第 2 章　软件工程基础知识

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

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

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

著名的软件工程专家 Boehm 于 1983 年提出了软件工程的 7 条基本原理：

（1）用分阶段的生命周期计划严格管理；

（2）坚持进行阶段评审；

（3）实行严格的产品控制；

（4）采用现代程序设计技术；

（5）结果应能清楚地审查；

（6）开发小组的人员应该少而精；

（7）承认不断改进软件工程实践的必要性。

软件工程方法学包含 3 个要素：方法、工具和过程。方法是指完成软件开发的各项任务的技术方法；工具是指为运用方法而提供的软件工程支撑环境；过程是指为获得高质量的软件所需要完成的一系列任务的框架。

根据考试大纲，在软件工程基础知识方面，要求考生掌握以下知识点：

- 软件需求分析与定义；
- 软件设计、测试与维护；
- 软件复用；
- 软件质量保证及质量评价；
- 软件配置管理；
- 软件开发环境；
- 软件过程管理。

本章主要介绍软件需求分析与定义，软件设计、测试与维护，软件质量保证及质量评价，软件配置管理，软件开发环境和软件过程管理方面的知识，有关软件复用的知识将在第 3 章介绍。

2.1　软件需求分析与定义

根据 Standish Group 对 23000 个项目进行的研究结果表明，28%的项目彻底失败，46%的项目超出经费预算或者超出工期，只有约 26%的项目获得成功。而在这些高达 74%的不成功项目中，有约 60%的失败是源于需求问题，也就是差不多有一半的项目都遇到了这个问题，这一可怕的现象不得不让我们对需求分析引起高度的重视。

2.1.1　软件需求与需求过程

1. 什么是软件需求

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

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

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

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

另外，在大量与需求相关的书籍、文章中有一些诸如业务需求、用户需求之类的词语，把很多读者搞得术语混淆，下面我们一起来看看这些概念。

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

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

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

也就是说，这分别对应于需求的 3 个不同的层次，从目标到具体，从整体到局部，从概念到细节。这些不同层次、不同类型的需求描述之间的关系如图 2-1 所示。

2. 需求工程

需求工程是一个包括创建和维护系统需求文档所必需的一切活动的过程，通常包括需求开发和需求管理两大工作。

（1）**需求开发：**包括需求捕获、需求分析、编写规格说明书和需求验证四个阶段。在这个阶段需要确定产品所期望的用户类型、获取每种用户类型的需求、了解实际用户任务和目标，以及这些任务所支持的业务需求、分析源于用户的信息、对需求进行优先级分类、将所收集的需求编写成为软件规格说明书和需求分析模型、对需求进行评审等工作。

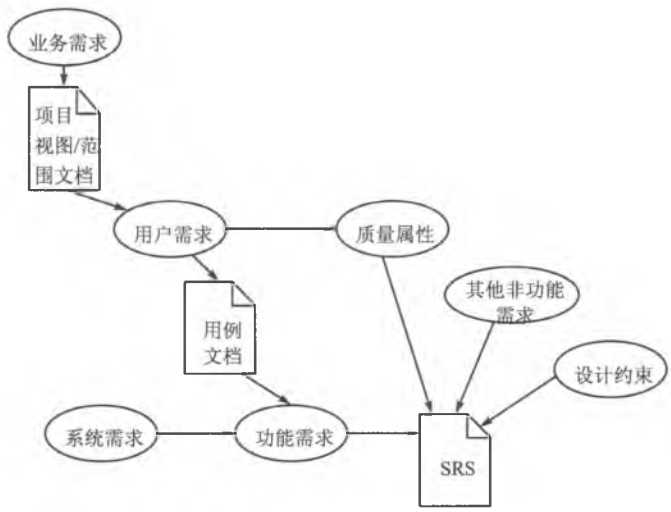

图 2-1　需求的组成结构示意图

（2）**需求管理：**通常包括定义需求基线、处理需求变更、需求跟踪等方面的工作。

而对于需求工程而言，最重要的还是需求开发，而需求开发总结起来就是包括需求捕获、需求分析、需求规格化、需求验证4个环节。需求捕获是为了收集需求信息，需求分析则是在需求捕获的基础上进行分析、建立模型，然后将其进行规格化形成《软件需求规格说明书》，最后再通过客户和管理层进行验证。

需求规格化的工作就是编制《软件需求规格说明书》，具体的方法和注意事项请参考有关文档编制的相关章节。而需求验证的工作则包括组织一个由不同代表组成的小组，对需求规格说明书和相关模型进行审查；以需求为依据编写测试用例，为确认测试做好准备；在需求的基础上，起草第一份用户手册；确定合格标准，也就是让用户描述什么样的产品才算是满足他们的要求和适合他们使用的。

2.1.2　需求调查与问题定义

需求调查与问题定义是看上去简单，做起来难的一件事。在很多人的印象中，需求调查，就是找用户聊聊说说，记个笔记。其实需求调查是否科学，准备是否充分，对调查出来的结果影响很大，这是因为大部分客户无法完整地讲述需求，而且也不可能看到系统的全貌。要想做好需求调查，必须清楚地了解 3 个问题。

- What：应该搜集什么信息。
- Where：从什么地方搜集这些信息。
- How：用什么机制或者技术来搜集这些信息。

接下来，我们就对这三个部分的内容进行一些更加详细的说明与描述。

1．要捕获的信息

一方面，需求分析员应该知道，从宏观的角度来看，要捕获的信息包括三大类：一是与问题域相关的信息（如业务资料、组织结构图、业务处理流程等）；二是与要求解决的问题相关的信息；三是用户对系统的特别期望与施加的任何约束信息。这样才能够有的放矢，不会顾此失彼。

另外一方面，需求分析员在开展具体需求捕获工作时，应该做到在此之前明确自己需要获得什么信息，这样才有可能获得所需信息，才知道工作进展是否顺利，是否完成了目标。

2．信息的来源

除了要明确地知道我们需要什么方面的信息，还要知道它们可以从哪里获得。通常情况下，这些需要的信息会藏于客户、原有系统、原有系统用户、新系统的潜在用户、原有产品、竞争对手的产品、领域专家、技术法规与标准里。

面对这么多种可能，在具体的实践中该从何下手呢？其实也很简单，首先从人的角度来说，应该首先对涉众（也就是风险承担人、项目干系人）进行分类，然后从每一类涉众中找到 1～2 名代表；而对于文档、产品而言，则更容易有选择地查阅。

结合前面讲述的内容，在制订需求捕获计划的时候，不妨列出一个表格，左边写上想了解的信息，右边跟上认为可能的来源，这样就能够建立一一对应的关系，使得需求捕获工作更加有的放矢，也更加高效。

3．需求捕获技术

当我们知道需要去寻找什么信息，并且也找到了信息的来源地，接下来就需要选择合适的技术进行需求捕获了。在此，我们列举出一些最常用的需求捕获技术。

（1）用户访谈。

用户访谈是最基本的一种需求捕获手段，也是最基本的一种手段。其形式包括结构化和非结构化两种，结构化是指事先准备好一系列问题，有针对地进行；而非结构化则

是只列出一个粗略的想法，根据访谈的具体情况发挥。最有效的访谈是结合这两种方法进行，毕竟不可能把什么都一一计划清楚，应该保持良好的灵活性。

准备问题：进行用户访谈之前，最好先对要询问的问题进行一些准备。准备的方法是围绕着想要获取的信息展开，设计一系列的问题，按顺序组织起来。而且还要预先准备好记录方式，主要包括本人记录、第三人记录或者是录音/录像的形式，不过采用录音/录像的方式应该征得被访谈者的同意，而且这种方法虽然看上去比较有效，不容易丢失信息，但这也会给后面的整理工作带来一定的工作量和难度。

访谈时的技巧：在访谈时一定要注意措辞得当，在充分尊重被访谈者的基础上，尽量避免出现“我不知你在说什么”，“我是来帮助你更好地工作”这样的言语，否则将会破坏访谈的气氛，从而使访谈的效率大打折扣。在访谈时一定要注意保持轻松的气氛，选择客户有充裕时间的时段进行，在说话、问问题时应该尽量采用易于理解、通俗化的语言。另外，值得注意的是，分析人员应该在进行访谈之前进行一些相关领域的知识培训，充分阅读相关材料，以保证自己有较专业的理解与认识，让被访谈者能够信任你。

