。

5．设计软件结构

把模块组织成良好的层次系统，顶层模块调用它的下层模块以实现程序的完整功能，每个下层模块再调用更下层的模块，从而完成程序的一个子功能，最下层的模块完成最具体的功能。软件结构，即由模块组成的层次系统可以用层次图或结构图来描绘。

6．设计数据库

对于需要使用数据库的那些应用系统，应该在需求分析阶段所确定的系统数据需求的基础上，进一步设计数据库。

7．制定测试计划

在软件开发的早期阶段考虑测试问题，能促使软件设计人员在设计时注意提高软件的可测试性。

8．书写文档

用正式的文档记录总体设计的结果，在这个阶段应该完成的文档通常有下述几种。

（1）系统说明，主要包括用系统流程图描绘的系统构成方案，组成系统的物理元素清单，成本／效益分析；对最佳方案的概括描述，精化的数据流图，用层次图或结构图描绘的软件结构，用IP0图或其他工具（例如，PDL语言）简要描述的各个模块的算法，模块问的接口关系，以及需求、功能和模块三者之间的交叉参照关系等。

（2）用户手册。

（3）测试计划，包括测试策略，测试方案，预期的测试结果，测试进度计划等。

（4）详细的实现计划。

（5）数据库设计结果。

9．审查和复审

最后对总体设计结果进行严格的技术审查，在技术审查通过后再由客户从管理角度进行复审。

三、设计原理

1．模块化

（1）模块

模块是由边界元素限定的相邻程序元素的序列，而且有一个总体标识符代表它。模块是构成程序的基本构件。过程、函数、子程序和宏等，都可作为模块。面向对象方法学中的对象是模块，对象内的方法也是模块。模块是构成程序的基本构件。

（2）模块化

模块化就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。模块化是为了使一个复杂的大型程序能被人的智力所管理，是软件应该具备的唯一属性。

（3）优点

①使软件结构清晰，不仅容易设计也容易阅读和理解。

②使软件容易测试和调试，有助于提高软件的可靠性。

③提高软件的可修改性。

④有助于软件开发工程的组织管理。

2．抽象

（1）定义

把在现实世界中一定事物、状态或过程之间的共性集中和概括起来，暂时忽略它们之间的差异，这就是抽象或者说抽象就是抽出事物的本质特性而暂时不考虑它们的细节。抽象是人类在认识复杂现象的过程中使用的最强有力的思维工具。

（2）原理

处理复杂系统的唯一有效的方法是用层次的方式构造和分析它。一个复杂的动态系统先可以用一些高级的抽象概念构造和理解，这些高级概念又可以用一些较低级的概念构造和理解，不断循环进行下去，直至最低层次的具体元素。

（3）层次分析

①在抽象的最高层次使用问题环境的语言，以概括的方式叙述问题的解法；

②在较低抽象层次采用更过程化的方法，把面向问题的术语和面向实现的术语结合起来叙述问题的解法；

③在最低的抽象层次用可以直接实现的方式叙述问题的解法。

（4）在软件工程中的应用

软件工程过程的每一步都是对软件解法的抽象层次的一次精化。

①在可行性研究阶段，软件作为系统的一个完整部件；

②在需求分析期间，软件解法是使用在问题环境内熟悉的方式描述的；

③由总体设计向详细设计过渡时，抽象的程度也就随之减少了；

④源程序写出来以后，也就达到了抽象的最低层。

（5）优点

①简化了软件的设计和实现；

②提高了软件的可理解性和可测试性；

③使得软件更容易维护。

3．逐步求精

（1）定义

逐步求精是软件工程技术的基础。可以把逐步求精定义为：为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。逐步求精是人类解决复杂问题时采用的基本方法。

（2）要求

逐步求精是一项把一个时期内必须解决的种种问题按优先级排序的技术。逐步求精方法确保每个问题都将被解决，而且每个问题都将在适当的时候被解决，但任何时候一个人都不需要同时处理7个以上知识块。

（3）原理

①逐步求精最初是一种自顶向下的设计策略。程序的体系结构是通过逐步精化处理过程的层次而设计出来的。通过逐步分解对功能的宏观陈述而开发出层次结构，直至最终得出用程序设计语言表达的程序。

②求精实际上是细化过程。人们从在高抽象级别定义的功能陈述（或信息描述）开始，也就是说，该陈述仅仅概念性地描述了功能或信息，但是并没有提供功能的内部工作情况或信息的内部结构。

（4）抽象与求精

抽象与求精是一对互补的概念。抽象使得设计者能够说明过程和数据，同时却忽略了低层细节。求精则帮助设计者在设计过程中逐步揭示出低层细节。这两个概念都有助在设计演化过程中创造出完整的设计模型。

4．信息隐藏和局部化

（1）定义

信息隐藏原理是在设计和确定的模块中，使一个模块内包含的信息（过程和数据）对于不需要这些信息的模块来说，是不能访问的。

（2）局部化与信息隐藏

①局部化的概念和信息隐藏概念是密切相关的。局部化是指把一些关系密切的软件元素物理地放得彼此靠近。在模块中使用局部数据元素是局部化的一个例子，局部化有助于实现信息隐藏。

②“隐藏”意味着有效的模块化可以通过定义一组独立的模块而实现，这些独立的模块彼此间仅仅交换那些为了完成系统功能而必须交换的信息。

（3）优点

绝大多数数据和过程对于软件的其他部分而言是隐藏的，在修改期间由于疏忽而引入的错误就很少可能传播到软件的其他部分。

5．模块独立

（1）定义

开发具有独立功能而且和其他模块之间没有过多的相互作用的模块，就可以做到模块独立。使得每个模块完成一个相对独立的特定子功能，并且和其他模块之间的关系很简单。模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。

（2）重要性

①具有独立的模块的软件比较容易开发出来。

②独立的模块比较容易测试和维护。

（3）耦合

①定义

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度、进入或访问一个模块的点，以及通过接口的数据。模块间的耦合程度强烈影响着系统的可理解性、可测试性、可靠性和可维护性。耦合衡量不同模块彼此之间互相依赖（连接）的紧密程度。

②分类

a．完全独立

如果两个模块中的每一个都能独立地工作而不需要另一个模块的存在，则称它们彼此完全独立，耦合程度最低。但是，在一个软件系统中不可能所有模块之间都没有任何连接。

b．数据耦合

如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，则称它们是数据耦合。数据耦合是低耦合，系统中至少必须存在这种耦合

c．控制耦合

如果两个模块彼此间通过参数交换信息，并且传递的信息中包含控制信息（这种控制信息可以以数据的形式出现），则称它们是控制耦合。控制耦合是中等程度的耦合，它增加了系统的复杂程度。控制耦合往往是多余的，可用数据耦合代替它。

d．特征耦合

如果整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素，则称它们是特征耦合。在这种情况下，被调用的模块可以使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制，从而给计算机犯罪提供了机会。

e．公共环境耦合

如果两个或多个模块通过一个公共数据环境相互作用，则称它们是公共环境耦合。公共环境耦合的复杂程度随耦合的模块个数增加而增加。

如果只有两个模块有公共环境，那么这种耦合有下面两种可能：

第一，一个模块往公共环境送数据，另一个模块从公共环境取数据，属于数据耦合的一种形式，是比较松散的耦合。

第二，两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。

f．内容耦合

内容耦合是最高程度的耦合，当出现下列情况之一时，就发生了内容耦合。

第一，一个模块访问另一个模块的内部数据；

第二，一个模块不通过正常入口而转到另一个模块的内部；

第三，两个模块有一部分程序代码重叠；

第四，一个模块有多个入口。

③设计原则

耦合是影响软件负责程度的一个重要因素，在设计时应追求尽可能松散耦合的系统，力求做到低耦合，尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境耦合的范围，完全不用内容耦合。

（4）内聚

①定义

内聚标志着一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展，即理想内聚的模块只做一件事情。设计时应该力求做到高内聚，内聚衡量一个模块内部各个元素彼此结合的紧密程度。

②内聚与耦合的关系

a．内聚和耦合是密切相关的，模块内的高内聚往往意味着模块间的松耦合。

b．内聚和耦合都是进行模块化设计的工具，但内聚更重要，应该把更多注意力集中到提高模块的内聚程度上。

③分类

a．低内聚

第一，偶然内聚

如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。有时在写完一个程序之后，发现一组语句在两处或多处出现，于是把这些语句作为一个模块以节省内存，这样就出现了偶然内聚的模块。

第二，逻辑内聚

如果一个模块完成的任务在逻辑上属于相同或相似的一类，则成为逻辑内聚。

第三，时间内聚

如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。时间关系在一定程度上反映了程序的某些实质，所以时间内聚比逻辑内聚好一些。

b．中内聚

第一，过程内聚

如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。使用程序流程图作为工具设计软件时，往往得到的是过程内聚的模块。

第二，通信内聚

如果模块中所有元素都使用同一个输入数据和（或）产生同一个输出数据，则称为通信内聚。

c．高内聚

第一，顺序内聚

如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。根据数据流图划分模块时，通常得到顺序内聚的模块。

第二，功能内聚

如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

四、启发规则

1．改进软件结构提高模块独立性

设计出软件的初步结构后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。

2．模块规模应该适中

（1）过大的模块往往是由于分解不充分，但是进一步分解必须符合问题结构，分解后不应该降低模块独立性。

（2）过小的模块开销大于有效操作，而且模块数目过多将使系统接口复杂。因此过小的模块有时不值得单独存在。

3．深度、宽度、扇出和扇入都应适当

（1）深度

深度表示软件结构中控制的层数，能粗略地标志一个系统的大小和复杂程度。深度和程序长度之间应该有粗略的对应关系，当然这个对应关系是在一定范围内变化的。如果层数过多则应该考虑是否有许多管理模块过分简单了，能否适当合并。

（2）宽度

宽度是软件结构内同一个层次上的模块总数的最大值。宽度越大系统越复杂。对宽度影响最大的因素是模块的扇出。

（3）扇出

扇出是一个模块直接控制的模块数目。在设计时应注意：

①扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块，扇出太大一般是因为缺乏中间层次，应适当增加中间层次的控制模块。

②扇出过小可把下级模块分解成若干个子功能模块，或合并到它的上级模块中去。分解模块或合并模块必须符合问题结构，不能违背模块独立原理。

③一个设计得好的典型系统的平均扇出通常是3或4。

（4）扇入

扇入表明有多少个上级模块直接调用它，扇入越大则共享该模块的上级模块数目越多。但是，不能违背模块独立原理单纯追求高扇入。

设计得很好的软件结构通常顶层扇出比较高，中层扇出较少，底层扇入到公共的实用模块中去（底层模块有高扇入）。

4．模块的作用域应该在控制域之内

（1）定义

①模块的作用域定义为受该模块内一个判定影响的所有模块的集合。

②模块的控制域是这个模块本身以及所有直接或间接从属于它的模块的集合。

（2）规则

在一个设计得很好的系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块。

（3）修改方法

①把做判定的点往上移

②把在作用域内但不在控制域内的模块移到控制域内

到底采用哪种方法改进软件结构，需要根据具体问题统筹考虑。一方面应该考虑哪种方法更现实，另一方面应该使软件结构能最好地体现问题原来的结构。

5．力争降低模块接口的复杂程度

模块接口复杂是软件发生错误的一个主要原因。应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。接口复杂或不一致（即看起来传递的数据之间没有联系）是紧耦合或低内聚的征兆，应该重新分析这个模块的独立性。

6．设计单入口单出口的模块

这条规则警告软件工程师不要使模块间出现内容耦合。当从顶部进入模块并且从底部退出来时，软件是比较容易理解的，因此也是比较容易维护的。

7．模块功能应该可以预测

模块的功能应该能够预测，但也要防止模块功能过分局限。

（1）可预测

如果一个模块可以当做一个黑盒子，即只要输入的数据相同就产生同样的输出，这个模块的功能就是可以预测的。由于内部存储器对于上级模块而言是不可见的，所以这样的模块既不易理解又难于测试和维护。

（2）过分局限

如果一个模块任意限制局部数据结构的大小，过分限制在控制流中可以做出的选择或者外部接口的模式，那么这种模块的功能就过分局限，使用范围也就过分狭窄了。

五、描绘软件结构的图形工具

1．层次图

（1）定义

层次图用来描绘软件的层次结构。层次图中的一个矩形框代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系。图5-1是层次图的一个例子，最顶层的方框代表正文加工系统的主控模块，它调用下层模块完成正文加工的全部功能；第二层的每个模块控制完成正文加工的一个主要功能，第二层的模块又可以调用下一层的模块完成具体的工作。



图5-1 正文加工系统的层次图

（2）适用性

层次图很适于在自顶向下设计软件的过程中使用。通常用层次图作为描绘软件结构的文档。

2.HIPO图

（1）定义

HIPO图是“层次图加输入/处理/输出图”的缩写。使HIPO图具有可追踪性，在层次图里除了最顶层的方框之外，每个方框都加了编号。图5-1加了编号后得到图5-2。

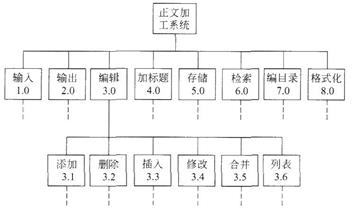


图5-2 带编号的层次图（H图）

（2）特征

①HIPO图和层次图中每个方框相对应，应有一张IPO图描绘这个方框代表的模块的处理过程。

②HIPO图中的每张IPO图内都应该明显地标出它所描绘的模块在H图中的编号。

3．结构图

（1）定义

结构图是进行软件结构设计的一个有力工具，和层次图类似，也是描绘软件结构的图形工具，图5-3、图5-4、图5-5均是结构图的例子。

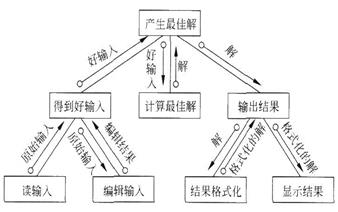


图5-3 产生最佳解的一般结构

①基本符号

a．结构图中一个方框代表一个模，框内注明模块的名字或主要功能；

b．方框之间的箭头（或直线）表示模块的调用关系；

c．尾部是空心圆表示传递的是数据，实心圆表示传递的是控制信息。

②特殊符号

a．表示当模块M中某个判定为真时调用模块A，为假时调用模块B，如图5-4所示；

b．表示模块M循环调用模块A、B和C，如图5-5所示。



图5-4 判定为真时调用A，为假时调用B

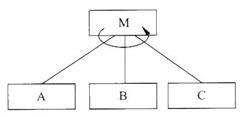


图5-5 模块M循环调用模块A、B、C

（2）与层次图的共性

①层次图和结构图不严格表示模块的调用次序。

②层次图和结构图不指明什么时候调用下层模块。

③层次图和结构图只表明模块调用那些模块，至于模块内有没有其他成分则没有表示。

（3）适用性

利用IPO图或数据字典中的信息得到模块调用时传递的信息，从而由层次图导出结构图的过程，是检查设计正确性和评价模块独立性的好方法。

六、面向数据流的设计方法

1．概念

（1）定义

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。

（2）目标

面向数据流的设计方法的目标是给出设计软件结构的一个系统化的途径。

（3）信息流的类型

①变换流

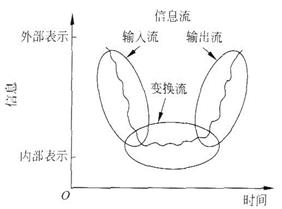


图5-6 变换流

图5-6是一个标准的变换流，变换流具有如下特征：

　 a．信息沿输入通路进入系统，同时由外部形式变换成内部形式；

b．进入系统的信息通过变换中心，经加工处理后沿输出通路变成外部形式离开软件系。

②事务流

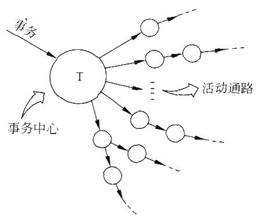


图5-7 事务流

事务流是“以事务为中心的”的数据流，即数据沿输入通路到达一个处理T，这个处理根据输入数据的类型在若干个动作序列中选出一个来执行。图5-7中的处理T称为事务中心，它完成下述任务：

a．接收输入数据；

b．分析每个事务以确定它的类型；

c．根据事务类型选取一条活动通路。

2．设计过程

图5-8说明了使用面向数据流方法逐步设计的过程。

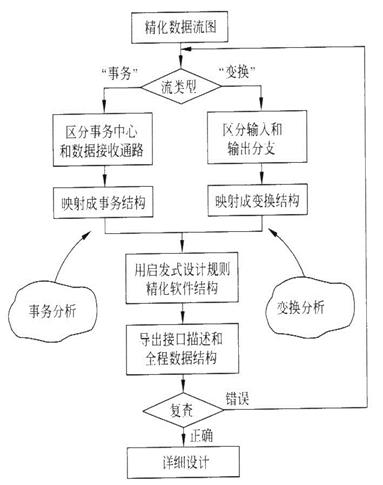


图5-8 面向数据流方法的设计过程

注意：任何设计过程都不是机械的一成不变的。

3．变换分析

（1）例子

假设的仪表板将完成下述功能：

①通过模数转换实现传感器和微处理机接口。

②在发光二极管面板上显示数据。

③指示每小时英里数（mph），行驶的里程，每加仑油行驶的英里数（mpg）等。

④指示加速或减速。

⑤超速警告：如果车速超过55英里/小时，则发出超速警告铃声。

（2）设计步骤

①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。应该对需求分析阶段得出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。

②复查并精化数据流图

a．目的

确保数据流图给出了目标系统的正确的逻辑模型，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。

b．实际操作

假设在需求分析阶段产生的数字仪表板系统的数据流图如图5-9所示。这个数据流图对于软件结构设计的第一次分割而言已经足够详细了，故不需要精化可进行下一个步骤。



图5-9 数字仪表板系统的数据流图

③确定数据流图具有变换特性还是事务特性

a．方法

第一，应该根据数据流图中占优势的属性，确定数据流的全局特性。

第二，还应该把具有和全局特性不同的特点的局部区域孤立出来，以后可以按照这些子数据流的特点精化根据全局特性得出的软件结构。

b．实际操作

从图5-9可以看出，数据沿着两条输入通路进入系统，然后沿着5条通路离开，没有明显的事务中心。因此可以认为这个信息流具有变换流的总特征。

④确定输入流和输出流的边界，从而孤立出变换中心

a．方法

输入流和输出流的边界和对它们的解释有关，即不同设计人员可能会在流内选取稍微不同的点作为边界的位置。

b．实际操作

对于汽车数字仪表板的例子，设计人员确定的流的边界如图5-10所示。

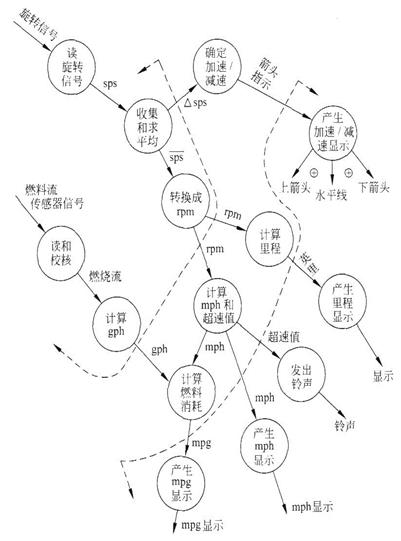


图5-10 具有边界的数据流图

⑤第一级分解

a．定义

软件结构代表对控制的自顶向下的分配，分解就是分配控制的过程。

b．方法

对于变换流的情况，数据流图被映射成一个特殊的软件结构，这个结构控制输入、变换和输出等信息处理过程。

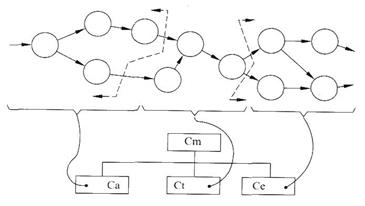


图5-11 第一级分解的方法

图5-11说明了第一级分解的方法。位于软件结构最顶层的控制模块Cm协调下述从属的控制功能：

第一，输入信息处理控制模块Ca，协调对所有输入数据的接收。

第二，变换中心控制模块Ct，管理对内部形式的数据的所有操作。

第三，输出信息处理控制模块Ce，协调输出信息的产生过程。

c．实际操作

对于数字仪表板的例子，第一级分解得出的结构如图5-12所示。每个控制模块的名字表明了为它所控制的那些模块的功能。



图5-12 数字仪表板系统的第一级分解

⑥第二级分解

a．定义

第二级分解是把数据流图中的每个处理映射成软件结构中一个适当的模块。

b．方法

图5-13表示进行第二级分解的普遍途径。

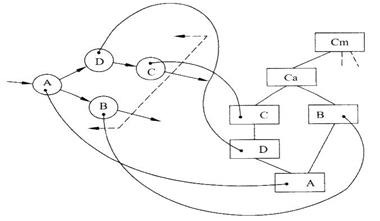


图5-13 第二级分解的方法

第一，从变换中心的边界开始逆着输入通路向外移动，把输入通路中每个处理映射成软件结构中Ca控制下的一个低层模块；

第二，沿输出通路向外移动，把输出通路中每个处理映射成直接或间接受模块Ce控制的一个低层模块；

第三，最后把变换中心内的每个处理映射成受Ct控制的一个模块。

c．实际操作

第一，对于数字仪表板系统的例子，第二级分解的结果分别用图5-14、图5-15和图5-16描绘。

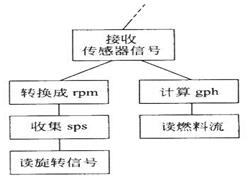


图5-14 未经精化的输入结构

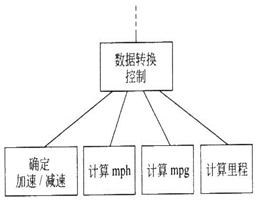


图5-15 未经精化的变换结构



图5-16 未经精化的输出结构

第二，应该为每个模块写一个简要说明，如下：

进出该模块的信息（接口描述）；模块内部的信息；过程陈述，包括主要判定点及任务等；对约束和特殊特点的简短讨论。

⑦使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

a．目的

为了产生合理的分解，得到尽可能高的内聚、尽可能松散的耦合，更为了得到一个易于实现、易于测试和易于维护的软件结构，应该对初步分割得到的模块进行再分解或合并。

b．实际操作

数字仪表板的例子中，经过修改后的软件结构画在图5-17中。修改的步骤如下：

第一，输入结构中的模块“转换成rpm”和“收集sps”可以合并。

第二，模块“确定加速/减速”可以放在模块“计算mph”下面，以减少耦合。

第三，模块“加速/减速显示”可以相应地放在模块“显示mph”的下面。

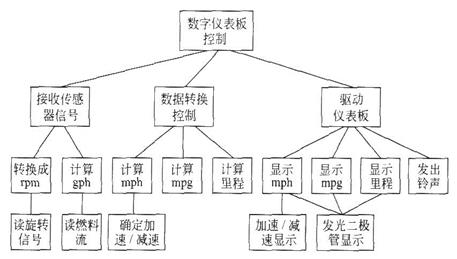


图5-17 精化后的数字仪表板系统的软件结构

4．事务分析

（1）适用性

在数据流具有明显的事务特点时，即有一个明显的事务中心时，采用事务分析方法为宜。

（2）设计

事务分析的设计步骤和变换分析的设计步骤大部分类似，主要差别仅在于由数据流图到软件结构的映射方法不同，如图5-18所示。

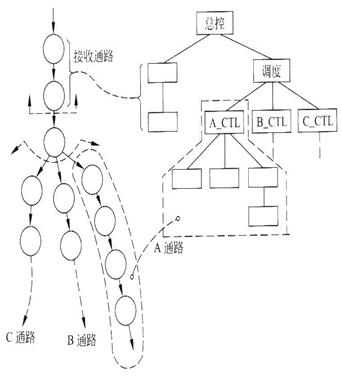


图5-18 事务分析的映射方法

①事务流映射成的软件结构包括一个接收分支和一个发送分支。发送分支的结构包含一个调度模块，它控制下层的所有活动模块。映射接收分支结构的方法是从事务中心边界开始，把沿接收流通路的处理映射成模块。数据流图中的每个活动流通路都应该映射成与它的流特征相对应的结构。

②一般说来，如果数据流不具有显著的事务特点，最好使用变换分析；反之，如果具有明显的事务中心，则应该采用事务分析技术。但是，机械地遵循变换分析或事务分析的映射规则，很可能会得到一些不必要的控制模块，如果它们确实用处不大，那么可以而且应该把它们合并。反之，如果一个控制模块功能过分复杂，则应该分解为两个或多个控制模块，或者增加中间层次的控制模块。

5．设计优化

（1）设计优化时应遵循的原则：

①应该开发能够满足所有功能和性能要求，而且按照设计原理和启发式设计规则衡量是值得接收的软件。

②应该在设计的早期阶段尽量对软件结构进行精化。

③结构简单通常既表示设计风格优雅，又表明效率高。

④对于时间是决定性因素的应用场合，可能需要在详细设计阶段或在编写程序的过程中进行优化。

（2）可以采用以下几种方法对时间起决定性作用的软件进行优化。

①在不考虑时间因素的前提下开发并精化软件结构。

②在详细设计阶段选出最耗费时问的那些模块，仔细地设计它们的处理过程（算法），以求提高效率。

③使用高级程序设计语言编写程序。

④在软件中孤立出那些大量占用处理机资源的模块。

⑤必要时重新设计或用依赖于机器的语言重写上述大量占用资源的模块的代码，以求提高效率。

**5.2　课后习题详解**

1．为每种类型的模块耦合举一个具体例子。

答：（1）数据耦合：计算机网络属于松耦合系统。

（2）控制耦合：遥控器与电器。

（3）公共环境耦合：多机系统。

（4）内容耦合：汇编程序模块。

2．为每种类型的模块内聚举一个具体例子。

答：（1）逻辑内聚

一个子程序将打印季度开支报告、月份开支报告和日开支报告．具体打印哪一个，将由传入的控制标志决定，这个子程序具有逻辑内聚性，因为它的内部逻辑是由输进去的外部控制标志决定的。

（2）时间内聚

若一个模块包含了需要在同一时间段中执行的多个任务，则称该模块的内聚为时间内聚。例如，将多个变量的初始化放在同一个模块中实现，或将需要同时使用的多个库文件的打开操作放在同一个模块中，都会产生时间内聚的模块。

（3）过程内聚

一个子程序，它产生读取雇员的名字，然后是地址，最后是它的电话号码。这种顺序之所以重要，仅仅是因为它符合用户的要求，用户希望按这种顺序进行屏幕输入。另外一个子程序将读取关于雇员的其它信息。这个子程序是过程内聚性，因为是由一个特定顺序而不是其它任何原因，把这些操作组合在一起的。

（4）顺序内聚

一个按给出的生日计算雇员年龄、退休时间的子程序，如果它是利用所计算的年龄来确定雇员将要退休的时间，那么它就具有顺序内聚性。而如果它是分别计算年龄和退休时间的，但使用相同生日数据，那它就只具有通讯内聚性。

（5）功能内聚

计算雇员年龄并给出生日的子程序就是功能内聚性的，因为它只完成一项工作，而且完成得很好。

（6）偶然内聚

一个模块完成一组任务，这些任务彼此间即使有关系，关系也比较松散，就叫做偶然内聚。

3．用面向数据流的方法设计下列系统的软件结构。

（1）储蓄系统。

（2）机票预订系统。

（3）患者监护系统。

答：（1）储蓄系统

①数据流图，如图5-19和5-20所示

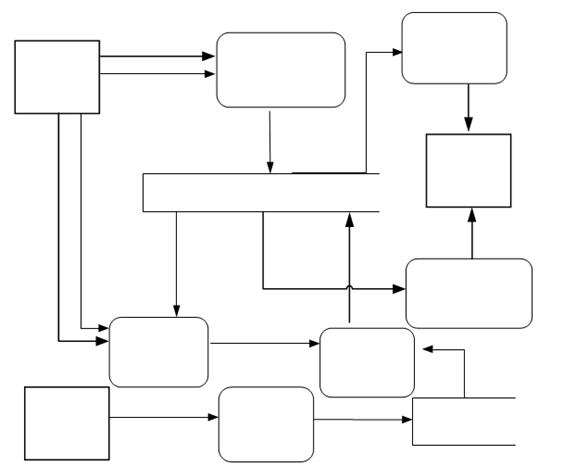


图5-19 储蓄系统数据流图（a）

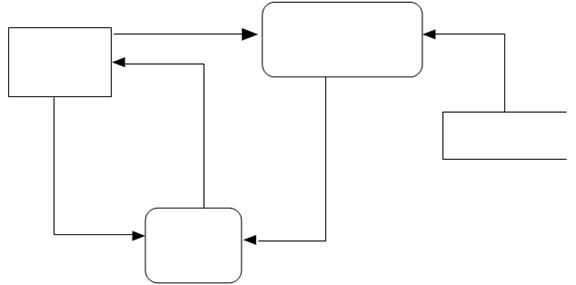


图5-20 储蓄系统数据流图（b）

②软件结构图，如图5-21所示。

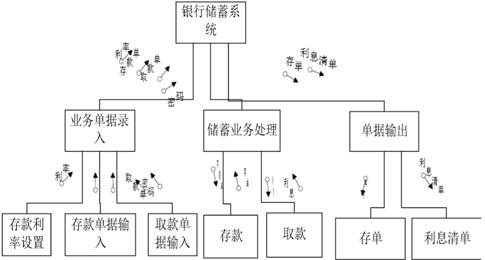


图5-21 储蓄系统软件结构图

（2）机票预定系统

①数据流图，如图5-22、5-23、5-24、5-25、5-26所示。

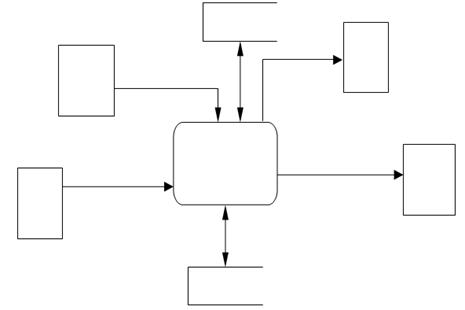


图5-22 机票预定系统数据流图（a）

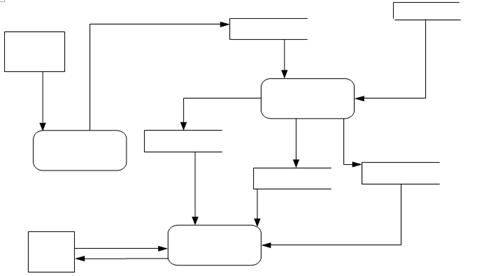


图5-23 机票预定系统数据流图（b）

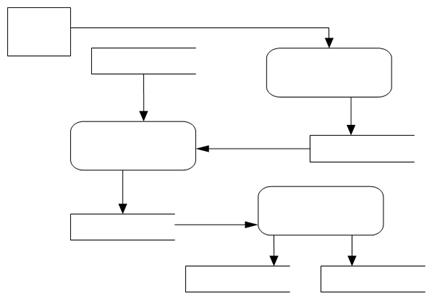


图5-24 机票预定系统数据流图（c）

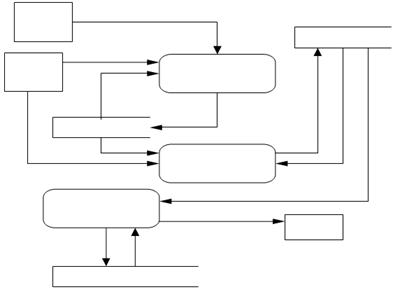


图5-25 机票预定系统数据流图（d）

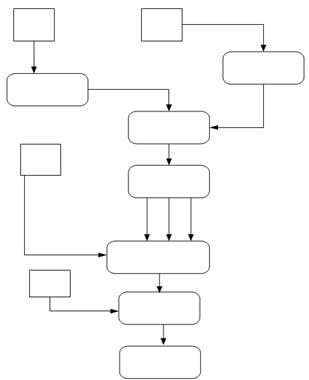


图5-26 机票预定系统数据流图（e）

②软件结构图，如图5-27所示。

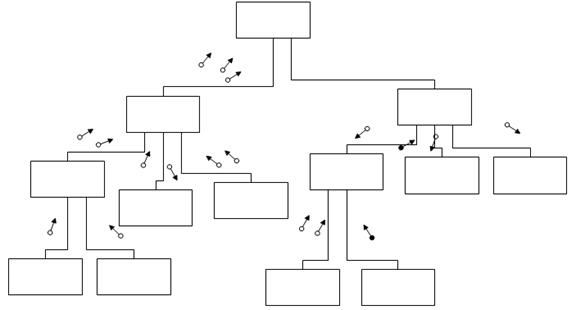


图5-27 机票预定系统软件结构图（f）

（3）患者监护系统

①数据流图，如图5-28所示。

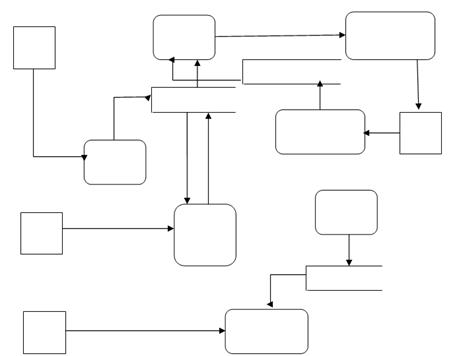


图5-28 患者监护系统数据流图

②软件结构图，如图5-29所示。

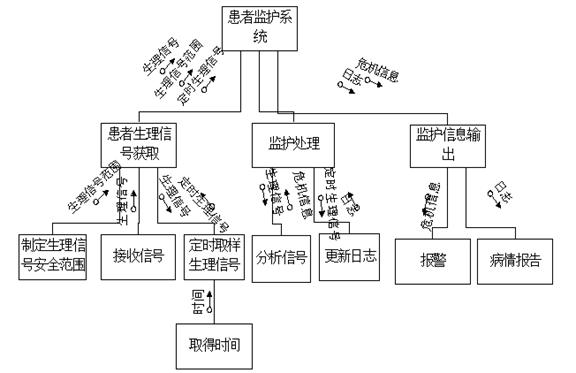


图5-29 患者监护系统软件结构图

4．美国某大学共有200名教师，校方与教师工会刚刚签订一项协议。按照协议，所有年工资超过$26000（含$26000）的教师工资将保持不变，年工资少于$26000的教师将增加工资，所增加的工资数按下述方法计算：给每个由此教师所赡养的人（包括教师本人）每年补助$100，此外，教师有一年工龄每年再多补助$50，但是，增加后的年工资总额不能多于$26000。

教师的工资档案储存在行政办公室的光盘上，档案中有目前的年工资、赡养的人数、雇用日期等信息。需要写一个程序计算并印出每名教师的原有工资和调整后的新工资。要求：

（1）画出此系统的数据流图。

（2）写出需求说明。

（3）设计上述的工资调整程序（要求用HIPO图描绘设计结果），设计时分别采用下述两种算法，并比较这两种算法的优缺点：

（a）搜索工资档案数据，找出年工资少于$26000的人，计算新工资，校核是否超过$26000，储存新工资，印出新旧工资对照表；

（b）把工资档案数据按工资从最低到最高的次序排序，当工资数额超过$26000时即停止排序，计算新工资，校核是否超过限额，储存新工资，印出结果。

（4）所画出的数据流图适用于哪种算法？

答：（1）数据流程图，如图5-30所示。

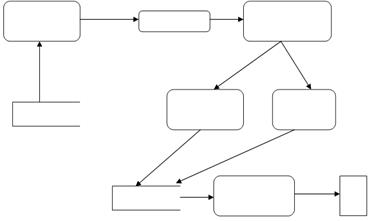


图5-30 工资系统数据流图

（2）需求说明

①任务概述

对于年工资未超过$26000的教师涨工资，即每赡养一人补助$100，每有一年工龄补助$50，涨工资的上限为$26000。

②数据描述

采用工资表、VF

③功能需求

a．档案管理

包括获得教师当前工资、获取赡养人数、获取工龄、更新教师信息。

b．数据计算

包括计算工资、根据限额确定工资。

c．查询管理

包括排序、查询

d．打印报表

打印工资对照单

（3）HIPO图，如图5-31和5-32所示。

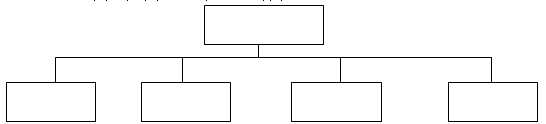


图5-31 工资系统HIPO图（a）

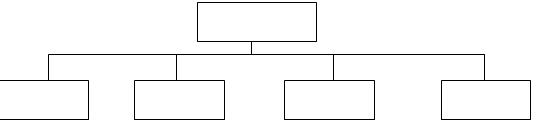


图5-32 工资系统HIPO图（b）

（4）所化的流程图适合（a）种画法

5．下面将给出两个人玩的扑克牌游戏的一种玩法，试设计一个模拟程序，它的基本功能是：

（1）发两手牌（利用随机数产生器）。

（2）确定赢者和赢牌的类型。

（3）模拟N次游戏，计算每种类型牌赢或平局的概率。要求用HIPO图描绘设计结果并且画出高层控制流程图。

扑克牌游戏规则如下：

（1）有两个人玩，分别称为A和B。

（2）一副扑克牌有52张牌，4种花色（方块、梅花、红桃和黑桃），每种花色的牌的点数按升序排列有2，3，4，…，10，J，Q，K，A等13种。

（3）给每个人发3张牌，牌面向上（即，亮牌），赢者立即可以确定。

（4）最高等级的一手牌称为同花，即3张牌均为同一种花色，最大的同花牌是同一种花色的Q、K、A。

（5）第二等级的牌称为顺子，即点数连续的3张牌，最大的顺子是花色不同的Q、K、A。

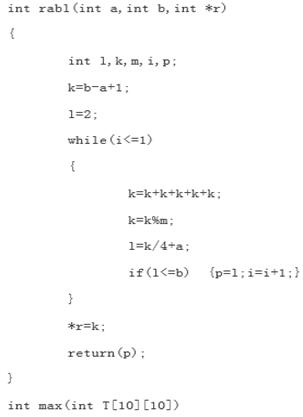
（6）第三等级的牌是同点，即点数相同的3张牌，最大的同点是A、A、A。

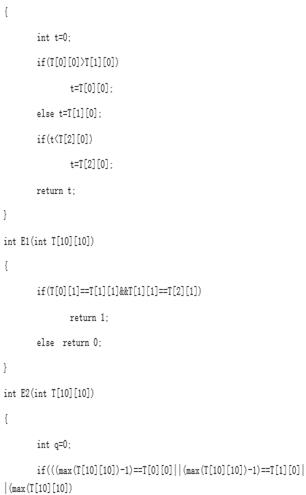
（7）第四等级的牌是对子，即3张牌中有两张点数相同，最大的对子是A、A、K。

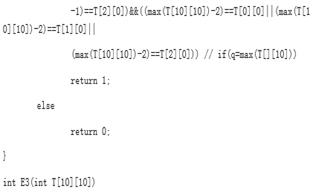
（8）第五等级的牌是杂牌，即除去上列4等之外的任何一手牌，最大的杂牌是不同花色的A、K、J。