应该询问的问题：在设计询问的问题时，应该考虑：自己的问题是否相关？回答是否正式？对方是回答这些问题的合适人选吗？是否问了过多的问题？是否还有更多的问题要问被访者？另外，还可以在询问过程中询问被访者还希望自己问他什么问题，还应该见哪些人？

总的来说，用户访谈具有良好的灵活性，有较宽广的应用范围，但是也存在着许多困难，诸如客户经常较忙，难以安排到时间；面谈时信息量大，记录较为困难；沟通需要很多技巧，同时需要分析员有足够的领域知识；另外，在访谈时会遇到一些对于组织来说比较机密和敏感的话题。因此，这看似简单的技术，也需要分析人员拥有足够多的经验和较强的沟通能力。

（2）用户调查。

正如前面讲到的，用户访谈时最大的难处在于很多关键的人员时间有限，不容易安排过多的时间；而且客户面经常较广，不可能一一访谈。因此，我们就需要借助“用户调查”这一方法，通过精心设计要问的问题，然后下发到相关的人员手里，让他们填写答案。这样可以有效地克服前面提到的两个问题。

但是与用户访谈相比，用户调查最大的不足就是缺乏灵活性；而且双方未见面，分析人员无法从他们的表情等其他动作来获取一些更隐性的信息；还有就是客户有可能在心理上会不重视一张小小的表格，不认真对待从而使得反馈的信息不全面。基于上述原因，因此较好的做法是将这两种技术结合使用。具体地讲，就是先设计问题，制作成用户调查表，下发填写完后，进行仔细的分组、整理、分析，以获得基础信息，然后再针对这个结果进行小范围的用户访谈，作为补充。

（3）现场观摩。

俗话说得好，百闻不如一见，对于许多较为复杂的流程和操作而言，是比较难以用言语表达清楚的，而且这样做也会显得很低效。针对这一现象，分析团队可以就一些较

复杂、较难理解的流程、操作采用现场观摩的方法来获取需求。

具体来说，就是走到客户的工作现场，一边观察，一边听客户的讲解，甚至可以安排人员跟随客户工作一小段时间。这样就可以使得分析人员更加直观地理解需求。

（4）文档考古。

对于一些数据流程比较复杂的，工作表单较多的项目，有时是难以通过说，或者通过观察来了解需求细节的。这个时候就可以借助于“文档考古”的方法，也就是对历史存在的一些文档进行研究，考古一词正是形象地说明了其主要的工作重心是结合已经填写完毕的、也就是带有数据的文件、表单、报告，从中获得所需的信息。

不过当你准备采用该方法时，也要记住这个方法的主要风险，那就是历史的文档可能与新系统的流程、数据有一些不吻合的地方，并且还可能承载一些原有系统的缺陷。要想有效地避免和发现这些问题，需要分析人员能够运用自己的聪明才智，将其与其他需求捕获技术结合对照。还有一个负面因素就是，这些历史的文档中记载的信息有可能涉及客户的商业秘密，因此对数据信息的保密也是分析人员基本的职业道德。

（5）联合讨论会。

这是一种相对来说成本较高的需求获取方法，但也是十分有效的一种。它通过联合各个关键客户代表、分析人员、开发团队代表一起，通过有组织的会议来讨论需求。通常该会议的参与人数为 6～18 人，召开时间为 1～5 小时。

在会议之前，应该将与讨论主题相关的材料提前分发给所有要参加会议的人。在会议开始之后，首先应该花一些时间让所有的与会者互相认识，以使交流在更加轻松的气氛下进行。会议的最初，就是针对所列举的问题进行逐项专题讨论，然后对原有系统、类似系统的不足进行开放性交流，第三步则是大家在此基础上对新的解决方案进行一番设想，在过程中将这些想法、问题、不足之处记录下来，形成一个要点清单。第四步就是针对这个要点清单进行整理，明确优先级，并进行评审。

这种联合讨论会将会起到群策群力的效果，对于一些最有歧义的问题和对需求最不清晰的领域都是十分有用的一种技术。而最大的难度就是会议的组织，要做到言之有物，气氛开放，否则将难以达到预想的效果。

4．需求捕获的策略

在整个需求过程中，需求捕获、需求分析、需求规格化、需求验证 4 个阶段不是瀑布式的发展，而是应该采用迭代式的演化过程。也就是说，在进行需求捕获时，不要期望一次就将需求收集完，而是应该捕获到一些信息后，进行相应的需求分析，并针对分析中发现的疑问和不足，带着问题再进行有针对性的需求捕获工作。

2.1.3　可行性研究

进行可行性研究，其主要的目的是回答一个问题，即所提出的项目是否可以完成。需要注意的是，可行性研究毕竟不是解决问题，而是研究问题的范围，探索这个问题是

不是值得去解决，根据现有的情况是否有能力，是否有可能找到较好的、成本效益合算的解决方案。

现在很多的开发团队把软件的最终开发结果是否符合需求规格说明书作为项目的成功标准，其实这是很片面的。真正成功的项目是满足客户的目标，为客户带来了预想的价值增长。而什么样的解决方案能够真正满足客户的需要，实现客户的目标呢？这就需要大量的需求分析与可行性研究工作。

1．可行性研究工作的基础

在可行性研究工作开始之前，系统分析员应该协助客户一起完成“问题定义”工作，也就是先明确系统要做什么？即 What 问题，也就是上一小节所解决的内容。

问题定义的关键是清晰地界定出问题的内容、性质，以及系统的目标、规模等内容，并形成完整的书面报告。系统分析员在这个过程中将使用各种需求调查技术研究问题、引导问题，从而形成完整的问题定义。

2．可行性研究工作的任务

可行性研究工作的目标大家已经了解，那么具体要完成什么工作，达到什么样的目的，也就是说从何着手呢？其实可行性研究工作的任务包括以下 3 个方面。

（1）**技术可行性：**现有的技术是否能够有效地解决该问题？是否有多种不同的解决方案？现有的技术力量是否达到？

（2）**经济可行性：**所有可能的解决方案所需投入的成本能否超过它的开发成本？它的成本效益分析结果如何？投资回报率如何？

（3）**社会可行性：**该解决方案是否符合企业实际情况？是否符合员工利益？是否符合相关法规和行业规范？

完成可行性研究工作之后，必须将这些成果文档化地记录下来，形成《可行性研究报告》。

3．可行性研究工作的步骤

那么具体而言，可行性研究工作包括哪些事项呢？该如何着手？都需要哪些人参与呢？下面我们就可行性研究工作的步骤做一个总结性阐述。

（1）核实问题定义与目标。

开始正式进行可行性研究工作之前，首先要做的一个工作，就是对该项工作的基础——问题定义再次核实。具体来说，就是仔细阅读问题定义的相关材料，对该问题所涉及的领域知识进行学习、考证，然后通过走访相关人员进行验证与核实。

这一步骤的关键目标是：使问题定义更加清晰、明确、没有歧义性，并且对系统的目标、规模，以及相关约束与限制条件做出更加细致的定义，确保可行性研究小组的所有成员达成共识。

（2）研究分析现有系统。

对现有系统的仔细分析与研究是十分重要的一项工作，因为它是新系统开发的最好参照物，对其的充分分析有助于新系统的开发。这么说的原因，主要是基于以下几点考虑：

- 现有系统已实现的功能通常也是新系统要实现功能的一部分。
- 新系统一定是在现有系统基础上的升华，毕竟如果旧系统没有问题，就不会有新系统开发的需求。
- 另外，现有系统的运行成本分析的结果是衡量新系统的经济可行性的重要参照物。

从字面上的理解会容易产生一个常见的误区，就是认为现有系统一定是软件系统，其实这里的“现有系统”不仅包括旧的软件系统，还包括旧的非计算机系统。

（3）为新系统建模。

在问题定义和对现有系统研究的基础上，开始对新的系统进行建模，建模的目的是为了获得一个对新系统的框架认识、概念性认识。通常可以采用以下几种技术。

- 系统上下文关系范围图：其实也就是 DFD（数据流图）的 0 层图，将系统与外界实体（可能是人、可能是外部系统）的关系（主要是数据流和控制流）体现出来，从而清晰地界定出系统的范围，实现共识。
- 实体-关系图（E-R）：这是系统的数据模型，这个阶段并不需要生成完整的 E-R 图，而是找到主要的实体，以及实体以间的关系即可。
- 用例模型：这是系统的一个动态模型，以 Actor 和 use-case 整理出系统的主要功能框架，这个阶段应该大部分都处于概念级，每个用例也无需花太多的时间确定细节，只要能够勾画出系统的雏形即可。
- 域模型：采用 OO 的思想，对于系统中主要的实体类找到，并说明实体类的主要特征和它们之间的关系。
- IPO 表：采用传统的结构化思想，从输入、处理、输出的角度进行描述系统。