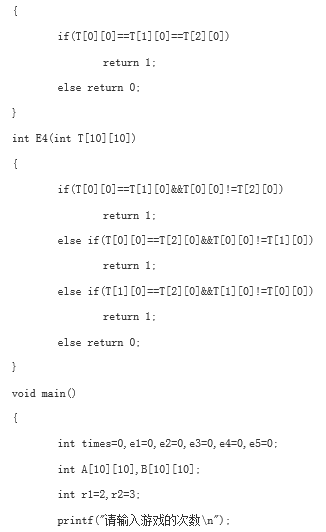
（9）若两人的牌类型不同，则等级高者胜；若等级相同，则点数高者胜；若点数也相同，则为平局。

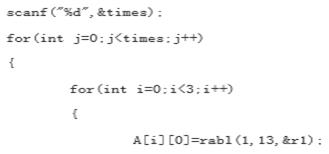
答：（1）程序代码如下：

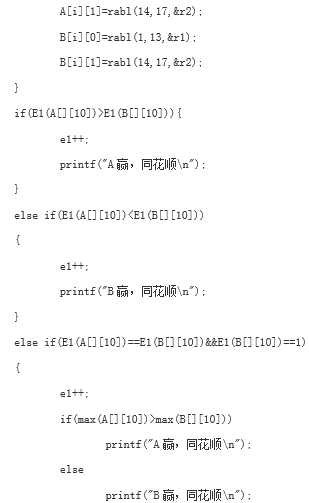


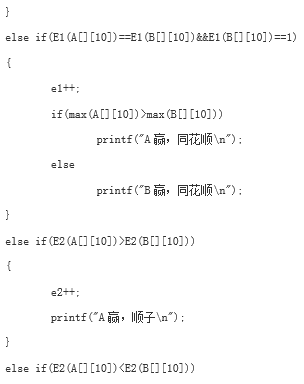


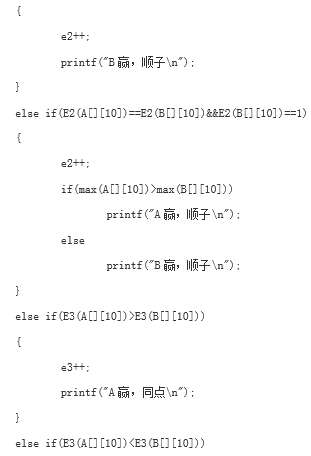


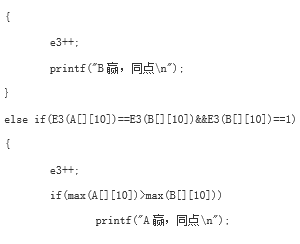


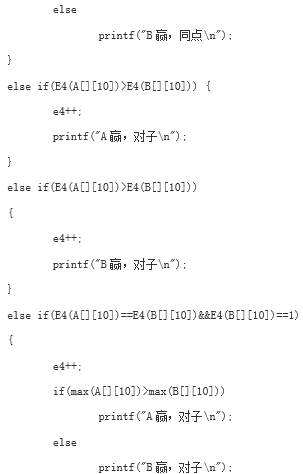


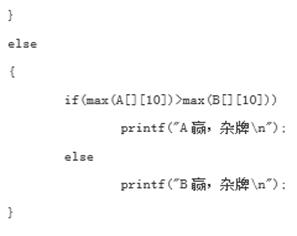


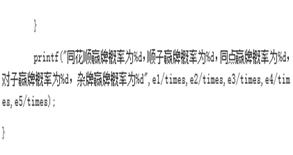












（2）高层控制流程图，如图5-33所示。

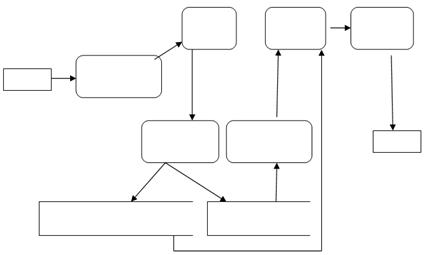


图5-33 高层控制流程图

**第6章　详细设计**

**6.1　复习笔记**

一、详细设计概述

1．目标

（1）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，即经过这个阶段的设计工作，应该得出对目标系统的精确描述，从而在编码阶段可以把这个描述直接翻译成用某种程序设计语言书写的程序。

（2）详细设计的目标不仅仅是逻辑上正确地实现每个模块的功能，更重要的是设计出的处理过程应该尽可能简明易懂。

2．工作任务

详细设计阶段的任务是要设计出程序的“蓝图”，以后程序员将根据这个“蓝图”写出实际的程序代码。

3．关键技术

详细设计的结果基本上决定了最终的程序代码的质量。结构程序设计技术是实现详细设计目标的关键技术，也是详细设计的逻辑基础。

二、结构程序设计

1．控制结构

（1）基本控制结构

三种基本的控制结构是“顺序”、“选择”和“循环”，它们的流程图分别为图6-1（a），6-1（b）和6-1（c）。实际上使用顺序结构和循环结构完全可以实现选择结构，因此，理论上最基本的控制结构只有两种。

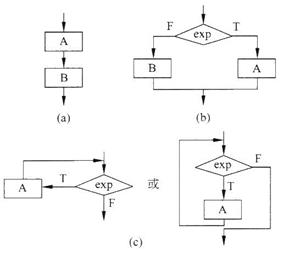


图6-1 三种基本的控制结构

（2）扩展的控制结构

DO UNTIL和DO CASE

DO UNTIL和DO CASE的流程图分别是图6-2（a）和图6-2（b）。

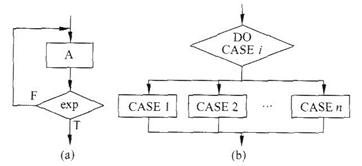


图6-2 其他常用的控制结构

（3）修正的控制结构

LEAVE和BREAK

LEAVE或BREAK结构是受限制的前向GO TO语句，用于转移到循环结构后面的语句。当需要立即从循环（甚至是嵌套的循环）中转移出来时，使用LEAVE或BREAK结构，不仅方便而且会使效率提高很多。

（4）分类

①经典的结构程序设计

只允许使用顺序、IF THEN ELSE型分支和DO WHILE型循环这三种基本控制结构。

②扩展的结构程序设计

除了基本控制结构之外，还允许使用DO CASE型多分支结构和DO UNTIL型循环结构。

③修正的结构程序设计

除了扩展的结构程序设计之外，再允许使用LEAVE（或BREAK）结构。

2．定义

（1）经典定义

如果一个程序的代码块仅仅通过顺序、选择和循环这三种基本控制结构进行连接，并且每个代码块只有一个入口和一个出口，则称这个程序是结构化的。

（2）改进的定义

结构程序设计是尽可能少用GO TO语句的程序设计方法。最好仅在检测出错误时才使用GO TO语句，而且应该总是使用前向GO TO语句。

三、人机界面设计

1．重要性

（1）人机界面设计是接口设计的一个重要的组成部分。对于交互式系统来说，人机界面设计和数据设计、体系结构设计及过程设计一样重要。

（2）人机界面的设计质量，直接影响用户对软件产品的评价，从而影响软件产品的竞争力和寿命。

2．设计问题

在设计人机界面的过程中，几乎总会遇到下述4个问题：系统响应时间、用户帮助设施、出错信息处理和命令交互。

（1）系统响应时间

①定义

系统响应时间指从用户完成某个控制动作，到软件给出预期的响应之间的这段时间。是许多交互式系统用户经常抱怨的问题。

②属性

系统响应时间有两个重要属性，分别是长度和易变性。

a．长度：如果系统响应时间过长，用户就会感到紧张和沮丧。但是，当用户工作速度是由人机界面决定的时候，系统响应时间过短也不好，这会迫使用户加快操作节奏，从而可能会犯错误。

b．易变性：指系统响应时间相对于平均响应时间的偏差，在许多情况下，这是系统响应时间的更重要的属性。即使系统响应时间较长，响应时间易变性低也有助于用户建立起稳定的工作节奏。

（2）用户帮助设施

①意义

交互式系统的每个用户几乎都需要帮助，当遇到复杂问题时需要查看用户手册以寻找答案。通过使用用户帮助设施，使得用户无须离开用户界面就能解决自己的问题。

②分类

常见的帮助设施可分为集成的和附加的两类，普遍认为，集成的帮助设施优于附加的帮助设施。

a．集成的

集成的帮助设施从一开始就设计在软件里面，通常，它对用户工作内容是敏感的，因此用户可以从与刚刚完成的操作有关的主题中选择一个请求帮助。显然，这可以缩短用户获得帮助的时间，增加界面的友好性。集成的帮助设施优于附加的帮助设施

b．附加的

附加的帮助设施是在系统建成后再添加到软件中的，在多数情况下它实际上是一种查询能力有限的联机用户手册。

③设计时需解决的问题

具体设计帮助设施时，必须解决下述的一系列问题：

a．在用户与系统交互期间，是否在任何时候都能获得关于系统任何功能的帮助信息？有两种选择：提供部分功能的帮助信息和提供全部功能的帮助信息。

b．用户怎样请求帮助？有3种选择：帮助菜单，特殊功能键和HELP命令。

c．怎样显示帮助信息？有3种选择：在独立的窗口中，指出参考某个文档（不理想）和在屏幕固定位置显示简短提示。

d．用户怎样返回到正常的交互方式中？有两种选择：屏幕上的返回按钮和功能键。

e．怎样组织帮助信息？有3种选择：平面结构（所有信息都通过关键字访问），信息的层次结构（用户可在该结构中查到更详细的信息）和超文本结构。

（3）出错信息处理

①定义

出错信息和警告信息，是出现问题时交互式系统给出的“坏消息”。出错信息设计得不好，将向用户提供无用的甚至误导的信息，反而会加重用户的挫折感。

②属性

交互式系统给出的出错信息或警告信息，应该具有下述属性：

a．信息应该用用户可以理解的术语描述问题。

b．信息应该提供有助于从错误中恢复的建设性意见。

c．信息应该指出错误可能导致哪些负面后果，以便用户检查是否出现了这些问题，并在确实出现问题时及时解决。

d．信息应该伴随着听觉上或视觉上的提示。

e．信息不能带有指责色彩，也就是说，不能责怪用户。

注意：当确实出现了问题的时候，有效的出错信息能提高交互式系统的质量，减轻用户的挫折感。

（4）命令交互

命令行曾经是用户和系统软件交互的最常用的方式，并且也曾经广泛地用于各种应用软件中。

①设计时需解决的问题

在提供命令交互时，需要考虑的问题如下：

a．是否每个菜单选项都有对应的命令？

b．采用何种命令形式？有3种选择：控制序列（例如，Ctrl+P），功能键和输入命令。

c．学习和记忆命令的难度有多大？忘记了命令怎么办？

d．用户是否可以定制或缩写命令？

②命令宏机制

利用命令宏机制，用户可以用自己定义的名字代表一个常用的命令序列。需要使用这个命令序列时，用户无须依次输入每个命令，只需输入命令宏的名字就可以顺序执行它所代表的全部命令。

3．设计过程

用户界面设计是一个迭代的过程，也就是说，通常先创建设计模型，再用原型实现这个设计模型，并由用户试用和评估，然后根据用户意见进行修改。

（1）设计流程

①创建第一级原型；

②用户试用并评估该原型，直接向设计者表述对界面的评价；

③设计者根据用户意见修改设计并实现下一级原型；

④持续进行上述过程，直到用户感到满意，不需要再修改界面设计时为止。

（2）评估标准

①系统及其界面的规格说明书的长度和复杂程度，预示了用户学习使用该系统所需要的工作量。

②命令或动作的数量、命令的平均参数个数或动作中单个操作的个数，预示了系统的交互时间和总体效率。

③设计模型中包含的动作、命令和系统状态的数量，预示了用户学习使用该系统时需要记忆的内容的多少。

④界面风格、帮助设施和出错处理协议，预示了界面的复杂程度及用户接受该界面的程度。

4．人机界面设计指南

用户界面设计主要依靠设计者的经验，总结众多设计者的经验得出的设计指南，有助于设计者设计出友好、高效的人机界面。人机设计指南主要包括一般交互指南、信息显示指南、数据输入指南。

（1）一般交互指南（全局性的）

一般交互指南涉及信息显示、数据输入和系统整体控制，因此，这类指南是全局性的，忽略它们将承担较大风险。一般的交互指南如下：

①保持一致性

应该为人机界面中的菜单选择、命令输入、数据显示以及众多的其他功能，使用一致的格式。

②提供有意义的反馈

应向用户提供视觉的和听觉的反馈，以保证在用户和系统之间建立双向通信。

③在执行有较大破坏性的动作之前要求用户确认

如果用户要删除一个文件，或覆盖一些重要信息，或终止一个程序的运行，应该给出“您是否确实要……”的信息，以请求用户确认他的命令。

④允许取消绝大多数操作

UNDO或REVERSE功能曾经使众多终端用户避免了大量时间浪费。每个交互式系统都应该能方便地取消已完成的操作。

⑤减少在两次操作之间必须记忆的信息量

不应该期望用户能记住在下一步操作中需使用的一大串数字或标识符。应该尽量减少记忆量。

⑥提高对话、移动和思考的效率

应该尽量减少用户按键的次数，设计屏幕布局时应该考虑尽量减少鼠标移动的距离，应该尽量避免出现用户问“这是什么意思”的情况。

⑦允许犯错误

系统应该能保护自己不受严重错误的破坏。

⑧按功能对动作分类，并据此设计屏幕布局

下拉菜单的一个主要优点就是能按动作类型组织命令。实际上，设计者应该尽力提高命令和动作组织的“内聚性”。

⑨提供对用户工作内容敏感的帮助设施

⑩用简单动词或动词短语作为命令名

过长的命令名难于识别和记忆，也会占用过多的菜单空间。

（2）信息显示指南

如果人机界面显示的信息是不完整的、含糊的或难于理解的。则该应用系统显然不能满足用户的需求。可以用多种不同方式“显示”信息：用文字、图形和声音；按位置、移动和大小；使用颜色、分辨率和省略。信息显示的设计指南如下：

①只显示与当前工作内容有关的信息。

②不要用数据淹没用户，应该用便于用户迅速吸取信息的方式来表示数据。

③使用一致的标记、标准的缩写和可预知的颜色。

④允许用户保持可视化的语境。

⑤产生有意义的出错信息。

⑥使用大小写、缩进和文本分组以帮助理解。

⑦使用窗口分隔不同类型的信息。

⑧使用“模拟”显示方式表示信息，以使信息更容易被用户提取。

⑨高效率地使用显示屏。

（3）数据输入指南

用户的大部分时间用在选择命令、输入数据和向系统提供输入。在许多应用系统中，键盘仍然是主要的输入介质，但是，鼠标、数字化仪和语音识别系统正迅速地成为重要的输入手段。数据输入指南如下：

①尽量减少用户的输入动作。

②保持信息显示和数据输入之间的一致性。

③允许用户自定义输入。

④交互应该是灵活的，并且可调整成用户最喜欢的输入方式。

⑤使在当前动作语境中不适用的命令不起作用。

⑥让用户控制交互流。

⑦对所有输入动作都提供帮助。

⑧消除冗余的输入。

四、过程设计的工具

1．定义

描述程序处理过程的工具称为过程设计的工具，它们可以分为图形、表格和语言3类。基本要求是能提供对设计的无歧义的描述，即应该能指明控制流程、处理功能、数据组织以及其他方面的实现细节，从而在编码阶段能把对设计的描述直接翻译成程序代码。

2．分类

（1）程序流程图

①符号表示

程序流程图又称为程序框图，它是历史最悠久、使用最广泛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。程序流程图中使用的符号表示如图6-3所示。

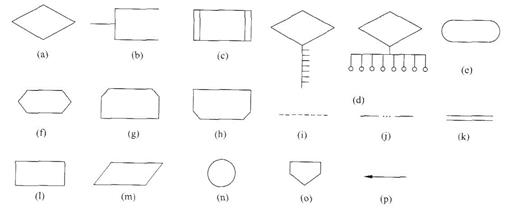


图6-3 程序流程图中使用的符号

各个符号代表的含义如下：

（a）选择（分支）；

（b）注释；

（c）预先定义的处理；

（d）多分支；

（e）开始或停止；

（f）准备；

（g）循环上界限；

（h）循环下界限；

（i）虚线；

（j）省略符；

（k）并行方式；

（1）处理；

（m）输入输出；

（n）连接；

（o）换页连接；

（p）控制流。

②优点

对控制流程的描绘很直观，便于初学者掌握。

③缺点

a．程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构。

b．程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制。

c．程序流程图不易表示数据结构。

（2）盒图（N-S图）

①特点

出于要有一种不允许违背结构程序设计精神的图形工具的考虑，Nassi和Shneiderman提出了盒图，又称为N-S图。它有下述特点：

a．功能域明确，可以从盒图上一眼就看出来；

b．不可能任意转移控制；

c．很容易确定局部和全程数据的作用域；

d．很容易表现嵌套关系，也可以表示模块的层次结构。

②表示

图6-4给出了结构化控制结构的盒图表示，也给出了调用子程序的盒图表示方法。

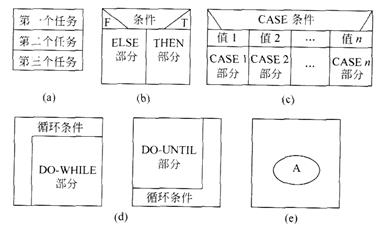


图6-4 盒图的基本符号

其中各个图形代表的人结构如下：

（a）顺序结构；

（b）IF\_THEN\_ELSE型分支；

（c）CASE型多分支；

（d）循环结构；

（e）调用子程序A。

③优点

盒图没有箭头，因此不允许随意转移控制。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。

（3）PAD图

①符号表示

PAD是问题分析图（problem analysis diagram）的英文缩写，自1973年由日本日立公司发明以后，已得到一定程度的推广。它用二维树形结构的图来表示程序的控制流，将这种图翻译成程序代码比较容易。图6-5给出PAD图的基本符号。

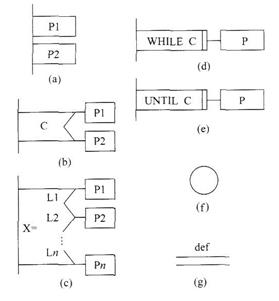


图6-5 PAD图的基本符号

各个符号代表的含义如下：

（a）顺序（先执行Pl后执行P2）；

（b）选择（IF C THEN Pl ELSE P2）；

（c）CASE型多分支；

（d）WHILE型循环（WHILE C DO P）；

（e）UNTIL型循环（REPEAT P UNTIL C）；

（f）语句标号；

（g）定义。

②优点

a．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。

b．PAD图所描绘的程序结构十分清晰。图中最左面的竖线是程序的主线，即第一层结构。随着程序层次的增加，PAD图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线。PAD图中竖线的总条数就是程序的层次数。

c．用PAD图表现程序逻辑，易读、易懂、易记。PAD图是二维树形结构的图形，程序从图中最左竖线上端的结点开始执行，自上而下，从左向右顺序执行，遍历所有结点。

d．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，从而可省去人工编码的工作，有利于提高软件可靠性和软件生产率。

e．即可用于表示程序逻辑，也可用于描绘数据结构。

f．PAD图的符号支持自顶向下、逐步求精方法的使用。开始时设计者可以定义一个抽象的程序，随着设计工作的深入而使用def符号逐步增加细节，直至完成详细设计（如图6-6所示，图6-6（a）表示初始的PAD图；6-6（b）表示使用def符号细化处理框P2）。

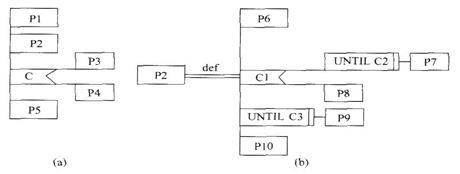


图6-6 使用PAD图提供的定义功能来逐步求精的例子

（4）判定表

①适用性

算法中包含多重嵌套的条件选择时，判定表能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。

②组成

一张判定表由四部分组成：

a．左上部列出所有条件；

b．左下部是所有可能做的动作；

c．右上部是表示各种条件组合的一个矩阵；

d．右下部是和每种条件组合相对应的动作。

注意：判定表右半部的每一列实质上是一条规则，规定了与特定的条件组合相对应的动作。

③实例分析

假设某航空公司规定，乘客可以免费托运重量不超过30kg的行李。当行李重量超过30kg时，对头等舱的国内乘客超重部分每公斤收费4元，对其他舱的国内乘客超重部分每公斤收费6元，对外国乘客超重部分每公斤收费比国内乘客多一倍，对残疾乘客超重部分每公斤收费比正常乘客少一半。

表6-1 用判定表表示计算行李费的算法



【解析】如表6-1所示。在表的右上部分中T表示它左边那个条件成立，F表示条件不成立，空白表示这个条件成立与否并不影响对动作的选择。判定表右下部分中画×表示做它左边的那项动作，空白表示不做这项动作。只要行李重量不超过30kg，不论这位乘客持有何种机票，是中国人还是外国人，是残疾人还是正常人，一律免收行李费，这就是表右部第一列（规则l）表示的内容。当行李重量超过30kg时，根据乘客机票的等级、乘客国籍及是否残疾人而使用不同算法计算行李费，这就是从规则2到规则9所表示的内容。

④优点

a．判定表能够简洁而又无歧义地描述处理规则。

b．判定表和布尔代数或卡诺图结合起来使用，可以更加直观、简洁、清晰的描述规则。

⑤缺点

a．不能同时清晰地表示出问题的顺序性和重复性。

b．初次接触这种工具的人理解它需要有一个学习过程。

c．数据元素增多时，判定表的简洁程度大幅下降。

（5）判定树

判定树是判定表的变种，是一种比较常用的系统分析和设计的工具，它的优缺点如下：

①优点

a．判定树能清晰地表示复杂的条件组合与应做的动作之间的对应关系。

b．判定树的形式简单到不需任何说明，一眼就能看出其含义，易于掌握和使用。

②缺点

a．简洁性差

如图6-7所示，数据元素的同一个值往往要重复写多遍，越接近树的叶端重复次数越多。

b．画判定树时分枝的次序可能对最终画出的判定树的简洁程度有较大影响。

如图6-7所示，如果不是把行李重量做为第一个分枝，而是将它作为最后一个分枝，则画出的判定树将有16片树叶而不是只有9片树叶。

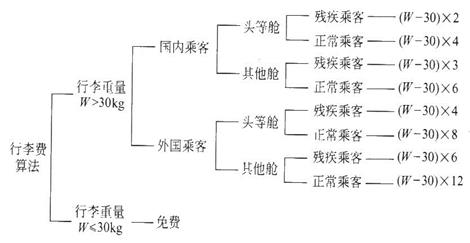


图6-7 用判定树表示计算行李费的算法

（6）过程设计语言

①定义

过程设计语言（PDL），即伪码，它是用正文形式表示数据和处理过程的设计工具。PDL具有严格的关键字外部语法，用于定义控制结构和数据结构。PDL表示实际操作和条件的内部语法通常又是灵活自由的，可以适应各种工程项目的需要。

②特点

a．关键字的固定语法，它提供了结构化控制结构、数据说明和模块化的特点。

b．自然语言的自由语法，它描述处理特点。

c．数据说明的手段。应该既包括简单的数据结构，又包括复杂的数据结构表或层次的。

d．模块定义和调用的技术，应该提供各种接口描述模式。

③优点

a．可作为注释直接插在源程序中。有助于保持文档和程序的一致性，提高文档的质量。

b．可以使用普通的正文编辑程序或文字处理系统，很方便地完成PDL的书写和编辑工作。

c．已经有自动处理PDL的程序存在，且可以自动由PDL生成程序代码。

④缺点

不如图形工具形象直观，描述复杂的条件组合与动作间的对应关系时，较复杂，不如判定表清晰简单。

五、面向数据结构的设计方法

1．相关概念

（1）目的

面向数据结构的设计方法的最终目标是得出对程序处理过程的描述。

（2）适用性

面向数据结构的设计方法最适合于在详细设计阶段使用，即在完成了软件结构设计之后，可以使用面向数据结构的方法来设计每个模块的处理过程。

2.Jackson图

（1）分类

Jackson方法和Warnier方法是最著名的两个面向数据结构的设计方法，Jackson图根据程序中数据元素彼此间的逻辑关系可分为顺序结构、选择结构和重复结构3类，具体如下：

①顺序结构

顺序结构的数据由一个或多个数据元素组成，每个元素按确定次序出现一次。图6-8是表示顺序结构的Jackson图的一个例子。

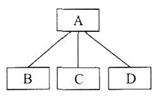


图6-8 A由B、C、D3个元素顺序组成

注意：图中每个元素只出现一次，出现的次序依次是B、C和D。

②选择结构

选择结构的数据包含两个或多个数据元素，每次使用这个数据时按一定条件从这些数据元素中选择一个。图6-9是表示3个中选1个结构的Jackson图。

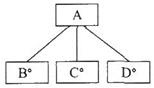


图6-9 根据条件A是B或C或D中的某一个

注意：在B、C和D的右上角有小圆圈做标记。

③重复结构

重复结构的数据，根据使用时的条件由一个数据元素出现零次或多次构成。图6-10是表示重复结构的Jackson图。

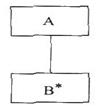


图6-10 A由B出现N次（N≥O）组成

注意：在B的右上角有星号标记。

（2）优点

Jackson图有下述优点：

①便于表示层次结构，而且是对结构进行自顶向下分解的有力工具。

②形象直观可读性好。

③既能表示数据结构也能表示程序结构。

（3）缺点

①用这种图形工具表示选择或重复结构时，选择条件或循环结束条件不能直接在图上表示出来，影响了图的表达能力，也不易直接把图翻译成程序。

②框间连线为斜线，不易在行式打印机上输出。

3．改进的Jackson图

（1）表示

图6-11中给出了改进的Jackson图。

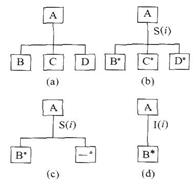


图6-11 改进的Jackson图

①顺序结构

如图6-11（a）所示，B、C、D中任一个都不能是选择出现或重复出现的数据元素。

②选择结构

如图6-11（b）所示，S右面括号中的数字i是分支条件的编号。

③可选结构

如图6-11（c）所示，A或者是元素B或者不出现。

④重复结构

如图6-11（d）所示，循环结束条件的编号为i。

（2）与层次图的比较

①层次图中的一个方框代表一个模块；而Jackson图即使在描绘程序结构时，一个方框也并不代表一个模块。

②层次图表现的是调用关系，通常一个模块除了调用下级模块外，还完成其他操作；而Jackson图表现的是组成关系。

4.Jackson方法

（1）设计步骤

Jackson结构程序设计方法基本上由下述5个步骤组成：

①分析并确定输入数据和输出数据的逻辑结构，并用Jackson图描绘这些数据结构。

②找出输入数据结构和输出数据结构中有对应关系的数据单元。所谓有对应关系是指有直接的因果关系，在程序中可以同时处理的数据单元。

③用下述3条规则从描绘数据结构的Jackson图导出描绘程序结构的Jackson图。

a．为每对有对应关系的数据单元，按照它们在数据结构图中的层次在程序结构图的相应层次画一个处理框。

b．根据输入数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。

c．根据输出数据结构中剩余的每个数据单元所处的层次，在程序结构图的相应层次分别为它们画上对应的处理框。

注意：描绘程序结构的Jackson图应该综合输入数据结构和输出数据结构的层次关系而导出来。在导出程序结构图的过程中，由于改进的Jackson图规定在构成顺序结构的元素中不能有重复出现或选择出现的元素，因此可能需要增加中间层次的处理框。

④列出所有操作和条件，并且把它们分配到程序结构图的适当位置。

⑤用伪码表示程序（与Jackson图完全对应）。

（2）伪码

Jackson方法中使用的伪码和Jackson图是完全对应的，顺序、选择、重复结构对应的伪码如下：

①图6-11（a）所示的顺序结构对应的伪码如下，其中seq和end是关键字：

A seq

B

C

D

A end

②图6-11（b）所示的选择结构对应的伪码如下，其中select、or和end是关键字，cond1、cond2和cond3分别是执行B、C或D的条件：

A select cond1

B

A or cond2

C

A or cond3

D

A end

③图6-11（d）所示的重复结构对应的伪码如下，其中iter、until、while和end是关键字，cond是条件：

A iter until（或while）cond

B

A end

（3）实例分析

一个正文文件由若干个记录组成，每个记录是一个字符串。要求统计每个记录中空格字符的个数，以及文件中空格字符的总个数。要求的输出数据格式是，每复制一行输入字符串之后，另起一行印出这个字符串中的空格数，最后印出文件中空格的总个数。

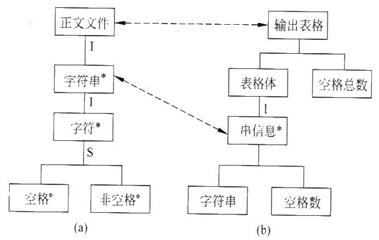


图6-12 表示输入输出数据结构的Jackson图

（a）输入数据结构；（b）输出数据结构

【解析】

①确定输入和输出数据的结构，如图6-12所示

②分析确定在输入数据结构和输出数据结构中有对应关系的数据单元

a．输出数据总是通过对输入数据的处理而得到的，因此在输入输出数据结构最高层次的两个单元（在这个例子中是“正文文件”和“输出表格”）总是有对应关系的。这一对单元将和程序结构图中最顶层的方框相对应。

b．每处理输入数据中一个“字符串”之后，就可以得到输出数据中一个“串信息”，它们都是重复出现的数据单元，而且出现次序和重复次数都完全相同，故“字符串”和“串信息”也是一对有对应关系的单元。

c．“字符”不可能和多个字符组成的“字符串”对应，和输出数据结构中其他数据单元也不能对应。单个空格并不能决定一个记录中包含的空格个数，因此没有对应关系。输入数据结构中余下的任何一个单元在输出数据结构中都找不到对应的单元。

d．在图6-12中用一对虚线箭头把有对应关系的数据单元连接起来，以突出表明这种对应关系。

③从数据结构图导出程序结构图

a．“统计空格”是在程序结构图的最顶层与“正文文件”和“输出表格”对应的处理框。

b．“程序体”和“印总数”是在程序结构图的第二层与“表格体”和“空格总数”对应的处理框。

c．“处理字符串”是在程序结构图的第三层与“字符串”和“串信息”对应的处理框。

d．“印字符串”、“分析字符”及“印空格数”在程序结构图的第四层与“字符串”、“字符”及“空格数”对应的处理框。由于“字符”是重复出现的数据单元，因此“分析字符”也应该是重复执行的处理。在“分析字符”这个处理框上面又增加了一个处理框“分析字符串”。

e．最后得到的程序结构图为图6-13所示。

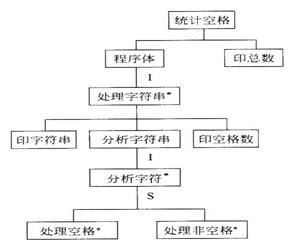


图6-13 描绘统计空格程序结构的Jackson图

④列出所有操作和条件，并且把它们分配到程序结构图的适当位置

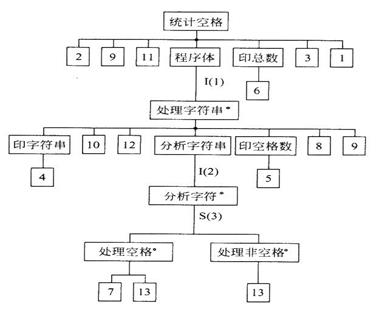


图6-14 把操作和条件分配到程序结构图的适当位置

把需要的操作和条件分配到程序结构图的适当位置，结果为图6-14。统计空格个数需要的全部操作和条件如下：

a．停止；

b．打开文件；

c．关闭文件；

d．印出字符串；

e．印出空格数目；

f．印出空格总数；

g．sum：=sum+1；

h．totalsum=totalsum+sum；

i．读入字符串；

g．sum：=0；

k．totalsum：=0；

l．pointer：=l；

m．pointer=pointer+11；

n．文件结束；

o．字符串结束；

p．字符是空格。

⑤用伪码表示程序处理过程

统计空格

打开文件

读入字符串

totalsum：=0

程序体 iter until 文件结束

处理字符串seq

印字符串 seq

　 印出字符串

印字符串end

sum：=0

pointer：=1

分析字符串 iter until 字符串结束

　 分析字符select字符是空格

　 处理空格seq

sum：=sum+1

pointer：=pointer+1

　 处理空格end

　 分析字符or字符不是空格

　处理非空格seq

pointer：=pointer+1

处理非空格end

分析字符end

　 分析字符串end

　 印空铬数seq

　 印出空格数目

　 印空格数end

　 totalsum：=totalsum+sum

　 读入字符串

处理字符串end

\　 程序体end

　 印总数seq

印出空格总数

　 印总数end

　 关闭文件

　 停止

统计空格end

六、程序复杂程度的定量度量

1．价值

（1）把程序的复杂程度乘以适当数可估算出软件中错误的数量以及开发需要的工作量。

（2）定量度量的结果可以用来比较两个不同的设计或两个不同算法的优劣。

（3）程序的定量的复杂程度可以作为模块规模的精确限度。

2.McCabe方法

（1）流图

①定义

流图实质上是“退化了的”程序流程图，它仅仅仅描绘程序的控制流程，完全不表现对数据具体操作以及分支或循环的具体条件，流图通常被用来突出表示程序的控制流。

②程序流程图映射成流图

图6-15说明了把程序流程图映射成流图的方法。

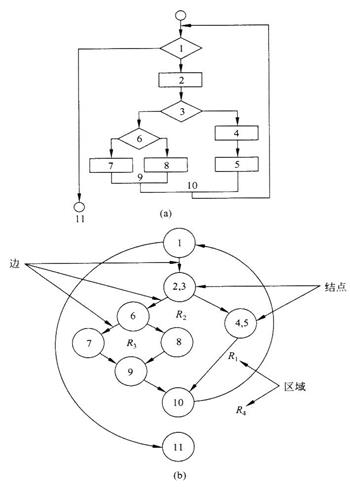


图6-15 把程序流程图映射成流图

（a）程序流程图；（b）流图

a．流图中用圆表示结点，一个圆代表一条或多条语句。程序流程图中的一个顺序的处理框序列和一个菱形判定框，可以映射成流图中的一个结点。

b．流图中的箭头线称为边，代表控制流。流图中一条边必须终止于一个结点，即使这个结点并不代表任何语句。

c．由边和结点围成的面积称为区域，计算区域数时应包括图外部未被围起来的区域。

③PDL翻译成流图

a．基本情况

图6-16是用PDL表示的处理过程及与之对应的流图。

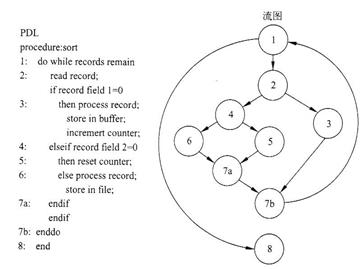


图6-16 由PDL翻译成的流图

b．包含复合条件

复合条件，就是在条件中包含了一个或多个布尔运算符。在这种情况下，应把复合条件分解为若干个简单条件，每个简单条件对应流图中一个结点。包含条件的结点称为判定节点，从每个判定结点引出两条或多条边。图6-17是由包含复合条件的PDL片断翻译成的流图。

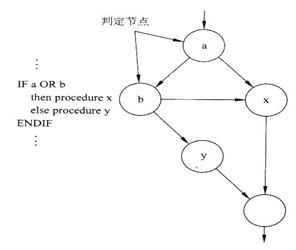


图6-17 由包含复合条件的PDL映射成的流图

（2）环形复杂度

①定义

McCabe方法根据程序控制流的复杂程度定量度量程序的复杂程度，这样度量出的结果称为程序的环形复杂度。

②计算方法

环形复杂度定量度量程序的逻辑复杂度，可以用下述3种方法中的任何一种来计算环形复杂度：

a．流图中线性无关的区域数等于环形复杂度。

b．流图G的环形复杂度V（G）=E-N+2，其中，E是流图中边的条数，N是结点数。

c．流图G的环形复杂度V（G）=P+1，其中，P是流图中判定结点的数目。

③环形复杂度的用途

程序的环形复杂度取决于程序控制流的复杂程度，即取决于程序结构的复杂程度。当程序内分支数或循环个数增加时，环形复杂度也随之增加，因此它是对测试难度的一种定量度量，也能对软件最终的可靠性给出某种预测。

④环形复杂度的规模

环形复杂度高的程序往往是最困难、最容易出问题的程序。模块规模以V（G）≤10为宜，即V(G)=10是模块规模的一个科学精确的上限。

3.Halstead方法

（1）定义

Halstead方法是根据程序中运算符和操作数的总数来度量程序的复杂程度。

（2）方法

①程序长度N定义（N1为程序中运算符出现的总次数，N2为操作数出现的总次数）：

N=N1+N2

②预测程序长度的公式（使用的不同运算符的个数n1，不同操作数的个数n2）：

H=n1log2n1+n2log2n2

③预测程序中包含错误的个数的公式：

E=N log2（n1+n2）/3000

**6.2　课后习题详解**

1．假设只有SEQUENCE和DO-WHILE两种控制结构，怎样利用它们完成 IF THEN ELSE操作？

答：转化如下：

　 K=1

　 DO WHILE （条件 AND K EQ 1）

程序块1

K=K+1

　 END DO

　 DO WHILE （（NOT条件）AND K EQ 1）

程序块2

K=K+1

　 END DO

2．假设只允许使用SEQUENCE和IF-THEN-ELSE两种控制结构，怎样利用它们完成DO WHILE操作？

答：转化如下：

　 label:　 IF（条件）THEN

　 程序块

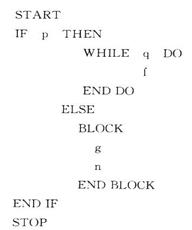
　 GOTO label

ELSE

　 程序块

END IF

3．画出下列伪码程序的程序流程图和盒图：



答：（1）流程图如图6-18所示。

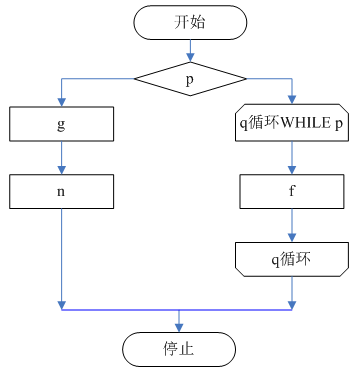


图6-18 从伪码变成的程序流程图

（2）该程序的盒图如图6-19所示。

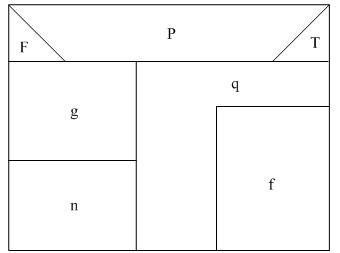


图6-19 从伪码变成的程序盒图

4．图6-20给出的程序流程图代表一个非结构化的程序，问：



图6-20 一个非结构化程序

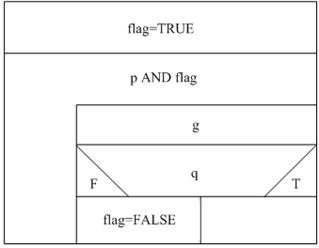
（1）为什么说它是非结构化的？

（2）设计一个等价的结构化程序。

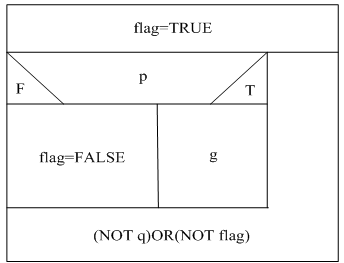
（3）在（2）题的设计中使用附加的标志变量flag了吗？若没用，再设计一个使用flag的程序；若用了，再设计一个不用flag的程序。

答：（1）通常所说的结构化程序，是按照狭义的结构程序的定义衡量，符合定义规定的程序。图示的程序的循环控制结构有两个出口，显然不符合狭义的结构程序的定义，因此是非结构化的程序。

（2）使用附加的标志变量flag，至少有两种方法可以把该程序改造为等价的结构化程序，图6-21描绘了等价的结构化程序的盒图。



（a）解法1



（b）解法2

图6-21 与该图等价的结构化程序（用flag）

（3）不使用flag把该程序改造为等价的结构化程序的方法如图6-22所示。

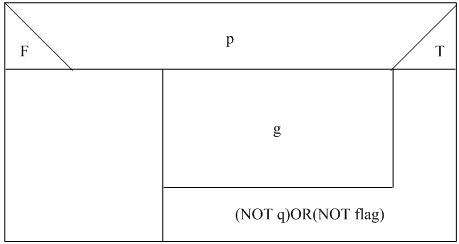
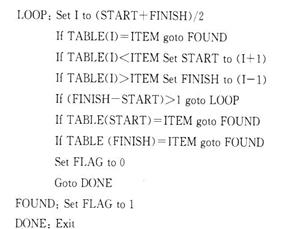


图6-22 与该图等价的结构化程序（不用flag）

5．研究下面的伪码程序：



要求：

（1）画出程序流程图。

（2）程序是结构化的吗？说明理由。

（3）若程序是非结构化的，设计一个等价的结构化程序并且画出程序流程图。

（4）此程序的功能是什么？它完成预定功能有什么隐含的前提条件吗？

答：（1）该程序流程图如图6-23所示。

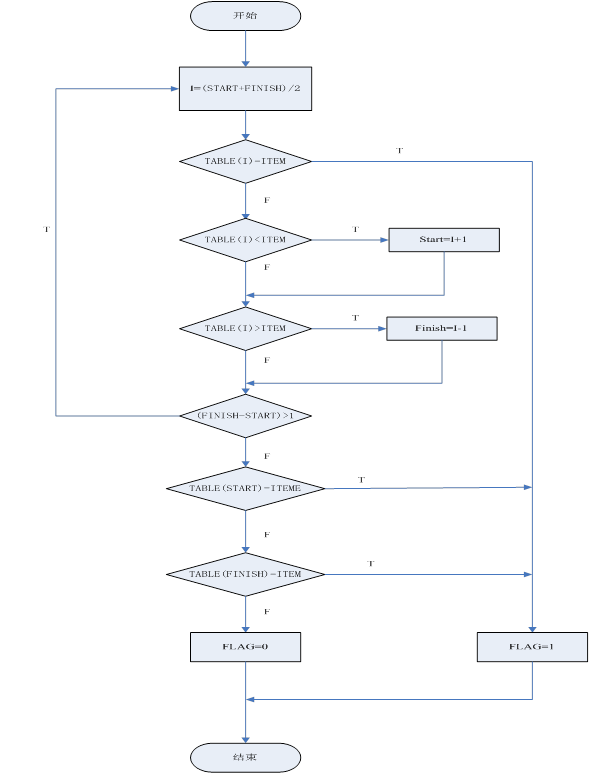


图6-23 程序流程图（a）

（2）该程序不是结构化的，结构化的程序只有一个入口和一个出口，而该程序的流程途中有两个出口。

（3）等价的结构化程序的流程图如图6-24所示。

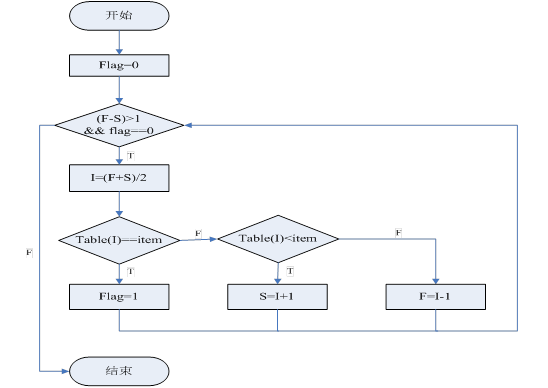


图6-24 程序流程图（b）

（4）此程序有二分查找的功能，它完成预定功能的隐含前提条件是现有序列为从小到大顺序排好的有序序列。

6．用Ashcroft-Manna技术可以将非结构化的程序转换为结构化程序，图6-25是一个转换的例子。

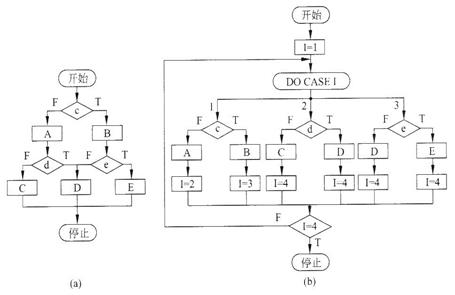


图6-25用 Ashcroft Manna技术的例子

（a）非结构化设计；（b）转换成的等价的结构化设计

（1）能否从这个例子总结出Ashcroft Manna技术的一些基本方法？

（2）进一步简化图6-25（b）给出的结构化设计。

答：（1）从这个例子中看出，Ashcroft-Manna技术的基本方法是，当待改造的序含有嵌套的非结构化的IF语句时，改造后的程序中增加DO-CASE语句和DO-UNTIL语句，并增加一个辅助变量I，I的初始值为1。最外层的IF语句在I=1时执行，执行完这个IF语句后把I赋值为随后应该执行的内层IF语句所对应的CASE标号值。DO-CASE语句的最大分支数（可执行的最大标号值）等于IF语句的个数。当执行完最内层的IF语句之后，把I赋值为可执行的最大标号值加1，而DO-UNTIL 循环的结束条件就是I等于这个值。

（2）进一步简化后的结构化程序的流程图如图6-26下所示。

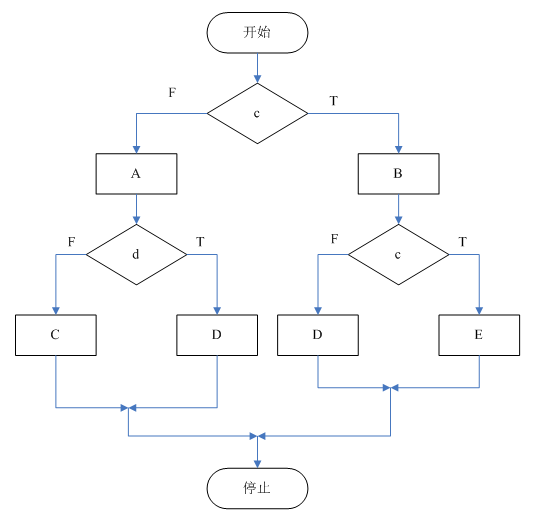


图6-26 进一步简化后的结构化程序

7．某交易所规定给经纪人的手续费计算方法如下：总手续费等于基本手续费加上与交易中的每股价格和股数有关的附加手续费。如果交易总金额少于1000元，则基本手续费为交易金额的8.4％；如果交易总金额在1000元～l0000元之间，则基本手续费为交易金额的5％，再加34元；如果交易总金额超过10000元，则基本手续费为交易金额的4％加上134元。当每股售价低于14元时，附加手续费为基本手续费的5％，除非买进、卖出的股数不是100的倍数，在这种情况下附加手续费为基本手续费的9％。当每股售价在14元到25元之间时，附加手续费为基本手续费的2％，除非交易的股数不是100的倍数，在这种情况下附加手续费为基本手续费的6％。当每股售价超过25元时，如果交易的股数零散（即不是100的倍数），则附加手续费为基本手续费的4％，否则附加手续费为基本手续费的1％。

要求：

（1）用判定表表示手续费的计算方法。

（2）用判定树表示手续费的计算方法。

答：（1）判定表如图6-27所示。

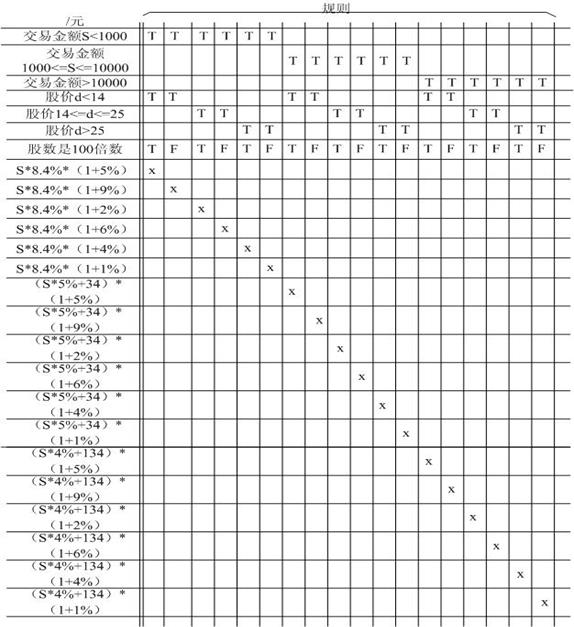


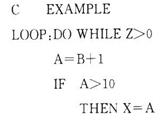
图6-27 判定表

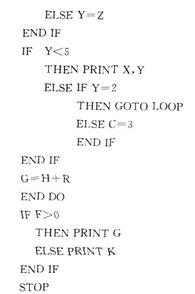
（2）判定树如图6-28所示。



图6-28 判定树

8．画出下列伪码程序的流图，计算它的环形复杂度。这个程序的逻辑有什么问题吗？





答：（1）先画出图示。

①该伪码的流程图如图6-29所示。

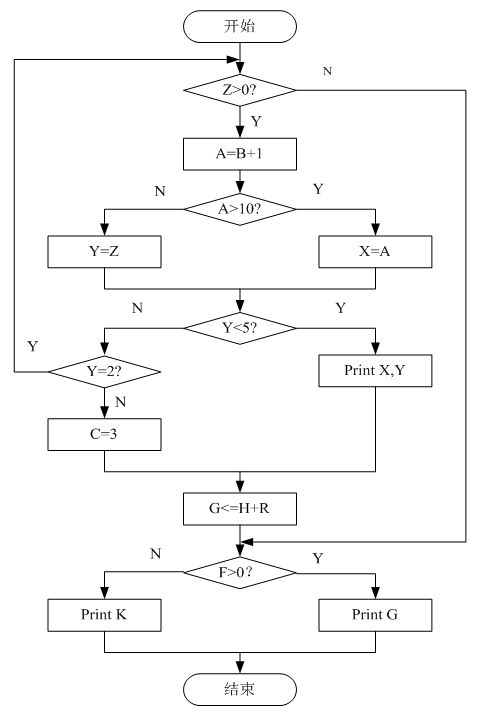


图6-29 程序流程图

②根据该程序流程图，可得该程序流图如图6-30所示。

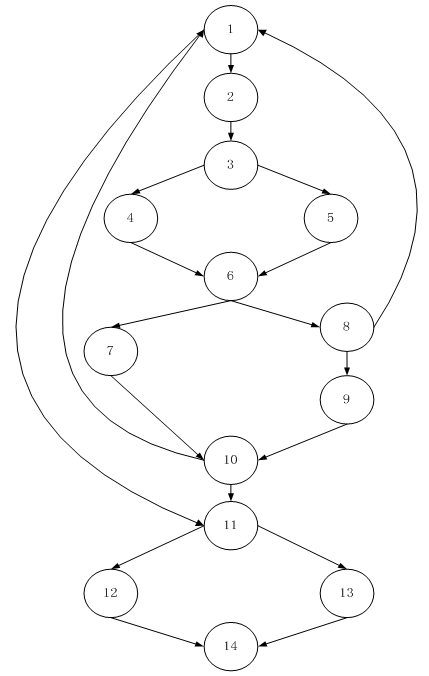


图6-30 程序流图

（2）计算

环形复杂度V（G）=流图中的区域数=流图中判定结点数目+1=5+1=6

（3）分析

该算法问题在于控制最外层循环的变量Z不仅没有经过初始化，并且在该循环内部没有任何有可能该变Z的语句。因此，该段代码中的WHILE循环部分代码要么不会执行，要么可能出现死循环。

9．把统计空格程序的Jackson图（图6-31）改画为等价的程序流程图和盒图。

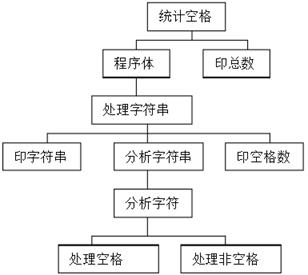


图6-31 统计空格程序的Jackson图

答：（1）等价的程序流程图如图6-32所示。

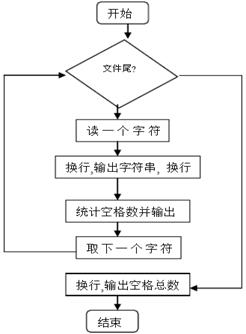


图6-32 统计空格程序的程序流程图

（2）等价的盒图如图6-33所示。



图6-33 统计空格程序的盒图

10．人机对话由操作员信息和系统信息交替组成。假设一段对话总是由操作员信息开始以系统信息结束，用Jackson图描绘这样的人机对话过程。

答：人机对话过程如图6-33、6-34、6-35、6-36所示。

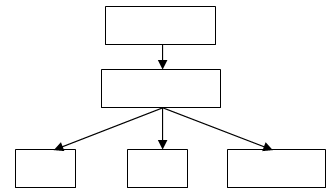


图6-33 人机对话Jackson图（a）

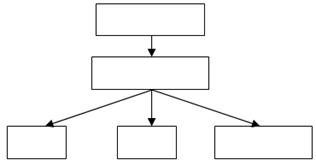


图6-34 人机对话Jackson图（b）

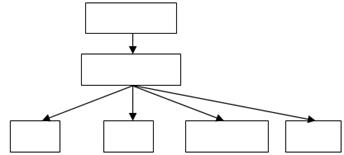


图6-35 人机对话Jackson图（c）

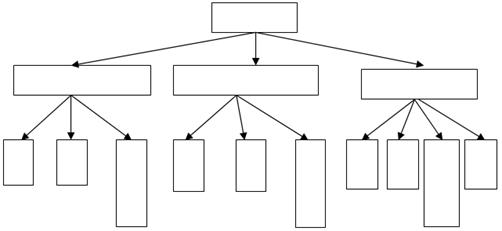


图6-36 人机对话Jackson图（d）

**第7章　实　现**

**7.1　复习笔记**

一、编码

1．定义

编码是把软件设计结果翻译成用程序设计语言书写的程序，是对设计的进一步具体化，因此程序的质量主要取决于软件设计的质量。

2．选择程序设计语言

（1）重要性

①程序设计语言的特点必然会影响人的思维和解题方式。

②程序设计语言会影响人和计算机通信的方式和质量。

③程序设计语言会影响其他人阅读和理解程序的难易程度。

④所选用的程序设计语言的特点将对程序的可靠性、可读性、可测试性和可维护性产生深远的影响。

（2）适宜的程序设计语言的优点

①能使根据设计去完成编码时困难最少。

②可以减少需要的程序测试量。

③可以得出更容易阅读和更容易维护的程序。

（3）选择程序设计语言的理想标准

①选用高级语言编写写程序。

②选用的高级语言应该有理想的模块化机制，以及可读性好的控制结构和数据结构。

③选用语言特点应该使编译程序能够尽可能多地发现程序中的错误。

④选用的高级语言应该有良好的独立编译机制。

（4）选择程序设计语言的实用标准

①系统用户的要求

如果所开发的系统由用户负责维护，用户通常要求用他们熟悉的语言书写程序。

②可以使用的编译程序

运行目标系统的环境中可以提供的编译程序往往限制了可以选用的语言的范围。

③可以得到的软件工具

如果某种语言有支持程序开发的软件工具可以利用，则目标系统的实现和验证都变得比较容易。

④工程规模

如果工程规模很庞大，现有的语言又不完全适用，那么设计并实现一种供这个工程项目专用的程序设计语言，可能是一个正确的选择。

⑤程序员的知识

虽然对于有经验的程序员来说，学习一种新语言并不困难，但是要完全掌握一种新语言却需要实践。如果和其他标准不矛盾，那么应该选择一种已经为程序员所熟悉的语言。

⑥软件可移植性要求

如果目标系统将在几台不同的计算机上运行，或者预期的使用寿命很长，那么选择一种标准化程度高、程序可移植性好的语言就是很重要的。

⑦软件的应用领域

所谓的通用程序设计语言实际上并不是对所有应用领域都同样适用。

3．编码风格

源程序代码的逻辑简明清晰、易读易懂是好程序的一个重要标准，为了做到这一点，应该遵循下述规则。

（1）程序内部的文档

①定义

程序内部的文档包括恰当的标识符、适当的注解和程序的视觉组织等。

②命名规则

a．选取含义鲜明的名字，使它能正确地提示程序对象所代表的实体。

b．使用缩写时，缩写规则应该一致，并且应该给每个名字加注解。

③注释规则

a．在每个模块开始处应有序言性注解，简述功能、算法、接口、重要数据和开发简史。

b．程序中间应有一段与代码有关的注解，主要解释代码的必要性。

c．对于用高级语言书写的源程序，应该利用注解提供一些额外的信息。

d．应该用空格或空行清楚地区分注解和程序。

e．注解的内容一定要正确，错误的注解会妨碍对程序的理解。

④程序清单的布局规则

应利用适当的阶梯形式使程序的层次结构清晰明显。

（2）数据说明

虽然在设计期间已经确定了数据结构的组织和复杂程度，然而数据说明的风格却是在写程序时确定的。为了使数据更容易理解和维护，应该遵循的一些简单的原则如下：

①数据说明的次序应该标准化。有次序就容易查阅，因此能够加速测试、调试和维护的过程。

②多个变量名在一个语句中说明时，应该按字母顺序排列这些变量。

③使用复杂的数据结构，应用注解说明用设计语言实现这个数据结构的方法和特点。

（3）语句构造

①原则

设计期间确定了软件的逻辑结构，然而个别语句的构造却是编写程序的一个主要任务。构造语句时应该遵循的原则是，每个语句都应该简单而直接，不能为了提高效率而使程序变得过分复杂。

②使语句简单明了的规则

a．不要为了节省空间而把多个语句写在同一行。

b．尽量避免复杂的条件测试。

c．尽量减少对“非”条件的测试。

d．避免大量使用循环嵌套和条件嵌套。

e．利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

（4）输入输出

在设计和编写程序时，输入输出风格需要遵循的规则：

①对所有输入数据都进行检验。

②检查输入项重要组合的合法性。

③保持输入格式简单。

④使用数据结束标记，不要要求用户指定数据的数目。

⑤明确提示交互式输入的请求，详细说明可用的选择或边界数值。

⑥当程序设计语言对格式有严格要求时，应保持输入格式一致。

⑦设计良好的输出报表。

⑧给所有输出数据加标志。

（5）效率

①定义

效率主要指处理机时间和存储器容量两个方面。

②原则

a．效率是性能要求，应该在需求分析阶段确定效率方面的要求。

b．效率是靠好设计来提高的。

c．效率和简单程度是一致的，不要牺牲程序的清晰性和可读性来不必要地提高效率。

③表现

a．程序运行时间

写程序的风格会对程序的执行速度和存储器要求产生影响。在把详细设计结果翻译成程序时，需要遵守以下规则：

第一，写程序之前先简化算术的和逻辑的表达式。

第二，仔细研究嵌套的循环，以确定是否有语句可以从内层往外移。

第三，尽量避免使用多维数组。

第四，尽量避免使用指针和复杂的表。

第五，使用执行时间短的算术运算。

第六，不要混合使用不同的数据类型。

第七，尽量使用整数运算和布尔表达式。在效率是决定性因素的应用领域，尽量使用有良好优化特性的编译程序，以自动生成高效目标代码。

b．存储器效率

第一，使用能保持功能域的结构化控制结构，是提高效率的好方法。

第二，在要求使用最少的存储单元时，应选用有紧缩存储器特性的编译程序。

第三，提高执行效率的技术能提高存储器效率。提高存储器效率的关键是“简单”。

c．输入输出的效率

第一，输入输出都应该有缓冲，以减少用于通信的额外开销。

第二，对二级存储器选用最简单的访问方法。

第三，二级存储器的输入输出应以信息组为单位进行。

第四，如果“超高效的”输入输出很难被人理解，则不应采用这种方法。

二、软件测试基础

1．软件测试的目标

（1）测试是为了发现程序中的错误而执行程序的过程。

（2）好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案。

（3）成功的测试是发现了至今为止尚未发现的错误的测试。

（4）软件测试的根本目标是尽可能多地发现并排除软件中潜藏的错误，最终把一个高质量的软件系统交给用户使用。

注意：测试只能查找出程序中的错误，不能证明程序中没有错误。

2．软件测试准则

（1）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程序不能满足用户需求的那些错误。

（2）应该远在测试开始之前就制定出测试计划。实际上，一旦完成了需求模型就可以着手制定测试计划，在建立了设计模型之后就可以立即开始设计详细的测试方案。因此，在编码之前就可以对所有测试工作进行计划和设计。

（3）应用Pareto原理，即测试发现的错误中80％可能是由程序20％的模块造成的。

（4）应该从“小规模”测试开始，并逐步进行“大规模”测试。首先重点测试单个程序模块，然后把测试重点转向在集成的模块簇中寻找错误，最后在整个系统中寻找错误。

（5）穷举测试是不可能的，即测试只能证明程序中有错误，不能证明程序中没有错误。

（6）为了达到最佳的测试效果，应该由独立的第三方从事测试工作。所谓“最佳效果”是指有最大可能性发现错误的测试。

3．测试方法

测试任何产品都有两种方法：如果已经知道了产品应该具有的功能，可以通过测试来检验是否每个功能都能正常使用；如果知道产品的内部工作过程，可以通过测试来检验产品内部动作是否按照规格说明书的规定正常进行。前一种方法称为黑盒测试，后一种方法称为白盒测试。

（1）白盒测试

白盒测试（结构测试）的前提是可以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的逻辑测试程序，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。

（2）黑盒测试

黑盒测试（功能测试）把程序看作一个黑盒子，完全不考虑程序的内部结构和处理过程。黑盒测试是在程序接口进行的测试，只检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接收输入数据并产生正确的输出信息，程序运行过程中能否保持外部信息的完整性。

4．测试步骤

（1）模块测试

模块测试（单元测试）是把每个模块作为一个单独的实体来测试，检验其正确性。目的是保证每个模块作为一个单元能正确运行。模块测试所发现的是编码和详细设计的错误。

（2）子系统测试

子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间的协调和通信是这个测试过程中的主要问题，即子系统测试着重测试模块的接口。

（3）系统测试

系统测试是把经过测试的子系统装配成一个完整的系统来测试。在这个过程不仅应该发现设计核编码的错误，还应该验证系统确实能提供需求说明书中指定的功能，而且系统动态特性也符合预定要求。系统测试发现的往往是软件设计中的错误，也可能发现需求说明中的错误。

注意：不论是子系统测试还是系统测试，都兼有检测和组装两重含义，都属于集成测试。

（4）验收测试

验收测试（确认测试）把软件系统作为单一的实体进行测试，它是在用户积极参与下进行的，而且主要使用实际数据进行测试。验收测试的目的是验证系统确实能够满足用户的需要，验收测试发现的是系统需求说明书中的错误。

（5）平行运行

平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果。具体目的有如下几点：

①可以在准生产环境中运行新系统而又不冒风险。

②用户能有一段熟悉新系统的时间。

③可以验证用户指南和使用手册之类的文档。

④能够以准生产模式对新系统进行全负荷测试，可以用测试结果验证性能指标

5．测试阶段的信息流

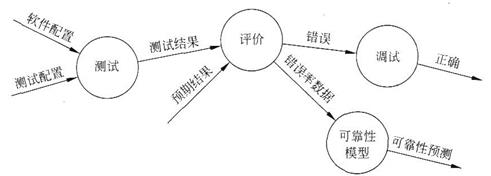


图7-1 测试阶段的信息流

图7-1描绘了测试阶段的信息流，具体内容如下：

（1）输入信息

①软件配置

包括需求说明书、设计说明书和源程序清单等。

②测试配置

包括测试计划和测试方案。测试方案应包括测试时使用的输入数据（测试用例）、每组输入数据预定要检验的功能、每组输入数据预期应该得到的正确输出。测试配置是软件配置的一个子集，最终交出的软件配置应该包括上述测试配置以及测试的实际结果和调试的记录。

（2）调试

比较测试得出的实际结果和预期的结果，如果两者不一致则很可能是程序中有错误。设法确定错误的准确位置并且改正它，这就是调试的任务。与测试不同，通常由程序的编写者负责调试。

（3）收集和评价测试结果

在对测试结果进行收集和评价的时候，如果经常出现要求修改设计的严重错误，那么软件的质量和可靠性是值得怀疑的，应该进一步仔细测试。反之，如果看起来软件功能完成得很正常，遇到的错误也很容易改正，则仍然应该考虑两种可能：

①软件的可靠性是可以接受的；

②所进行的测试尚不足以发现严重的错误。

（4）软件可靠性模型

软件可靠性模型使用错误率数据估计将来出现错误的情况，并进而对软件可靠性进行预测。

三、单元测试（模块测试）

单元测试集中检测软件设计的最小单元—模块，它和编码属于软件过程的同一个阶段。在编写出源程序代码并通过了编译程序的语法检查之后，就可以用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。单元测试主要使用白盒测试技术，而且对多个模块的测试可以并行地进行，包括人工测试和计算机测试两种。

1．测试重点

（1）模块接口

①模块接口的数据流是否能正常进出。

②参数的数目、次序、属性或单位系统与变元是否一致。

③是否修改了只作输入用的变元。

④全局变量的定义和用法在各个模块中是否一致。

（2）局部数据结构

对于模块来说，局部数据结构是常见的错误来源。应该仔细设计测试方案，以便发现局部数据说明、初始化、默认值等方面的错误。

（3）重要的执行通路

①选择最有代表性、最可能发现错误的执行通路进行测试。

②设计测试方案来发现由于错误计算、不正确的比较或不适当的控制流而造成的错误。

（4）出错处理通路

当评价出错处理通路时，应着重测试可能发生的错误为：

①对错误的描述是难以理解的。

②记下的错误与实际遇到的错误不同。

③在对错误进行处理之前，错误条件已经引起系统干预。

④对错误的处理不正确。

⑤描述错误的信息不足以帮助确定造成错误的位置。

（5）边界条件

边界测试是单元测试中最重要的任务。软件常常在它的边界上失效，例如，处理n元数组的第n个元素时，或做到1次循环中的第1次重复时，往往会发生错误。使用刚好小于、刚好等于和刚好大于最大值或最小值的数据结构、控制量和数据值的测试方案，非常可能发现软件中的错误。

2．测试方法

（1）代码审查

①定义

人工测试源程序可以由程序的编写者本人非正式地进行，也可以由审查小组正式进行。后者称为代码审查，是一种非常有效的程序验证技术。

②流程

a．组建审查小组。

审查小组由组长（能力强且没有参与任务的人员）、程序的设计者、程序的编写者、程序的测试者组成。

b．审查会议

第一，审查前，小组成员应该先研究设计说明书。

第二，审查会上由程序的编写者解释他是怎样用程序代码实现这个设计的。

第三，其他成员仔细倾听讲解并力图发现其中的错误。

第四，对照程序设计常见错误清单，分析审查程序并记录错误。

③优点

a．一次审查会上可以发现许多错误。

b．不需要每次发现一个错误就进行验证，减少了系统验证的总工作量。

④与计算机测试的关系

对于查找某些类型的错误来说，人工测试比计算机测试更有效；对于其他类型的错误来说则刚好相反。因此，人工测试和计算机测试是互相补充，相辅相成的，缺少其一都会使查找错误的效率降低。

（2）计算机测试

①方法

模块并不是一个独立的程序，因此必须要为每个单元测试开发驱动软件和（或）存根软件，具体如下：

a．驱动程序

接收测试数据，把这些数据传送给被测试的模块，并且印出有关的结果。

b．存根程序

代替被测试的模块所调用的模块，它使用被它代替的模块的接口，做最少量的数据操作，印出对入口的检验或操作结果，并且把控制归还给调用它的模块。

②例子

图7-2是一个正文加工系统的部分层次图，假定要测试其中编号为3.0的关键模块。

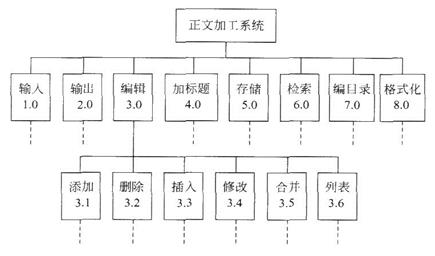


图7-2 正文加工系统的层次图

a．分析

第一，需要有一个测试驱动程序来调用它。这个驱动程序说明必要的变量，接收测试数据字符串，并且设置正文编辑模块的编辑功能。

第二，需要有存根程序简化地模拟下层模块。

b．设计

计算机测试所需要程序的伪码，如图7-3所示。

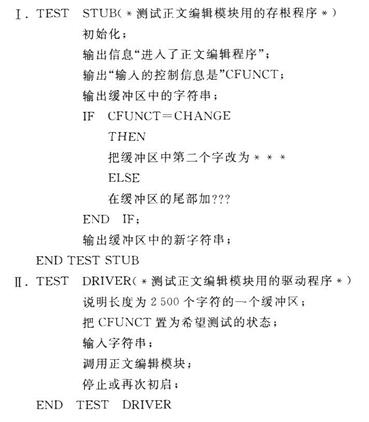


图7-3 计算机测试的程序

四、集成测试

1．概念

（1）方法分类

集成测试是测试和组装软件的系统化技术，由模块组装成程序时有两种方法：非渐增式方法和渐增式方法。

①非渐增式测试

先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。

②渐增式测试

把下一个要测试的模块同已经测试好的那些模块结合起来进行测试，测试完以后再把下一个应该测试的模块结合进来测试，每次增加一个模块。渐增式测试同时完成单元测试和集成测试。

（2）非渐增式测试的缺点

①把所有模块放在一起，测试者面对的情况十分复杂。

②在庞大的程序中诊断定位一个错误非常困难。

③一旦改正一个错误之后，又会遇到新的错误，没有穷尽。

（3）渐增式测试的优点

①把程序划分成小段来构造和测试，比较容易定位和改正错误。

②对接口可以进行更彻底的测试。

③可以使用系统化的测试方法。

2．渐增式测试的策略

当使用渐增方式把模块结合到程序中去时，有自顶向下和自底向上两种集成策略。

（1）自顶向下集成

①定义

从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于主控制模块的模块组装到程序结构中时，使用深度优先的策略或宽度优先的策略。自顶向下集成方法是一个日益为人们广泛采用的测试核组装软件的途径。

②步骤

a．对主控制模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块。

b．根据选的结合策略（深度优先或宽度优先），每次用一个实际模块代换一个存根程序。

c．在结合进一个模块的同时进行测试。

d．为了保证加入模块没有引进新的错误，可能需要进行回归测试。

③结合策略

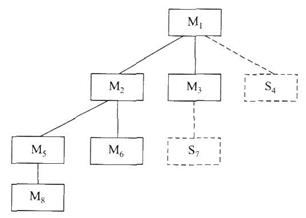


图7-4 自顶向下结合

a．深度优先

深度优先的结合方法先组装在软件结构的一条主控制通路上的所有模块。步骤如下（参看图7-4）：

第一，选择一条主控制通路取决于应用的特点（如，选取左通路）。

第二，结合模块M1、M2、M5。

第三，把M8或M6结合进来。

第四，构造中央的和右侧的控制通路。

b．宽度优先

宽度优先的结合方法是沿软件结构水平地移动，把处于同一个控制层次上的所有模块组装起来。步骤如下（参看图7-4）：

第一，结合模块M2、M3、M4。

第二，结合下一个控制层次中的模块M5、M6、M7。

第三，继续进行下去，直到所有模块都被结合进来为止。

（2）自底向上集成

①定义

自底向上测试从软件结构最低层的模块开始组装和测试。因为是从底部向上结合模块，总能得到所需的下层模块处理功能，所以不需要存根程序。

②自底向上结合的步骤

a．把低层模块组合成实现某个特定的软件子功能的族。

b．写一个用于测试的控制程序，协调测试数据的输入和输出。

c．对由模块组成的子功能族进行测试。

d．去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成大的子功能族。

③实例分析

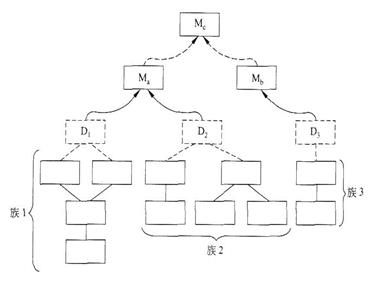


图7-5 自底向上结合

图7-5描绘了自底向上的结合过程，分析如下：

a．把模块组合成族1、族2和族3。

b．使用驱动程序（图中用虚线方框表示）对每个子功能族进行测试。

c．族1和族2中的模块附属于模块Ma，去掉驱动程序D1和D2，把这两个族直接同Ma连接起来。同样在和模块Mb结合之前去掉族3的驱动程序D3。

d．Ma和Mb这两个模块都与模块Mc结合起来。

3．两种集成策略的比较

（1）自顶向下集成

①优点

a．不需要测试驱动程序。

b．能够在测试阶段的早期实现并验证系统的主要功能。

c．能在早期发现上层模块的接口错误。

②缺点

a．需要存根程序，可能遇到与此相联系的测试困难。

b．低层关键模块中的错误发现较晚。

c．在早期不能充分展开人力。

（2）字底向上集成

①优点

a．不需要存根程序，不会遇到与存根程序相联系的测试困难。

b．能较早的发现低层关键模块的错误。

c．能在早期充分展开人力。

②缺点

a．需要测试驱动程序。

b．不能在测试阶段早期实现并验证系统的主要功能。

c．在早期不能发现上层模块的接口错误。

4．其他集成测试策略

在测试实际的软件系统时，应该根据软件的特点以及工程进度安排，选用适当的测试策略。一般说来，纯粹自顶向下或纯粹自底向上的策略可能都不实用，人们在实践中创造出许多混合策略。

（1）改进的自顶向下测试方法

基本上使用自顶向下的测试方法，但在早期使用自底向上的方法测试软件中的少数关键模块。缺点是测试关键模块时需要驱动程序。

（2）混合法

对软件结构中较上层使用的自顶向下方法与对软件结构中较下层使用的自底向上方法相结合。

5．回归测试

（1）定义

回归测试是指重新执行已经做过的测试的某个子集，以保证上述这些变化没有带来非预期的副作用。它可以用于保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误的测试活动。

（2）方法

①通过重新执行全部测试用例的一个子集人工地进行。

②利用捕获回放工具，捕获测试用例和实际运行结果，然后回放，并比较运行结果。

（3）回归测试集

回归测试集（已执行过的测试用例的子集）包括下述3类不同的测试用例：

①检测软件全部功能的代表性测试用例。

②专门针对可能受修改影响的软件功能的附加测试。

③针对被修改过的软件成分的测试。

五、确认测试

1．概念

确认测试也称为验收测试，它的目标是验证软件的有效性。

（1）验证

验证指的是保证软件正确地实现了某个特定要求的一系列活动。

（2）确认

确认指的是为了保证软件确实满足了用户需求而进行的一系列活动。

（3）软件有效性

如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的。

2．确认测试的范围

（1）要求

①必须有用户积极参与，或者以用户为主进行。用户应该参与设计测试方案，使用用户界面输入测试数据并且分析评价测试的输出结果。在验收之前由开发单位对用户进行培训。

②确认测试通常使用黑盒测试法。应该仔细设计测试计划和测试过程，测试计划包括要进行的测试的种类及进度安排，测试过程规定了用来检测软件是否与需求一致的测试方案。通过测试和调试要保证软件能满足所有功能要。

（2）结果

①功能和性能与用户要求一致，软件是可以接受的。

②功能和性能与用户要求有差距。

3．软件配置复查

（1）目的

复查的目的是保证软件配置的所有成分都齐全，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，而且已经编好目录。

（2）要求

①在确认测试过程中应该严格遵循用户指南及其他操作程序。

②必须仔细记录发现的遗漏或错误，并且适当地补充和改正。

4.Alpha和Beta测试

（1）Alpha测试

Alpha测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试，且开发者负责记录发现的错误和遇到的问题。即Alpha测试是在受控的环境中进行的。

（2）Beta测试

①定义

Beta测试由软件的最终用户们在一个或多个客户场所进行。开发者通常不在Beta测试的现场，即Beta测试是软件在开发者不能控制的环境中的“真实”应用。

②测试过程

a．用户记录在Beta测试过程中遇到的问题，并且定期把这些问题报告给开发者。

b．开发者在接收到在Beta测试期间报告的问题之后，对软件产品进行必要的修改。

c．开发者向全体客户发布最终的软件产品。

六、白盒测试技术

设计测试方案是测试阶段的关键技术问题。所谓测试方案包括具体的测试目的，应该输入的测试数据和预期的结果。通常又把测试数据和预期的输出结果称为测试用例。其中最困难的问题是设计测试用的输入数据。

设计测试方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。已经研究出许多设计测试数据的技术，这些技术各有优缺点，没有哪一种是最好的，更没有哪一种可以代替其余所有技术；同一种技术在不同的应用场合效果可能相差很大，因此，通常需要联合使用多种设计测试数据的技术。

1．逻辑覆盖

（1）定义

逻辑覆盖是对一系列测试过程的总称，这组测试过程逐渐进行越来越完整的通路测试。

（2）分类

①语句覆盖

语句覆盖的含义是选择足够多的测试数据，使被测程序中每个语句至少执行一次。

②判定覆盖

判定覆盖（分支覆盖）含义是不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次。

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。

④判定/条件覆盖

判定/条件覆盖的含义是选取足够多的测试数据，使得判定表达式中的每个条件都取到各种可能的值，而且每个判定表达式也都取到各种可能的结果。它同时满足判定覆盖和条件覆盖。

⑤条件组合覆盖

条件组合覆盖要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。满足条件组合覆盖，也一定满足判定覆盖、条件覆盖和判定/条件覆盖标准。

⑥点覆盖

连通图G的子图G′是连通的，而且包含G的所有结点，则称G′是G的点覆盖。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，即点覆盖标准和语句覆盖标准是相同的。

⑦边覆盖

连通图G的子图G″是连通的，而且包含G的所有边，则称G″是G的边覆盖。为满足边覆盖的测试标准，要求选取足够多测试数据，使程序执行路径至少经过流图中每条边一次，即边覆盖与判定覆盖是相同的。

⑧路径覆盖

路径覆盖含义是选取足够多测试数据，使程序的每条可能路径都至少执行一次，如果程序图中有环，则要求每个环至少经过一次。

2．控制结构测试

（1）基本路径测试

①定义

基本路径测试是TomMcCabe提出的一种白盒测试技术。使用这种技术设计测试用例时，首先计算程序的环形复杂度，并用该复杂度为指南定义执行路径的基本集合，从该基本集合导出的测试用例可以保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。