再次请大家注意，这个阶段的所有模型是不够精确的，只是一个框架，要达到一个宏观的角度，否则将陷入无休止的工作中。

（4）客户复核。

可以这么说，在第（3）步中建立起来的系统模型是系统分析员眼里的问题定义，那么这与客户心目中的问题是否一致呢？因此，完整的系统模型建立之后，一个十分重要的工作就是与客户一起进行复核。当然，由于这个时候模型将成为讨论和分析的基础，因此使用客户更容易接受的模型将显得十分重要。

如果在这个过程中，发现模型与客户的目标有不一致的地方，就应该再次通过访谈、现场观摩、对现有系统分析等手段进行了解，然后在此基础上修改模型。由此也可以看书，（1）～（4）的步骤是一个循环，周而复始，直至客户确认了新的系统模型为止。

（5）提出并评价解决方案。

前面的工作还是停留在“系统解决什么问题”上，只是更加清晰地进行了定义和说明。当客户与系统分析员对要解决的问题有了一个清晰的共识之后，接下来的工作就是提出解决方案，这也是系统分析员很重要的工作之一。

在这个阶段，应该尽量列举出各种可行的解决方案，并且对这些解决方案的优点、缺点做一个综合性的评价，以便下一步决策。需要注意的是，对那些明显不可行的，如技术上还没有相应的办法、经济角度明显不可行的、违背企业或行业实际情况的解决方案应该直接过滤掉。

（6）确定最终推荐的解决方案。

明确地指出该项目是否可行，如果可行，什么方面是最合理的？由于对这两个问题的回答，是可行性分析研究工作的核心目标。因此在各种解决方案提出之后，接下来应该从中选中一个最合理、最可行的解决方案，更加详细地说明理由，并且还要对其进行更加完善的成本/效益分析。

具体来说，成本效益分析可以分成两个部分的内容。

① 成本估计。对于软件系统而言，主要的成本是人力资源成本，另外还包括一些直接的费用、设备采购的费用等。相对而言，硬件设备，以及其他的一些相关费用的评估会比较容易一些，最难的是人力资源的成本分析。因此，对系统工作量（用人月、人年等单位进行说明）进行合理、科学的评估，并在此基础上进行计算是很必要的。

要想准确地估算出工作量，通常可以借助的工具是历史数据和经验模型。历史数据就是指你所在的开发团队以往从事项目的情况，可以通过以类似系统项目做参照的方法。而经验模型则是一些软件工作量估算方面的研究总结，常用的有功能点分析、COCOMO 分析等。

通常，可以分成以下几步进行：

- 首先，进行工作任务分解，将目标细化。
- 然后，就每一个工作任务包，与具体的开发团队进行共同分析，使用功能点分析法或其他相关的分析法进行估算，并且将估算的结果与类似项目的历史数据进行比较，做出微调。
- 使用 COCOMO 分析，计算出相应的人月数。在这个过程中，还应该根据类似项目的历史数据推出项目组队的平均产能，从而使估算值更有代表性。

另外，有一点是十分重要的，在这个阶段是不可能得到精确的估算值的，这个观念必须让开发团队、管理层和客户很清晰地认识到。否则，即使违心地给出精确的估算值也显得没有任何意义。

这个阶段的工作，对于系统分析员而言，很重要的知识技能在于软件度量与估算方法、技术的掌握方面。

② 效益分析。有了估算出来的开发成本以后，就可以进行效益分析了。在做效益分析之前应该首先对该系统应用之后，将会带来的直接、间接收益，以及成本降低的具

体数额进行量化。并且通过经济学的相关模型来进行分析，这要求系统分析人员能够在该方面有一定的知识积累。通常进行效益分析时要借助以下几个概念。

- 货币的时间价值：比如说现在的 1 元钱，与未来的 1 元钱，其代表的价值是不同的。通常是使用利率的形式来表现这个价值。我们用 F 代表未来的价值，P 代表现在的价值，那么可以总结为：$F=P(1+i)^n$ 和 $P=F/(1+i)^n$。其中 i 代表年利率，n 代表年数。
- 投资回收期：投资回收期的意思就是要多少年才能够将投资回收，越短越有利。
- 纯收入：衡量系统价值的一个很重要的指标就是纯收入，而纯收入指的是整个生命周期之内系统的累计经济效益（折成现值）与投资之差。
- 投资回报率：当投资额、每年预计可获得的经济效益这两个数据有了的时候，就可以计算投资回报率（ROI），公式如下所示：

$$P=F_1/(1+j)+F_2/(1+j)^2+\cdots+F_n/(1+j)^n$$

其中，P 代表总投资额，F_i 是第 i 年年底的收益，n 是系统使用寿命，j 就是投资回报率。

（7）草拟开发计划。

在上一步，我们对主要推荐的解决方案进行了详细的成本效益分析，对工作内容和工作量都有了一个详细的了解，接下来需要制订一个最粗略的开发计划，说明开发所需的资源、人员和时间进度安排。这也将作为可行性分析的一个重要依据和立项开发后制订项目计划的基础。

（8）以书面的形式提交《可行性分析报告》并进行审查。

最后就是将这些研究的结果整理成文，提交客户和管理层，进行审查通过。在这个阶段还应该制作一些相应的讲义，为客户和管理层做介绍和说明。

2.1.4　需求分析

在细化地说明需求分析之前，我们先温习一下分析的定义：所谓分析就是通过对问题域的研究，获得对该领域特性及存在于其中（需要解决）的问题特性的透彻理解并用文档说明。

从上面的定义中，我们可以知道需求分析的关键在于对问题域的研究与理解。为了便于理解问题域，现代软件工程方法所推荐的做法是对问题域进行抽象，将其分解为若干的基本元素，然后对元素之间的关系进行建模。

1．需求分析的工作任务

前面对分析的定义相对比较抽象，不太易于理解，不太容易用来指导具体的操作。其实用更通俗的话来说，需求分析就是提炼、分析和仔细审查已经收集到的需求，以确保所有的涉众都明白其含义并找出其中的错误、遗漏或其他不足的地方。

在 Karl E.Wiegers 的经典名作《软件需求》一书中指出，需求分析的工作通常包括以下七个方面。

（1）**绘制系统上下文范围关系图：**这种关系图是用于定义系统与系统外部实体间的界限和接口的简单模型，它可以为需求确定一个范围。其实就是 DFD 的 0 层图，图 2-2 就是一个实例。

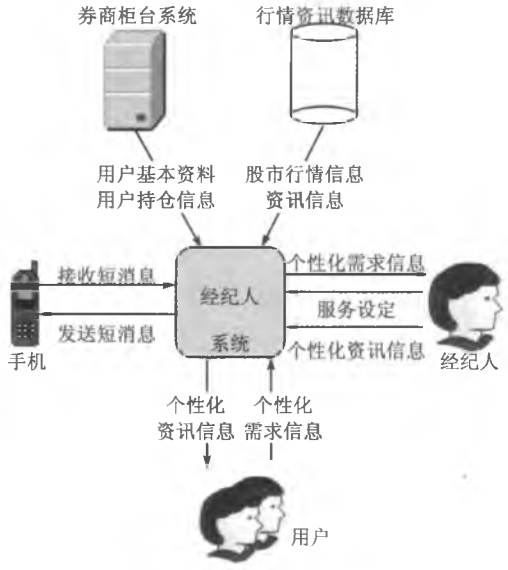

图 2-2　系统上下文范围关系图

（2）**创建用户接口原型：**由于用户界面对于一个系统来说是十分重要的，因此在需求分析阶段通过快速开发工具开发一个可抛弃原型，或者通过 PowerPoint、Authorware 等演示工具制作一个演示原型，甚至可以用纸笔画出一些关键的界面接口示意图，这些都会帮助客户更好地理解所要解决的问题，更好地理解需求。

（3）**分析需求的可行性：**对所有获得的需求进行成本、性能、技术实现方面的可行性研究，以及这些需求项是否与其他的需求项有冲突，是否有对外的依赖关系等。

（4）**确定需求的优先级：**这是一个很重要的工作，迭代开发已经成为现代软件工程方法论的一个基础，而需求的优先级是制订迭代计划的一个最重要的依据。对于需求优先级的描述，可以采用满意度/非满意度指标进行说明（满意度：取值 1～5，表示当需求被实现时用户的满意程度。不满意度：取值 1～5，表示当需求未被实现时用户的不满意程度。）