②步骤

a．根据过程设计结果画出相应的流图。

b．计算流图的环形复杂度。

c．确定线性独立路径（至少包含一条在定义该路径之前不曾用过的边）的基本集合。

d．设计可强制执行基本集合中每条路径的测试用例。

注意：某些独立路径不能以独立的方式测试，即程序的正常流程不能形成独立执行该路径所需要的数据组合。

（2）条件测试

①关系表达式

一个简单条件是一个布尔变量或一个关系表达式，在布尔变量或关系表达式之前还可能有一个NOT（┐）算符。关系表达式的形式如下：

E1<关系算符>E2

其中，E1和E2是算术表达式，而<关系算符>是下列算符之一：<，≤，=，≠，>或≥。复合条件由两个或多个简单条件、布尔算符和括弧组成。布尔算符有OR（∣），AND（&）和NOT（┐）。不包含关系表达式的条件称为布尔表达式。

②条件错误的类型

a．布尔算符错；

b．布尔变量错；

c．布尔括弧错；

d．关系算符错；

e．算术表达式错。

③条件测试的优点

a．容易度量条件的测试覆盖率。

b．程序内条件的测试覆盖率可指导附加测试的设计。

④条件测试的目的

条件测试的目的不仅是检测程序条件中的错误，而且是检测程序中的其他错误。

⑤条件测试策略

a．分支测试

对于复合条件C来说，C的真分支和假分支以及C中的每个简单条件，都应该至少执行一次。域测试要求对一个关系表达式执行3个或4个测试。包含n个变量的布尔表达式需要2n个测试。

b．BRO测试

BRO测试利用条件C的条件约束来设计测试用例。包含n个简单条件的条件C的条件约束定义为（D1，D2，…，Dn），其中Di（0<i<n）表示条件C中第i个简单条件的输出约束。对于布尔变量B来说，B的输出约束指出，B必须是真（t）或假（f）；对于关系表达式来说，用符号>，=和<指定表达式的输出约束。

（3）循环测试

①定义

循环是绝大多数软件算法的基础，但是，在测试软件时却往往未对循环结构进行足够的测试。循环测试是一种白盒测试技术，它专注于测试循环结构的有效性

②分类

在结构化的程序中通常只有3种循环，即简单循环、串接循环和嵌套循环。

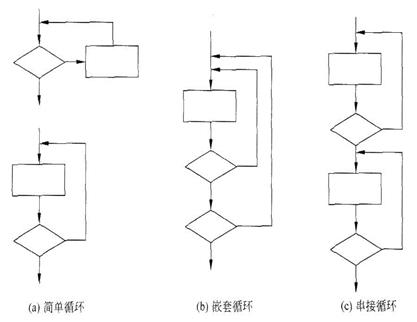


图7-6 三种循环

a．简单循环

结构如图7-6（a）所示。

b．嵌套循环

结构如图7-6（b）所示。可以通过以下步骤减少测试数：

第一，从最内层循环开始测试，把所有其他循环都设置为最小值。

第二，对最内层循环使用简单循环测试方法。使内层循环的迭代参数取最小值，并为越界值或非法值增加一些额外的测试。

第三，由内向外，对下一个循环进行测试，但保持所有其他外层循环为最小值，其他嵌套循环为“典型”值。

第四，继续进行下去，直到完成所有循环。

c．串接循环

结构如图7-6（c）所示。如果串接循环的各个循环都彼此独立，则可以使用前述的测试简单循环的方法来测试串接循环；循环不独立时，使用测试嵌套循环的方法来测试串接循环。

七、黑盒测试技术

1．概念

黑盒测试着重测试软件功能。黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型的错误。

（1）目的

黑盒测试力图发现下述类型的错误：

①功能不正确或遗漏了功能。

②界面错误。

③数据结构错误或外部数据库访问错误。

④性能错误。

⑤初始化和终止错误。

（2）适用性

白盒测试在测试过程的早期阶段进行，黑盒测试主要用于测试过程的后期。

（3）设计测试方案需要考虑的问题

①怎样测试功能的有效性？

②哪些类型的输入可构成好测试用例？

③系统是否对特定的输入值特别敏感？

④怎样划定数据类的边界？

⑤系统能够承受什么样的数据率和数据量？

⑥数据的特定组合将对系统运行产生什么影响？

（4）测试用例的标准

①能够减少为达到合理测试所需要设计的测试用例的总数。

②能够告诉人们，是否存在某些类型的错误，而不是仅仅指出与特定测试相关的错误是否存在。

2．技术方法

（1）等价划分

①定义

等价划分是一种黑盒测试技术，这种技术把程序的输入域划分成若干个数据类，据此导出测试用例。一个理想的测试用例能独自发现一类错误。

②目的

等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目。

③流程

a．划分数据的等价类

第一，需要研究程序的功能说明，从而确定输入数据的有效等价类和无效等价类。

第二，在确定输入数据的等价类时常常还需要分析输出数据的等价类。

第三，在划分等价类时还应考虑编译程序的检错功能。

b．根据等价类设计测试方案

第一，设计一个新的测试方案以尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步骤直到所有有效等价类都被覆盖为止。

第二，设计一个新的测试方案，使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止。

注意：通常程序发现一类错误后就不再检查是否还有其他错误，因此，应该使每个方案只覆盖一个无效的等价类。

④等价类划分规则

a．如果规定了输入值的范围，则可划分一个有效的等价类（输入值在此范围内），两个无效的等价类（输入值小于最小值或大于最大值）。

b．如果规定了输入数据的个数，则类似地也可以划分出一个有效的等价类和两个无效的等价类。

c．如果规定了输入数据的一组值，而且程序对不同输入值做不同处理，则每个允许的输入值是一个有效的等价类，此外还有一个无效的等价类（任一个不允许的输入值）。

d．如果规定了输入数据必须遵循的规则，则可以划分出一个有效的等价类（符合规则）和若干个无效的等价类（从各种不同角度违反规则）。

e．如果规定了输入数据为整型，则可以划分出正整数、零和负整数3个有效类。

f．如果程序的处理对象是表格，则应该使用空表，以及含一项或多项的表。

（2）边界值分析

使用边界值分析方法设计测试方案首先应该确定边界情况，选取的数据应该刚好等于、稍小于和稍大于等价类边界值，即应该选取刚好等于、稍小于和稍大于等价类边界值的数据作为测试数据，而不是选取每个等价类内的典型值或任意值作为测试数据。

（3）错误推测

①定义

错误推测法基本思想是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案。

②输入组合

a．定义

输入组合是综合考虑各个输入数据相结合产生的测试功效，减少遗漏输入数据易于出错的情况。

b．选择途径

第一，利用判定表或判定树为工具，列出输入数据各种组合与程序应作的动作之间的对应关系，然后为判定表的每一列至少设计一个测试用例。

第二，把计算机测试和人工检查代码结合起来。

八、调试

1．定义

调试（纠错）作为成功测试的后果出现，即调试是在测试发现错误之后排除错误的过程。调试是把症状和原因联系起来的尚未被人深入认识的智力过程。

2．调试过程

（1）流程

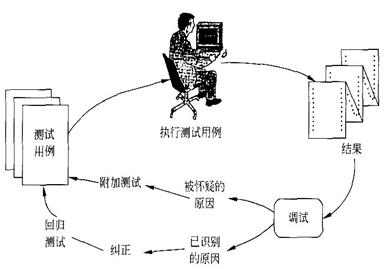


图7-7 调试过程

如图7-7所示，调试过程从执行一个测试用例开始，评估测试结果，如果发现实际结果与预期结果不一致，表明在软件中存在着隐藏的问题。调试过程试图找出产生症状的原因，以便改正错误。

（2）调试的结果

①找到了问题的原因并把问题改正和排除掉了。

②没找出问题的原因。

没找出问题的原因时，调试人员可以猜想一个原因，并设计测试用例来验证，重复这个过程直至找到原因并改正了错误。

（3）软件错误的特征

①症状和产生症状的原因可能在程序中相距甚远。

②当改正了另一个错误之后，症状可能暂时消失了。

③症状可能实际上并不是由错误引起的。

④症状可能是由不易跟踪的人为错误引起的。

⑤症状可能是由定时问题而不是由处理问题引起的。

⑥可能很难重新产生完全一样的输入条件。

⑦症状可能时有时无。

⑧症状可能是由分布在许多任务中的原因引起的，这些任务运行在不同的处理机上。

3．调试途径

（1）蛮干法

①基本思路

按照“让计算机自己寻找错误”的策略，这种方法印出内存的内容，激活对运行过程的跟踪，并在程序中到处都写上WRITE（输出）语句，在生成的信息海洋的某个地方发现错误原因的线索。

②适用性

蛮干法是寻找软件错误原因的最低效的方法。仅当所有其他方法都失败了的情况下，才应该使用这种方法。

（2）回溯法

①基本思路

从发现症状的地方开始，人工沿程序的控制流往回追踪分析源程序代码，直到找出错误原因为止。

②适用性

当调试小程序时回溯法非常有效的。但随着程序规模的扩大，应该回溯的路径数目也变得越来越大，以至彻底回溯变成完全不可能了。

（3）原因排除法

①对分查找法

如果已经知道每个变量在程序内若干个关键点的正确值，可用赋值语句或输入语句在程序中点附近“注入”这些变量的正确值，然后运行程序并检查所得到的输出。如果输出结果是正确的，则错误原因在程序的前半部分；反之，错误原因在程序的后半部分。对错误部分重复使用这个方法，直到把出错范围缩小到容易诊断的程度为止。

②归纳法

归纳法是从个别现象推断出一般性结论的思维方法。使用这种方法调试程序时，首先把和错误有关的数据组织起来进行分析。然后导出对错误原因的一个或多个假设，并利用已有的数据来证明或排除这些假设。

③演绎法

演绎法从一般原理或前提出发，经过排除和精化的过程推导出结论。采用这种方法调试程序时，首先设想出所有可能的出错原因，然后用测试来排除每一个假设的原因。

4．调试准则

（1）仔细分析程序出错处的逻辑模式，找出该错误出现的所有地方。

（2）在改正错误前应仔细研究源程序，以评估逻辑和数据结构的耦合程度。

（3）修改软件产品的同时改进开发软件产品的软件过程，避免今后在程序中出现错误。

九、软件可靠性

1．基本概念

测试阶段的根本目标是消除错误，保证软件的可靠性。

（1）软件可靠性

软件可靠性是程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。

（2）软件的可用性

软件可用性是程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

（3）比较

①可靠性意味着在0到t这段时间间隔内系统没有失效。

②可用性只意味着在时刻t，系统是正常运行的。

（4）MTTF和MTTR

①概念

a．平均维修时间MTTR

表示修复一个故障平均需要用的时间，取决于维护人员的技术水平和对系统的熟悉程度，也和系统的可维护性有重要关系。

b．平均无故障时间MTTF

表示系统按规格说明书规定成功地运行的平均时间，主要取决于系统中潜伏的错误的数目，因此和测试的关系十分密切。

②计算公式

如果在一段时间内，软件系统故障停机时间分别为td1，td2，…，正常运行时间分别为tu1，tu2，…，则系统的稳态可用性为：

IMG_419

其中：

IMG_420

（7.1）式可以变成

IMG_421

2．估算平均无故障时间的方法

（1）符号

①ET：测试之前程序中错误总数；

②IT：程序长度（机器指令总数）；

③τ：测试（包括调试）时间；

④Ed（τ）：在0至τ期间发现的错误数；

⑤Ec（τ）：在0至τ期间改正的错误数。

（2）基本假定

①单位长度里的错误数ET/IT近似为常数。通常0.5×102≤ET/IT≤2×102。

②失效率正比于软件中潜藏的错误数，平均无故障时间MTTF与剩余的错误数成反比。

③假设发现的每一个错误都立即正确地改正了，即Ec（τ）= Ed（τ）。

a．剩余的错误数

IMG_422

b．单位长度程序中剩余的错误数

IMG_423

（3）估算平均无故障时间

①平均无故障时间与单位长度程序中剩余的错误数成反比（K经典值为200），即

IMG_424

②估算平均无故障时间的公式，可以评价软件测试进展情况。由（7.5）式可得：

IMG_425

（4）估计错误总数的方法

①植入错误法

a．基本思路

测试之前，由专人在程序中随机地植入一些错误；测试之后，根据发现的错误中原有的和植入的两种错误的比例，来估计程序中原有错误的总数E。

b．计算方法

假设人为地植入的错误数为Ns，经过一段时间的测试之后发现ns个植入的错误，还发现了n个原有的错误。则估计出程序中原有错误的总数为：

IMG_426

其中IMG_427即是错误总数ET的估计值。

②分别测试法

a．基本思路

假定所用的测试方案发现植入错误和发现原有错误的概率相同。

b．计算方法

在测试过程的早期阶段，由测试员甲和测试员乙分别测试同一个程序的两个副本，由另一名分析员分析他们的测试结果。用r表示测试时间，假设：

第一，τ=0时错误总数为B0；

第二，r=τ1时测试员甲发现的错误数为B1；

第四，τ=τ1时测试员乙发现的错误数为B2；

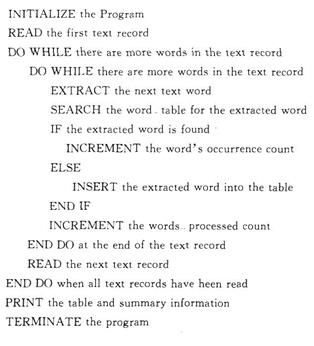
第五，τ=τ1时两个测试员发现的相同错误数为bc。

假定测试员乙发现有标记错误和发现无标记错误的概率相同，则可以估计出测试前程序中的错误总数为：

IMG_428

**7.2　课后习题详解**

1．下面给出的伪码中有一个错误。仔细阅读这段伪码，说明该伪码的语法特点，找出并改正伪码中的错误。字频统计程序的伪码如下：



答：（1）错误：INSERT the extracted word into the table在这个后面，没有给这个word的occurrence count赋值为1。

（2）语法特点：该伪代码在循环结构中嵌套循环结构，并且在嵌套的循环结构中又嵌套了IF ELSE结构。

2．研究下面给出的伪码程序，要求：

（1）画出它的程序流程图。

（2）它是结构化的还是非结构化的？说明理由。

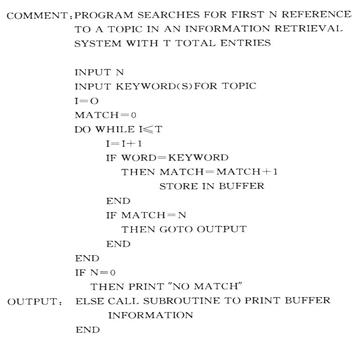
（3）若是非结构化的，则：

a．把它改造成仅用3种控制结构的结构化程序；

b．写出这个结构化设计的伪码；

c．用盒图表示这个结构化程序。

（4）找出并改正程序逻辑中的错误。



答：（1）程序流程图，如图7-8所示。



图7-8 程序流程图

（2）此程序是非结构化的，它有一个GOTO语句，并且是从一个循环体内转到循环体外的一条件语句内部。

（3）①修改后的伪码，如图7-9所示。

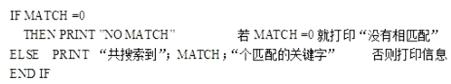
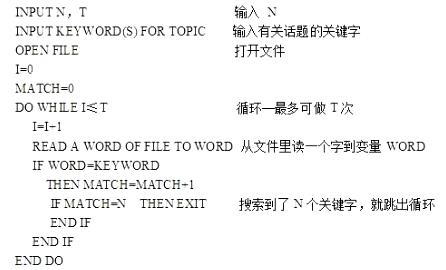


图7-9 修改后的伪码

②修改后程序的盒图，如图7-10所示。



图7-10 修改后程序的盒图

（4）程序中的错误：

①语句“IF WORD=KEYWORD”里的变量“WORD”没有预先赋值。

②程序中没有预先输入T的值。

③“N=0”应该是“MATCH=0”。

3．在第2题的设计中若输入的N值或KEYWORD不合理，会发生问题。

（1）给出这些变量的不合理值的例子。

（2）将这些不合理值输入程序会有什么后果？

（3）怎样在程序中加入防错措施，以防止出现这些问题？

答：（1）N=1，KEYWORD=# 。

（2）不合理的值输入程序后无法打印出任何信息。

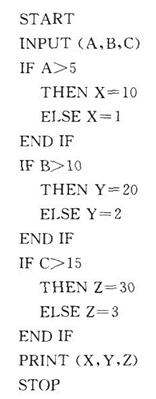
（3）可以通过加入循环来防止出现这些错误的问题。

4．回答下列问题。

（1）什么是模块测试和集成测试？它们各有什么特点？

（2）假设有一个由1000行FORTRAN语句构成的程序（经编译后大约有5000条机器指令），估计在对它进行测试期间将发现多少个错误？为什么？

（3）设计下列伪码程序的语句覆盖和路径覆盖测试用例：



答：（1）①模块测试时对每个单独的模块，分别用黑盒和白盒测试技术，测试它地 功能是否正确，检查模块控制结构中的特定路径并发现最大数量的错误。

②集成测试时把模块装配在一起形成完整的软件包，在装配的同事进行测试。

③模块测试的特点是主要应用白盒测试，对多个模块的测试可以并发的行进；集成测试的特点是在测试过程中可能发生接口问题。

（2）通常发现错误的比值在0.005~0.02之间，即在测试前每1000条指令中大约有5~20个错误。所以在对1000行FORTRAN语句构成的程序进行测试期间将发现的错误个数大约有25~100个。

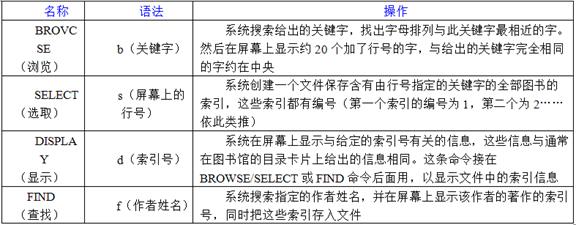
（3）语句覆盖用例（如图7-11所示）：①A=5，B=10，C=15；②A=6，B=11，C=16；

　 条件覆盖用例（如图7-11所示）：①A=5，B=10，C=15；②A=6，B=11，C=16。



图7-11 测试用例

5．某图书馆有一个使用CRT终端的信息检索系统，该系统有下列4个基本检索命令：



要求：

（1）设计测试数据以全面测试系统的正常操作；

（2）设计测试数据以测试系统的非正常操作。

答：（1）测试系统正常操作的测试数据

①顺序执行下列三个命令：

b（KEYWORD）

s（L）

d（N）

其中，KEYWORD是正确的关键字；L是执行命令b后在屏幕上显示的约20个行号中的一个（至少应该使L分别为第一个、最后一个和中央一个行号）；N是执行命令S后列出的索引号中的一个（至少应该使N分别为第一个、最后一个和中央一个索引号）。

针对若干个不同的KEYWORD重复执行上述命令序列。

②顺序执行下列两个命令：

f（NAME）

d（N）

其中，NAME是已知的作者姓名；N是执行命令f后列出的索引号中的一个（至少应该使 N分别为第一个、最后一个和中央一个索引号）。

针对若干个不同的NAME重复执行上述命令序列。

（2）测试系统非正常操作的测试数据

①用过长的关键字作为命令b的参数，例如，b（reliability software and hardware combined）

预期的输出：系统截短过长的关键字，例如，上列命令中的关键字可能被截短为 reliability software

②用不正确的关键字作为命令b的参数，例如，b（AARDVARK）

预期的输出：显示出最接近的匹配结果，例如，执行上列命令后可能显示

1.AARON，J ULES（book）

③用比执行命令b后列出的最大行号大1的数作为命令S的参数

预期的输出：“命令s的参数不在行号列表中”

④用数字和标点符号作为命令b和命令f的参数

预期的输出：“参数类型错”

⑤用字母字符作为命令s和命令d的参数

预期的输出：“参数类型错”

⑥用0和负数作为命令S和命令d的参数

预期的输出：“参数数值错”

⑦命令顺序错，例如，没执行命令b就执行命令s，或没执行命令s就执行命令d

预期的输出：“命令顺序错”

⑧命令语法错，例如，遗漏命令名b、s、d或f；或命令参数没用圆括号括起来

预期的输出：“命令语法错”

⑨命令参数空，例如，b（）、s（）、d（）或f（）

预期的输出：系统提供默认参数或给出出错信息

⑩使用拼错了的作者姓名作为命令f的参数

预期的输出：“找不到这位作者的著作”

6．航空公司A向软件公司B订购了一个规划飞行路线的程序。假设自己是软件公司C的软件工程师，A公司已雇用自己所在的公司对上述程序进行验收测试。任务是根据下述事实设计验收测试的输入数据，解释选取这些数据的理由。

领航员向程序输入出发点和目的地，以及根据天气和飞机型号而初步确定的飞行高度。程序读入途中的风向风力等数据，并且制定出3套飞行计划（高度，速度，方向及途中的5个位置校核点）。所制定的飞行计划应做到燃料消耗和飞行时间都最少。

答：应该分别使用正常的输入数据和异常的输入数据作为验收测试数据。

（1）用正常的输入数据作为测试数据

①输入常规的出发点、目的地、5个位置校核点、高度、速度及飞机型号。

②针对5对不同的出发点和目的地，重复执行测试①。

③固定出发点、目的地、位置校核点、高度和速度，分别输入3～5种不同的飞机型号，重复执行测试①。

④固定出发点、目的地、位置校核点、高度和飞机型号，分别输入3～5个不同的速度，重复执行测试①。

⑤固定出发点、目的地、位置校核点、速度和飞机型号，分别输入3～5个不同的高度，重复执行测试①。

⑥固定出发点、目的地、高度、速度和飞机型号，分别输入3～5组不同的位置校核点，重复执行测试①。

⑦固定出发点、位置校核点、高度、速度和飞机型号，分别输入3～5个不同的目的地，重复执行测试①。

⑧固定目的地、位置校核点、高度、速度和飞机型号，分别输入3～5个不同的出发点，重复执行测试①。

⑨同时改变一对参数的值，其他参数的值固定，重复执行测试①。⑩同时改变三个参数的值，其他参数的值固定，重复执行测试①。⑥以适当的方式改变描述天气状况的数据，重复执行测试①。

（2）用边界数据值作为测试数据

①分别使用距离非常近和距离非常远的两个地点作为出发点和目的地。

②输入位置校核点的非常规组合。

③分别输入非常高和非常低的高度值。

④分别输入非常高和非常低的速度值。

⑤输入极其少见的飞机型号。

（3）用无效的数据作为测试数

①用由字母数字字符和控制字符混合在一起组成的字符串作为出发点或目的地。

②用数字0作为所有参数的值。

③用负数作为高度和速度的值。

7．严格说来，有两种不同的路径覆盖测试，分别为程序路径覆盖和程序图路径覆盖。这两种测试可分别称为程序的自然执行和强迫执行。所谓自然执行是指测试者（人或计算机）读入程序中的条件表达式，根据程序变量的当前值计算该条件表达式的值（真或假），并相应地分支。强迫执行是在用程序图作为程序的抽象模型时产生的一个人为的概念，它可以简化测试问题。强迫执行的含义是，一旦遇到条件表达式，测试者就强迫程序分两种情况（条件表达式的值为真和为假）执行。显然，强迫执行将遍历程序图的所有路径，然而由于各个条件表达式之间存在相互依赖的关系，这些路径中的某一些在自然执行时可能永远也不会进入。

为了使强迫执行的概念在实际工作中有用，它简化测试工具的好处应该超过它使用额外的不可能达到的测试用例所带来的坏处。在绝大多数情况下，强迫执行的测试数并不比自然执行的测试数大很多，此外，对强迫执行的定义实际上包含了一种技术，能够缩短在测试含有循环的程序时所需要的运行时间。

程序的大部分执行时间通常用于重复执行程序中的DO循环，特别是嵌套的循环。因此必须发明一种技术，使得每个DO循环只执行一遍。这样做并不会降低测试的功效，因为经验表明第一次或最后一次执行循环时最容易出错。

Laemmel|教授提出的自动测试每条路径的技术如下：

当编写程序时每个DO循环应该写成一种包含测试变量T和模式变量M的特殊形式，因此DO I = 1 TO 38应变成DO I TO M \* 38+（1-M）\*T可见，当M=0时处于测试模式，而M=1时处于正常运行模式。当处于测试模式时，令T=0则该循环一次也不执行，令T=1则该循环只执行一次。

类似地，应该使用模式变量和测试变量改写IF语句，例如

IF X+Y>0　 THEN Z=X

　 ELSE Z=Y

应变成

IF M\*（X+Y）+T>0　 THEN Z=X

　 ELSE Z=Y

正常运行时令M=1和T=0，测试期间令M=0，为测试THEN部分需令T=+1，测试ELSE部分则令T= -1。

要求：

（1）选取一个包含循环和IF语句的程序，用Laemmel技术修改这个程序，上机实际测试这个程序并解释所得到的结果。

（2）设计一个程序按照Laemmel技术自动修改待测试的程序。利用这个测试工具修改上一问中人工修改的程序，两次修改得到的结果一致吗？

（3）怎样把Laemmel技术推广到包含WHILE DO和REPEAT UNTIL语句的程序？

（4）试分析Laemmel技术的优缺点并提出改进意见。

答：（1）if （x+y）>2

　 A=x

　 else

　 A=y

　 改为：

　 if M\*（x+y）+T>2

　 A=X

　 else

　 A=y

　 结果一致。

（2）两次结构一致。

（3）WHILE DO 和IF ELSE修改技术类似。

（4）优点是可以使得结果更为精确；缺点是并不适用所有程序，有时会使得程序

运行变得十分复杂。

8．对一个包含10000条机器指令的程序进行一个月集成测试后，总共改正了15个错误，此时MTTF=10h；经过两个月测试后，总共改正了25个错误（第二个月改正了10个错误），MTTF=15h。

要求：

（1）根据上述数据确定MTTF与测试时间之间的函数关系，画出MTTF与测试时间τ的关系曲线。在画这条曲线时做了什么假设？

（2）为做到MTTF=100h，必须进行多长时间的集成测试？当集成测试结束时总共改正了多少个错误，还有多少个错误潜伏在程序中？

答：（1）MTTF与测试时间τ的关系曲线，如图7-12所示。

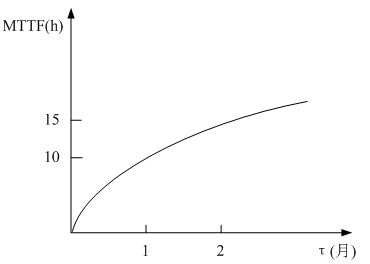


图7-12 MTTF与测试时间τ的关系

（2）根据估算平均无故障时间的公式可得：

1/K（ET/10000-15/10000）=10

1/K（ET/10000-25/10000）=15

计算可得：K=33.3，ET=45

当MTTF=100h时，有

1/333（45/10000-Ec/10000）=100

计算可得，Ec=42，按前两个月测试改错的进度估算，需进行3个月的集成测试。

当测试结束时，共改正了42个错误，还有3个错误潜伏在程序中。

9．如对一个长度为100000条指令的程序进行集成测试期间记录下下面的数据：

（a）7月1日：集成测试开始，没有发现错误。

（b）8月2日：总共改正100个错误，此时MTTF=0.4h

（c）9月1日：总共改正300个错误，此时，MTTF=2h

根据上列数据完成下列各题：

（1）估计程序中的错误总数。

（2）为使MTTF达到10h，必须测试和调试这个程序多长时间？

（3）画出MTTF和测试时间τ之间的函数关系曲线。

答：（1）根据估算平均无故障时间的公式可得：

1/K（ET/100000-100/100000）=0.4

1/K（ET/100000-300/100000）=2

计算可得：K＝1000，ET=350，即程序中的错误总数为350。

（2）当MTTF＝10h时，有

1/K（350/100000- Ec/100000）=10

计算可得，Ec=340，按前两个月测试改错的进度估算，还进行2个月的集成测试。

（3）MTTF和测试时间τ之间的函数关系曲线，如图7-13所示。

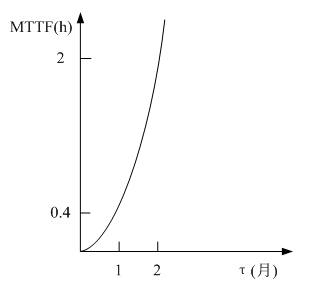


图7-13 MTTF与测试时间τ的关系

10．在测试一个长度为24000条指令的程序时，第一个月由甲、乙两名测试员各自独立测试这个程序。经一个月测试后，甲发现并改正20个错误，使MTTF达到10h。与此同时，乙发现24个错误，其中6个甲也发现了。以后由甲一个人继续测试这个程序。问：

（1）刚开始测试时程序中总共有多少个潜藏的错误？

（2）为使MTTF达到60h，必须再改正多少个错误？还需用多长测试时间？

（3）画出MTTF与集成测试时间τ之间的函数关系曲线。

答：（1）根据公式：B0=B2B1/bc，可得：

B0＝20\*24/6=80，即刚开始测试时程序中总共有80个错误。

（2）根据估算平均无故障时间的公式可得：

1/K（80/24000-20/24000）=10

1/K（80/24000- Ec/24000）=60

计算可得，K=40、Ec=70，即还需要改正50个错误。根据甲改正进度，还需要3个月。

（3）MTTF与集成测试时间τ之间的函数关系曲线，如图7-14所示。



图7-14 MTTF与测试时间τ的关系

**第8章　维　护**

**8.1　复习笔记**

一、软件工程的目的

软件工程的主要目的就是要提高软件的可维护性，减少软件维护所需要的工作量，降低软件系统的总成本。

二、软件维护的概念

1．定义

软件维护是在软件已经交付使用后，为了改正错误或满足新的需要而修改软件的过程，是软件生命周期的最后一个阶段，其基本任务是保证软件在一个相当长的时期能够正常运行。

2．分类

（1）改正性维护

在任何大型程序的使用期间，用户必然会发现程序错误，并且把他们遇到的问题报告给维护人员。把诊断和改正错误的过程称为改正性维护。

（2）适应性维护

适应性维护，也就是为了和变化了的环境适当地配合而进行的修改软件的活动，是既必要又经常的维护活动。

（3）完善性维护

为了满足用户在使用团建过程中提出的新功能或者修改已有功能的建议，需要进行完善性维护。这项维护活动通常占软件维护工作的大部分。

（4）预防性维护

当为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件，这项维护活动通常称为预防性维护。

注意：上述4类维护活动都必须应用于整个软件配置，维护软件文档与维护软件的可执行代码是同样重要的。

三、软件维护的特点

1．结构化维护与非结构化维护差别巨大

（1）非结构化维护

①流程

a．软件配置的唯一成分是程序代码，维护活动从艰苦地评价程序代码开始；

b．由于程序内部文档不足，而对于软件结构、全程数据结构、系统接口、性能和设计约束等会产生误解；

c．对程序代码所做的改动的后果是难于估量的；

d．没有测试方面的文档，不可能进行回归测试。

②特点

非结构化维护需要付出很大代价，这种维护方式是没有使用良好定义的方法学开发出来的软件的必然结果。

（2）结构化维护

①流程

a．有一个完整的软件配置存在，维护工作从评价设计文档开始；

b．确定软件重要的结构特点、性能特点以及接口特点；

c．估量要求的改动将带来的影响，并且计划实施途径；

d．修改设计并且对所做的修改进行仔细复查；

e．编写相应的源程序代码；

f．使用在测试说明书中包含的信息进行回归测试；

g．把修改后的软件再次交付使用。

②特点

结构化维护是在软件开发的早期应用软件工程方法学的结果。有了软件的完整配置能减少精力的浪费并且能提高维护的总体质量。

2．维护的代价高昂

软件维护中无形的代价有：

（1）因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机。

（2）当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满。

（3）由于维护时的改动，在软件中引入了潜伏的错误，从而降低了软件的质量。

（4）当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱。

（5）生产率的大幅度下降。

3．维护存在的问题

与软件维护有关的绝大多数问题，都可归因于软件定义和软件开发的方法有缺点。在软件生命周期的头两个时期没有严格而又科学的管理和规划，几乎必然会导致在最后阶段出现问题。与软件维护有关的部分问题如下：

（1）理解别人写的程序非常困难，而且困难程度随着软件配置成分的减少而迅速增加。

（2）需要维护的软件往往没有合格的文档，或者文档资料显著不足。

（3）当要求对软件进行维护时，不能指望由开发人员给人们仔细说明软件。

（4）绝大多数软件在设计时没有考虑将来的修改。

（5）软件维护不是一项吸引入的工作。

四、软件维护过程

1．定义

维护过程本质上是修改和压缩了的软件定义和开发过程。软件维护过程可以描述为：

（1）建立一个维护组织；

（2）确定报告和评价的过程；

（3）为每个维护要求规定一个标准化的事件序列；

（4）建立一个适用于维护活动的记录保管过程；

（5）规定复审标准。

2．具体过程

（1）维护组织

每个维护要求都通过维护管理员转交给熟悉该产品的系统管理员去评价。系统管理员是被指定去熟悉一小部分产品程序的技术人员。系统管理员对维护任务做出评价之后，由变化授权人决定应该进行的活动。

（2）维护报告

①维护要求表

软件维护人员通常给用户提供空白的维护要求表（软件问题报告表），这个表格由要求一项维护活动的用户填写。由维护管理员和系统管理员评价用户提交的维护要求表。软件要求表是一个外部产生的文件，它是计划维护活动的基础，应该用标准化的格式表达所有软件维护要求。

②软件修改报告

a．满足维护要求表中提出的要求所需要的工作量。

b．维护要求的性质。

c．这项要求的优先次序。

d．与修改有关的事后数据。

（3）维护的事件流

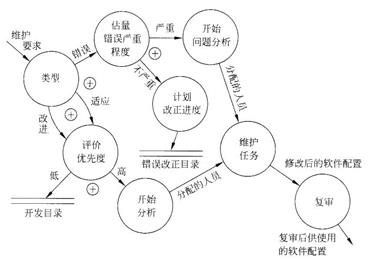


图8-1 维护阶段的事件流

图8-1描绘了由一项维护要求而引出的一串事件。

维护过程需要的技术工作有：

①修改软件设计；

②复查；

③必要的代码修改；

④单元测试和集成测试；

⑤验收测试；

⑥复审；

⑦处境复查。

（4）保存维护记录

在保存维护记录阶段，值得保存的数据：

①程序标识；

②源语句数；

③机器指令条数；

④使用的程序设计语言；

⑤程序安装的日期；

⑥自从安装以来程序运行的次数；

⑦自从安装以来程序失效的次数；

⑧程序变动的层次和标识；

⑨因程序变动而增加的源语句数；

⑩因程序变动而删除的源语句数；

⑪每个改动耗费的人时数；

⑫程序改动的日期；

⑬软件工程师的名字；

⑭维护要求表的标识；

⑮维护类型；

⑯维护开始和完成的日期；

⑰累计用于维护的人时数；

⑱与完成的维护相联系的纯效益。

（5）评价维护活动

①度量标准

a．每次程序运行平均失效的次数。

b．用于每一类维护活动的总人时数。

a．平均每个程序、每种语言、每种维护类型所做的程序变动数。

d．维护过程中增加或删除一个源语句平均花费的人时数。

e．维护每种语言平均花费的人时数。

f．一张维护要求表的平均周转时间。

g．不同维护类型所占的百分比。

②作用

根据对维护工作定量度量的结果，可以做出关于开发技术、语言选择、维护工作量规划、资源分配及其他许多方面的决定，而且可以利用这样的数据去分析评价维护任务。

五、软件的可维护性

1．定义

可维护性指的是维护人员理解、改正、改动或改进这个软件的难易程度。提高可维护性是支配软件工程方法学所有步骤的关键目标。

2．决定软件可维护性的因素

维护就是在软件交付使用后进行的修改，修改之前必须理解待修改的对象，修改之后应该进行必要的测试，以保证所做的修改是正确的。如果是改正性维护，还必须预先进行调试以确定错误的具体位置。决定软件可维护性的因素主要有下述5个：

（1）可理解性

①定义

软件可理解性表现为外来读者理解软件的结构、功能、接口和内部处理过程的难易程度。

②影响因素

a．模块化（模块结构良好，高内聚，松耦合）；

b．详细的设计文档；

c．结构化设计；

d．程序内部的文档；

e．高级程序设计语言等。

（2）可测试性

诊断和测试的容易程度取决于软件容易理解的程度。

①影响因素

a．良好的文档；

b．软件结构；

c．可用的测试工具和调试工具；

d．以前设计的测试过程。

②要求

维护人员需要得到在开发阶段用过的测试方案，以便进行回归测试。在设计阶段应该尽力把软件设计成容易测试和容易诊断的。

③衡量标准

对于程序模块来说，可以用程序复杂度来度量它的可测试性。模块的环形复杂度越大，可执行的路径就越多，全面测试它的难度就越高。

（3）可修改性

耦合、内聚、信息隐藏、局部化、控制域与作用域的关系等，都影响软件的可修改性。

（4）可移植性

①定义

软件可移植性是指把程序从一种计算环境（硬件配置和操作系统）转移到另一种计算环境的难易程度。

②提高可移植性的方法

把与硬件、操作系统以及其他外部设备有关的程序代码集中放到特定的程序模块中，可以把因环境变化而必须修改的程序局限在少数程序模块中，从而降低修改的难度，提高可移植性。

（5）可重用性

①定义

重用是指同一事物不做修改或稍加改动就在不同环境中多次重复使用。

②对可维护性的影响

a．提高软件可靠性，较少改正性维护

可重用的软件构件在开发时都经过很严格的测试，可靠性比较高，且在每次重用过程中都会发现并清除一些错误，构件将变成实质上无错误的。

b．降低适应性和完善性维护的难度

很容易修改可重用的软件构件使之再次应用在新环境中。

3．文档

（1）重要性

文档是影响软件可维护性的决定因素。由于长期使用的大型软件系统在使用过程中必然会经受多次修改，所以文档比程序代码更重要。

（2）特点

①必须描述如何使用这个系统，没有这种描述时即使是最简单的系统也无法使用。

②必须描述怎样安装和管理这个系统。

③必须描述系统需求和设计。

④必须描述系统的实现和测试，以便使系统成为可维护的。

（3）分类

软件系统的文档可以分为用户文档和系统文档两类。

①用户文档

用户文档主要描述系统功能和使用方法，并不关心这些功能是怎样实现的。文档的结构方式应该使用户能够方便地根据需要阅读有关的内容。内容包括：

a．功能描述；

b．安装文档；

c．使用手册；

d．参考手册；

e．操作员指南。

②系统文档

系统文档是指从问题定义、需求说明到验收测试计划这样一系列和系统实现有关的文档。描述系统设计、实现和测试的文档对于理解程序和维护程序来说是极端重要的。

4．可维护性复审

（1）可维护性

可维护性是所有软件都应该具备的基本特点，必须在开发阶段保证软件具有可理解性、可测试性、可修改性、可重用性、可移植性。在软件工程过程的每一个阶段都应该考虑并努力提高软件的可维护性，在每个阶段结束前的技术审查和管理复审中，应该着重对可维护性进行复审。

（2）复审的任务

①需求分析阶段的复审

a．对将来要改进的部分和可能会修改的部分加以注意并指明；

b．讨论软件的可移植性问题，并且考虑可能影响软件维护的系统界面。

②正式的和非正式的设计阶段的复审

a．从容易修改、模块化和功能独立的目标出发，评价软件的结构和过程；

b．对将来可能修改的部分预作准备。

③编码阶段

a．强调编码风格和内部说明文档这两个影响可维护性的因素；

b．尽量使用可重用的软件构件，如开发新的构件，也应该注意提高构件的可重用性。

④测试阶段

a．保证软件配置的所有成分是完整的、一致的和可理解的；

b．为便于修改和管理，进行编目归档。

⑤维护阶段

保证维护是针对整个软件配置，不只是修改源程序代码。

（3）复审的必要性

在软件再次交付使用之前，对软件配置进行严格的复审，则可大大减少文档的问题。事实上，某些维护要求可能并不需要修改软件设计或源程序代码，只是表明用户文档不清楚或不准确，因此只需要对文档做必要的维护。

六、预防性维护

1．定义

预防性维护指的是把今天的方法学应用到昨天的系统上，以支持明天的需求。

2．方法

（1）反复多次地做修改程序的尝试，以实现所要求的修改。

（2）通过仔细分析程序尽可能多地掌握程序的内部工作细节，以便更有效地修改它。

（3）在深入理解原有设计的基础上，用软件工程方法重新设计、重新编码和测试那些需要变更的软件部分，即局部软件再工程。

（4）以软件工程方法学为指导，对程序全部重新设计、重新编码和测试，为此可以使用CASE工具来帮助理解原有的设计，即软件再工程。

3．必要性

（1）维护一行源代码的代价可能是最初开发该行源代码代价的14～40倍。

（2）重新设计软件体系结构时使用了现代设计概念，对将来的维护可能有很大的帮助。

（3）由于现有的程序版本可作为软件原型使用，开发生产率可大大高于平均水平。

（4）用户具有较多使用该软件的经验，能够很容易地搞清新的变更需求和变更的范围。

（5）利用逆向工程和再工程的工具，可以使一部分工作自动化。

（6）在完成预防性维护的过程中可以建立起完整的软件配置。

七、软件再工程过程

1．典型软件再工程模型

典型的软件再工程过程模型如图8-2所示，该模型定义了6类活动。在某些情况下这些活动以线性顺序发生，但也并非总是这样。

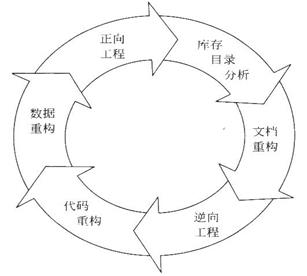


图8-2 软件再工程过程模型

2．六类活动

（1）库存目录分析

①内容

每个软件组织都应该保存其拥有的所有应用系统的库存目录。该目录包含关于每个应用系统的基本信息。

②预防性维护的对象

对库中每个程序都做逆向工程或再工程是不现实的，下述3类程序有可能成为预防性维护的对象。

a．预定将使用多年的程序。

b．当前正在成功地使用着的程序。

c．在最近的将来可能要做重大修改或增强的程序。

③方法

仔细分析库存目录，按照业务重要程度、寿命、当前可维护性、预期的修改次数等标准，把库中的应用系统排序，从中选出再工程的候选者，然后明智地分配再工程所需要的资源。

（2）文档重构

①建立文档非常耗费时间，选择有价值的程序重新建立文档。如果一个程序是相对稳定的，正在走向其有用生命的终点，而且可能不会再经历什么变化，那么，让它保持现状是一个明智的选择。

②采用“使用时建文档”的方法，只针对系统中当前正在修改的那部分建立完整文档。

③对于必须进行重构的关键业务，则应设法把文档工作减少到必需的最小量。

（3）逆向工程

软件的逆向工程是分析程序以便在比源代码更高的抽象层次上创建出程序的某种表示的过程，即逆向工程是一个恢复设计结果的过程，逆向工程工具从现存的程序代码中抽取有关数据、体系结构和处理过程的设计信息。

（4）代码重构

①适用性

针对某些老程序具有比较完整、合理的体系结构，但个体模块的编码方式却是难于理解、测试和维护的情况，可以重构可疑模块的代码。

②流程

a．用重构工具分析源代码，标注出和结构化程序设计概念相违背的部分；

b．然后重构有问题的代码；

c．复审和测试生成的重构代码并更新代码文档。

③特点

a．重构不修改整体的程序体系结构，仅关注个体模块的设计细节和在模块中定义的局部数据结构。

b．重构扩展到模块边界之外并涉及软件体系结构，则变成了正向工程。

（5）数据重构

①定义

数据重构发生在相当低的抽象层次上，是一种全范围的再工程活动。数据重构始于逆向工程活动，分解当前使用的数据体系结构，必要时定义数据模型，标识数据对象和属性，并从软件质量的角度复审现存的数据结构。

②必要性

对数据体系结构差的程序很难进行适应性修改和增强。对应用系统来说，数据体系结构比源代码本身对程序的长期生存力有更大影响。

（6）正向工程

①定义

正向工程（革新或改造）应用软件工程的原理、概念、技术和方法来重新开发某个现有的应用系统。在大多数情况下，被再工程的软件不仅重新实现现有系统的功能，而且加人了新功能和提高了整体性能。

②作用

正向工程不仅从现有程序中恢复设计信息，还使用该信息去改变或重构现有系统，以提高其整体质量。

**8.2　课后习题详解**

1．软件的可维护性与哪些因素有关？在软件开发过程中应该采取哪些措施来提高软件产品的可维护性？

答：（1）软件的可维护性与软件的可理解性、可测试性、可修改性、可重用性、可移植性有关。

（2）提高软件产品的可维护性的措施有如下几点：

①建立明确的软件直连目标。

②使用先进的软件开发技术和工具。

③建立明确的质量保证。

④选择可维护的程序设计语言。

⑤改进程序文档。

2．假设自己的任务是对一个已有的软件做重大修改，而且只允许从下述文档中选取两份：（a）程序的规格说明；（b）程序的详细设计结果（自然语言描述加上某种设计工具表示）；（c）源程序清单（其中有适当数量的注解）。

应选取哪两份文档？为什么这样选取？打算怎样完成交给自己的任务？

答：应该选取（a）和（b）两份文档，理由如下：

（1）“对一个已有的软件做重大修改”意味着对软件功能做较大变更或增加较多新功能，这往往需要修改软件的体系结构。因此，了解原有软件的总体情况是很重要的。程序的规格说明书准确地描述了对软件系统的数据要求、功能需求、性能需求、可靠性和可用性要求、出错处理需求、接口需求、约束、逆向需求及将来可能提出的需求，对了解已有软件的总体情况有很大帮助。在对已有软件做重大修改之前仔细阅读、认真研究这份文档，可以避免许多修改错误。因此，应该选取（a）程序的规格说明。

（2）有经验的软件工程师通过阅读含有适当数量注解的源程序，不难搞清程序的实现算法，没有描述详细设计结果的文档并不会给维护工作带来太大困难。此外，为了修改程序代码，原有程序的清单是必不可少的。因此，应该选取（c）源程序清单。

3．分析预测在下列系统交付使用以后，用户可能提出哪些改进或扩充功能的要求。如果由自己来开发这些系统，在设计和实现时将采取哪些措施，以方便将来的修改？

（1）储蓄系统

（2）机票预订系统

（3）患者监护系统

答：（1）储蓄系统

一般大型软件的维护成本远高于开发成本若干倍，所以在设计时就应该考虑到维护成本，而且在开发过程中应该使用标准的程序设计语言和标准的操作系统接口，可以大大提高软件的可维护性，也可以减少软件存在的错误。

（2）机票预订系统

人多的时候系统瘫痪几率就大，在开发人员不在场的情况下，很容易出现系统错误，维护软件也是很困难的，也会给很多人带来不便，所以，在设计过程中应该科学的管理规划还有合理设计模块，使各个模块的独立性越高，这样对软件的改进越方便，也便于快速纠错。

（3）患者监护系统

应该考虑它的完善性和预防性，要能满足用户在使用过程中的增加和修改工作，还要为了改善未来的可维护性或可靠性而修改。更要考虑系统数据的隐秘和安全，随时备份。

**第9章　面向对象方法学引论**

**9.1　复习笔记**

一、面向对象方法学概述

1．要点

面向对象方法学已经成为人们在开发软件时首选的范型。面向对象技术已成为当前最好的软件开发技术。

（1）基本原则

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，使开发软件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，使描述问题的问题空间（问题域）与实现解法的解空间（求解域）在结构上尽可能一致。

（2）定义

面向对象方法是一种以数据或信息为主线，把数据和处理相结合的方法，即把对象作为由数据及可以施加在这些数据上的操作所构成的统一体。面向对象的方法可以用下列方程来概括：

OO = object + classes + inheritance + communication with messages

（3）要点

①对象

面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的软件对象由比较简单的对象组合而成。用对象分解取代了传统方法的功能分解，对象是从客观世界中的实体抽象而来的，是不固定的

②类

把所有对象都划分成各种对象类，每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性，是对象的状态信息。类中定义的方法，是允许施加于该类对象上的操作，是该类所有对象共享的，并不需要为每个对象都复制操作的代码。

③继承性

按子类与父类的关系，把若干个对象类组成一个层次结构的系统。子类自动具有和上层的父类相同的数据和方法，而且低层的特性将屏蔽高层的同名特性。

④封装性

对象彼此之间仅能通过传递消息互相联系。对象是进行处理的主体，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。一切局部于该对象的私有信息，都被封装在该对象类的定义中，就好像装在一个不透明的黑盒子中一样，在外界是看不见的，更不能直接使用，

2．面向对象方法学的优点

（1）与人类习惯的思维方法一致

①以对象为核心，开发出的软件系统由对象组成。

②设计的主要思路是使用现实世界的概念抽象地思考问题从而自然地解决问题。

③基本原则是按照人类习惯的思维方法建立问题域和求解域模型。

④抽象机制使用户在利用计算机软件系统解决复杂问题时使用习惯的抽象思维工具。

⑤对象分类过程，支持从特殊到一般的归纳思维过程。

⑥继承特性，支持从一般到特殊的演绎思维过程。

⑦提供了随着对系统认识的逐步深入和具体化，而逐步设计和实现该系统的可能性。

（2）稳定性好

面向对象方法基于构造问题领域的对象模型，以对象为中心构造软件系统。它的基本作法是用对象模拟问题领域中的实体，以对象间的联系刻画实体间的联系。因为面向对象的软件系统的结构是根据问题领域的模型建立起来的，而不是基于对系统应完成的功能的分解，当对系统的功能需求变化时不会引起软件结构的整体变化，仅需要作一些局部性的修改。

（3）可重用性好

①定义

重用技术是用已有的零部件装配新的产品。重用是提高生产率的最主要的方法。对象是比较理想的模块和可重用的软件成分。

②方法

a．创建该类的实例，从而直接使用它。

b．从该类派生出一个满足当前需要的新类。

（4）较易开发大型软件产品

面向对象方法学开发软件时，把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，不仅降低了开发的技术难度，而且使开发工作的管理变容易了。

（5）可维护性好

用传统方法和面向过程语言开发出来的软件很难维护，是长期困扰人们的一个严重问题，是软件危机的突出表现。下面几个因素使得面向对象方法所开发的软件可维护性好：

①面向对象的软件稳定性比较好。

②面向对象的软件比较容易修改。

③面向对象的软件比较容易理解。

④易于测试和调试。

二、面向对象的概念

1．对象

（1）对象的形象表示

“对象”是面向对象方法学中使用的最基本的概念，对象的形象表示如图9-1所示。

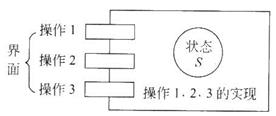


图9-1 对象的形象表示

图9-1形象地描绘了具有3个操作的对象。面向对象方法学中的对象是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。对象可以作的操作表示它的动态行为。

（2）对象的定义

①面向对象程序设计的角度

对象是具有相同状态的一组操作的集合。在应用领域中有意义的、与所要解决的问题有关系的任何事物都可以作为对象，它既可以是具体的物理实体的抽象，也可以是人为的概念，或者是任何有明确边界和意义的东西。

②结构化的角度

对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体，这个封装体有可以唯一地标识它的名字，而且向外界提供一组服务。对象中的数据表示对象的状态，一个对象的状态只能由该对象的操作来改变。每当需要改变对象的状态时，只能由其他对象向该对象发送消息。对象响应消息时，按照消息模式找出与之匹配的方法，并执行该方法。

③动态角度（对象的实现机制）

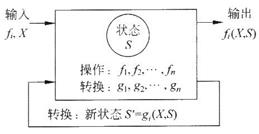


图9-2 用自动机模拟对象

对象是一台自动机。具有内部状态S，操作fi（i=1,2，···，n），且与操作fi对应的状态转换函数为gi（i=1,2，···，n）的一个对象，可以用图9-2所示的自动机来模拟。

（3）对象的特点

①以数据为中心

操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态有关。

②对象是主动的

对象进行处理的主体。不能从外部直接加工它的私有数据，而是必须通过它的公有接口向对象发消息，请求它执行它的某个操作，处理它的私有数据。

③实现了数据封装

私有的数据完全被封装在内部，对外是隐藏的、不可见的，对私有数据的访问或处理只能通过公有的操作进行。为了使用对象内部的私有数据，只需知道数据的取值范围和可以对该数据施加的操作。一个对象类型也可以看作是一种抽象数据类型。

④具有并行性

对象是描述其内部状态的数据及可以对这些数据施加的全部操作的集合。不同对象各自独立地处理自身的数据，彼此通过发消息传递信息完成通信。因此，本质上具有并行工作的属性。

⑤模块独立性好

对象是由数据及可以对这些数据施加的操作所组成的统一体，而且对象是以数据为中心的，操作围绕对其数据所需做的处理来设置，没有无关的操作。因此，对象内部各种元素彼此结合得很紧密，内聚性相当强。它与外界的联系比较少，对象之间的耦合比较松。

2．其他概念

（1）类

“类”是对具有相同数据和相同操作的一组相似对象的定义，即类是对具有相同属性和行为的一个或多个对象的描述，包括对怎样创建该类的新对象的说明。类是支持继承的抽象数据类型，而对象就是类的实例。

（2）实例

实例就是由某个特定的类所描述的一个具体的对象。类是对具有相同属性和行为的一组相似的对象的抽象，类在现实世界中并不能真正存在。实际上类是建立对象时使用的“样板”，按照这个样板所建立的一个个具体的对象，就是类的实际例子，通常称为实例。

（3）消息

消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。一个消息由接收消息的对象、消息选择符、零个或多个变元组成。

（4）方法

方法就是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，响应消息的方法。

（5）属性

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例都有自己特有的属性值。

（6）封装

①定义

封装是把数据和实现操作的代码集中起来放在对象内部。封装也就是信息隐藏，通过封装对外界隐藏了对象的实现细节。

②特点

a．有一个清晰的边界

有一个清晰的边界。所有私有数据和实现操作的代码都被封装在这个边界内，从外面看不见更不能直接访问。

b．有确定的接口（即协议）

这些接口就是对象可以接受的消息，只能通过向对象发送消息来使用它。

c．受保护的内部实现

实现对象功能的细节（私有数据和代码）不能在定义该对象的类的范围外访问。

注意：对象类实质上是抽象数据类型。类把数据说明和操作说明与数据表达和操作实现分离开了，使用者只需知道它的说明，就可以使用它。

（7）继承

①定义

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是子类自动地共享基类中定义的数据和方法的机制。

②原理

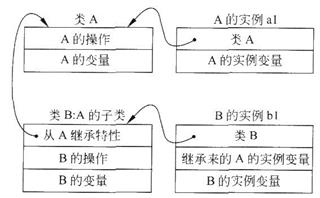


图9-3 实现继承机制的原理

图9-3中以A、B两个类为例，B类是从A类派生出来的子类，它除了具有自己定义的特性之外，还从父类A继承特性。创建A类的实例a1时，a1以A类为样板建立实例变量，并不从A类中复制所定义的方法。创建B类的实例b1的时候，b1既要以B类为样板建立实例变量，又要以A类为样板建立实例变量，b1所能执行的操作既有B类中定义的方法，又有A类中定义的方法，这就是继承。

③特点

a．继承具有传递性；

b．低层的性质将屏蔽高层的同名性质。

④分类

a．单继承：一个类只允许有一个父类，即类等级为树形结构。

b．多重继承：允许一个类有多个父类。

⑤实现软件重用

a．使用抽象的类开发出一般性问题的解；

b．在派生类中增加少量代码使一般性的解具体化。

（8）多态性

①定义

多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象也可以发送给子类对象。即在类等级的不同层次中可以共享一个方法的名字，不同层次中的每个类各自按自己的需要来实现这个行为。

②动态联编

在运行时刻根据接收消息的对象所属于的类，决定执行哪个特定的版本，这称为动态联编，也叫滞后联编。

③优点

a．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。

b．显著提高了软件的可重用性和可扩充性。

（9）重载

①分类

a．函数重载

在同一作用域内的若干个参数特征不同的函数可以使用相同的函数名字。

b．运算符重载

同一个运算符可以施加于不同类型的操作数上面。

当参数特征不同或被操作数的类型不同时，实现函数的算法或运算符的语义是不相同的。

②实现

a．函数重载是通过静态联编（先前联编）实现的。

b．运算符重载是在编译时根据被操作数的类型，决定使用该算符的哪种语义。

③优点

进一步提高了面向对象系统的灵活性和可读性。

三、面向对象建模

1．模型

（1）定义

模型是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。模型由一组图示符号和组织这些符号的规则组成，利用它们来定义和描述问题域中的术语和概念。

（2）作用

①模型是一种思考工具，利用这种工具可以把知识规范地表示出来。

②模型可以帮助人们思考问题、定义术语、在选择术语时作出适当的假设，并且有助于保持定义和假设的一致性。

2．建模的重要性

由于建模忽略了事物的非本质属性，因此模型比原始事物更容易操作。对于那些因过分复杂而不能直接理解的系统，特别需要建立模型，模型通过把系统的重要部分分解成人的头脑一次能处理的若干个子部分，从而减少了系统的复杂程度。

3．模型分类

（1）对象模型：描述系统数据结构。

（2）动态模型：描述系统控制结构。

（3）功能模型：描述系统功能。

这3种模型都涉及数据、控制和操作等共同的概念，只不过每种模型描述的侧重点不同。这3种模型从3个不同但又密切相关的角度模拟目标系统，它们各自从不同侧面反映了系统的实质性内容，综合起来则全面地反映了对目标系统的需求。

四、对象模型

1．概念

（1）定义

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以及对象彼此间的关系的映射，描述了系统的静态结构。对象模型为建立动态模型和功能模型，提供了实质性的框架。

（2）工具

使用UML提供的类图来建立对象模型。在UML中，类的实际含义是一个类及属于该类的对象。

2．类图的基本符号

类图描述类及类与类之间的静态关系。类图是一种静态模型，它是创建其他UML图的基础。一个系统可以由多张类图来描述，一个类也可以出现在几张类图中。

（1）定义类

①表示



图9-4 表示类的图形符号

UML中类的图形符号为长方形，用两条横线把长方形分戒上、中、下3个区域，3个区域分别放类的名字、属性和服务，如图9-4所示。

②命名规则

类名是一类对象的名字。命名是否恰当对系统的可理解性影响相当大，因此，为类命名时应该遵守以下几条准则：

a．使用标准术语。应该使用在应用领域中人们习惯的标准术语作为类名，不要随意创造名字。

b．使用具有确切含义的名词。尽量使用能表示类的含义的13常用语作名字，不要使用空洞的或含义模糊的词作名字。

c．必要时用名词短语作名字。为使名字的含义更准确，必要时用形容词加名词或其他形式的名词短语作名字。

注意：类名应该是富于描述的、简洁的而且无二义性的。

（2）定义属性

UML描述属性的语法格式如下：

可见性 属性名：类型名=初值{性质串}

①可见性

属性的可见性分为公有的（+）、私有的（-）和保护的（#），如果未声明可见性，则表示该属性的可见性尚未定义。

注意：没有默认的可见性。

②类型名

表示该属性的数据类型，它可以是基本数据类型或是是用户自定义的类型。属性名和类型名之间用冒号（：）分隔。

③赋值

在创建类的实例时应给其属性赋值，如果给某个属性定义了初值，则该初值可作为创建实例时这个属性的默认值。类型名和初值之间用（=）隔开。

④性质串

性质串明确地列出该属性所有可能的取值，不同值之间用逗号隔开。性质串用花括号（{}）括起来。

（3）定义服务

服务也就是操作，UML描述操作的语法格式如下：

可见性 操作名（参数表）：返回值类型{性质串}

①可见性

操作的可见性分为公有的（+）、私有的（-）和保护的（#），如果未声明可见性，则表示该操作的可见性尚未定义。

注意：没有默认的可见性。

②参数表

用逗号分隔的形式参数的序列。描述一个参数的语法如下：

参数名：类型名=默认值

在类中也可以定义类作用域操作，在类图中表示带下划线的操作。这类操作只能存取本类的类作用域属性。

3．表示关系的符号

类与类之间通常有关联、泛化（继承）、依赖和细化4种关系。

（1）关联

①定义

关联表示两个类的对象之间存在某种语义上的联系。

②关联的角色



图9-5 关联的角色

在任何关联中都会涉及参与此关联的对象所扮演的角色，在某些情况下显式标明角色名有助于别人理解类图。如图9-5所示。如果没有显式标出角色名，则意味着用类名作为角色名。

③普通关联

a．定义

普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示。

b．表示

IMG_448

图9-6 普通关联示例

第一，普通关联的图示符号是连接两个类之间的直线，如图9-6所示。

第二，关联是双向的，可为关联起一个名字。在名字前面（或后面）加一个表示关联方向的黑三角。

第三，在表示关联的直线两端可以写上重数，它表示该类有多少个对象与对方的一个对象连接。未明确标出关联的重数，则默认重数是1。

④限定关联

a．定义

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一，或从多对多简化成多对一。

b．表示

IMG_449

图9-7 一个受限的关联

在类图中把限定词放在关联关系末端的一个小方框内。如图9-7所示，利用限定词“文件名”表示了目录与文件之间的关系，利用限定词把一对多关系简化成了一对一关系。

c．意义

限定提高了语义精确性，增强了查询能力。

⑤关联类

a．定义

为了说明关联的性质，可能需要一些附加信息。关联类可以用来记录相关信息。

b．表示

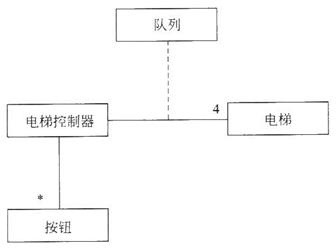


图9-8 关联类示例

如图9-8所示。关联中的每个连接与关联类的一个对象相联系。关联类通过一条虚线与关联连接。

（2）聚集

聚集（聚合）是关联的特例。表示类与类之间的关系是整体与部分的关系。在陈述需求时使用的“包含”、“组成”、“分为……部分”等字句，往往意味着存在聚集关系。除了一般聚集之外，还有两种特殊的聚集关系，分别是共享聚集和组合聚集。

①共享聚集

a．定义

如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集。

b．表示

IMG_451

图9-9 共享聚集示例

如图9-9所示。一般聚集和共享聚集的图示符号，都是在表示关联关系的直线末端紧挨着整体类的地方画一个空心菱形。

②组合聚集

a．定义

如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失，则该聚集称为组合聚集（组成）。

b．表示

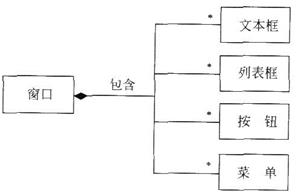


图9-10 组合聚集示例

如图9-10所示，组成关系用实心菱形表示。

（3）泛化

UML中的泛化关系就是继承关系，它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。在UML中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素。

①普通泛化

a．抽象类

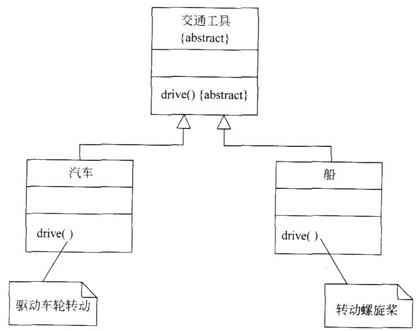


图9-11 抽象类示例

没有具体对象的类称为抽象类。抽象类通常都有抽象操作，作用来指定该类的所有子类应具有哪些行为。如图9-11所示，表示抽象类是在类名下方附加一个标记值{abstract}，表示抽象操作是在操作标记后面跟随一个性质串{abstract}。

b．具体类

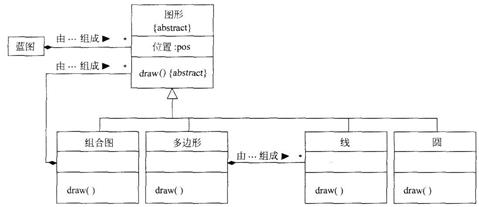


图9-12 复杂类图示例

具体类有自己的对象，并且该类的操作都有具体的实现方法。如图9-12所示，是一个比较复杂的类图示例。

②受限泛化

a．定义

可以给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化。

b．约束

预定义的约束有4种：多重、不相交、完全和不完全。这些约束都是语意约束。

c．继承分类

第一，多重继承

一个子类可以同时多次继承同一个上层基类，如图9-13所示。

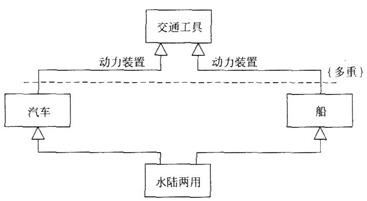


图9-13 多重继承示例

第二，不相交继承

一个子类不能多次继承同一个基类。一般的继承都是不相交继承。

第三，完全继承

父类的所有子类都已在类图中穷举出来了。

第四，不完全继承

父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护。是默认的继承关系。

（4）依赖和细化

①依赖关系

a．定义

依赖关系描述两个模型元素之间的语义连接关系：其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素。

b．表示

IMG_456

图9-14 友元依赖关系

如图9-14所示，在UML类图中用带箭头的虚线连接有依赖关系的两个类，箭头指向独立的类。在虚线上可以带一个版类标签，具体说明依赖的种类。

②细化关系

a．定义

对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系。

b．表示

IMG_457

图9-15 细化关系示例

如图9-15所示，细化的图示符号为由元素B指向元素A的一端为空心三角形的虚线。

c．适用性

细化用来协调不同阶段模型之间的关系，表示各个开发阶段不同抽象层次的模型之间的相关性，常常用于跟踪模型的演变。

六、动态模型

1．概念

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。

2．建模

用UML提供的状态图来描绘对象的状态、触发状态转换的事件以及对象的行为。每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型，即动态模型是基于事件共享而互相关联的一组状态图的集合。

七、功能模型

1．概念

（1）定义

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。

（2）组成

功能模型由一组数据流图组成。

2．用例图

UML提供的用例图也是进行需求分析和建立功能模型的强有力工具。在UML中把用用例图建立起来的系统模型称为用例模型。

（1）定义

用例模型描述的是外部行为者所理解的系统功能。用例模型的建立是系统开发者和用户反复讨论的结果，它描述了开发者和用户对需求规格所达成的共识。

（2）表示

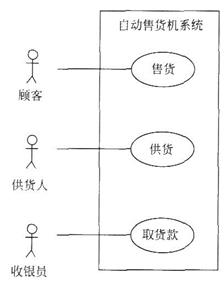


图9-16 自动售货机系统用例图

图9-16是自动售货机系统的用例图。

①系统

a．定义

系统被看作是一个提供用例的黑盒子，内部如何工作、用例如何实现，这些对于建立用例模型来说都是不重要的。

b．表示

系统用方框表示，其边线表示系统的边界，用于划定系统的功能范围，定义了系统所具有的功能。描述该系统功能的用例置于方框内，代表外部实体的行为者置于方框外。

②用例

a．定义

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。在UML中把用例定义成系统完成的一系列动作。

b．表示

在UML中，椭圆代表用例。用例通过关联与行为者连接，关联指出一个用例与哪些行为者交互，这种交互是双向的。

c．特征

第一，用例代表某些用户可见的功能，实现一个具体的用户目标。

第二，用例总是被行为者启动的，并向行为者提供可识别的值。

第三，用例必须是完整的。

注意：用例是一个类，它代表一类功能而不是使用该功能的某个具体实例。用例的实例是系统的一种实际使用方法，通常把用例的实例称为脚本。脚本是系统的一次具体执行过程。

③行为者

a．定义

行为者是指与系统交互的人或其他系统，它代表外部实体。使用用例并且与系统交互的任何人或物都是行为者。行为者代表一种角色，而不是某个具体的人或物。

b．表示

在UML中，线条人代表行为者。在用例图中用直线连接行为者和用例，表示两者之间交换信息，称为通信联系。行为者触发用例，并与用例交换信息。单个行为者可与多个用例联系，一个用例也可与多个行为者联系。

④用例间的关系

UML用例之间主要有扩展和使用两种关系，它们是泛化关系的两种不同形式。

a．扩展关系

向一个用例中添加一些动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者继承前者的一些行为，通常把后者称为扩展用例。如图9-17所示，用例之间的扩展关系图示为带版类《扩展》的泛化关系。

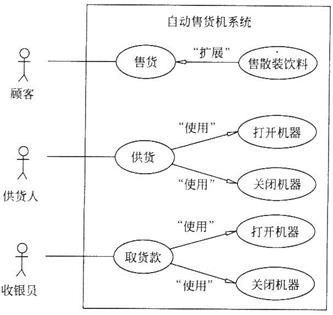


图9-17 含扩展和使用关系的用例图

b．使用关系

一个用例使用另一个用例时，这两个用例之间就构成了使用关系。如图9-17所示，用例之间的使用关系用带版类《使用》的泛化关系表示。

c．两种关系的异同

第一，都是从几个用例中抽取那些公共的行为并放入一个单独的用例中，而这个用例被其他用例使用或扩展。

第二，使用和扩展的目的是不同的。在描述一般行为的变化时采用扩展关系。

第三，在两个或多个用例中出现重复描述又想避免这种重复时，采用使用关系。

3．用例建模

（1）创建用例模型

用例集中的每个用例都是对系统的一个潜在的需求，一个用例模型由若干幅用例图组成。创建用例模型的工作包括：定义系统，寻找行为者和用例，描述用例，定义用例之间的关系，确认模型。其中，寻找行为者和用例是关键。

（2）寻找行为者和用例

①寻找行为者

为获取用例首先要找出系统的行为者，可以通过请系统的用户回答一些问题的办法来发现行为者。

②寻找用例

一旦找到了行为者，就可以通过请每个行为者回答下述问题来获取用例：

a．行为者需要系统提供哪些功能？行为者自身需要做什么？

b．行为者是否需要读取、创建、删除、修改或存储系统中的某类信息？

c．系统中发生的事件需要通知行为者吗？行为者需要通知系统某些事情吗？从功能观点看，这些事件能做什么？

d．行为者的日常工作是否因为系统的新功能而被简化或提高了效率？

七、三种模型比较

1．三种模型

（1）三种模型相互补充、相互配合，使得人们对系统的认识更加全面。

（2）对象模型是最基本最重要的，它为其他两种模型奠定了基础。

2．关系

（1）针对每个类建立的动态模型，描述了类实例的生命周期或运行周期。

（2）状态转换驱使行为发生，这些行为在数据流图中被映射成处理，在用例图中被映射成用例，它们同时与类图中的服务相对应。

（3）功能模型中的处理对应于对象模型中的类所提供的服务。

（4）数据流图中的数据存储，以及数据的源点/终点，通常是对象模型中的对象。

（5）数据流图中的数据流，往往是对象模型中对象的属性值，也可能是整个对象。

（6）用例图中的行为者，可能是对象模型中的对象。

（7）功能模型中的处理可能产生动态模型中的事件。

（8）对象模型描述了数据流图中的数据流、数据存储以及数据源点/终点的结构。

**9.2　课后习题详解**

1．什么是面向对象方法学？它有哪些优点？

答：（1）面向对象方法学是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类解决问题的方法与过程，使描述问题的问题空间与实现解法的解空间在结构上尽可能一致的方法学科。

（2）面向对象方法学优点：

①与人类习惯的思维方法一致。

②面向对象软件稳定性好。

③面向对象软件可重用性好。

④较易开发大型软件产品。

⑤可维护性好。

2．什么是“对象”？它与传统的数据有何异同？

答：（1）对象是对问题域中某个实体的抽象。

（2）相对于传统数据结构的静态被处理，对象既有静态的属性，也有动态的行为，是进行处理的主体。

3．什么是“类”？

答：类是对具有相同数据结构和相同操作的一组组相似对象的定义，即类是对具有相同属性和行为的一个或多个对象的描述，包括对怎样创建该类的新对象的说明。

4．什么是“继承”？

答：继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是子类自动地共享基类中定义的数据和方法的机制。

5．什么是模型？开发软件为何要建模？

答：（1）模型是为了理解事物而对事物作出的一种抽象，是对事物的一种无歧义的书面描述。模型由一组图示符号和组织这些符号的规则组成，利用它们来定义和描述问题域中的术语和概念。模型是一种思考工具，利用这种工具可以把知识规范地表示出来。

（2）由于建模忽略了事物的非本质属性，因此模型比原始事物更容易操作。对于那些因过分复杂而不能直接理解的系统，特别需要建立模型，模型通过吧系统的重要部分分解成人的头脑一次能处理的若干个子部分，从而减少了系统的复杂程度。

6．什么是对象模型？建立对象模型时主要使用哪些图形符号？这些符号的含义是什么？

答：（1）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以及对象彼此间的关系的映射，描述了系统的静态结构。

（2）通常使用UML提供的类图来建立对象模型。

（3）在UML中“类”的实际含义是。一个类以及属于该类的对象。

7．什么是动态模型？建立动态模型时主要使用哪些图形符号？这些符号的含义是什么？

答：（1）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对象的合法变化序列。

（2）在UML中，使用状态图和事件追踪图来建立动态模型。

（3）其符号的含义为：

①状态图表示需要考察的对象的动态行为。

②事件追踪图表示其运行规律和行为规则。

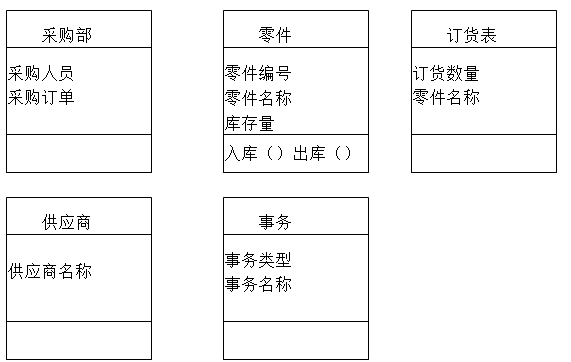
8．什么是功能模型？建立功能模型时主要使用哪些图形符号？

答：（1）功能模型表示软件系统的功能性质，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。

（2）在UML中主要使用用例图来建立功能模型，一幅用例图包括的模型元素有系统、行为者、用例以及用例之间的关系。

9．试用面向对象观点分析、研究第2章中给出的订货系统的例子。在这个例子中有哪些类？试建立订货系统的对象模型。

答：订货系统中的类如下：



10．建立订货系统的用例模型。

答：仓库管理员通过放在仓库中的终端把零件入库或出库事务报告给订货系统，系统接收到事务信息之后应该处理事务；采购员需要使用订货系统提供的产生报表功能，以获取订货报表。用例图如图9-18所示。

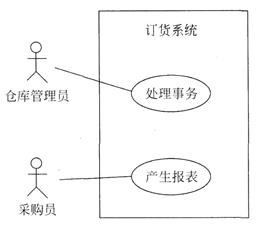


图9-19 订货系统用例图

**第10章　面向对象分析**

**10.1　复习笔记**

一、面向对象分析的基本过程

1．概述

（1）定义

面向对象分析，就是抽取和整理用户需求并建立问题域精确模型的过程。

（2）过程

①分析陈述用户需求的文件

a．作用

通过分析需求文件，可以发现和改正原始陈述中的二义性和不一致性，补充遗漏的内容，从而使需求陈述更完整、更准确。

b．方法

第一，需要反复多次地与用户协商、讨论、交流信息。

第二，通过调研了解现有的类似系统。

第三，快速建立起一个可在计算机上运行的原型系统，有助于正确地提炼出用户的需求。

②深入理解用户需求，抽象出目标系统的本质属性；

③用模型准确地表示。

（3）目的

全面深入地理解问题域和用户的真实需求，建立问题域的精确模型。

2．3个子模型与5个层次

（1）3个模型

面向对象建模得到的模型包含系统的三个要素，即静态结构、交互次序和数据变换。

①静态结构（对象模型）

解决任一问题，都需要从客观世界实体及实体间相互关系抽象出极有价值的对象模型。

②交互次序（动态模型）

当问题涉及交互作用和时序时，动态模型是重要的。

③数据变换（功能模型）

解决运算量很大的问题，则涉及重要的功能模型。

（2）5个层次

复杂问题的对象模型由5个层次组成：主题层、类与对象层、结构层、属性层和服务层。这5个层次一层比一层显现出对象模型的更多细节。如图10-1所示。



图10-1 复杂问题的对象模型的5个层次

①主题

主题是指导读者理解大型、复杂模型的一种机制。即通过划分主题把一个大型、复杂的对象模型分解成几个不同的概念范畴。

②7±2原则

心理研究表明，人类的短期记忆能力一般限于一次记忆5～9个对象，这就是著名的7±2原则。面向对象分析从两个方面来体现这条原则：

a．控制可见性：控制读者能见到的层次数目来控制可见性。

b．指导读者注意力：增加了主题层，可从高层次描述总体模型，并指导读者的注意力。

③面向对象分析顺序

a．寻找类与对象；

b．识别结构；

c．识别主题；

d．定义属性；

e．建立动态模型；

f．建立功能模型；

g．定义服务。

④建模要点

a．面向对象分析不可能严格按照顺序线性进行。

b．必须在反复分析中对初始模型中不准确、不完整和错误的内容加以扩充和更正。

c．仔细研究类似问题的分析结果，尽可能重用这些结果。

二、需求陈述

1．内容

（1）阐明“做什么”而不是“怎样做”。

（2）描述用户的需求而不是提出解决问题的方法。

（3）指出哪些是系统必要的性质，哪些是任选的性质。

（4）避免对设计策略施加过多的约束，不描述系统的内部结构。

（5）描述系统性能及系统与外界环境交互协议。

（6）描述采用的软件工程标准、模块构造准则、将来的扩充以及可维护性要求等方面。

2．书写规范

（1）做到语法正确，而且应该慎重选用名词、动词、形容词和同义词。

（2）必须把需求与实现策略区分开，后者不是问题域的本质性质。

（3）需求陈述可简可繁。

（4）避免出现具有二义性的、不完整的、不一致的内容。

三、建立对象模型

1．概念

（1）对象模型

面向对象分析的首要工作，是建立问题域的对象模型。对象模型描述了现实世界中的“类与对象”以及它们之间的关系，表示了目标系统的静态数据结构。

（2）先建立对象模型的原因

①静态数据结构对应用细节依赖较少，比较容易确定。

②当用户的需求变化时，静态数据结构相对来说比较稳定。

（3）信息来源

需求陈述、应用领域的专业知识、客观世界的常识，是建立对象模型时的主要信息来源。

（4）典型的建模步骤

①确定对象类和关联（对于大型复杂问题还要进一步划分出若干个主题）；

②给类和关联增添属性，以进一步描述它们；

③使用适当的继承关系进一步合并和组织类。

2．确定类与对象

（1）找出候选的类与对象

①客观事物分类

对象是对问题域中有意义的事物的抽象，它们既可能是物理实体，也可能是抽象概念。客观事物可分为下述5类：

a．可感知的物理实体；

b．人或组织的角色；

c．应该记忆的事件；

d．两个或多个对象的相互作用；

e．需要说明的概念。

②非正式分析

以用自然语言书写的需求陈述为依据，把陈述中的名词作为类与对象的候选者，用形容词作为确定属性的线索，把动词作为服务的候选者。这种方法确定的候选者是非常不准确的，其中往往包含大量不正确或不必要的事物，需要经过更进一步的严格筛选。