（5）**为需求建立模型：**也就是建立分析模型，这些模型的表现形式主要是图表加上少量的文字描述，正所谓一图抵千字，图形化地描述需求使其更加清晰、易懂。根据采用的分析技术的不同，采用的图也不同，例如，OO 技术下的用例模型和域模型；面向数据分析技术下的 E-R 图；结构化分析技术下的数据流图等。

（6）**创建数据字典**：数据字典是对系统用到的所有数据项和结构进行定义，以确保开发人员使用统一的数据定义。

（7）**使用质量功能调配（QFD）**：这是在需求优先级基础上的一个升华，其原理与满意度/非满意度指标十分接近。它通过将产品特性、属性与对客户的重要性联系起来，QFD 将需求分为三类：期望需求，即如果缺少会让其感到不满意的需求；普通需求；兴奋需求：实现了客户会感到惊喜，但没有也不会遭到责备。

2．需求建模

根据在需求方面的权威 Alan Davis 的见解，仅仅单一地看需求并不能提供对需求的完全理解，而是需要把用文字表示的需求和用图形表示的需求结合起来。而用图形表示需求，就是需求建模，获得分析模型。分析模型有助于检测需求的不一致性、模糊性、错误及遗漏。有一点需要提醒读者注意的是，迄今为止，尚未发现一种分析建模技术能够将所有的内容涵盖，我们需要根据实际情况进行有效的组合。

进行需求建模时，首先应该建立一个概念，那就是分析模型的基础是分析元素，而分析元素则来源于客户所陈述的需求。而且，我们应该尽可能地利用 CASE 工具创建、维护、发布需求模型，以保持其可控性。

2.1.5　流行的需求分析方法论

需求分析的方法可谓种类繁多，不过如果按照分解的方式不同，可以很容易地划分出几种大类型。我们先从分析方法发展的历史，对其建立一个概要性的认识。

结构化分析方法（Structured Analysis，SA）：最初的分析方法都不成体系，而且通常都只包括一些笼统的告诫，在 20 世纪 70 年代分析技术发展的分水岭终于出现了。这时人们开始尝试使用标准化的方法，开发和推出各种名为“结构化分析”的方法论，Tom DeMacro 是这个领域最有代表性和权威性的专家。

软系统方法：这是一个过渡性的方法论，并未真正流行过，它的出现只是证明了结构化分析方法的一些不足。因为结构化分析方法采用的相对形式化的模型不仅与社会观格格不入，而且在解决“不确定性”时显得十分无力。最有代表性的软系统方法是 Checkland 方法。

面向对象分析方法（Object Oriented Analysis，OOA）：在 20 世纪 90 年代，结构化方法的不足在面对多变的商业世界时，显得更加苍白无力，这促使了 OOA 的迅速发展。

面向问题域的分析（Problem Domain Oriented Analysis，PDOA）：随着技术的应用和发展发现面向对象分析方法也存在着很多的不足，应运而生了一些新的方法论，PDOA 就是其中一种。不过 PDOA 尚在研究阶段，并未广泛应用。

1. 结构化分析

结构化分析与面向对象分析方法之间的最大差别是：结构化分析方法把系统看做一个过程的集合体，包括人完成的和电脑完成的；而面向对象方法则把系统看成一个相互影响的对象集。结构化分析方法的特点是利用数据流图来帮助人们理解问题，对问题进行分析。

结构化分析一般包括以下工具，在本节的随后部分将对它们一一做简单介绍。

- 数据流图（Data Flow Diagram，DFD）；
- 数据字典（Data Dictionary，DD）；
- 结构化语言；
- 判定表；
- 判定树。

结构化系统分析方法从总体上看是一种强烈依赖数据流图的自顶向下的建模方法。它不仅是需求分析技术，也是完成需求规格化的有效技术手段。

在介绍具体的结构化分析方法之前，我们先对如何进行结构化分析做一个总结性描述，以帮助大家更好地应用该方法。

（1）**研究“物质环境”。**首先，应画出当前系统（可能是非计算机系统，或是半计算机系统）的数据流图，说明系统的输入、输出数据流，说明系统的数据流情况，以及经历了哪些处理过程。在这个数据流图中，可以包括一些非计算机系统中数据流及处理的命名，例如，部门名、岗位名、报表名等。这个过程可以帮助分析员有效地理解业务环境，在与用户的充分沟通与交流中完成。

（2）**建立系统逻辑模型。**当物理模型建立完成之后，接下来的工作是画出相对真实系统的等价逻辑数据流图。在前一步骤建立的数据流图的基础上，将所有自然数据流都转成等价的逻辑流：例如将现实世界的报表改成存储在计算机系统中的文件里；又如将现实世界中“送往总经理办公室”改为“报送报表”。

（3）**划清人机界限。**最后，我们确定在系统逻辑模型中，哪些将采用自动化完成，哪些仍然保留手工操作。这样就可以清晰地划清系统的范围。

2. 数据流图

数据流图是一种图形化的系统模型，它在一张图中展示信息系统的主要需求，即输入、输出、处理（过程）、数据存储。由于从DFD中可以很容易地一眼看出系统紧密结合的各个部分，而且整个图形模式只有五个符号需要记忆，所以深受分析人员的喜爱，因而广为流行。

正如图2-3所示，DFD中包括以下几个基本元素。

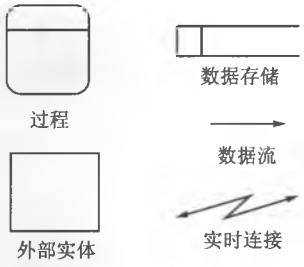

图 2-3　数据流图的符号集

- 过程：一步步地执行指令，完成输入到输出的转换。
- 外部实体：系统之外的数据源或目的。
- 数据存储：存放数据的地方，一般是文件、数据库等形式。
- 数据流：从一处到另一处的数据流向，如从输入或输出到一个过程的数据流。
- 实时连接：当过程执行时，外部实体与过程之间的来回通信。

（1）数据流图的层次。正如前面提到的，结构化分析的思路是依赖于数据流图进行自顶而下的分析。这也是因为系统通常比较复杂，很难在一张图上将所有的数据流和加工描述清楚。因此，数据流图提供一种表现系统高层和低层概念的机制。也就是先绘制一张较高层次的数据流图，然后在此基础上，对其中的过程（处理）进行分解，分解成若干独立的、低层次的、详细的数据流图，而且可以这样逐一地分解下去，直至系统被清晰地描述出来。

（2）Context 图。Context 图，也就是在 2.1.4 节中提到的系统上下文范围关系图。这是描述系统最高层结构的 DFD 图。它的特点是，将整个待开发的系统表示为一个过程，将所有的外部实体和进出系统的数据流都画在一张图中。

图 2-3 就是一个 Context 图的实例，只不过在绘制时做了一些处理，使得它看上去更加直观易懂，图 2-4 也是一个 Context 图的例子。

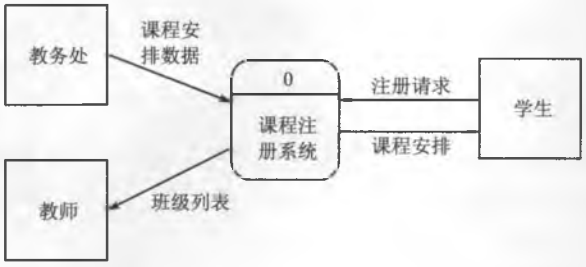

图 2-4　Context 图实例

Context 图用来描述系统有什么输入、输出数据流，与哪些外部实体直接相关，可以把整个系统的范围勾画出来。

（3）逐级分解。当完成了 Context 图的建模后，就可以在此基础上进行进一步分解。下面我们以图 2-4 为例，进行再分解，在对原有流程了解的基础上，可以得到图 2-5。

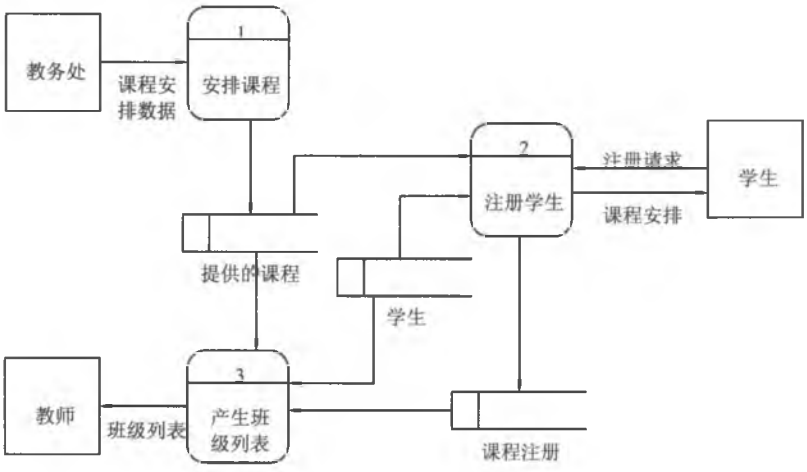

图 2-5　DFD 0 层图

图 2-5 是在 Context 的基础上做的第一次分解，而在 Context 只有一个过程，那就是系统，我们将其编号为 0。接下来对 Context 图进行分解，其实就是对这个编号为 0 的过程进行更细化的描述，在这里引入了新的过程、数据存储，为了能够区分其位于的级别，在这个层次上的过程将以 1、2、3 为序列进行编号。

正是由于这是对过程 0 的分解，因此也称之为 DFD 0 层图。而我们可以根据需要对 DFD 0 层图上的过程（编号为 1、2、3）进行如法炮制的分解，称之为 DFD 1 层图，DFD 1 层图中引入的新过程，其编号规则就是 1.1，1.2…，以及 2.1，2.2…，依此类推，直到完成分析工作。

另外，这里存在一个很关键的要点，由于 DFD 0 层图是 Context 的细化，因此所有的输入和输出应该与 Context 完全一致，否则就说明存在着错误。

（4）如何画 DFD。DFD 的绘制是一个自顶向下、由外到里的过程，通常按照以下几个步骤进行。

- 画系统的输入和输出：在图的边缘标出系统的输入、输出数据流。这一步其实是决定研究的内容和系统的范围。在画的时候，可以先将尽可能多的输入、输出画出来，然后再删除多余的，增加遗漏的。
- 画数据流图的内部：将系统的输入、输出用一系列的处理连接起来，可以从输入数据流画向输出数据流，也可以从中间画出去。
- 为每一个数据流命名：命名的好坏与数据流图的可理解性密切相关，应避免使用空洞的名字。
- 为加工命名：注意应用动宾短语。

不考虑初始化和终点，暂不考虑出错路径等细节，不画控制流和控制信息。

3. 细化记录 DFD 部件

为了能够更好地描述 DFD 的部件，结构化分析方法还引入了数据字典、结构化语言，以及决策树、决策表等方法。通过使用这些工具，能够对数据流图中描述不够清晰的地方进行有效的补充。

（1）结构化语言。结构化语言是结构化编程语言与自然语言的有机结合，可以采用顺序结构、分支结构、循环结构等机制，同时还说明加工的处理流程。该技术通常用来描述一些重要的、复杂的过程的程序逻辑逻辑。表 2-1 所示是一个使用结构化语言描述的例子。

表 2-1　使用结构化语言描述的例子

```
IF 分数>=60 Then
IF 分数<80 Then
   成绩=C
     ELSE
         IF 分数<90 Then
            成绩=B
         Else
            成绩=A
         EndIf
     EndIf
  ELSE
IF 分数>=50 Then
   成绩=D
    ELSE
       成绩=E
    EndIf
 EndIf
```

（2）决策表和决策树。决策表是一种处理逻辑的表格表示方法，其中包括决策变量、决策变量值、参与者或公式。与上例对应的决策表示例如表 2-2 所示。

表 2-2　决策表示例

分数>=60	是			否	
分数	>=80		<80	>=50	<50
分数	>=90	<90			
成绩	A	B	C	D	E

而决策树则使用像树枝一样的线条对过程逻辑进行图表化的描述。与上例相应的决策表如图2-6所示。

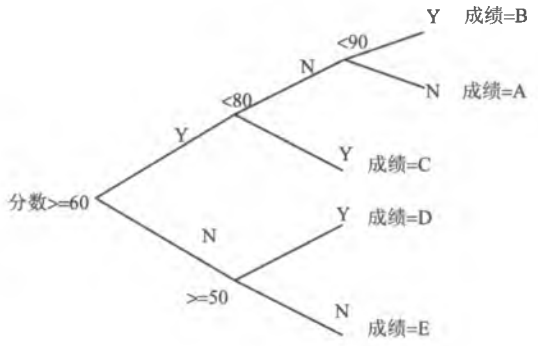

图2-6　决策树示例

很显然，应用这两种手段来描述复杂决策逻辑，要远远优于使用结构化语言。而这两种技术也各有优劣，决策表更严密，而决策树更易读。分析人员可以根据自己的实际需要来灵活选择应用。

（3）数据字典。数据字典技术是一种很实用、有效的表达数据格式的手段。它是对所有与系统相关的数据元素的一个有组织的列表和精确的、严格的定义，使用户和系统分析员对输入、输出、存储成分和中间计算机有共同的理解。通常数据字典的每一条目中包括以下信息。

- 名称：数据或控制项、数据存储或外部实体的主要名称，如果有别名的还应该将别名列出来。
- 何处使用/如何使用：使用数据或控制项的加工列表，以及如何使用。
- 内容描述：说明该条目内容组成，通常采用以下符号进行说明。
  - =：由…构成。
  - +：和，代表顺序连接的关系。
  - [|]：或，代表从中选择一个。
  - {}*：n次重复。
  - ()：代表可选的数据项。
  - *…*：表示特定限制的注释。
- 补充信息：关于数据类型、默认值、限制等信息。

表2-3是一个数据字典的实例。

表2-3　数据字典的实例

客户基本信息=客户编号+客户名称+身份证号码+手机+小灵通+家庭电话
客户编号={0…9}^{8}

客户名称={字}^{4}

身份证号码=[{0…9}^{15}|{0…9}^{18}]

手机=[{0…9}^{11}|{0…9}^{12}]

小灵通=（区号）+本地号

家庭电话=（区号）+本地号

办公电话=（区号）+本地号

区号={0…9}^{4}

本地号=[{0…9}^{7}|{0…9}^{8}]

4．实体-关系图

传统的系统开发方法都把重点集中在新系统的数据存储需求上，包括数据实体、数据实体的属性，以及它们之间的关系。而描述这些东西的最好形式就是借助实体-关系图（Entity Relationship Diagram，E-R 图）。

（1）实体。由于所有的系统都包括数据，而且是大量的数据。因此，我们在开发系统时，需要一个概念来抽象地表示一组相类似的事物的所有实例，我们称这个概念为实体，在 E-R 图中使用一个圆角的矩形来表示，如图 2-7（a）所示。从这个概念上看，与面向对象分析方法中的类有些相似。不过，由于在这里只关心数据，因此实体通常是需要存储的数据。与类一样，实体也有实例，表示实体的一个具体值，称为实体实例。

由于实体是用来存储数据的，因此需要描述它具体存储什么数据，这些具体的数据叫做属性，用来描述实体的性质或特征，可以直接在圆角矩形中填入这些属性值，如图 2-7（b）所示。另外还应该为属性定义合法的值，这个定义包括数据类型、域，以及默认值，这些描述可以直接跟在属性值的后面，如图 2-7（c）所示。关于数据类型、域的说明如表 2-4 所示。

表 2-4　数据类型与域说明

数据类型	类型说明	域
NUMBER	任何数、实数或整数	对于整数，指定范围 {最大～最小} 对于实数，指定范围和精度
TEXT	一个字符串	TEXT（属性的最大长度）
MEMO	不确定大小的 TEXT，常用于不定长的信息	无
DATA	各种格式的日期	MMDDYYYY 或者 MMYYYY 等
TIME	各种格式的时间	HHMMT 或者 HHMM 等
YES/NO	布尔型变量，是或者否	{YES，NO}　{ON，OFF}
VALUE SET	一个有值集合	{值 1，值 2，…，值 n} {代表代码及含义的表}
IMAGE	任何图形或图像	无