③提取隐含的类与对象

（2）筛选出正确的类与对象

筛选时主要依据下列标准，删除不正确或不必要的类与对象。

①如果两个类表达了同样的信息，则应该保留在此问题域中最富于描述力的名称。

②需要把与本问题密切相关的类与对象放进目标系统中。

③系统无须记忆笼统的、泛指的名词信息，把这些笼统的或模糊的类去掉。

④把描述的是其他对象属性的词从候选类与对象中去掉。

⑤慎重考虑既可作为名词，又可作为动词的词，以便正确地决定把它们作为类还是作为类中定义的操作。本身具有属性，需独立存在的操作，应该作为类与对象。

⑥应该去掉仅和实现有关的候选的类与对象。

3．确定关联

（1）关联

①定义

两个或多个对象之间的相互依赖、相互作用的关系就是关联。在需求陈述中使用的描述性动词或动词词组，通常表示关联关系。

②确定关联的重要性

分析确定关联，能促使分析员考虑问题域的边缘情况，有助于发现尚未被发现的类与对象。

注意：在分析确定关联的过程中，不必花过多的精力去区分关联和聚集，聚集是一种特殊的关联，是关联的一个特例。

（2）步骤

①初步确定关联

a．直接提取动词短语得出的关联。

b．需求陈述中隐含的关联。

c．根据问题域知识得出的关联。

②筛选

筛选时主要根据下述标准删除候选的关联：

a．已删去的类之间的关联

如果在分析确定类与对象的过程中已经删掉了某个候选类，则与这个类有关的关联也应该删去，或用其他类重新表达这个关联。

b．与问题无关的或应在实现阶段考虑的关联

应该把处在本问题域之外的关联 与实现密切相关的关联删去。

c．瞬时事件

关联应该描述问题域的静态结构，而不应该是一个瞬时事件

d．三元关联

三个或三个以上对象间的关联，可分解为二元关联或用词组描述成限定的关联。

e．派生关联

去掉那些可以用其他关联定义的冗余关联。

③改进

可以从以下几个方面进一步完善经筛选后余下的关联。

a．正名：仔细选择含义更明确的名字作为关联名。

b．分解：为了能够适用于不同的关联，必要时应该分解以前确定的类与对象。

c．补充：发现了遗漏的关联就应该及时补上。

d．标明重数：应该初步判定各个关联的类型，并粗略地确定关联的重数。

4．划分主题

（1）定义

在开发大型、复杂系统的过程中，为了降低复杂程度，把系统再进一步划分成几个不同的主题，即在概念上把系统包含的内容分解成若干个范畴。

（2）针对不同类型的方法

①规模小的系统

可能无须引入主题层。

②含有较多对象的系统

首先识别出类与对象和关联，然后划分主题，并用它作为指导开发者和用户观察整个模型的一种机制。

③规模大的系统

首先由高级分析员粗略地识别对象和关联，然后初步划分主题，经进一步分析，对系统结构有更深入的了解之后，再进一步修改和精炼主题。

（3）原则

①按问题领域而不是用功能分解方法来确定主题。

②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。

5．确定属性

（1）属性

属性是对象的性质，借助于属性人们能对类与对象和结构有更深入更具体的认识。

注意：在分析阶段不要用属性来表示对象间的关系，使用关联能够表示两个对象间的任何关系，而且把关系表示得更清晰、更醒目。

（2）确定属性步骤

①分析

a．在需求陈述中用名词词组表示属性，用形容词表示可枚举的具体属性。

b．借助于领域知识和常识分析需要的属性。

c．仅考虑与具体应用直接相关的属性，不要考虑那些超出所要解决的问题范围的属性。

d．首先找出最重要的属性，以后再逐渐把其余属性增添进去。

e．不要考虑那些纯粹用于实现的属性。

②选择

从初步分析确定下来的属性中删掉不正确的或不必要的属性。有以下几种常见情况：

a．误把对象当作属性

如果某个实体的独立存在比它的值更重要，则应把它作为一个对象而不是对象的属性。

b．误把关联类的属性当作一般对象的属性

如果某个性质依赖于某个关联链的存在，则该性质是关联类的属性，在分析阶段不应把它作为一般对象的属性。

c．把限定误当成属性

如果把某个属性值固定下来以后能减少关联的重数，则应该考虑把这个属性重新表达成一个限定词。

d．误把内部状态当成了属性

如果某个性质是对象的非公开的内部状态，则应该从对象模型中删除这个属性。

e．过于细化

在分析阶段应该忽略那些对大多数操作都没有影响的属性。

f．存在不一致的属性

类应该是简单而且一致的。如果得出一些看起来与其他属性毫不相关的属性，则应该考虑把类分解成两个不同的类。

6．识别继承关系

（1）建立继承关系的方式

确定了类中应该定义的属性之后，就可以利用继承机制共享公共性质，并对系统中众多的类加以组织。可以使用以下两种方式建立继承关系。

①自底向上

抽象出现有类的共同性质泛化出父类，这个过程实质上模拟了人类归纳思维的过程。

②自顶向下

把现有类细化成更具体的子类，这模拟了人类的演绎思维过程。从应用域中常常能明显看出应该做的自顶向下的具体化工作。

（2）多重继承

①作用

利用多重继承可以提高共享程度，但增加了概念上以及实现时的复杂程度。

②要点

a．指定一个主要父类，从它继承大部分属性和行为；

b．次要父类只补充一些属性和行为。

7．反复修改

（1）必要性

软件开发过程就是一个多次反复修改、逐步完善的过程。仅仅经过一次建模过程很难得到完全正确的对象模型。

（2）面向对象在修改中的优点

面向对象的概念和符号在整个开发过程中都是一致的，比使用结构分析、设计技术更容易实现反复修改、逐步完善的过程。

四、建立动态模型

1．概念

（1）适用性

①对于仅存储静态数据的系统来说，动态模型并没有什么意义。

②在开发交互式系统时，动态模型却起着很重要的作用。

③收集输入信息是系统的主要工作时，则在开发时建立正确的动态模型是至关重要的。

（2）步骤

①编写典型交互行为的脚本。

②从脚本中提取出事件，确定触发每个事件的动作对象以及接受事件的目标对象。

③排列事件发生的次序，确定每个对象的状态及状态间的转换关系，用状态图描绘。

④比较各个对象的状态图，确保事件之间的匹配。

2．编写脚本

（1）定义

脚本是指系统在某一执行期间内出现的一系列事件。脚本描述用户与目标系统之间的一个或多个典型的交互过程。编写脚本的过程，就是分析用户对系统交互行为的要求的过程。

（2）目的

编写脚本的目的是保证不遗漏重要的交互步骤，有助于确保交互过程的正确性、清晰性。

（3）内容

脚本描写的范围主要由编写脚本的具体目的决定，既可以包括系统中发生的全部事件，也可以只包括由某些特定对象触发的事件。

（4）方法

①编写正常情况的脚本；

②考虑特殊情况；

③考虑出错情况。

3．设想用户界面

大多数交互行为都可以分为应用逻辑和用户界面两部分，通常，系统分析员首先集中精力考虑系统的信息流和控制流，而不是首先考虑用户界面。

（1）重要性

用户界面的美观程度、方便程度、易学程度以及效率等，是用户使用系统时最先感受到的。用户界面的好坏往往对用户是否喜欢、是否接受一个系统起很重要的作用。

（2）目的

这个阶段用户界面的细节并不太重要，重要的是在这种界面下的信息交换方式。目的是确保能够完成全部必要的信息交换，而不会丢失重要的信息。

（3）方法

快速地建立起用户界面的原型，供用户试用与评价。

4．画事件跟踪图

（1）必要性

用自然语言书写的脚本往往不够简明，而且有时在阅读时会有二义性。为了有助于建立动态模型，需要画出事件跟踪图。

（2）步骤

①确定事件

a．提取出所有外部事件

第一，找出正常事件、异常事件和出错条件（传递信息的对象的动作也是事件）。

第二，把对控制流产生相同效果的事件组合为一类事件，并取一个唯一的名字。

b．区分出每类事件的发送对象和接受对象

②画出事件跟踪图

a．一条竖线代表一个对象；

b．每个事件用一条水平的箭头线表示；

c．箭头方向从事件的发送对象指向接受对象；

d．时间从上向下递增；

e．用箭头线在垂直方向上的相对位置表示事件发生的先后，不表示事件间的时间差。

5．画状态图

（1）定义

状态图描绘事件与对象状态的关系。当对象接受了一个事件以后，它的下个状态取决于当前状态及所接受的事件。由事件引起的改变称为“转换”。一张状态图描绘了一类对象的行为，它确定了由事件序列引出的状态序列。

（2）适用性

对于仅响应与过去历史无关的那些输入事件，或者把历史作为不影响控制流的参数类的对象，状态图是不必要的。

（3）方法

①仅考虑事件跟踪图中指向某条竖线的那些箭头线。把这些事件作为状态图中的有向边，边上标以事件名。

②两个事件之间的间隔就是一个状态，每个状态取个有意义的名字。从事件跟踪图中当前考虑的竖线射出的箭头线，是这条竖线代表的对象达到某个状态时所做的行为。

③根据一张事件跟踪图画出状态图后，再把其他脚本的事件跟踪图合并到该图中。

④考虑完正常事件后再考虑边界情况和特殊情况，包括在不适当时候发生的事件。

6．审查动态模型

（1）检查系统级的完整性和一致性。

（2）审查每个事件，跟踪它对系统中各个对象所产生的效果，保证与每个脚本都匹配。

五、建立功能模型

1．定义

功能模型表明了系统中数据之间的依赖关系，以及有关的数据处理功能，它由一组数据流图组成。在建立了对象模型和动态模型之后再建立功能模型。

2．画出基本系统模型图

基本的系统模型有下述两部分组成：

（1）数据源点/终点

数据源点输入的数据和输出到数据终点的数据，是系统与外部世界间交互事件的参数。

（2）处理框

处理框代表了系统加工、变换数据的整体功能。

3．画出功能级数据流图

把基本系统模型中单一的处理框分解成若干个处理框，以描述系统加工、变换数据的基本功能，就得到功能级数据流图。

4．描述处理框功能

（1）要点

着重描述每个处理框所代表的功能，而不是实现功能的具体算法。

（2）分类

①说明性描述（更重要）

说明性描述规定了输入值和输出值之间的关系，以及输出值应遵循的规律。

②过程性描述

过程性描述则通过算法说明“做什么”。

六、定义服务

（1）常规行为

无须在类图中显式表示读、写该类每个属性的常规操作

（2）从事件导出的操作

状态图中发往对象的事件也就是该对象接收到的消息，因此该对象必须有由消息选择符指定的操作，这个操作修改对象状态并启动相应的服务。

（3）与数据流图中处理框对应的操作

数据流图中的每个处理框都与一个对象上的操作相对应。应该仔细对照状态图和数据流图，以便更正确的确定对象应该提供的服务。

（4）利用继承减少冗余操作

应该尽量利用继承机制以减少所需的服务数目。

**10.2　课后习题详解**

1．用面向对象方法分析研究本书习题2第2题中描述的储蓄系统，试建立它的对象模型、动态模型和功能模型。

答：（1）对象模型

储蓄系统的对象模型，如图10-2所示。

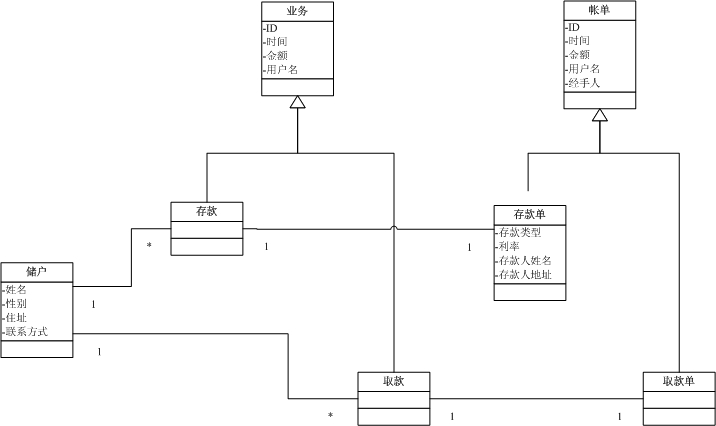


图10-2 储蓄系统的对象模型

（2）动态模型

①编写脚本

a．正常情况脚本

第一，储户有存款要求，填写存款单，包含储户个人信息，存款金额和存款类型；

第二，业务员查收存款，审核存款与存款单存款金额吻合；

第三，存款单生效；

第四，储户有取款要求，填写取款单，包含个人账号、密码（待定）和存款金额；

第五，业务员审核存款，验证储户身份，确定储户存款金额 > = 取款金额；

第六，审核通过，取款单生效；

第七，系统打印利息清单，业务员把本金和利息返回储户。

b．异常情况脚本

第一，储户有存款要求，填写存款单，包含储户个人信息，存款金额和存款类型；

第二，业务员查收存款，审核存款与存款单存款金额不符；

第三，存款单作废；

第四，储户有取款要求，填写取款单，包含个人账号、密码（待定）和存款金额；

第五，业务员审核存款，验证储户身份，发现储户存款金额 < = 取款金额；

第六，取款单作废。

②事件追踪图

储蓄系统的事件追踪图，如图10-3所示。

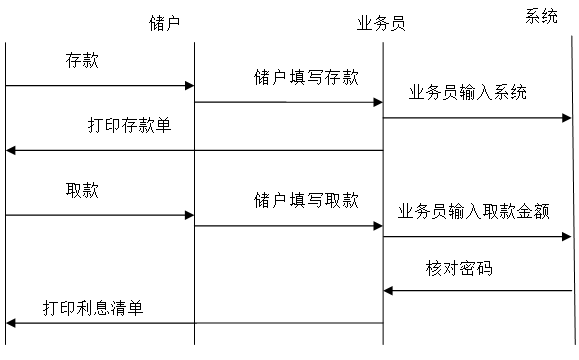


图10-3 储蓄系统的事件追踪图

（3）功能模型

储蓄系统的数据库流图，如图10-4所示。

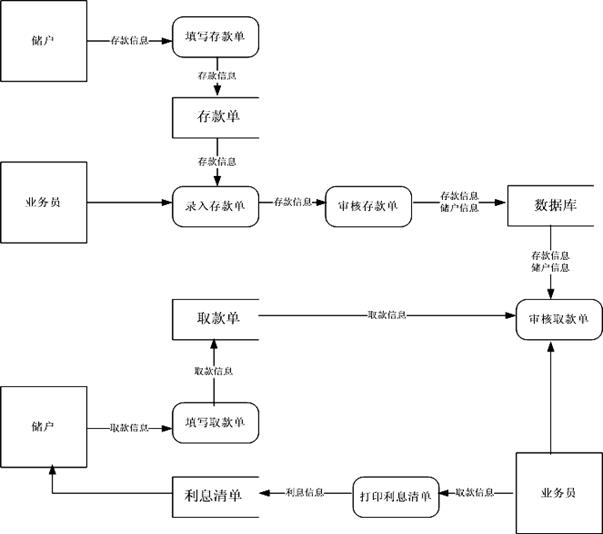


图10-4 储蓄系统的数据库流图

2．用面向对象方法分析研究本书习题2第3题中描述的机票预订系统，试建立它的对象模型、动态模型和功能模型。

答：略。

3．用面向对象方法分析研究本书习题2第4题中描述的患者监护系统，试建立它的对象模型、动态模型和功能模型。

答：略。

4．下面是自动售货机系统的需求陈述，试建立它的对象模型、动态模型和功能模型：

自动售货机系统是一种无人售货系统。售货时，顾客把硬币投入机器的投币口中，机器检查硬币的大小、重量、厚度及边缘类型。有效的硬币是一元币、五角币、一角币、五分币、二分币和一分币。其他货币都被认为是假币。机器拒绝接收假币，并将其从退币孔退出。当机器接收了有效的硬币之后，就把硬币送入硬币储藏器中。顾客支付的货币根据硬币的面值进行累加。

自动售货机装有货物分配器。每个货物分配器中包含零个或多个价格相同的货物。顾客通过选择货物分配器来选择货物。如果货物分配器中有货物，而且顾客支付的货币值不小于该货物的价格，货物将被分配到货物传送孔送给顾客，并将适当的零钱返回到退币孔。如果分配器是空的，则和顾客支付的货币值相等的硬币将被送回到退币孔。如果顾客支付的货币值少于所选择的分配器中货物的价格，机器将等待顾客投进更多的货币。如果顾客决定不买所选择的货物，他投放进的货币将从退币孔中退出。

答：（1）对象模型

①建立流程

a．候选类

自动售货机、顾客、硬币、投币口、假币、退币孔、硬币储藏器、货物分配器、货物、货物传送孔、零钱。

b．筛选优化类

第一，硬币、假币、零钱是货币的属性。属于系统内部“检验”功能处理的内容，它们与“货物”都属于物流，应该去掉。

第二，顾客决定买不买所选货物，需要一个按钮通知硬币储藏器，所以应增加一个“选择按钮”。

第三，进行投币检验需要一个检验器。

c．类集合

自动售货机、顾客、投币口、退币口、硬币储藏器、货物分配器、货物传送孔、硬币暂存检查器、选择按钮。

②对象模型

自动售货机系统的对象模型，如图10-5所示。

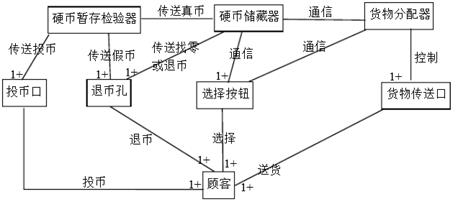


图10-5 自动售货机系统的对象模型

（2）动态模型

①编写脚本

a．正常情况脚本

第一，顾客投入了5元真硬币；

第二，投币机检验，钱币合格，送往硬币储藏器，储藏器进行累加；

第三，顾客按“选择按钮”，指定一种3.5元的货物；

第四，按钮将信息送往货物分配器；

第五，货物分配器检查货物是否还有。还有，则将该货物的价格及所购数量计算后通知硬币储藏器需款总数；

第六，硬币储藏器将找零的硬币送往退币孔；

第七，货物分配器将应交付的货物送往货物传送口。

b．异常情况脚本1（投币不足）

第一，顾客投入了5元真硬币；

第二，投币机检验，钱币合格，送往硬币储藏器，储藏器进行累加；

第三，顾客按“选择按钮”，指定一种6元的货物；

第四，按钮将信息送往货物分配器；

第五，货物分配器检查货物是否还有。还有，则将该货物的价格及所购数量计算后通知硬币储藏器需款总数；

第六，硬币储藏器将对投款数与需款数进行对比后不发出任何信息，机器处于等待状态。

c．异常情况脚本2（投入假币）

第一，顾客投入了5元真硬币；

第二，投币机检验，其中有假币，投币机即刻将所投币送往退币孔。

d．异常情况脚本3（无货）

第一，顾客投入了5元真硬币；

第二，投币机检验，钱币合格，送往硬币储藏器，储藏器进行累加；

第三，顾客按“选择按钮”，指定一种3.5元的货物；

第四，按钮将信息送往货物分配器；

第五，货物分配器检查货物是否还有。没有货物，则将无货信息送往硬币储藏器；

第六，硬币储藏器将5元硬币送往退币孔。

e．异常情况脚本4（顾客决定不买）

第一，顾客投入了5元真硬币；

第二，投币机检验，钱币合格，送往硬币储藏器，储藏器进行累加；

第三，顾客按“选择按钮”，决定不买了；

第四，按钮将信息送往货物分配器；

第五，硬币储藏器将5元硬币送往退币孔。

②状态图

自动售货机系统的状态图，如图10-6所示。

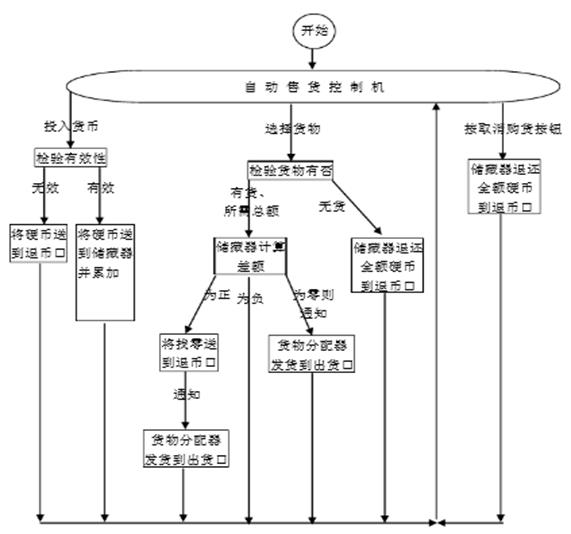


图10-6 自动售货机系统的状态图

（3）功能模型

自动售货机系统的程序流程图，如图10-7所示。

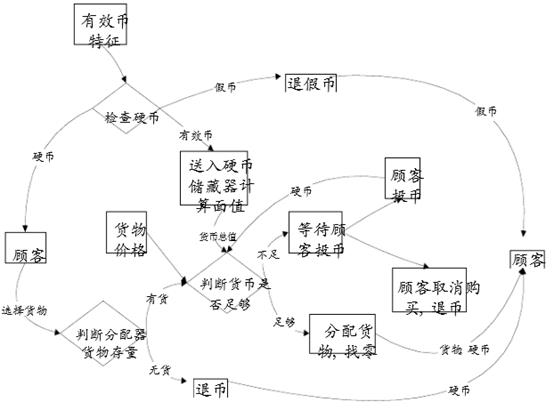


图10-7 自动售货机系统的程序流程图

**第11章　面向对象设计**

**11.1　复习笔记**

一、面向对象设计的概念

1．定义

设计是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程。从面向对象分析到面向对象设计是一个逐渐扩充模型的过程，即面向对象设计就是用面向对象观点建立求解域模型的过程。

2．设计与分析的关系

（1）分析结果可以直接映射成设计结果，而在设计过程中又会加深和补充对系统需求的理解，进一步完善分析结果。

（2）分析和设计活动是一个多次反复迭代的过程。

（3）分析是提取和整理用户需求，并建立问题域精确模型的过程。设计则是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程。

3．分类

（1）系统设计：确定实现系统的策略和目标系统的高层结构。

（2）对象设计：确定解空间中的类、关联、接口形式及实现服务的算法。

二、面向对象设计的准则

1．优秀设计的定义

优秀设计是权衡了各种因素，从而使得系统在其整个生命周期中的总开销最小的设计。优秀软件设计的一个主要特点就是容易维护。

2．准则

（1）模块化

面向对象软件开发模式支持了把系统分解成模块设计的原理，对象是面向对象软件系统中的模块，它是把数据结构和操作这些数据的方法紧密地结合在一起所构成的模块。

（2）抽象

面向对象的程序设计语言不仅支持过程抽象，而且支持数据抽象，对象类实际上是具有继承机制的抽象数据类型，它对外开放的公共接口构成了类的规格说明（协议），这种接口规定了外界可以使用的合法操作符，利用这些操作符可以对类实例中包含的数据进行操作。

①规格说明抽象

使用者无须知道操作符的实现算法和类中数据元素的具体表示方法，就可以通过这些操作符使用类中定义的数据，这种抽象称为规格说明抽象。

②参数化抽象

指当描述类的规格说明时并不具体指定所要操作的数据类型，而是把数据类型作为参数，使得类的抽象程度更高，应用范围更广，可重用性更高。

（3）信息隐藏

在面向对象的软件中，信息隐藏通过对象的封装来实现，即类结构分离了接口与实现，从而支持了信息隐藏。对于类，属性的表示方法和操作的实现算法都是隐藏的。

（4）弱耦合

耦合是指一个软件结构内不同模块之间互连的紧密程度。在面向对象方法中，对象是最基本的模块，因此，耦合主要指不同对象之间相互关联的紧密程度。弱耦合是优秀设计的一个重要标准。

①交互耦合

对象间的耦合通过消息连接来实现，则这种耦合是交互耦合。要使交互耦合尽可能松散，必须遵守下述准则。

a．尽量降低消息连接的复杂程度。应该尽量减少消息中包含的参数个数，降低参数的复杂程度。

b．减少对象发送或接收的消息数。

②继承耦合

继承是一般类与特殊类之间耦合的一种形式。通过继承关系结合起来的基类和派生类构成了系统中粒度更大的模块，因此，它们彼此之间应该结合得越紧密越好。

（5）强内聚

内聚衡量一个模块内各个元素彼此结合的紧密程度，在设计时应该力求做到高内聚。在面向对象设计中存在下述3种内聚：

①服务内聚

一个服务应该完成一个且仅完成一个功能。

②类内聚

设计类的准则是，一个类应该只有一个用途，它的属性和服务应该是高内聚的。如果某个类有多个用途，应该把它分解成多个专用的类。

③一般—特殊内聚

设计出的一般—特殊结构应该是对相应的领域知识的正确抽取。紧密的继承耦合与高度的一般—特殊内聚是一致的。

（6）可重用

软件重用是提高软件开发生产率和目标系统质量的重要途径。重用基本上从设计阶段开始。重用有两方面的含义：

①尽量使用已有的类。

②如果需要创建新类，则在设计这些新类的协议时应该考虑将来的可重复使用性。

三、启发规则

1．必要性

总结人类使用面向对象方法学开发软件的经验，可以得出几条启发规则，它们往往能帮助软件开发人员提高面向对象设计的质量。

2．具体规则

（1）设计结果应该清晰易懂

使设计结果清晰、易读、易懂是提高软件可维护性和可重用性的重要措施。保证设计结果清晰易懂的主要因素如下：

①用词一致

应该使名字与它所代表的事物一致，而且应该尽量使用人们习惯的名字。不同类中相似服务的名字应该相同。

②使用已有的协议

如果开发同一软件的其他设计人员已经建立了类的协议，或者在所使用的类库中已有相应的协议，则应该使用这些已有的协议。

③减少消息模式的数目

如果已有标准的消息协议，设计人员应该遵守这些协议。

④避免模糊的定义

一个类的用途应该是有限的，而且应该从类名可以较容易地推出它的用途。

（2）一般—特殊结构的深度应适当

①使类等级中包含的层次数适当，类等级中包含的层次保持在7±2。

②不能仅从方便编码的角度出发随意创建派生类，应该使一般—特殊结构与领域知识或常识保持一致。

（3）设计简单的类

应该尽量设计小而简单的类，以便开发和管理。为使类保持简单，应注意以下几点：

①避免包含过多的属性

属性过多通常表明这个类过分复杂了，它所完成的功能可能太多了。

②有明确的定义

为了使类的定义明确，分配给每个类的任务应该简单，最好能使用一两个简单语句描述它的任务。

③简化对象之间的合作关系

如果需要多个对象协同配合才能做好一件事，则破坏了类的简明性和清晰性。

④不要提供太多服务（公共服务不超过7个）

一个类提供的服务过多，同样表明这个类过分复杂。典型地，一个类提供的公共服务不超过7个。

⑤划分“主题”。

（4）使用简单的协议

一般来说，消息中的参数不要超过3个。通过复杂消息相互关联的对象是紧耦合的，对一个对象的修改往往导致其他对象的修改。

（5）使用简单的服务

①避免使用复杂的服务。

②需要在服务中使用CASE语句时，应用一般—特殊结构代替这个类。

（6）把设计变动减至最小

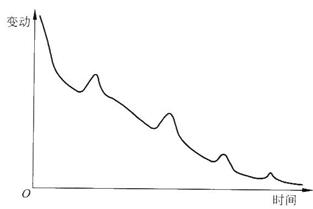


图11-1 理想的设计变动情况

理想的设计变动曲线如图11-1所示。即在设计的早期阶段，变动较大，随着时间推移，设计方案日趋成熟，改动也越来越小了。

四、软件重用

1．概述

（1）重用

重用也叫再用或复用，是指同一事物不作修改或稍加改动就多次重复使用。软件重用可分为以下3个层次：

①知识重用；

②方法和标准的重用；

③软件成分的重用。

其中知识重用以及方法和标准的重用层次属于知识工程研究的范畴。

（2）软件成分的重用级别

①代码重用

a．定义

代码重用是调用库中的模块。

b．形式

第一，源代码剪贴

是最原始的重用形式，复制或修改源代码时可能出错，且存在严重的配置管理问题。

第二，源代码包含

配置管理问题有所缓解，所有包含它的程序都必须重新编译。

第三，继承

无须修改已有的代码，就可扩充或具体化在库中找出的类。基本上不存在配置管理问题。

②设计结果重用

重用某个软件系统的设计模型（求解域模型），有助于把一个应用系统移植到完全不同的软硬件平台上。

③分析结果重用

是一种更高级别的重用，重用某个系统的分析模型。特别适用于用户需求未改变，但系统体系结构发生了根本变化的场合。

（3）典型的可重用软件成分

可能被重用的软件成分主要有以下10种：

①项目计划

软件项目计划的基本结构和许多内容都是可以跨项目重用的。

②成本估计

在只做极少修改或根本不做修改的情况下，重用对该功能的成本估计结果。

③体系结构

创建一组类属的体系结构模板，并把那些模板作为可重用的设计框架。

④需求模型和规格说明

类和对象的模型、规格说明、用传统软件工程方法开发的分析模型是重用的候选者。

⑤设计

用传统方法开发的体系结构、数据、接口和过程设计结果，是重用的候选者。

⑥源代码

用兼容的程序设计语言书写的、经过验证的程序构件，是重用的候选者。

⑦用户文档和技术文档

即使针对的应用是不同的，也经常有可能重用用户文档和技术文档的大部分。

⑧用户界面

这可能是最广泛被重用的软件成分，GUI（图形用户界面）软件经常被重用。因为它可占到一个应用程序的60％的代码量。

⑨数据

被重用的数据包括：内部表、列表和记录结构，以及文件和完整的数据库。

⑩测试用例

一旦设计或代码构件将被重用，相关的测试用例应该“附属于”它们也被重用。

2．类构件

（1）可重用软构件的特点

为使软构件也像硬件集成的电路那样，能在构造各种各样的软件系统时方便地重复使用，就必须使它们满足下列要求：

①模块独立性强

具有单一、完整的功能，且经过反复测试被确认是正确的。它应该是一个不受或很少受外界干扰的封装体，其内部实现在外面是不可见的。

②具有高度可塑性

可重用的软构件必须具有高度可裁剪性，即必须提供为适应特定需求而扩充或修改已有构件的机制，而且所提供的机制必须使用起来简单方便。

③接口清晰、简明、可靠

软构件应该提供清晰、简明、可靠的对外接口，而且还应该有详尽的文档说明，以方便用户使用。

（2）类构件的重用方式

面向对象技术中的“类”，是比较理想的可重用软构件，称之为类构件。类构件有实例重用、继承重用和多态重用3种重用方式。

①实例重用

除了用已有的类为样板直接创建该类的实例之外，还可以用几个简单的对象作为类的成员创建出一个更复杂的类。

②继承重用

当已有的类构件不能通过实例重用方式满足当前系统的需求时，利用继承机制从已有类派生出符合需要的子类，是安全修改已有的类构件并获得可在当前系统中使用的类构件的有效手段。

③多态重用

在设计类构件时应把注意力集中在下列这些可能妨碍重用的操作上：

a．与表示方法有关的操作。

b．与数据结构、数据大小等因素有关的操作。

c．与外部设备有关的操作。

d．实现算法在将来可能会改变的核心操作。

3．软件重用的效益

（1）质量

随着每一次重用，都会有一些错误被发现并被清除，构件的质量也会随之改善。随着时间的推移，构件将变成实质上无错误的。重用给软件产品的质量和可靠性带来实质性的提高。

（2）生产率

把可重用的软件成分应用于软件开发的全过程时，创建计划、模型、文档、代码和数据所花费的时间将减少，从而用较少的投入给客户提供相同级别的产品，故生产率得到了提高。

（3）成本

软件重用带来的净成本节省可以用下式估算：

C=Cs-Cr-Cd

其中，Cs是项目从头开发时所需要的成本；Cr是与重用相关联的成本；Cd是交付给客户的软件的实际成本。

与重用相关联的成本C，主要包括下述成本：

①领域分析与建模的成本。

②设计领域体系结构的成本。

③为方便重用而增加的文档的成本。

④维护和完善可重用的软件成分的成本。

⑤为从外部获取构件所付出的版税和许可证费用。

⑥创建及运行重用库的费用。

⑦对设计和实现可重用构件的人员的培训费用。

五、系统分解

1．分解的思想

在设计比较复杂的应用系统时，先把系统分解成若干个较小部分，然后分别设计每个部分。这样做有利于降低设计的难度，有利于分工协作，也有利于维护人员对系统理解和维护。

2．子系统

（1）定义

系统的主要组成部分称为子系统，通常根据所提供的功能来划分子系统。

（2）划分原则

①根据所提供的功能来划分子系统，子系统数目应该与系统规模基本匹配。

②各个子系统之间应该具有尽可能简单、明确的接口。

③应该尽量减少子系统彼此间的依赖性。

3．分解面向对象设计模型

（1）表示



图11-2 典型的面向对象设计模型

典型的面向对象设计模型，如图11-2所示。

①面向对象设计模型由主题、类与对象、结构、属性、服务5个层次组成。这5个层次一层比一层表示的细节更多，可以把这5个层次想象为整个模型的水平切片。

②面向对象设计模型在逻辑上都由4大部分组成，分别对应于组成目标系统的4个子系统，即问题域子系统、人机交互子系统、任务管理子系统和数据管理子系统。

（2）子系统间交互方式

在软件系统中，子系统之间的交互有两种可能的方式，分别是客户—供应商（Client-supplier）关系和平等伙伴（peer-to-peer）关系。

①客户—供应商关系（较好）

作为“客户”的子系统调用作为“供应商”的子系统，后者完成某些服务工作并返回结果。作为客户的子系统必须了解作为供应商的子系统的接口，后者却无须了解前者的接口。

②平等伙伴关系

每个子系统都可能调用其他子系统，每个子系统都必须了解其他子系统的接口。由于各个子系统需要相互了解对方的接口，子系统之间的交互复杂，且还可能存在通信环路。

注意：单项交互比双向交互更容易理解，也更容易设计和修改，因此应该尽量使用客户—供应商关系。

（3）组织系统的方案

把子系统组织成完整的系统时，可以使用水平层次组织和垂直块组织两种方案。

①层次组织

a．定义

软件系统组织成一个层次系统，每层是一个子系统。上层在下层的基础上建立，下层为实现上层功能而提供必要的服务。每一层内所包含的对象，彼此间相互独立，而处于不同层次上的对象，彼此间有关联。在上、下层之间存在客户—供应商关系。低层子系统提供服务，上层子系统使用下层提供的服务。

b．模式分类

第一，封闭式

每层子系统仅仅使用其直接下层提供的服务。降低了各层次之间的相互依赖性，更容易理解和修改。

第二，开放式

子系统可以使用处于其下面的任何一层子系统所提供的服务。优点是减少了需要在每层重新定义的服务数目，使系统更高效更紧凑。但其不符合信息隐藏原则。

②块状组织

把软件系统垂直地分解成若干个相对独立的、弱耦合的子系统，一个子系统相当于一块，每块提供一种类型的服务。

③层次和块的组合

当混合使用层次结构和块状结构时，同一层次可以由若干块组成，而同一块也可以分为若干层。

④设计系统的拓扑结构

典型的拓扑结构有管道形、树形、星形等。应采用与问题结构相适应的、尽可能简单的拓扑结构，以减少子系统之间的交互数量。

六、设计问题域子系统

1．概念

（1）面向对象分析所得出的问题域精确模型，为设计问题域子系统建立了完整的框架。

（2）保持面向对象分析所建立的问题域结构。

（3）面向对象设计仅需从实现角度对问题域模型做一些补充或修改。

（4）问题域子系统过分复杂庞大时，应该把它进一步分解成若干个更小的子系统。

2．对问题域模型进行的处理

（1）调整需求

①需进行修改的情况

a．用户需求或外部环境发生了变化。

b．分析员对问题域理解不透彻或不能完整、准确地反映用户的真实需求。

②方法

简单地修改面向对象分析结果，然后再把这些修改反映到问题域子系统中。

（2）重用已有的类

重用已有类的主要步骤：

①在已有类中找出与问题域内某个最相似的类作为被重用的类。

②从被重用的类派生出问题域类。

③简化对问题域类的定义（从被重用的类继承的属性和服务无须再定义）。

④修改与问题域类相关的关联，必要时改为与被重用的类相关的关联。

（3）把问题域类组合在一起

①增添一个根类而把若干个问题域类组合在一起。

②引入根类或基类的办法，可以为一些具体类建立一个公共的协议。

（4）增添一般化类以建立协议

在设计过程中常常发现，一些具体类需要有一个公共的协议，可以引入附加类以便建立这个协议。

（5）调整继承层次

如果面向对象分析模型中包含了多重继承关系，然而所使用的程序设计语言却并不提供多重继承机制，则必须修改面向对象分析的结果。具体如下：

①多重继承机制

使用多重继承机制时，应该避免出现属性及服务的命名冲突。

②单继承机制

使用多重继承机制时，必须把面向对象分析模型中的多重继承结构转换成单继承结构。

七、设计人机交互子系统

1．概念

（1）主要内容

在面向对象设计过程中，对系统的人机交互子系统进行详细设计，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。

（2）重要性

人机界面设计得好，则会使系统对用户产生吸引力，用户在使用系统的过程中会感到兴奋，能够激发用户的创造力，提高工作效率；人机界面设计得不好，用户在使用过程中就会感到不方便、不习惯，甚至会产生厌烦和恼怒的情绪。