另外，为了能够区别每一个实体实例，经常需要设置一个标志符或键，而键就是其中的一个或一组属性，它们对每个实体实例具有唯一的值，通常在 E-R 图中的属性值

后面加上 Primary Key 或 Alternate Key 等描述，如图 2-7（d）所示。

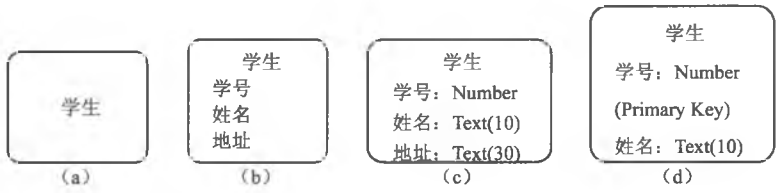

图 2-7　实体表示图例

（2）关系。实体和属性都不是孤立存在的，它们各自代表的事物互相交互，并且互相影响，共同支持业务任务。关系是存在于一个或多个实体之间的自然业务联系。关系可以链接实体的一个事件，也可以是纯粹的逻辑关系。

通常情况下，我们还需要对关系的多重性进行说明，这也就是基数。基数定义了一个实体相对于另一个关联实体的某个具体值的最小和最大具体值数量。因为所有的关系都是双向的，因此在两个方向上都需要定义。基数定义如图 2-8 所示。

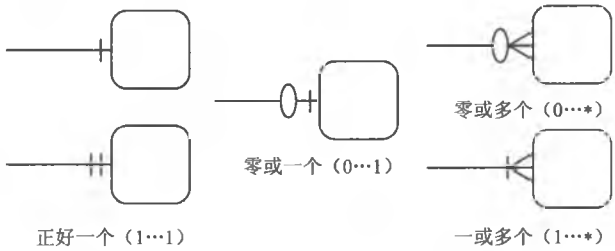

图 2-8　关系的基数表示法

结构化分析方法为开发者和客户提供一个直观易懂的模型，能够对实现理解问题域这一基本的分析目标以支持。但也存在着很多的先天不足：

- 对问题域的研究力度不够大；
- 分析与设计之间缺乏清晰的界限；
- 没有一个真正的功能规格说明；
- 需求实质上是根据满足该需求的某一特定系统的内部设计来加以说明的；
- 内部设计的开发使用的则是不可靠的内部设计技术-功能分解；
- 不适用于很多类型的应用。

可以这么说，结构化分析方法在很大程度上推动了分析技术的发展，但又被更合适的技术逐渐取代，不过，结构化分析方法中的具体工具仍然有很广泛的应用空间。

5．面向问题域的分析

相对而言，面向问题域的分析是一项很新的技术，还处于研究阶段，相关的文档资

料还不多。与 SA 和 OOA 相比，面向问题域的分析更多地强调描述，而较少强调建模。它的描述大致分为以下两个部分。

（1）关注问题域：用一个文档对含有问题域进行相关的描述，并列出需在该域中求解的问题列表，即需求列表。只有这个文档是在分析时产生的。

（2）关注解系统的待求行为：用一个文档对解系统（即系统实现）的待求行为进行描述。该文档将在需求规格说明时再完成。

在面向问题域的分析方法中，对整个过程有着一个清晰的定义：

- 搜集基本的信息并开发问题框架，以建立问题域的类型；
- 在问题框架类型的指导下，进一步搜集详细信息并给出一个问题域相关特性的描述。

基于以上两点，收集并用文档说明新系统的需求。

从上面的描述中，我们可以看出，问题框架是面向问题域分析的核心元素。问题框架是将问题域建模成一系列相互关联的子域，而一个子域可以是那些可能算是精选出来的问题域的一部分，也可以把问题框架视为开发 Context 图，但不同的是 Context 图的建模对象是针对解系统的，而问题框架则是针对问题域的。也就是说，问题框架的目标就是大量地捕获更多有关问题域的信息。

2.2　软件设计

从功能上的划分来看，软件设计应该是软件设计师的工作，但作为一名项目管理师，首先自己必须懂得软件设计的基本原则和理论，掌握基本的软件设计方法，具有一定的软件设计经验。

2.2.1　软件设计基本原则

1. 信息隐蔽

在一节不和谐的课堂里，老师叹气道：“要是坐在后排聊天的同学能像中间打牌的同学那么安静，就不会影响到前排睡觉的同学”。

这个故事告诉我们，如果不想让坏事传播开来，就应该把坏事隐藏起来，“家丑不可外扬”就是这个道理。为了尽量避免某个模块的行为去干扰同一系统中的其他模块，在设计模块时就要注意信息隐藏。应该让模块仅仅公开必须要让外界知道的内容，而隐藏其他一切内容。

在软件设计中同样有信息隐蔽原则。Parnas 提出：在概要设计时列出将来可能发生变化的因素，并在模块划分时将这些因素放到个别模块的内部。也就是说，每个模块的实现细节对于其他模块来说是隐蔽的，模块中所包含的信息（包括数据和过程）不允许其他不需要这些信息的模块使用。这样，在将来由于这些因素变化而需修改软件时，只

需修改这些个别的模块，其他模块不受影响。信息隐蔽技术不仅提高了软件的可维护性，而且也避免了错误的蔓延，改善了软件的可靠性。现在信息隐蔽原则已成为软件工程学中的一条重要原则。

2．模块独立性

软件设计中的模块独立性是指软件系统中每个模块只涉及软件要求的具体子功能，而和软件系统中其他的模块接口是简单的。模块独立的概念是模块化、抽象、信息隐蔽和局部化概念的直接结果。

一般采用两个准则度量模块独立性，即模块间耦合和模块内聚。

耦合是模块之间的相对独立性（互相联系的紧密程度）的度量。模块之间的联系越紧密，联系越多，耦合性就越高，而其模块独立性就越弱。

内聚是模块功能强度（一个模块内部各个元素彼此结合的紧密程度）的度量。一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高，相对地，它与其他模块之间的耦合性就会减低，而模块独立性就越强。由此可见，模块独立性比较强的模块应是高内聚低耦合的模块。

（1）内聚。内聚是信息隐蔽功能的自然扩展。内聚的模块在软件过程中完成单一的任务，同程序其他部分执行的过程交互很少，简而言之，内聚模块（理想情况下）应该只完成一件事。在设计模块时应尽量争取高内聚。

一般模块的内聚性分为七种，如图2-9所示。

一般认为，巧合（偶然）、逻辑和时间上的聚合是低聚合性的表现；信息的聚合则属于中等聚合性；顺序的和功能的聚合是高聚合性的表现。表2-5列出了各类聚合性与模块各种属性的关系。

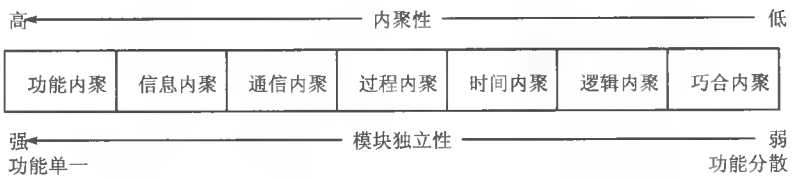

图2-9　模块的内聚性

表2-5　各类聚合性与模块各种属性的关系

	内部联系	清 晰 性	可重用性	可修改性	可理解性
偶然聚合性	很差	差	很差	很差	很差
逻辑聚合性	很差	很差	很差	很差	差
时间聚合性	差	中	很差	中	中
信息聚合性	中	好	差	中	中
顺序聚合性	好	好	中	好	好
功能聚合性	好	好	好	好	好

- 功能内聚（Functional Cohesion）。一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的，则称该模块为功能内聚模块。它是内聚程度最高的，也是模块独立性最强的模块。
- 信息内聚（Informational Cohesion）。这种模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该执行哪一个功能。由于这个模块的所有功能都是基于同一个数据结构（符号表），因此，它是一个信息内聚的模块。信息内聚模块可以看成是多个功能内聚模块的组合，并且达到信息的隐蔽。即把某个数据结构、资源或设备隐蔽在一个模块内，不为别的模块所知晓。
- 通信内聚（Communication Cohesion）。如果一个模块内各功能部分都使用了相同的输入数据，或产生了相同的输出数据，则称之为通信内聚模块。通常，通信内聚模块是通过数据流图来定义的。
- 过程内聚（Procedural Cohesion）。使用流程图作为工具设计程序时，把流程图中的某一部分划出组成模块，就得到过程内聚模块。例如，把流程图中的循环部分、判定部分、计算部分分成三个模块，这三个模块都是过程内聚模块。
- 时间内聚（Classical Cohesion）。时间内聚又称为经典内聚。这种模块大多为多功能模块，但模块的各个功能的执行与时间有关，通常要求所有功能必须在同一时间段内执行。例如，初始化模块和终止模块。
- 逻辑内聚（Logical Cohesion）。这种模块把几种相关的功能组合在一起，每次被调用时，由传送给模块的判定参数来确定该模块应执行哪一种功能。
- 巧合内聚（Coincidental Cohesion）。巧合内聚又称为偶然内聚。当模块内各部分之间没有联系，或者即使有联系，这种联系也很松散，则称这种模块为巧合内聚模块，它是内聚程度最低的模块。

（2）耦合。耦合是程序结构中模块相互关联的度量。耦合取决于各个模块间接口的复杂程度、调用模块的方式，以及哪些信息通过接口。

耦合的强度依赖于以下几个因素：

- 一个模块对另一个模块的调用；
- 一个模块向另一个模块传递的数据量；
- 一个模块施加到另一个模块的控制的多少；
- 模块之间接口的复杂程度。

一般模块之间可能的连接方式有 7 种，它们构成耦合性的 7 种类型，如图 2-10 所示。

耦合是影响软件复杂程度的一个重要因素，在软件设计过程中，应尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，完全不用内容耦合。表 2-6 列出了各类耦合性与模块各种属性的关系。

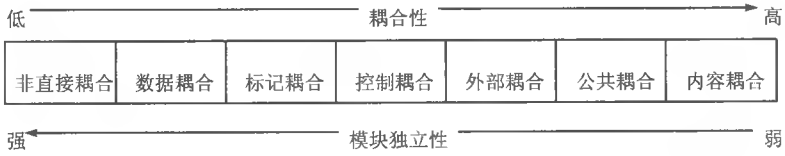

图 2-10　模块之间的耦合性

表 2-6　各类耦合性与模块各种属性的关系

	对修改的敏感性	可重用性	可修改性	可理解性
内容耦合	很强	很差	很差	很差
公共耦合	强	很差	中	很差
外部耦合	一般	很差	很差	中
控制耦合	一般	差	差	差
标记耦合	不一定	中	中	中
数据耦合	不一定	好	好	好

- 非直接耦合（Nondirective Coupling）。如果两个模块之间没有直接关系，它们之间的联系完全是通过主模块的控制和调用来实现的，这就是非直接耦合。这种耦合的模块独立性最强。
- 数据耦合（Data Coupling）。如果一个模块访问另一个模块时，彼此之间是通过简单数据参数（不是控制参数、公共数据结构或外部变量）来交换输入、输出信息的，则称这种耦合为数据耦合。
- 标记耦合（Stamp Coupling）。如果一组模块通过参数表传递记录信息，就是标记耦合。这个记录是某一数据结构的子结构，而不是简单变量。
- 控制耦合（Control Coupling）。如果一个模块通过传送开关、标志、名字等控制信息，明显地控制选择另一模块的功能，就是控制耦合。
- 外部耦合（External Coupling）。一组模块都访问同一全局简单变量而不是同一全局数据结构，而且不是通过参数表传递该全局变量的信息，则称为外部耦合。
- 公共耦合（Common Coupling）。若一组模块都访问同一个公共数据环境，则它们之间的耦合就称为公共耦合。公共的数据环境可以是全局数据结构、共享的通信区、内存的公共覆盖区等。公共耦合的复杂程度随耦合模块的个数增加而显著增加。若只是两模块间有公共数据环境，则公共耦合有两种情况。松散公共耦合和紧密公共耦合。
- 内容耦合（Content Coupling）。如果发生下列情形，两个模块之间就发生了内容耦合：一个模块直接访问另一个模块的内部数据；一个模块不通过正常入口转到另一模块内部；两个模块有一部分程序代码重叠（只可能出现在汇编语言中）；一个模块有多个入口。

2.2.2　结构化设计方法

结构化设计方法是基于模块化、自顶向下逐层细化、结构化程序设计等程序设计技术基础上发展起来的，该方法实施的过程如下。

（1）总结出系统应有的功能，对一个功能，从功能完成的过程考虑，将各个过程列出，标志出过程转向和传递的数据。这样，可以将所有的过程都画出来。

（2）细化数据流。确定应该记录的数据。

（3）分析各过程之间的耦合关系，合理地进行模块划分以提高它们之间的内聚性。实际上，对于这个练习，可以使模块具有信息内聚性。

1．系统结构图中的模块

在系统结构图中，不能再分解的底层模块称为原子模块。如果一个软件系统的全部实际加工都由原子模块来完成，而其他所有非原子模块仅仅执行控制或协调功能，这样的系统就是完全因子分解的系统。如果系统结构图是完全因子分解的，就是最好的系统。但实际上，这只是力图达到的目标，大多数系统做不到完全因子分解。

一般来说，结构图中可能出现如图 2-11 所示的 4 种类型的模块。

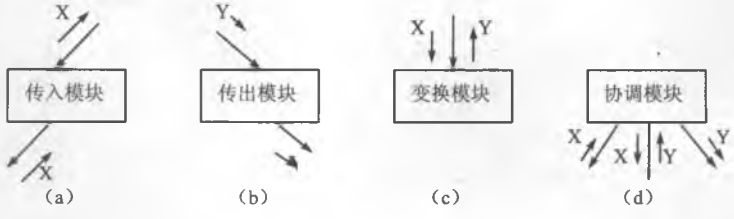

图 2-11　4 种模块类型

- 传入模块：图 2-11（a），从下属模块取得数据，经过某些处理，再将其传送给上级模块。它传送的数据流叫做逻辑输入数据流。
- 传出模块：图 2-11（b），从上级模块取得数据，进行某些处理，传送给下属模块。它传送的数据流叫做逻辑输出数据流。
- 变换模块：图 2-11（c），从上级模块取来数据，进行特定处理后，送回原上级模块。它加工的数据流叫做变换数据流。
- 协调模块：图 2-11（d），对其下属模块进行控制和管理的模块。在一个好的系统结构图中，协调模块应在较高层出现。

值得注意的是，结构图着重反映的是模块间的隶属关系，即模块间的调用关系和层次关系。它和程序流程图（常称为程序框图）有着本质的区别。程序流程图着重表达的是程序执行的顺序，以及执行顺序所依赖的条件。结构图则着眼于软件系统的总体结构，

它并不涉及模块内部的细节，只考虑模块的作用，以及它和上、下级模块的关系。而程序流程图则用来表达执行程序的具体算法。

没有学过软件开发技术的人，一般习惯于使用流程图编写程序，往往在模块还未做划分，程序结构的层次尚未确定以前，便急于用流程图表达他们对程序的构想。这就像造一栋大楼，在尚未决定建筑面积和楼屋有多少时，就已经开始砌砖了。这显然是不合适的。

Adele Goldberg 在 Succeeding with Objects 中叙述了一位犹太教教士在新年伊始的宗教集会上讲述的故事：

“一位教士登上一列火车，由于他经常乘坐这辆车，因此列车长认识他。教士伸手到口袋中掏车票。但没有找到，他开始翻他的行李。列车长阻止了他：‘教士，我知道您肯定有车票。现在别急着找。等找到后再向我出示。’但教士仍在找那张车票。当列车长再次见到他时，教士说：‘你不明白。我知道你相信我有车票，但……我要去哪里呢？’”

有太多项目失败就是因为它们没有明确的目标就开始了。

在结构化分析和设计技术中，通常存在着两种典型的问题类型，变换型问题和事务型问题，它们的数据流图和结构图都有明显的特征。下面分别讨论它们的数据流图形态及其映射成结构图的过程。

结构图（Structured Charts，SC）是准确表达程序结构的图形表示方法，它能清楚地反映出程序中各模块间的层次关系和联系。与数据流图反映数据流的情况不同，结构图反映的是程序中控制流的情况。图 2-12 为某大学教务管理系统的结构图。