注意：使用由原型支持的系统化的设计策略，是成功地设计人机交互子系统的关键。

2．设计策略

（1）分类用户

应该把将来可能与系统交互的用户按技能水平，或按职务，或按所属集团进行分类。

（2）描述用户

了解将来使用系统的每类用户的情况，把用户类型、使用目的、特征、关键的成功因素、技能水平、完成本职工作的脚本的信息记录下来。

（3）设计命令层次

①研究现有的人机交互含义和准则

设计图形用户界面时，应该遵守广大用户习惯的约定，这样才会被用户接受和喜爱。

②确定初始的命令层次

命令层次实质上是用抽象机制组织起来的、可供选用的服务的表示形式，设计命令层次时，通常先从对服务的过程抽象着手，然后进一步修改它们，以适合具体应用环境的需要。

③精化命令层次

为进一步修改完善初始的命令层次，应该考虑次序、整体部分关系、宽度和深度等因素。

（4）设计人机交互类

人机交互类与所使用的操作系统及编程语言密切相关

八、设计任务管理子系统

1．设计的必要性

（1）许多对象之间往往存在相互依赖关系。

（2）在实际使用的硬件中，可能仅由一个处理器支持多个对象。

2．设计步骤

（1）分析并发性

①并发性

如果两个对象彼此间不存在交互，或它们同时接受事件，则它们在本质上是并发的。

②方法

a．通过面向对象分析建立起来的动态模型，是分析并发性的主要依据。

b．通过检查各个对象的状态图及它们之间交换的事件，能够把若干个非并发的对象归并到一条控制线中。

③控制线

控制线是一条遍及状态图集合的路径，在这条路径上每次只有一个对象是活动的。在计算机系统中用进程实现控制线。把多个任务的并发执行称为多任务。

（2）设计任务管理子系统

①确定事件驱动型任务

a．定义

某些任务是由事件驱动的，这类任务可能主要完成通信工作。

b．具体任务

第一，任务处于睡眠状态，等待来自数据线或其他数据源的中断。

第二，一旦接收到中断就唤醒了该任务，接收数据并把数据放入内存缓冲区或其他目的地，通知需要知道这件事的对象，然后该任务又回到睡眠状态。

②确定时钟驱动型任务

a．定义

某些任务每隔一定时间间隔就被触发以执行某些处理。

b．具体任务

第一，任务设置了唤醒时间后进入睡眠状态，等待来自系统的中断。

第二，接收到这种中断，任务就被唤醒并做它的工作，通知有关的对象，然后该任务又回到睡眠状态。

③确定优先任务

优先任务可以满足高优先级或低优先级的处理需求。

a．高优先级

有些服务是优先级的，为了在严格限定的时间内完成，把这类服务分离成独立的任务。

b．低优先级

与高优先级相反，有些服务是低优先级的，属于低优先级处理。设计时用额外的任务把其分离出来。

④确定关键任务

a．定义

关键任务是有关系统成功或失败的关键处理，这类处理通常都有严格的可靠性要求。

b．处理方法

在设计过程中用额外的任务把这样的关键处理分离出来，以满足高可靠性处理的要求。

⑤确定协调任务

a．定义

当系统中存在3个以上任务时，就应该增加一个任务，用它作为协调任务。

b．优缺点

优点：有助于把不同任务之间的协调控制封装起来。

缺点：会增加系统的总开销。

c．工具

使用状态转换矩阵可以比较方便地描述该任务的行为。这类任务仅做协调工作，不要让它再承担其他服务工作。

⑥尽量减少任务数

a．定义

必须仔细分析和选择每个确实需要的任务。应该使系统中包含的任务数尽量少。

b．原因

设计者为了自己处理时的方便而轻率地定义过多的任务。这样做加大了设计工作的技术复杂度，并使系统变得不易理解，也加大了系统维护的难度。

⑦确定资源需求

a．通过计算系统载荷，来估算所需要的固件的处理能力。

b．综合权衡一致性、成本、性能以及未来的可扩充性和可修改性，决定资源需求。

ｃ．综合考虑各种因素，以决定哪些子系统用硬件实现，哪些子系统用软件实现。

九、设计数据管理子系统

1．概念

数据管理子系统是系统存储或检索对象的基本设施，它建立在某种数据存储管理系统之上，并且隔离了数据存储管理模式的影响。

2．选择数据存储管理模式

不同的数据存储管理模式有不同的特点，适用范围也不相同，设计者应该根据应用系统的特点选择适用的模式。

（1）文件管理系统

①优点

文件管理系统是操作系统的一个组成部分，使用它长期保存数据具有成本低和简单的优点。

②缺点

文件操作的级别低，为提供适当的抽象级别还必须编写额外的代码，不同操作系统的文件管理系统往往有明显差异。

（2）关系数据库管理系统

①理论基础

关系数据库管理系统的理论基础是关系代数。

②优点

a．理论基础坚实。

b．提供了各种最基本的数据管理功能，例如中断恢复，多用户共享，多应用共享，完整性，事务支持等。

c．为多种应用提供了一致的接口。

d．标准化的语言。

③缺点

a．运行开销大：即使完成简单的事务，也需要较长的时间。

b．不能满足高级应用的需求：关系数据库管理系统是为商务应用服务的，商务应用中数据量虽大但数据结构却比较简单。

c．与程序设计语言的连接不自然：SQL语言支持面向集合的操作，是一种非过程化的语言；然而大多数程序设计语言本质上却是过程性的，每次只能处理一个记录。

（3）面向对象数据库管理系统

①扩展的关系数据库管理系统

在关系数据库的基础上，增加了抽象数据类型和继承机制，此外还增加了创建及管理类和对象的通用服务。

②扩展的面向对象程序设计语言

扩充了面向对象程序设计语言的语法和功能，增加了在数据库中存储和管理对象的机制。可以使用统一的面向对象观点进行设计，不需要区分存储数据结构和程序数据结构。

3．设计数据管理子系统

设计数据管理子系统，既需要设计数据格式，又需要设计相应的服务。

（1）设计数据格式

①文件系统

a．定义第一范式表：列出每个类的属性表；把属性表规范成第一范式，从而得到第一范式表的定义。

b．为每个第一范式表定义一个文件。

c．测量性能和需要的存储容量。

d．修改原设计的第一范式，以满足性能和存储需求。

必要时把泛华结构的属性压缩在单个文件中，以减少文件数量；必要时把某些属性组合在一起，并用某种编码值表示这些属性，而不再使用独立的域表示每个属性。

②关系数据库管理系统

a．定义第三范式表：列出每个类的属性表；把属性表规范成第三范式，从而得出第三范式表的定义。

b．为每个第三范式表定义一个数据库表。

c．测量性能和需要的存储容量。

d．修改先前设计的第三范式，以满足性能和存储需求。

③面向对象数据库管理系统

a．扩展的关系数据库途径：使用与关系数据库管理系统相同的方法。

b．扩展的面向对象程序设计语言途径：不需要规范化属性的步骤。

（2）设计相应的服务

使用不同数据存储管理模式时的设计要点如下：

①文件系统

a．被存储的对象需要知道打开哪个文件，怎样把文件定位到正确的记录上，怎样检索出旧值，以及怎样用现有值更新它们。

b．定义一个ObjectServer类，并创建它的实例。

②关系数据库管理系统

a．被存储的对象，应该知道访问哪些数据库表，怎样访问所需要的行，怎样检索出旧值，以及怎样用现有值更新它们。

b．定义一个ObjectServer类，并声明它的对象。

③面向对象数据库管理系统

a．扩展的关系数据库途径：与使用关系数据库管理系统时方法相同。

b．扩展的面向对象程序设计语言途径：无须增加服务，只需给长期保存的对象加个标记，然后由面向对象数据库管理系统负责存储和恢复这类对象。

十、设计类中的服务

1．确定类中应有的服务

（1）确定服务的总体思想

①对象模型是进行对象设计的基本框架。必须把动态模型中对象的行为以及功能模型中的数据处理转换成由适当的类所提供的服务。

②动态模型中状态图中的状态转换执行对象服务的结果。

③功能模型指明了系统必须提供的服务。

（2）确定操作目标对象的启发规则

①如果某个处理的功能是从输入流中抽取一个值，则该输入流就是目标对象。

②如果某个处理具有类型相同的输入流和输出流，而且输出流实质上是输入流的另一种形式，则该输入输出流就是目标对象。

③如果某个处理从多个输入流得出输出值，则该处理是输出类中定义的一个服务。

④如果某个处理把对输入流处理的结果输出给数据存储或动作对象，则该数据存储或动作对象就是目标对象。

（3）确定处理归属的启发规则

①如果处理影响或修改了一个对象，则最好把该处理与处理的目标联系在一起。

②考察处理涉及的对象类及这些类之间的关联，从中找出处于中心地位的类。

2．设计实现服务的方法

（1）设计实现服务的算法

主要考虑下列因素：

①算法复杂度：选用复杂度较低（效率较高）的算法，但不能过分追求高效率，应以能满足用户需求为准。

②容易理解与容易实现：容易理解与容易实现的要求往往与高效率有矛盾，设计者应该对这两个因素适当折衷。

③易修改：预测将来可能做的修改，并在设计时预先做些准备。

（2）选择数据结构

选择能够方便、有效地实现算法的物理数据结构。

（3）算法与数据结构的关系

主要考虑下列因素：

①分析问题寻找数据特点，提炼出所有可行有效的算法。

②定义与所提炼算法相关联的数据结构。

③依据此数据结构进行算法的详细设计。

④进行一定规模的实验与评测。

⑤确定最佳设计。

（4）定义内部类和内部操作

增添一些用来存放在执行算法过程中所得出的中间结果的类，其需求陈述中没有提到。复杂操作往往可以用简单对象上的更低层操作来定义，因此，在分解高层操作时常常引入新的低层操作。

十一、设计关联

1．关联

（1）定义

在对象模型中，关联是联结不同对象的纽带，它指定了对象相互间的访问路径。

（2）确定实现关联的策略

①选定一个全局性的策略统一实现所有关联。

②分别为每个关联选择具体的实现策略，以与它在应用系统中的使用方式相适应。

2．使用关联的方式

（1）关联的遍历

①单向遍历

②双向遍历

在使用原型法开发软件的时候，原型中所有关联都应该是双向的，以便于增加新的行为，快速地扩充和修改原型。

（2）实现单向关联

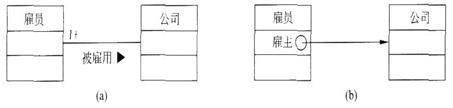


图11-3 用指针实现单向关联

（a）关联；（b）实现

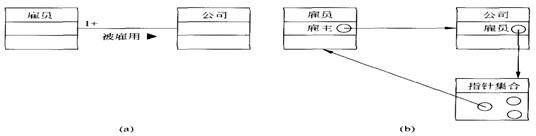


图11-4 用指针实现双向关联

（a）关联；（b）实现

①关联的重数是一元的（如图11-3所示），则实现关联的指针是一个简单指针。

②重数是多元的，则需要用一个指针集合实现关联（如图11-4所示）。

（3）实现双向关联

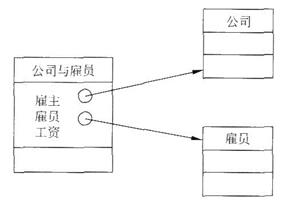


图11-5 用对象实现关联

①只用属性实现一个方向的关联

如果两个方向遍历的频度相差很大，而且需要尽量减少存储开销和修改时的开销，则这是一种很有效的实现双向关联的方法。

②两个方向的关联都用属性实现

这种方法能实现快速访问。当访问次数远远多于修改次数时，这种实现方法很有效，如图11-4所示。

③用独立的关联对象实现双向关联

关联对象不属于相互关联的任何一个类，它是独立的关联类的实例，如图11-5所示。

（4）关联对象的实现

①定义

用一个关联类来保存描述关联性质的信息，关联中的每个连接对应关联类的一个对象。

②方法

a．对于一对一的关联，关联对象可以与参与关联的任一个对象合并。

b．对于一对多的关联，关联对象可以与多端对象合并。

c．对于多对多的关联，关联链的性质不可能只与一个参与关联的对象有关。

十二、设计优化

1．确定优先级

（1）必要性

系统的各项质量指标并不是同等重要的，必须确定各项质量指标的优先级，以便在优化设计时制定折中方案。系统的整体质量与制定的折中方案密切相关。最终产品成功与否，在很大程度上取决于是否选择好了系统目标。

（2）方法

在效率和清晰度之间寻求适当的折中方案。在折中方案中设置的优先级应当是模糊的。

2．提高效率的几项技术

（1）增加冗余关联以提高访问效率。

（2）调整查询次序。

（3）保留派生属性。

3．调整继承关系

（1）继承关系

继承关系能够为一个类族定义一个协议，并能在类之间实现代码共享以减少冗余。一个基类和它的子孙类在一起称为一个类继承。在面向对象设计中，建立良好的类继承是非常重要的。利用类继承能够把若干个类组织成一个逻辑结构。

（2）建立类继承

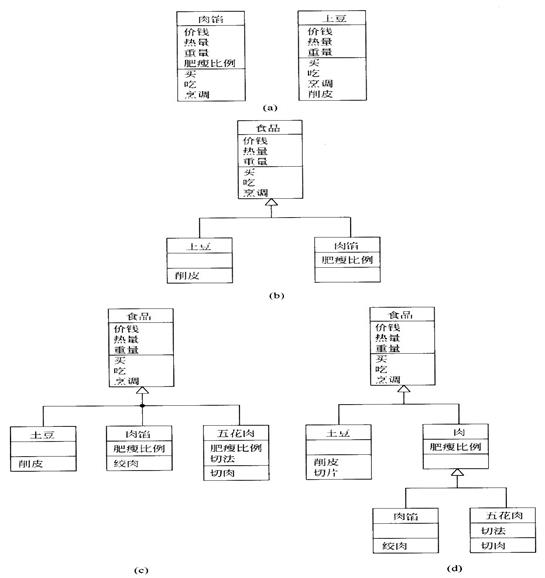


图11-6 设计类继承的例子

（a）先创建一些具体类；（b）归纳出抽象娄；

（c）进一步具体化；（d）再次归纳

①抽象与具体

在设计类继承时，很少使用纯粹自顶向下的方法，通常的做法如下：

a．创建满足具体用途的类，然后对它们进行归纳。

b．归纳出一些通用的类以后，根据需要再派生出具体类。

c．进行一些具体化的工作后，再次归纳。

②为提高继承程度而修改类定义

在一组相似的类中存在公共的属性和公共的行为时，可以把这些公共的属性和行为抽取出来放在一个共同的祖先类中，供其子类继承，如图11-6（a）和（b）所示。在对现有类进行归纳的时候，要注意下述两点：

a．不能违背领域知识和常识。

b．应该确保现有类的协议不变。

③利用委托实现行为共享

仅当存在真实的一般—特殊关系（子类确实是父类的一种特殊形式）时，利用继承机制实现行为共享才是合理的。

**11.2　课后习题详解**

1．面向对象设计应该遵循哪些准则？简述每条准则的内容，并说明遵循这条准则的必要性。

答：面向对象设计应该遵循如下六条准则：

（1）模块化

对象是面向对象软件系统中的模块，它是把数据结构和操作这些数据的方法紧密地结合在一起所构成的模块。

（2）抽象

面向对象的程序设计语言不仅支持过程抽象，而且支持数据抽象，对象类实际上是具有继承机制的抽象数据类型。

（3）信息隐藏

在面向对象的软件中，信息隐藏通过对象的封装来实现，即类结构分离了接口与实现，从而支持了信息隐藏。

（4）弱耦合

①交互耦合

对象间的耦合通过消息连接来实现，则这种耦合是交互耦合。要使交互耦合尽可能松散。

②继承耦合

继承是一般类与特殊类之间耦合的一种形式。通过继承关系结合起来的基类和派生类构成了系统中粒度更大的模块，因此，它们彼此之间应该结合得越紧密越好。

（5）强内聚

①服务内聚

一个服务应该完成一个且仅完成一个功能。

②类内聚

设计类的准则是，一个类应该只有一个用途，它的属性和服务应该是高内聚的。

③一般—特殊内聚

设计出的一般—特殊结构应该是对相应的领域知识的正确抽取。紧密的继承耦合与高度的一般—特殊内聚是一致的。

（6）可重用

软件重用是提高软件开发生产率和目标系统质量的重要途径。重用基本上从设计阶段开始。重用有两方面的含义：

①尽量使用已有的类。

②如果需要创建新类，则在设计这些新类的协议时应该考虑将来的可重复使用性。

2．简述有助于提高面向对象设计质量的每条主要启发规则的内容和必要性。

答：（1）有助于提高面向对象设计质量的主要启发规则如下所述：

①设计结果应该清晰易懂

a．用词一致。

b．使用已有的协议。

c．减少消息模式的数目。

d．避免模糊的定义。

②一般—特殊结构的深度应适当

③设计简单的类

a．不要包含过多的属性。

b．有明确的定义。为使类的定义明确，分配给每个类的任务应该简单。

c．简化对象之间的合作关系。

d．不要提供太多的服务。

④使用简单的协议

⑤使用简单的服务

⑥把设计变动减至最小

（2）启发规则的必要性

人们使用面向对象方法学开发软件的历史虽然不长，但也积累了一些经验。总结这些经验得出了几条启发规则，它们往往能帮助软件开发人员提高面向对象设计的质量。

3．为什么说类构件是目前比较理想的可重用软构件？它有哪些重用方式？

答：（1）可重用的软构件应具备的特点：

①模块独立性强。

②具有高度可塑性。

③接口清晰、简明、可靠。

（2）类构件的重用方式：

①实例重用。

②继承重用。

③多态重用。

4．试用面向对象方法，设计本书第2章中给出的订货系统的例子。

答：略。

5．试用面向对象方法，设计本书习题2第2题中描述的储蓄系统。

答：使用面向对象方法设计出的储蓄系统的对象模型如图11-7所示：

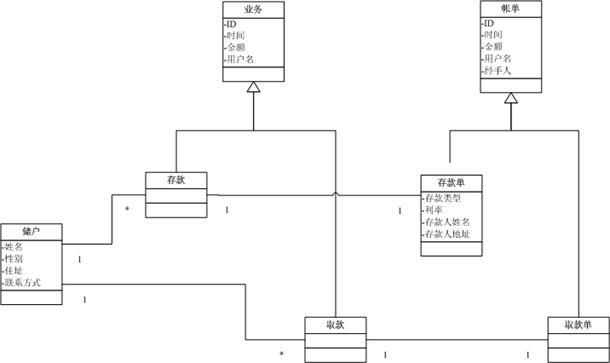


图11-7 储蓄系统的对象模型

6．试用面向对象方法，设计本书习题2第3题中描述的机票预订系统。

答：略。

7．试用面向对象方法，设计本书习题2第4题中描述的患者监护系统。

答：略。

8．有若干行C语言代码，要求统计出该代码中共有多少个关键字？试设计出相关算法和数据结构。

注：C语言的关键集合如下（32个）：

auto double int struct break else long switch case enum register typedef char extern return union const float short unsigned continue for signed void default goto sizeof volatile do if while static

答：C语言程序如下：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define Key\_Num 32

#define Len\_Max 8

#define Len\_Min 2

const char Key[Key\_Num][Len\_Max+1]={

“auto”,“double”,“int”,“struct”,break”,“else”,“long”,“switch”,“case”,“enum”,“register”,

“typedef”,“char”,“extern”,“return”,“union”,“const”,“float”,“short”,“unsigned”,“continue”,

“for”,“signed”,“void”,“default”,“goto”,“sizeof”,“volatile”,“do”,“if”,“while”,“static”

};

int count[Key\_Num];

char input[Len\_Max+2];

int n=sizeof（input）;

void CheckKeyWord(){

int i,result;

for(i=0;i<n;i++)

{

result=strcmp(input,Key[i]);

if(result==0){

count[i]++;

break;

}

if(result<0)

break;

}

int main(){

FILE \*fi;

int i,inQuote;

fi=fopen(“xxx.c”, “r”)

if(fi==NULL)

return 1;

i=inQuote=0;

while(fscanf(fi, “%c”,&input[i])==1){

if(input[i]==’\\’){

fscanf(fi, “%\*c”);

if(inQuote==0&&i>=Len\_Min&&i<=Len\_Max){

input[i]=’\0’;

CheckKeyWord();

}

i=0;

}

else if(input[i]==’\’){

if(inQuote==0&&i>=Len\_Min&&i<=Len\_Max){

input[i]=’\0’;

CheckKeyWord();

}

i=0;

inQuote=1- inQuote;

}

else if(input[i]>=’a’&&input[i]<=’z’){

if(i<=Len\_Max)

i++;

}

else if(input[i]>=’A’&&input[i]<=’Z’||input[i]>=’0’&&input[i]<=’9’){

i=Len\_Max+1;

}

else{

if(inQuote==0&&i>=Len\_Min&&i<=Len\_Max){

input[i]=’\0’;

CheckKeyWord();

}

i=0;

}

}

fclose(fi);

}

**第12章　面向对象实现**

**12.1　复习笔记**

一、面向对象实现概述

1．主要任务

（1）把面向对象设计结果翻译成用某种程序语言书写的面向对象程序。

（2）测试并调试面向对象的程序。

2．面向对象程序质量的影响因素

（1）面向对象设计的质量；

（2）采用的程序语言的特点；

（3）程序设计风格。

3．保证软件可靠性的方法

保证软件可靠性的主要措施是软件测试。面向对象测试的目标是用尽可能低的测试成本发现尽可能多的软件错误。

二、程序设计语言

1．面向对象语言的优点

（1）面向对象设计结果的表示方式

①面向对象语言

编译程序可以自动把面向对象概念映射到目标程序中。

②非面向对象语言

必须由程序员自己把面向对象概念映射到目标程序中。

（2）优点

从面向对象观点看来，能够更完整、更准确地表达问题域语义的面向对象语言的语法是非常重要的，会带来以下3个重要优点：

①一致的表示方法

面向对象开发基于不随时间变化的、一致的表示方法。既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分。

②可重用性

既可重用面向对象分析结果，也可重用相应的面向对象设计和面向对象程序设计结果。

③可维护性

程序显式地表达问题域语义，对维护人员理解待维护的软件有很大帮助。在选择编程语言时，应该考虑的首要因素是哪个语言能最恰当地表达问题域语义。

2．面向对象语言的技术特点

纯面向对象语言着重支持面向对象方法研究和快速原型的实现，而混合型面向对象语言的目标则是提高运行速度和使传统程序员容易接受面向对象思想。成熟的面向对象语言通常都提供丰富的类库和强有力的开发环境。在选择面向对象语言时应该着重考虑以下几种技术特点。

（1）支持类与对象概念的机制

①内容

面向对象语言允许用户动态创建对象，并且可以用指针引用动态创建的对象。需要及时释放不再需要的对象所占用的内存。

②管理内存的方法

a．由语言的运行机制自动管理内存，即提供自动回收“垃圾”机制；

b．由程序员编写释放内存的代码。

注意：自动管理内存虽然方便且安全，但是必须采用先进的垃圾收集算法才能减少开销。

（2）实现聚集结构的机制

①使用指针；

②使用独立的关联对象。

注意：大多数现有的面向对象语言并不显式支持独立的关联对象，在这种情况下，增加内部指针可以方便地实现关联。

（3）实现泛化结构的机制

①实现继承的机制；

②解决名字冲突的机制，即处理在多个基类中可能出现的重名问题。

（4）实现属性和服务的机制

①实现属性的机制

a．支持实例连接的机制；

b．属性的可见性控制；

c．对属性值的约束。

②实现服务的机制

a．支持消息连接（表达对象交互关系）的机制；

b．控制服务可见性的机制；

c．动态联编（在发送消息前，无须知道接受消息的对象属于哪个类）。

（5）类型检查

①分类

a．弱类型

语言仅要求每个变量或属性隶属于一个对象。

b．强类型

语法规定每个变量或属性必须准确地属于某个特定的类。

②强类型语言优点

a．有利于在编译时发现程序错误，提高软件的可靠性。

b．增加了优化的可能性，提高软件的运行效率。

③适用性

使用强类型编译型语言开发软件产品，使用弱类型解释型语言快速开发原型。

（6）类库

①类型

大多数面向对象语言都提供一个实用的类库。某些语言本身并没有规定提供什么样的类库，而是由实现这种语言的编译系统自行提供类库，主要可分为以下几类：

a．包含实现通用数据结构的类，即包容类；

b．实现各种关联的类；

c．独立于具体设备的接口类；

d．用于实现窗口系统的用户界面类。

②重要性

存在类库，许多软构件就不必由程序员从头编写了，为实现软件重用带来很大方便。

（7）效率

提高面向对象语言效率的方法为：

①使用拥有完整类库的面向对象语言。

②优化查找继承树查找过程，从而实现高效率查找。

（8）持久保存对象

①原因

a．为实现在不同程序之间传递数据，需要保存数据；

b．为恢复被中断了的程序的运行，首先需要保存数据。

②方法

a．在类库中增加对象存储管理功能；

b．使程序设计语言语法与对象存储管理语法无缝集成。

（9）参数化类

①定义

参数化类是使用一个或多个类型去参数化一个类的机制，如果程序语言提供一种能抽象出这类共性的机制，则对减少冗余和提高可重用性是大有好处的。

②方法

a．定义一个参数化的类模板；

b．把数据类型作为参数传递进来。

（10）开发环境

软件工具和软件开发环境对软件生产率有很大影响，因此面向对象语言所提供的软件工具或开发环境应该至少包括编辑程序，编译程序或解释程序，浏览工具和调试器等。具体如下：

①编辑程序；

②编译程序或解释程序（最重要）；

③浏览工具；

④调试器。

其中，编译程序或解释程序是最基本、最重要的软件工具。编译与解释的差别主要是速度和效率不同，编译程序快且效率高，解释程序慢且效率低但易于调试。

3．选择面向对象语言的标准

（1）将来能否占主导地位

（2）可重用性

应该优先选用能够最完整、最准确地表达问题域语义的面向对象语言。

（3）类库和开发环境

①在类库中，应该考虑是否提供了类库、类库中提供了哪些有价值的类。

②在开发环境中，除了基本软件工具外，还应该提供类库编辑工具和浏览工具。

注意：语言、开发环境和类库这3个因素综合起来，共同决定了可重用性。

（4）其他因素

①对用户学习面向对象分析、设计和编码技术所能提供的培训服务；

②在使用这个面向对象语言期间能提供的技术支持；

③能提供给开发人员使用的开发工具、开发平台、发行平台；

④对机器性能和内存的需求；

⑤集成已有软件的容易程度等。

三、程序设计风格

1．概念

（1）良好的程序设计风格的重要性

①能明显减少维护或扩充的开销。

②有助于在新项目中重用已有的程序代码。

（2）良好的面向对象程序设计风格的内容

①传统的程序设计风格准则。

②为适应面向对象方法所特有的概念而必须遵循的一些新准则。

2．提高可重用性

（1）代码重用

①内部重用

即本项目内的代码重用，主要是找出设计中相同或相似的部分，然后利用继承机制共享它们。

②外部重用

即新项目重用旧项目的代码，需要有长远眼光，反复考虑，精心设计。

（2）主要准则

有助于实现内部重用和外部重用的程序设计准则如下：

①提高方法的内聚

一个方法应该只完成单个功能。如果某个方法涉及两个或多个不相关的功能，则应该把它分解成几个更小的方法。

②减小方法的规模

应该减小方法的规模。如果某个方法规模过大，则应该把它分解成几个更小的方法。

③保持方法的一致性

保持方法的一致性，有助于实现代码重用，功能相似的方法应该有一致的名字、参数特征、返回值类型、使用条件及出错条件等。

④把策略与实现分开

a．方法分类

第一，策略方法。

策略方法应该检查系统运行状态，并处理出错情况，它们并不直接完成计算或实现复杂的算法。其紧密依赖于具体应用。

第二，实现方法。

实现方法仅仅针对具体数据完成特定处理，用于实现复杂的算法。在执行过程中发现错误，它们只返回执行状态而不对错误采取行动。其相对独立于具体应用。

b．与实现分开

把算法的核心部分放在一个单独的具体实现方法中。为此需要从策略方法中提取出具体参数，作为调用实现方法的变元。

⑤全面覆盖

a．输入条件的各种组合都可能出现，应针对所有组合写出方法。

b．一个方法对空值、极限值及界外值等异常情况也应能够做出有意义的响应。

⑥尽量不使用全局信息

应该尽量降低方法与外界的耦合程度，不使用全局信息是降低耦合度的一项主要措施。

⑦利用继承机制

在面向对象程序中，使用继承机制是实现共享和提高重用程度的主要途径。

a．调用子过程

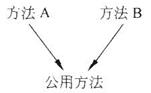


图12-1 通过调用公用方法实现代码重用

把公共的代码分离出来，构成一个被其他方法调用的公用方法。可以在基类中定义这个公用方法，供派生类中的方法调用，如图12-1所示。

b．分解因子

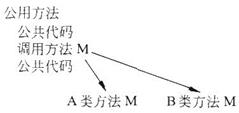


图12-2 通过因子分解实现代码重用

提高相似类代码可重用性的一个有效途径，是从不同类的相似方法中分解出不同的代码，把余下的代码作为公用方法中的公共代码，把分解出的因子作为名字相同算法不同的方法，放在不同类中定义，并被这个公用方法调用，如图12-2所示。

c．使用委托

继承关系的存在意味着子类“即是”父类，因此，父类的所有方法和属性都应该适用于子类，仅当确实存在一般—特殊关系时，使用继承才是恰当的，当逻辑上不存在一般—特殊关系时，为重用已有的代码，可以利用委托机制。

d．把代码封装在类中

重用通过其他方法编写的、解决同一类应用问题的程序代码的一个比较安全的途径是把被重用的代码封装在类中。

3．提高可扩充性

提高可扩充行的主要准则为：

（1）封装实现策略

应该把类的实现策略封装起来，对外只提供公有的接口，否则将降低今后修改数据结构或算法的自由度。

（2）不要用一个方法遍历多条关联链

一个方法应该只包含对象模型中的有限内容。违反这条准则将导致方法过分复杂，既不易理解，也不易修改扩充。

（3）避免使用多分支语句

可以利用DΟ\_CASE语句测试对象的内部状态，而不要用来根据对象类型选择应有的行为，否则在增添新类时将不得不修改原有的代码。

（4）精心确定公有方法

公有方法是向公众公布的接口。对这类方法的修改往往会涉及许多其他类，因此，修改公有方法的代价通常都比较高。为提高可修改性，降低维护成本，必须精心选择和定义公有方法私有方法是仅在类内使用的方法，通常利用私有方法来实现公有方法。删除、增加或修改私有方法所涉及的面要窄得多，因此代价也比较低。

4．提高健壮性

（1）健壮性

健壮性是在硬件故障、输入的数据无效或操作错误等意外环境下，系统能做出适当响应的程度。程序员在编写实现方法时，需要在健壮性与效率之间做出适当的折衷。

（2）提高健壮性的主要准则

①预防用户的操作错误

软件系统必须具有处理用户操作错误的能力。当用户在输入数据时发生错误，不应该引起程序运行中断，更不应该造成“死机”。

②检查参数的合法性

对公有方法，尤其应该着重检查其参数的合法性，因为用户在使用公有方法时可能违反参数的约束条件。

③不要预先确定限制条件

在设计阶段，往往很难准确地预测出应用系统中使用的数据结构的最大容量需求。因此，不应该预先设定限制条件。

④先测试后优化

为在效率与健壮性之间做出合理的折衷，应该在为提高效率而进行优化之前，先测试程序的性能。

四、测试策略

1．经典的测试策略

测试软件的经典策略是，从“小型测试”开始，逐步过渡到“大型测试”。用软件测试的专业术语描述，可以分为以下三步：

（1）单元测试；

（2）集成测试；

（3）确认测试、系统测试。

2．面向对象测试策略

（1）面向对象的单元测试

最小的可测试单元是封装起来的类和对象。一个类可以包含一组不同的操作，而一个特定的操作也可能存在于一组不同的类中。测试面向对象软件时，不能再孤立地测试单个操作，而应该把操作作为类的一部分来测试。在测试面向对象的软件时，传统的单元测试方法是不适用的，不能再在“真空”中（即孤立地）测试单个操作。

（2）面向对象的集成测试

①策略

因为在面向对象的软件中不存在层次的控制结构，传统的自顶向下或自底向上的继承策略是没有意义的。面向对象软件的集成测试主要有下述两种不同的策略：

a．基于线程的测试

把响应系统的一个输入或一个事件所需要的那些类集成起来。分别集成并测试每个线程，同时应用回归测试以保证没有产生副作用。

b．基于使用的测试

不使用服务器类的独立类，把独立类都测试完之后，再测试使用独立类的下一个层次的类（称为依赖类）。对依赖类的测试一层一层次地测试，直至把整个软件系统构造完为止。

②集群测试

集群测试是面向对象软件集成测试的一个步骤。在这个测试步骤中，用精心设计的测试用例检查一群相互协作的类，这些测试用例力图发现协作错误。

（3）面向对象的确认测试

在确认测试或系统测试层次，不再考虑类之间相互连接的细节。面向对象软件的确认测试也集中检查用户可见的动作和用户可识别的输出。为了导出确认测试用例，测试人员应该认真研究动态模型和描述系统行为的脚本，以确定最可能发现用户交互需求错误的情景。

五、设计测试用例

1．测试类的方法

（1）随机测试

通过执行一些随机产生的测试用例，来对类和对象进行测试的过程。

（2）划分测试

①优点

采用划分测试方法可以减少测试类时所需要的测试用例的数量。

②流程

a．把输入和输出分类；

b．设计测试用例以测试划分出的每个类别。

③方法

a．基于状态的划分：根据类操作改变类状态的能力来划分类操作。

b．基于属性的划分：根据类操作使用的属性来划分类操作。

c．基于功能的划分：根据类操作所完成的功能来划分类操作。

（3）基于故障的测试

首先推测软件中可能有的错误，然后设计出最可能发现这些错误的测试用例。

2．集成测试方法

（1）多类测试

和测试单个类相似，测试类协作可以使用随机测试方法和划分测试方法，以及基于情景的测试和行为测试来实现。

①随机测试

a．对每个客户类，使用类操作符列表来生成一系列随机测试序列。

b．对所生成的每个消息，确定协作类和在服务器对象中的对应操作符。

c．对服务器对象中的每个操作符，确定传递的消息。

d．对每个消息，确定下一层被调用的操作符，并把这些操作符结合进测试序列中。

②划分测试

a．应该扩充测试序列以包括那些通过发送给协作类的消息而被调用的操作。

b．根据与特定类的接口来划分类操作。

（2）从动态模型导出测试用例

类的状态图可以帮助人们导出测试该类的动态行为的测试用例。通过导出大量的测试用例，保证该类的所有行为都被适当地测试了。在类的行为导致与一个或多个类协作的情况下，应该使用多个状态图去跟踪系统的行为流。

**12.2　课后习题详解**

1．面向对象实现应该选用哪种程序设计语言？为什么？

答：（1）面向对象实现应该尽量选用面向对象语言来实现面向对象分析、设计的结果。

（2）原因

①一致的表示方法。面向对象开发基于不随时间变化的、一致的表示方法。既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分。

②可重用性。既可重用面向对象分析结果，也可重用相应的面向对象设计和面向对象程序设计结果。

③可维护性。程序显式地表达问题域语义，对维护人员理解待维护的软件有很大帮助。在选择编程语言时，应该考虑的首要因素是哪个语言能最恰当地表达问题域语义。

2．面向对象程序设计语言主要有哪些技术特点？

答：（1）支持类与对象概念的机制。

（2）实现聚集结构的机制。

（3）实现泛化结构的机制。

（4）实现属性和服务的机制。

（5）类型检查机制。

（6）类库。

（7）效率。

（8）持久保存对象的机制。

（9）参数化类的机制。

（10）开发环境。

3．选择面向对象程序设计语言时主要应该考虑哪些因素？

答：（1）将来能否占主导地位。

（2）可重用性。

（3）类库和开发环境。

（4）其他因素，包括售后服务、对运行环境的需求、集成已有软件的难易程度等。

4．良好的面向对象程序设计风格主要有哪些准则？

答：面向对象程序设计风格应遵循如下准则：

（1）提高重用性。

（2）提高可扩展性。

（3）提高健壮性。

5．测试面向对象软件时，单元测试、集成测试和确认测试各有哪些新特点？

答：（1）单元测试，是在类层面上的测试。由于继承和复合，类（或对象）在很多情况下已不再是单纯意义上的单个操作。因此，具体的测试将在多有与操作有关的每个子类语境中进行。

（2）集成测试，由于面向对象软件中类的成分直接和间接交互，使得传统测试放法已经失去意义。因此有两种策略可供选择，分别是基于线程的测试和基于使用的测试。

（3）确认测试，关注与用户可见的动作和用户识别的系统输出，但基于场景的测试总是主宰面向对象系统的确认测试。

6．测试面向对象软件时，主要有哪些设计单元测试用例的方法？

答：设计单元测试用例的方法主要有随机测试、划分测试、基于故障的测试。

（1）随机测试：通过执行一些随机产生的测试用例，来对类和对象进行测试的过程。

（2）划分测试：通过把输入和输出分类，设计测试用例以测试划分出的每个类别的过程。主要分为以下几种方法。

①基于状态的划分：根据类操作改变类状态的能力来划分类操作。

②基于属性的划分：根据类操作使用的属性来划分类操作。

③基于功能的划分：根据类操作所完成的功能来划分类操作。

（3）基于故障的测试：首先推测软件中可能有的错误，然后设计出最可能发现这些错误的测试用例。

7．测试面向对象软件时，主要有哪些设计集成测试用例的方法？

答：设计集成测试用例的方法主要有多类测试、从动态模型中导出测试用例。

（1）多类测试：多类测试可分为随机测试和划分测试两种。

①随机测试

a．对每个客户类，使用类操作符列表来生成一系列随机测试序列。

b．对所生成的每个消息，确定协作类和在服务器对象中的对应操作符。

c．对服务器对象中的每个操作符，确定传递的消息。

d．对每个消息，确定下一层被调用的操作符，并把这些操作符结合进测试序列中。

②划分测试

a．应该扩充测试序列以包括那些通过发送给协作类的消息而被调用的操作。

b．根据与特定类的接口来划分类操作。

（2）从动态模型中导出测试用例：类的状态图可以帮助人们导出测试该类的动态行为的测试用例。通过导出大量的测试用例，保证该类的所有行为都被适当地测试了。在类的行为导致与一个或多个类协作的情况下，应该使用多个状态图去跟踪系统的行为流。

8．测试面向对象软件时，主要有哪些设计确认测试用例的方法？

答：设计确认测试用例的方法主要有传统的黑盒方法、基于情景的方法。

（1）黑盒测试：黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。在测试中把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地就收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

（2）基于情景的方法：场景，是一种有假设条件的故事，可以辅助测试人员把一个复杂的问题或系统通过电影那样地过一遍。测试人员通过把整个场景都设想出来，在设想中的场景中进行的测试就是基于场景的测试。

9．试用C++语言实现（编程并测试）本书习题11第4题要求设计的订货系统。

答：略。

**第13章　软件项目管理**

**13.1　复习笔记**

一、软件项目管理总述

1．管理

管理是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。

2．软件项目管理

软件项目管理先于任何技术活动之前开始，并且贯穿于软件的整个生命周期之中。软件项目管理过程从一组项目计划活动开始，而制定计划的基础是工作量估算和完成期限估算。

二、估算软件规模

1．代码行技术

（1）定义

代码行技术依据以往开发类似产品的经验和历史数据，估计实现一个功能所需要的源程序行数。是一种比较简单的定量估算软件估摸的方法。

（2）方法

①把实现每个功能的源程序行数累加起来，可得到实现整个软件所需要的源程序行数。

②估计程序的最小规模（a）、最大规模（b）和最可能的规模（m），分别算出这3种规模的平均值后，再用下式计算程序规模的估计值：

IMG_478

③程序小时用的单位是代码行数（LOC）；程序大时用的单位是千行代码数（KLOC）。

（3）优点

①代码是所有软件开发项目都有的“产品”，而且很容易计算代码行数。

②有以往开发类似产品的历史数据可参考时，估计出的数值比较准确。

（4）缺点

①源程序仅是软件配置的一个成分，用它的规模代表整个软件的规模不太合理。

②用不同语言实现同一个软件所需要的代码行数并不相同。

③不适用于非过程语言。

2．功能点技术

（1）定义

功能点技术依据对软件信息域特性和软件复杂性的评估结果，估算软件规模。用功能点（FP）为单位度量软件规模。是为了克服代码行技术的缺点，提出来的新技术。

（2）信息域特性

①输入项数（Inp）：用户向软件输入的项数，这些输入给软件提供面向应用的数据。

②输出项数（Out）：软件向用户输出的项数，它们向用户提供面向应用的信息。

③查询数（Inq）：一次联机输入，它导致软件以联机输出方式产生某种即时响应。

④主文件数（Maf）：逻辑主文件（数据的一个逻辑组合）的数目。

⑤外部接口数（Inf）：机器可读的全部接口数量，用这些接口把信息传送给另一个系统。

（3）估算功能点的步骤

①计算未调整的功能点数UFP

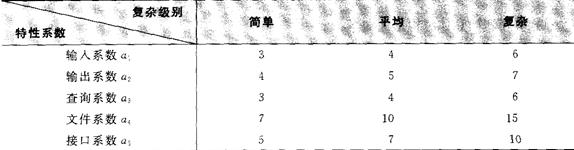
a．把产品信息域的每个特性都分类为简单级、平均级或复杂级，并根据其等级为每个特性分配一个功能点数。

b．用下式计算未调整的功能点数UFP：

IMG_479

其中，ai（1≤i≤5）是信息域特性系数，由相应特性的复杂级别决定，如表13-1所示。

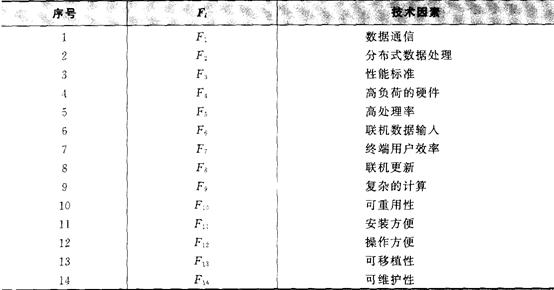
表13-1 信息域特性系数值



②计算技术复杂性因子TCF

这一步骤度量14种技术因素对软件规模的影响程度，在表13-2中列出了全部技术因素，并用Fi（1≤i≤14）代表这些因素。

表13-2 技术因素



a．根据软件的特点，为每个因素分配一个从0（不存在或对软件规模无影响）到5（有很大影响）的值。

b．用下式计算技术因素对软件规模的综合影响程度DI（0～70）：

IMG_482

c．技术复杂性因子TCF（0.65～1.35）由下式计算：

TCF=0.65+0.01×DI

③计算功能点数FP

用下式计算功能点数FP：

FP=UFP×TCF

功能点数与所用的编程语言无关，所以功能点技术比代码行技术更合理，但是，在判断信息域特性复杂级别和技术因素的影响程度时，功能点技术存在相当大的主观因素。

三、工作量估算

软件估算模型使用由经验导出的公式来预测软件开发工作量，工作量是软件规模的函数，工作量的单位通常是人月（pm）。没有一个估算模型可以适用于所有类型的软件和开发环境。

1．静态单变量模型

（1）形式

静态单变量模型的总体结构形式如下：

E=A+B×（eν）c

其中，A、B和C是由经验数据导出的常数，E是以人月为单位的工作量，eν是估算变量（KLOC或FP）。

（2）分类

①面向KLOC的估算模型

a．Walston\_Felix模型

E=5.2×（KLOC）0.91

b．Bailey\_Basili模型

E=5.5+0.73×（KLOC）1.16

c．Boehm简单模型

E=3.2×（KLOC）1.05

d．Doty模型（KLOC>9时适用）

E=5.288×（KLOC）1.047

②面向FP的估算模型

a．Albrecht ＆ Gaffney模型

E=-13.39+0.0545FP

b．Maston，Barnett和Mellichamp模型

E=585.7+15.12FP

2．动态多变量模型

①定义

动态多变量模型（软件方程式）是根据从4000多个当代软件项目中收集的生产率数据推导出来的。该模型把工作量看作软件规模和开发时间这两个变量的函数。

②形式

动态多变量估算模型的形式如下：

IMG_483

其中，E是以人月或人年为单位的工作量；t是以月或年为单位的项目持续时间；B是特殊技术因子，对于较小的程序（KLOC=5～15），B=0.16，对于超过70 KLOC的程序，B=0.39；P是生产率参数，它反映了以下因素对工作量的影响：

a．总体过程成熟度及管理水平。

b．使用良好的软件工程实践的程度。

c．使用的程序设计语言的级别。

d．软件环境的状态。

e．软件项目组的技术及经验。

f．应用系统的复杂程度。

3.COCOMO2模型

（1）3层模型

COCOMO是构造性成本模型（constructive cost model）的英文缩写。COCOMO2给出了3个层次模型，这3个层次的模型在估算工作量时，对软件细节考虑的详尽程度逐级增加。这3个模型如下：

①应用系统组成模型：主要用于估算构建原型的工作量，模型名字暗示在构建原型时大量使用已有的构件。

②早期设计模型：适用于体系结构设计阶段。

③体系结构模型：适用于完成体系结构设计之后的软件开发阶段。

（2）特点

①这个层次的模型在估算工作量时，对软件细节考虑的详尽程度逐级增加。

②这些模型既可以用于不同类型的项目，也可以用于同一个项目的不同开发阶段。

（3）形式

COCOMO2模型把软件开发工作量表示成代码行数（KLOC）的非线性函数：

IMG_484

其中：E是开发工作量（以人月为单位）；a是模型系数；KLOC是估计的源代码行数（以千行为单位）；b是模型指数；fi（i=1～17）是成本因素。

（4）与COCOMO模型所使用成本因素的区别

①新增加了4个成本因素，它们分别要求的是可重用性、需要的文档量、人员连续性和多地点开发。

②略去了原始模型中的两个成本因素。

③某些成本因素（分析员能力、平台经验、语言和工具经验）对生产率的影响增加了，另一些成本因素（程序员能力）的影响减小了。

（5）模型指数

COCOMO2采用了的b分级模型，是使用5个分级因素Wi（1≤i≤5），其中每个因素都划分成从甚低（Wi=5）到特高（wi=0）的6个级别，用下式计算b（1.01～1.26）的数值：

IMG_485

（6）COCOMO2的5个分级因素

①项目先例性：指出对于开发组织来说该项目的新奇程度。

②开发灵活性：反映出为实现预先确定的外部接口和为了及早开发出产品而增加的工作量。

③风险排除度：反映了重大风险已被消除的比例。

④项目组凝聚力：表明了开发人员相互协作时可能存在的困难。

⑤过程成熟度：反映了按照能力成熟度模型度量出的项目组织的过程成熟度。

四、进度计划

1．相关概念

（1）任务集合

一个有效的软件过程应该定义一个适用于当前项目的任务集合。一个任务集合包括一组软件工程工作任务、里程碑和可交付的产品。为一个项目所定义的任务集合，必须包括为获得高质量的软件产品而应该完成的所有任务，但是同时又不能让项目组承担不必要的工作。

（2）项目管理者的工作

①目标

定义全部项目任务，识别出关键任务，跟踪关键任务的进展状况，保证及时发现拖延进度的情况。

②方法

管理者必须制定一个足够详细的进度表，以便监督项目进度并控制整个项目。

（3）进度安排

①定义

软件项目的进度安排通过把工作量分配给特定的软件工程任务并规定完成各项任务的起止日期，从而将估算出的项目工作量分布于计划好的项目持续期内。进度计划将随着时间的流逝而不断演化。

②流程

a．在项目计划的早期，制定一个宏观的进度安排表，标识出主要的软件工程活动和这些活动影响到的产品功能。

b．随着项目的进展，把宏观进度表中的每个条目都精化成一个详细进度表，标识出完成一个活动所必须实现的一组特定任务，并安排好实现这些任务的进度。

2．估算开发时间

（1）利用成本估算模型估算开发时间

①Walston\_Felix模型：

T=2.5E0.35

②原始的COCOMO模型：

T=2.5E0.38

③COCOMO2模型：

T=3.0E0.33+0.2×（b-1.01）

④Putnam模型：

T=2.4E1/3

其中，E是开发工作量（以人月为单位）；T是开发时间（以月为单位）。

（2）特殊情况

①描述

随着开发小组规模的扩大，个人生产率将下降，以致开发时间与从事开发工作的人数并不成反比关系。

②原因

a．小组变得更大时，每个人需要用更多时间与组内其他成员讨论问题、协调工作，因此增加了通信开销。

b．如果在开发过程中增加小组人员，最初一段时间内项目组总生产率不仅不会提高反而会下降。因为新成员在开始时不是生产力，且在他们学习期间需花费小组其他成员的时间。

c．Brooks规律

向一个已经延期的项目增加人力，只会使得它更加延期。

（3）项目组规模与项目组总生产率的关系

①通信路径

项目组成员之间的通信路径数，由项目组人数和项目组结构决定。通信路径数大约在P～P2/2的范围内变化。

②平均生产力

某一个组员与其他组员通信的路径数在1～（P-1）的范围内变化。如果不与任何人通信时个人生产率为L，而且每条通信路径导致生产率减少1，则组员个人平均生产率为：

Lr=L-1（P-1）r

其中，r是对通信路径数的度量，o<r≤l。

③总生产率

对于一个规模为P的项目组，项目组的总生产率为：

Ltot=P（L-l（P-1）r）

对于给定的一组L，1和r的值，总生产率Ltot是项目组规模P的函数。存在一个最佳的项目组规模Popt这个规模的项目组其总生产率最高。

3.Gantt图

（1）例子

【例题】假设有一座陈旧的矩形木板房需要重新油漆。这项工作必须分3步完成：首先刮掉旧漆，然后刷上新漆，最后清除溅在窗户上的油漆。假设一共分配了15名工人去完成这项工作，然而工具却很有限：只有5把刮旧漆用的刮板，5把刷漆用的刷子，5把清除溅在窗户上的油漆用的小刮刀。怎样安排才能使工作进行得更有效呢？

【解析】每道工序需要的时间如表13-3所示。

表13-3 各道工序估计需用的时间（小时）



可以使用图13-1中Gantt图描绘上述流水作业过程：

①在时间为零时开始刮第1面墙上的旧漆；

②两小时后刮旧漆的工人转去刮第2面墙，同时另5名工人开始给第1面墙刷新漆；

③当给一面墙刷完新漆之后，第3组的5名工人立即清除溅在这面墙窗户上的漆。

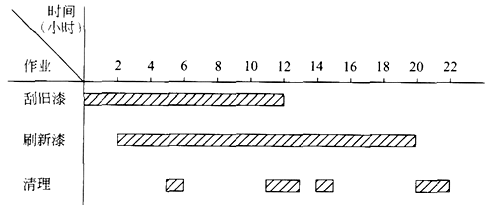


图13-1 旧木板房刷漆丁程的Gantt图

从图13-1可以看出，12小时后刮完所有旧漆，20小时后完成所有墙壁的刷漆工作，再过2小时后清理工作结束。因此全部工程在22小时后结束。

（2）优点

①很形象地描绘任务分解情况，以及每个子任务（作业）的开始时间和结束时间。

②容易掌握、容易绘制。

（3）缺点

①不能显式地描绘各项作业彼此间的依赖关系。

②进度计划的关键部分不明确，难于判定哪些部分应当是主攻和主控的对象。

③计划中有潜力的部分及潜力的大小不明确，往往造成潜力的浪费。

4．工程网络

（1）定义

工程网络可以描绘任务分解情况以及每项作业的开始时间和结束时间，它还显式地描绘各个作业彼此间的依赖关系。

（2）表示

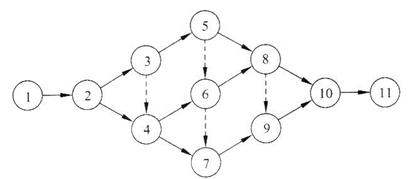


图13-2 工程网络

工程网路，如图13-2所示。

①用箭头表示作业，作业通常既消耗资源又需要持续一定时间。

②用圆圈表示事件（开始或结束），事件是明确定义的时间点，并不消耗时间和资源。

③用虚线箭头表示虚拟作业，虚拟作业是为了显式地表示作业之间的依赖关系。

5．估算工程进度

（1）完善工程网络

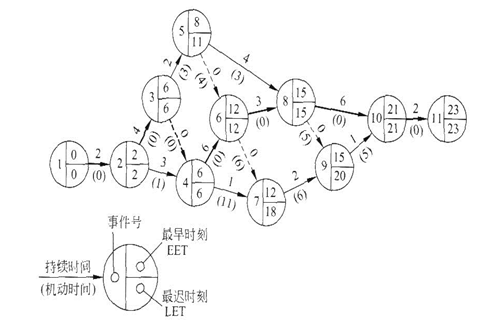


图13-3 旧木板房刷漆工程的完整的工程网络

（粗线箭头是关键路径）

①把每个作业估计需要使用的时间写在表示该项作业的箭头上方。

②为每个事件计算最早时刻EET和最迟时刻LET，分别写在表示事件的圆圈的右上角和右下角，如图13-3左下角的符号所示。

（2）最早时刻EET

①定义

事件的最早时刻是该事件可以发生的最早时间。通常工程网络中第一个事件的最早时刻定义为零，其他事件的最早时刻在工程网络上从左至右按事件发生顺序计算。

②计算规则

a．考虑进入该事件的所有作业。

b．对于每个作业都计算它的持续时间与起始事件的EET之和。

c．选取上述和数中的最大值作为该事件的最早时刻EET。

（3）最迟时刻LET

①定义

事件的最迟时刻是在不影响工程竣工时间的前提下，该事件最晚可以发生的时刻。最后一个事件的最迟时刻就是它的最早时刻。其他事件的最迟时刻在工程网络上从右至左按逆作业流的方向计算。

②计算规则

a．考虑离开该事件的所有作业。

b．从每个作业的结束事件的最迟时刻中减去该作业的持续时间。

c．选取上述差数中的最小值作为该事件的最迟时刻LET。

6．关键路径

（1）定义

由最早时刻和最迟时刻相同的事件定义了关键路径，如图13-3所示。关键事件必须准时发生，组成关键作业的实际持续时间不能超过估计的持续时间，否则工程就不能准时结束。

（2）特点

①处于关键路径之外的任务进度拖后，不会影响整个项目的完成时间。

②处于关键路径之中的任务进度拖后，则整个项目的完成日期就会拖后。

7．机动时间

（1）定义

不在关键路径上的作业有一定程度的机动余地一实际开始时间可以比预定时间晚一些，或者实际持续时间可以比预定的持续时间长一些，而并不影响工程的结束时间。

（2）计算

一个作业可以有的全部机动时间等于它的结束事件的最迟时刻减去它的开始事件的最早时刻，再减去这个作业的持续时间：

机动时间=（LET）结束-（EET）开始-持续时间

（3）表示

工程网络中每个作业的机动时间写在该项作业的箭头下面的括号里，如图l3-3所示。在制定进度计划时仔细考虑和利用工程网络中的机动时间，往往能够安排出既节省资源又不影响最终竣工时间的进度表。

五、人员组织

1．必要性

（1）项目成功的关键合理的组织人员，使他们有效地分工协作共同完成开发工作。

（2）项目组组织得越好，其生产率越高，而且产品质量也越好。

（3）项目组具有了凝聚力，成功的可能性就大大增加了。

2．典型的组织方式

现有的软件项目组的组织方式很多，通常，组织软件开发人员的方法，取决于所承担的项目的特点、以往的组织经验以及管理者的看法和爱好，主要有3种典型的组织方式。

（1）民主制程序员组

①定义

民主制程序员组中小组成员完全平等，享有充分民主，通过协商做出技术决策。即小组成员之间的通信是平行的，如果小组内有n个成员，则可能的通信信道共有n（n-1）/2条。

②要求

a．小组的人数不能太多（2～8名成员为宜）

小组规模小，可以减少通信问题、容易确定小组的质量标准、用民主方式确定的标准更容易被大家遵守、组员间关系密切、能够互相学习。

b．采用非正式的组织方式

名义上有一个组长，但是他和组内其他成员完成同样的任务。在这样的小组中，由全体讨论协商决定应该完成的工作，并且根据每个人的能力和经验分配适当的任务。

③优点

a．组员们对发现程序错误持积极的态度，有助于更快速地发现错误，提高代码质量。

b．组员们享有充分民主，小组凝聚力高、学术空气浓厚，有利于攻克技术难关。

④缺点

没有明确的权威指导开发过程，组员间将缺乏必要的协调，最终可能导致工程失败。

⑤适用性

所要开发的软件的技术难度较高时，采用民主制程序员组是适宜的。

（2）主程序员组

①定义

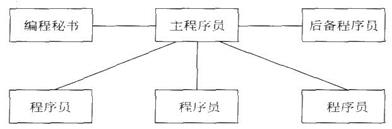


图13-4 主程序员组的结构

主程序员组用经验多、技术好、能力强的程序员作为主程序员，同时，利用人和计算机在事务性工作方面给主程序员提供充分支持，而且所有通信都通过一两个人进行。典型的主程序员组的组织形式如图13-4所示。

②核心人员及其分工

a．主程序员

既是成功的管理人员又是经验丰富、技术好、能力强的高级程序员，负责体系结构设计和关键部分的详细设计，并且负责指导其他程序员完成详细设计和编码工作。

b．后备程序员

技术熟练而且富于经验，协助主程序员工作并且在必要时接替主程序员的工作。具体工作是设计测试方案、分析测试结果及独立于设计过程的其他工作。

c．编程秘书

负责完成与项目有关的全部事务性工作。

注意：图13-4介绍的是20世纪70年代初期的主程序员组组织结构，现在的情况已经和当时大不相同了，程序员已经有了自己的终端或工作站，他们自己完成代码的输入、编辑、编译、链接和调试等工作，无须由编程秘书统一做这些工作。

③特点（优点）

a．专业化：该组每名成员仅完成他们擅长的工作。

b．层次性：主程序员指挥组员工作，并对项目全面负责

④缺点

符合主程序员、后备程序员、编辑秘书标准的人才在现实社会中并不容易雇佣到。

⑤适用性

采用主程序员组这种组织方式的程序一般具有以下几方面的特点：

a．软件开发人员多数比较缺乏经验。

b．程序设计过程中有许多事务性的工作。

c．多渠道通信很费时间，将降低程序员的生产率。

（3）现代程序员组

①主程序员由两个人共同担任

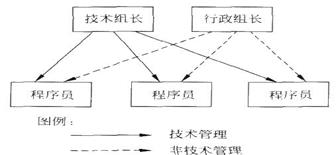


图13-5 现代程序员组的结构

组织结构如图13-5所示。

a．技术负责人

负责小组的技术活动，参与全部代码审查工作，并且对代码的各方面质量负责。

b．行政负责人

负责非技术性事务的管理决策。不参与代码审查，其职责是对程序员的业绩进行评价。

②制定针对公共职责范围内的事务的处理方案

③实行分组策略

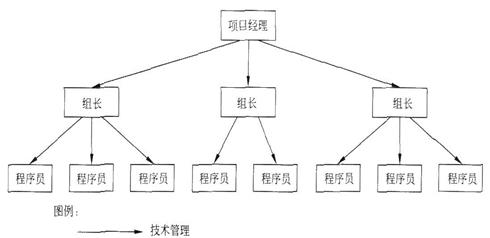


图13-6 大型项目的技术管理组织结构

采用分组策略，如图13-6所示。产品开发作为一个整体是在项目经理的指导下进行的，程序员向他们的组长汇报工作，而组长则向项目经理汇报工作。当产品规模更大时，可以适当增加中间管理层次。

④分散决定

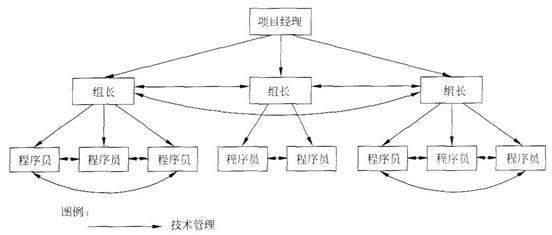


图13-7 包含分散决策的组织方式

在合适的地方采用分散做决定的方法，如图13-7所示。这样做有利于形成畅通的通信渠道，以便充分发挥每个程序员的积极性和主动性，集思广益攻克技术难关。

六、质量保证

1．软件质量

（1）定义

软件质量是软件与明确地和隐含地定义的需求相一致的程度，即软件质量是软件与明确地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。

（2）特点

①软件需求是度量软件质量的基础，与需求不一致就是质量不高。

②指定的开发标准定义了指导软件开发准则，没有遵守这些准则，会导致软件质量不高。

③软件满足明确描述的需求，但不满足隐含的需求，那么软件的质量是值得怀疑的。

（3）软件质量因素与产品活动

①软件质量因素

如表13-4所示，列出了软件质量因素的简明定义。

表13-4 软件质量因素的定义



②产品活动

可以把产品活动（倾向）分为产品运行、产品修改和产品转移。

③关系

软件质量因素和3种产品活动（倾向）之间的关系，如图13-8所示。



图13-8 软件质量因素与产品活动的关系

2．软件质量保证措施

（1）措施

①基于非执行的测试（复审或评审）：主要用来保证在编码前各阶段产生的文档的质量。

②基于执行的测试（软件测试）：在程序编写完后进行， 保证软件质量的最后一道防线。

③程序正确性证明：使用数学方法严格验证程序是否与对它的说明完全一致。

（2）参加软件质量保证的人员分类

①软件工程师：用先进的技术方法和度量，进行复审以及完成软件测试来保证软件质量。

②SQA小组：辅助软件工程师以获得高质量的软件产品。其从事的软件质量保证活动的主要是：计划，监督，记录，分析和报告，它通过确保软件过程的质量来保证软件产品的质量。

（3）技术复审

正式技术复审的优点是能较早发现软件错误，防止错误被传播到软件过程的后续阶段。包括走查和审查等具体方法。

①走查

a．走查组

走查组由4～6名成员组成。成员包括负责起草文档的人、负责该文档说明的管理员、客户代表、下阶段开发组的代表、SQA小组的代表（作为组长）。

b．要点

第一，为了能发现重大错误，走查组成员最好是经验丰富的高级技术人员。

第二，走查组成员应根据材料并列出不理解的术语和认为不正确的术语。

第三，走查组组长引导该组成员走查文档，力求发现尽可能多的错误。

第四，走查的时间最长不要超过2小时。

c．方式

第一，参与者驱动法

参与者按照事先准备好的列表，提出他们不理解的术语和认为不正确的术语。文档编写组的代表必须回答每个质疑。

第二，文档驱动法（更有效）

文档编写者向走查组成员仔细解释文档。走查组成员在此过程中针对事先准备好的问题或解释过程中发现的问题提出质疑。

②审查

a．审查组

查组由4人组成，分别是组长（既是管理人员又是技术负责人）、负责当前阶段开发工作的项目组代表、负责下一阶段开发工作的项目组代表、SQA小组的代表。

b．步骤

第一，综述

由负责编写文档的成员向审查组综述该文档。

第二，准备

评审员仔细阅读文档。

第三，审查

评审组仔细走查整个文档。

第四，返工

文档的作者负责解决在审查报告中列出的所有错误及问题。

第五，跟踪

组长必须确保所提出的每个问题都得到了圆满的解决。

c．与走查比较

第一，审查过程步数比走查多。

第二，审查过程每个步骤都是正规的仔细划分错误类型，并把这些信息运用在后续阶段的文档审查中以及未来产品的审查中。

d．重要性

审查是检测软件错误的一种好方法，利用审查可以在软件过程的早期阶段（修改成本低的阶段）发现并改正错误，即审查是一种经济有效的错误检测方法。

注意：走查的步骤比审查少，没有审查正规。

（4）程序正确性证明

①定义

正确性证明的基本思想是证明程序能完成预定的功能。应提供对程序功能的严格数学说明，然后根据程序代码证明程序确实能实现它的功能说明。

②方法

a．人工证明程序正确性

对于评价小程序可能有些价值，但在证明大型软件的正确性时，不仅工作量太大，而且在证明的过程中很容易包含错误。

b．自动系统

七、软件配置管理

1．相关概念

（1）定义

软件配置管理是在软件的整个生命期内管理变化的一组活动。其主要任务是控制变化，同时也负责各个软件配置项和软件各种版本的标识、软件配置审计以及对软件配置发生的任何变化的报告。

（2）目的

软件配置管理不同于软件维护，它的目标是使变化更正确且更容易被适应，在必须变化时减少所需花费的工作量。具体如下：

①标识变化；

②控制变化；

③确保适当地实现了变化；

④向需要知道这类信息的人报告变化。

（3）与维护的区别

维护是在软件交付给用户使用后才发生的，而配置管理是在软件项目启动时就开始，并且一直持续到软件退役后才终止的一组跟踪和控制活动。

2．软件配置

（1）软件配置项

软件过程的输出信息可以分为3类：

①计算机程序（源代码和可执行程序）。

②描述计算机程序的文档（供技术人员或用户使用）。

③数据（程序内包含的或在程序外的）。

（2）基线

①定义

基线是已经通过了正式复审的规格说明或中间产品，它可以作为进一步开发的基础，并且只有通过正式的变化控制过程才能改变它，即基线就是通过了正式复审的软件配置项。

②作用

基线有助于人们在不严重妨碍合理变化的前提下来控制变化。

（3）软件工具

把特定版本的编辑器、编译器和其他CASE工具，作为软件配置的一部分。为防止不同版本的工具产生结果不同，应把软件工具也基线化，并且列入到综合的配置管理过程之中。

3．软件配置管理过程

软件配置管理是软件质量保证的重要一环，它的主要任务是控制变化，同时也负责各个软件配置项和软件各种版本的标识、软件配置审计以及对软件配置发生的任何变化的报告，具体来说，软件配置管理主要有5项任务：标识、版本控制、变化控制、配置审计和报告。

（1）标识软件配置中的对象

①对象分类

a．基本对象：是软件工程师在分析、设计、编码或测试过程中创建出来的“文本单元”。

b．聚集对象：是基本对象和其他聚集对象的集合。

②要点

a．每个对象都有一组能唯一地标识它的特征：名字、描述、资源表和实现。

b．对象名是无二义性地标识该对象的一个字符串。

c．标识模式必须能无歧义地标识每个对象的不同版本。

（2）版本控制

①定义

版本控制使用规程和工具，以管理在软件工程过程中所创建的配置对象的不同版本。

②目标

借助于版本控制技术，用户能通过选择适当的版本来指定软件系统的配置。

③步骤

a．把属性和软件的每个版本关联起来。

b．描述一组所期望的属性（施加到系统上的功能变化的具体类型）来指定和构造所需要的配置。

（3）变化控制

①定义

变化控制把人的规程和自动工具结合起来，以提供一个控制变化的机制。

②过程

a．评估该变化在技术方面的得失、可能产生的副作用、对其他配置对象和系统功能的整体影响以及估算出的修改成本。

b．根据评估结果形成变化报告，供变化控制审批者审阅。

c．为每个被批准的变化都生成一个工程变化命令，描述将要实现的变化，必须遵守的约束以及复审和审计的标准。

d．把要修改的对象从项目数据库中提取出来，进行修改并应用适当的SQA活动。

e．把修改后的对象提交进数据库，用适当的版本控制机制创建该软件的下一个版本。

③主要功能

a．访问控制：决定哪个软件工程师有权访问和修改一个特定的配置对象。

b．同步控制：助于保证由两名不同的软件工程师完成的并行修改不会相互覆盖。

（4）配置审计

①正式的技术复审

正式的技术复审关注被修改后的配置对象的技术正确性。复审者审查该对象以确定它与其他软件配置项的一致性，并检查是否有遗漏或副作用。

②软件配置审计

软件配置审计通过评估配置对象的那些通常不在复审过程中考虑的特征，而成为对正式技术复审的补充。

（5）状态报告

①内容

a．发生的事件。

b．做的这件事的人。

c．事件是发生的时间。

d．产生的影响。

②作用

配置状态报告对大型软件开发项目的成功有重大贡献。配置状态报告通过改善所有相关人员之间的通信，帮助消除由于通信不精确、不及时所产生的严重问题。

八、能力成熟度模型

1．能力成熟度模型（CMM）

（1）定义

能力成熟度模型（CMM）是用于评价软件机构的软件过程能力成熟度的模型。

（2）目的

①为大型软件项目的招投标活动提供一种全面而客观的评审依据。

②应用于许多软件机构内部的过程改进活动中。

（3）基本思想

能力成熟度模型的基本思想是，由于问题是由人们管理软件过程的方法不当引起的，所以新软件技术的运用并不会自动提高软件的生产率和质量。

（4）作用

能力成熟度模型有助于软件开发机构建立一个有规律的、成熟的软件过程。改进后的软件过程将开发出质量更好的软件，使更多的软件项目免受时间延误和费用超支之苦。

（5）CMM在改进软件过程中所起的作用

①指导软件机构通过确定当前的过程成熟度并识别出对过程改进起关键作用的问题，明确过程改进的方向和策略。

②通过集中开展与过程改进的方向和策略相一致的一组过程改进活动，软件机构便能稳步而有效地改进其软件过程，使其软件过程能力得到循序渐进的提高。

（6）对能力成熟度划分的原因

①对软件过程的改进，是在完成一个又一个小的改进步骤基础上不断进行的渐进过程。

②这5个成熟度等级定义了一个有序的尺度，用以测量软件机构的软件过程成熟度和评价其软件过程能力，这些等级还帮助软件机构把应做的改进工作排出优先次序。

③成熟度等级是妥善定义的向成熟软件机构前进途中的平台，每个成熟度等级都为软件过程的继续改进提供了一个台阶。

2．能力成熟度的5个等级

（1）内容

①反映出软件机构为了达到从无序的、混乱的软件过程进化到有序的、有纪律的且成熟的软件过程的目的，必须经历的过程改进活动的途径。

②每个成熟度级别都是该软件机构沿着改进其过程的途径前进途中的一个台阶，后一个成熟度级别是前一个级别的软件过程的进化目标。

③每个成熟度级别中都包含一组过程改进的目标，满足这些目标后一个机构的软件过程就从当前级别进化到下一个成熟度级别。

（2）5个级别

①初始级（1级）

软件过程能力是不可预测的，其软件过程是不稳定的，产品质量只能根据相关人员的个人工作能力来预测。

②可重复级（2级）

软件项目的策划和跟踪是稳定的，已经为一个有纪律的管理过程提供了可重复以前成功实践的项目环境。

③已定义级（3级）

无论是管理活动还是工程活动都是稳定的。软件开发的成本和进度以及产品的功能和质量都受到控制，而且软件产品的质量具有可追溯性。

④已管理级（4级）

软件机构对软件过程和软件产品都建立了定量的质量目标，所有项目的重要的过程活动都是可度量的，软件过程在可度量的范围内运行。

⑤优化级（5级）

软件机构能够不断地改进其过程能力，既对现行的过程实例不断地改进和优化，又借助于新技术和新方法来实现未来的过程改进。这一级的软件机构是一个以防止出现缺陷为目标的机构，它有能力识别软件过程要素的薄弱环节，并有足够的手段改进它们。

**13.2　课后习题详解**

1．研究本书所述的订货系统，要求：

（1）用代码行技术估算本系统的规模；

（2）用功能点技术估算本系统的规模；

（3）用静态单变量模型估算开发本系统所需的工作量；

（4）假设由一个人开发本系统，试制定进度计划；

（5）假设由两个人开发本系统，试制定进度计划。

答：略。

2．研究本书习题中描述的储蓄系统，要求：

（1）用代码行技术估算本系统的规模；

（2）用功能点技术估算本系统的规模；

（3）用静态单变量模型估算开发本系统所需的工作量；

（4）假设由一个人开发本系统，试制定进度计划；

（5）假设由两个人开发本系统，试制定进度计划。

答：略。

3．下面叙述对一个计算机辅助设计（CAD）软件的需求：该CAD软件接受由工程师提供的二维或三维几何图形数据。工程师通过用户界面与CAD系统交互并控制它，该用户界面应该表现出良好的人机界面特征。几何图形数据及其他支持信息都保存在一个CAD数据库中。开发必要的分析、设计模块，以产生所需要的输出，这些输出将显示在各种不同的图形设备上。应该适当地设计软件，以便与外部设备交互并控制它们。所用的外部设备包括鼠标、数字化扫描仪和激光打印机。

（1）进一步精化上述要求，把CAD软件的功能分解成若干个子功能；

（2）用代码行技术估算每个子功能的规模；

（3）用功能点技术估算每个子功能的规模；

（4）从历史数据得知，开发这类系统的平均生产率是620LOC/pm，如果软件工程师的平均月薪是8000元，试估算开发本系统的工作量和成本；

（5）如果从历史数据得知，开发这类系统的平均生产率是6.5 FP/pm，试估算开发本系统的工作量和成本。

答：（1）经过对需求的进一步精化，分解出软件的下述7个主要的子功能。

①用户界面及控制机制。

②二维几何图形分析。

③三维几何图形分析。

④数据库管理。

⑤计算机图形显示机制。

⑥外部设备控制。

⑦设计分析模块。

（2）为了用代码行技术估算软件规模，应该针对每个子功能都分别估计出下述3个值：乐观值（即最小规模）、悲观值（即最大规模）和可能值（即最可能规模）。然后用加权平均法计算每个子功能的规模，结果如表13-5所示。

表13-5 代码行技术的估算表



（3）使用功能点技术估算软件规模时，对软件的分解是基于信息域特性而不是基于软件功能。

①表13-6给出了对5个信息域特性的估计值。为了计算未调整的功能点数，假设每个信息域特性的复杂度都是平均级别的。

②估计14个技术复杂性因素的值，并且计算DI的值，表13-7列出了得到的结果。

表13-6 估算未调整的功能点数



表13-7 估算技术复杂性因素



③用下式计算技术复杂性因子：

TCF=0.65+0.01×DI=1.14

④计算功能点数：

FP=UFP×TCF=318×1.14=363

（4）工作量和成本：

①用代码行技术估算，开发本系统的工作量为：

E=33200/620 ≈54（人月）

②开发本系统的成本为：

8000×54=432000（元）

（5）工作量和成本：

①用功能点技术估算，开发本系统的工作量为：

E=363/6.5≈56（人月）

②开发本系统的成本为：

8000×56=448000（元）

4．假设自己被指定为项目负责人，任务是开发一个应用系统，该系统类似于自己的小组以前做过的那些系统，但是规模更大且更复杂一些。客户已经写出了完整的需求文档。应选用哪种项目组结构？为什么？打算采用哪种（些）软件过程模型？为什么？

答：（1）由于待开发的应用系统类似于以前做过的系统，开发人员已经积累了较丰富的经验，没有多少技术难题需要攻克。为了减少通信开销，充分发挥技术骨干的作用，统一意志统一行动，提高生产率，加快开发进度，项目组的组织结构以基于主程序员组的形式为宜。

（2）针对待开发的系统，客户已经写出了完整的需求文档，项目组又有开发类似系统的经验，因此，可以采用广大软件工程师熟悉的瀑布模型来开发本系统。

5．假设自己被指派为一个软件公司的项目负责人，任务是开发一个技术上具有创新性的产品，该产品把虚拟现实硬件和最先进的软件结合在一起。由于家庭娱乐市场的竞争非常激烈，这项工作的压力很大。应选择哪种项目组结构？为什么？打算采用哪种（些）软件过程模型？为什么？

答：（1）由于是技术上具有创新性的产品，所以需要采用民主制程序员组，大家可以集思广益，共同攻关技术难题。

（2）要求把虚拟现实硬件和最先进的软件结合在一起，所以需要采用一种完整而且完美的模型进行开发，所以RUP最为合适。

6．假设自己被指派作为一个大型软件产品公司的项目负责人，工作是管理该公司已被广泛应用的字处理软件的新版本开发。由于市场竞争激烈，公司规定了严格的完成期限并且对外公布了。应选择哪种项目组结构？为什么？打算采用哪种（些）软件过程模型？为什么？

答：（1）应该选择现代程序员组，因为小组成员都能对发现程序错误持积极、主动的态度。能更好的适应竞争。

（2）大型软件应采用演化模型中的螺旋模型。

7．什么是软件质量？试叙述它与软件可靠性的关系。

答：（1）软件质量是软件与明确地和隐含地定义的需求相一致的程度。

（2）软件质量是软件与明确地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。软件可靠性是程序在给定的时间间隔内按照规格说明书的规定成功地运行的概率。

8．一个程序能既正确又不可靠吗？解释自己的答案。

答：（1）软件可靠性是程序在给定的时间间隔内按照规格说明书的规定成功地运行的概率。软件可靠性既包含正确性又包含健壮性，即不仅在预定环境下程序应该能正确地完成预期功能，而且在硬件发生故障、输入的数据无效或用户操作错误等意外环境下，程序也应该能作出适当的响应。

（2）如果一个程序在预定环境下能够正确地完成预期的功能，但是在意外环境下不能作出适当的响应，则该程序就是既正确又不可靠。

9．仅当每个与会者都在事先作了准备时，正式的技术复审才能取得预期的效果。如果自己是复审小组的组长，怎样发现事先没做准备的与会者？打算采取什么措施来促使大家事先做准备？

答：（1）可以通过如下关于软件配置复审因素的问题来发现实现没有做准备的与会者：

①变更指令中指令的变更是否完成？

②每个附加变更是否已经纳入到系统中？

③是否进行了正时技术复审？

④是否遵循软件工程标准？

⑤变更的软件配置项是否做了特殊标记而得到强调？

⑥是否注明变更日期和变更执行人员？

⑦软件配置属性是否反映了变更？

⑧是否遵循与变更有关的注释、记录及报告的软件配置管理规程？

⑨相关的软件配置是否都得到了同步更新？

（2）经常督促大家事先做充足的准备，平日里严格要求每次在准备开会前几天再次提醒每个与会者，开会时记录下每个与会者回答问题的准确程度，根据此内容对与会者进行适当的赏罚。

10．什么是基线？为什么要建立基线？

答：（1）基线是已经通过了正式复审的规格说明或中间产品，它可以作为进一步开发的基础，并且只有通过正式的变化控制过程才能改变它。

（2）建立基线的三大原因

①重视性

重视性是指及时返回并重新生成软件系统给定发布的能力，或是在项目中的早些时候重新生成开发环境的能力。

②可追踪性

可追踪性建立项目工件之间的前后继承关系，其目的在于确保设计满足要求、代码实施设计以及用正确代码编译可执行文件。

③报告

报告来源于一个基线内容同另一基线内容的比较。基线比较利于调试并生成发布说明。

11．配置审计和技术复审有何不同？可否把它们的功能放在一次复审中完成？

答：（1）正式的技术复审关注被修改后的配置对象的技术正确性。

（2）复审者审查该对象以确定它与其他软件配置项的一致性，并检查是否有遗漏或副作用。软件配置审计通过评估配置对象的那些通常不在复审过程中考虑的特征，而成为对正式技术复审的补充。

12.CMM的基本思想是什么？为什么要把能力成熟度划分成5个等级？

答：（1）CMM的基本思想是，由于软件危机是因对软件过程管理不善而引起的，所以新软件技术的运用并不会自动提高软件的生产率和质量，提高软件生产率和软件质量的关键是改进对软件过程的管理。

（2）对能力成熟度划分的原因

①对软件过程的改进，是在完成一个又一个小的改进步骤基础上不断进行的渐进过程。

②这5个成熟度等级定义了一个有序的尺度，用以测量软件机构的软件过程成熟度和评价其软件过程能力，这些等级还帮助软件机构把应做的改进工作排出优先次序。

③成熟度等级是妥善定义的向成熟软件机构前进途中的平台，每个成熟度等级都为软件过程的继续改进提供了一个台阶。