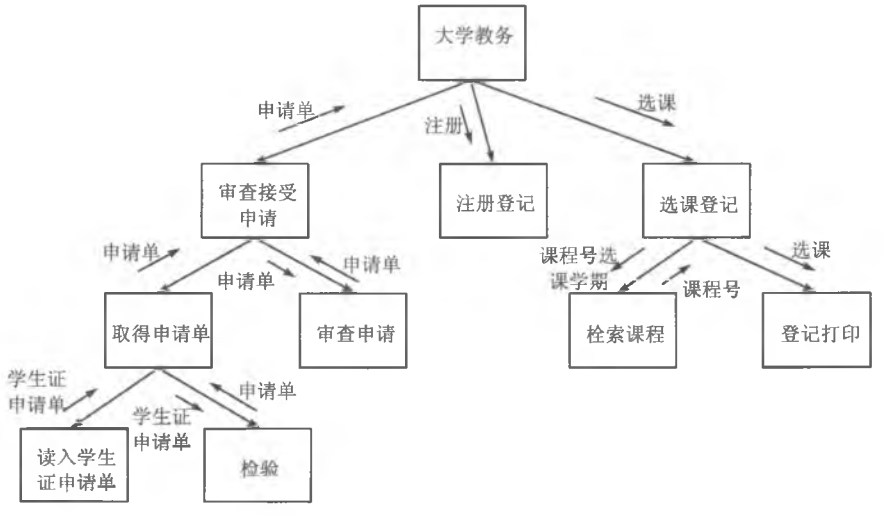

图 2-12　大学教务管理系统结构图

2. 系统结构图中的主要成分

结构图中的主要成分有四种。

（1）模块。以矩形框表示，框中标有模块的名字。对于已定义（或者已开发）的模块，则可以用双纵边矩形框表示，如图 2-13 所示。

图 2-13　模块的表示

（2）模块间的调用关系。两个模块，一上一下，以箭头相连，上面的模块是调用模块，箭头指向的模块是被调用模块，如图 2-14 中，模块 A 调用模块 B。在一般情况下，箭头表示的连线可以用直线代替。

（3）模块间的通信。以表示调用关系的长箭头旁边的短箭头表示，短箭头的方向和名字分别表示调用模块和被调用模块之间信息的传递方向和内容。如图 2-14 所示，首先模块 A 将信息 C 传给模块 B，经模块 B 加工处理后的信息 D 再传回给 A。

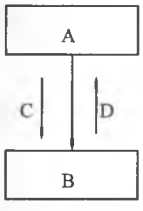

图 2-14　模块的调用关系及信息传递关系的表示

（4）辅助控制符号。当模块 A 有条件地调用模块 B 时，在箭头的起点标以菱形。模块 A 反复地调用模块 D 时，另加一环状箭头。如图 2-15 所示。

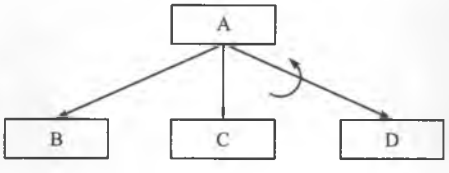

图 2-15　条件调用和循环调用的表示

在结构图中条件调用所依赖的条件和循环调用的循环控制条件通常都无需注明。

3．常用的系统结构图

常用的系统结构图有以下几种。

（1）变换型系统结构图。在数据处理问题中，我们通常会遇到这样一类问题，即从（程序）“外部”取得数据（例如从键盘、磁盘文件等），对取得的数据进行某种变换，然后再将变换得到的数据传回“外部”。其中取得数据这一过程称为传入信息（数据）

流程、变换数据的过程称为变换信息（数据）流程，传回数据过程称为传出信息（数据）流程，如图2-16所示。

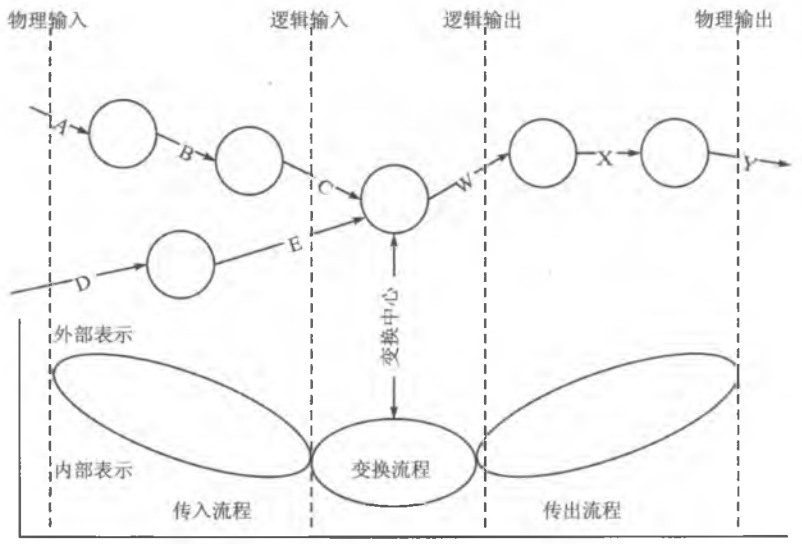

图2-16　变换型问题

当数据流图或其中某一段数据流表现出上述特征时，该数据流图或该段数据流图表示的是一个变换型问题。完成数据变换的处理单元叫变换中心。变换型问题数据流图基本形态及其对应的基本结构图分别如图2-17（a）和（b）所示。

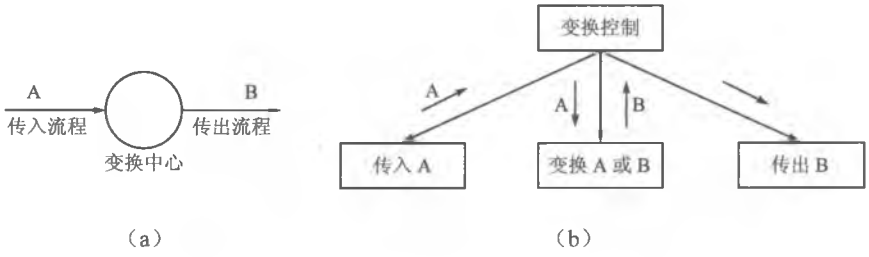

图2-17　基本变换型问题数据流图及其结构图

根据图2-17所示的基本映射关系所得到的图2-16所示变换型问题的结构图如图2-18所示。“变换控制”模块首先获得控制，然后控制沿着结构到达底层的“传入A”模块，物理输入数据A由“传入A”模块读入后，从底层逐步向上传送。在传送过程中，数据经“变换A成B”、“变换B成C”等模块的预处理，逐渐变换成纯粹的逻辑输入C、E。接着在“变换控制”模块的控制下，将逻辑输入经变换中心模块“变换C和E成W”处理后，变换成逻辑输出W，再从顶层逐步向下传送。在这一传送过程中，

数据经“变换 W 成 X”、“变换 X 成 Y”等模块的后处理，逐渐变换成适当的输出形式，最后由“传出 Y”模块完成物理输出。

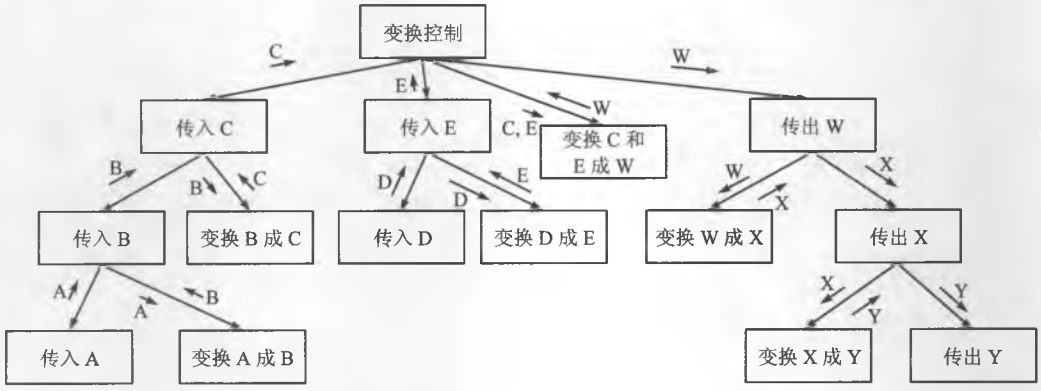

图 2-18　变换型问题结构图

（2）事务型系统结构图。在实际中，我们还常常会遇到另一类问题，即通常在接受某一项事务后，根据事务的特点和性质，选择分派给一个适当的处理单元，然后给出结果，如图 2-19 所示。

这类问题就是事务型问题。它的特点是，数据沿着接收分支把外部信息（数据）转换成一个事务项，然后计算该事务项的值，并根据它的值从多条数据流中选择其中的某一条数据流。发出多条数据流的处理单元叫事务中心。这类问题的典型结构图如图 2-20 所示。事务控制模块按所取得事务的类型，选择调用某一个处理事务模块。

（3）混合型系统结构图。在实际中，一些大型问题往往既不是单纯的变换型问题，也不是单纯的事务型问题，而是两种混合在一起的混合型问题。图 2-21 表示的就是一个混合型问题数据流图，图 2-22 是图 2-21 相应的结构图。

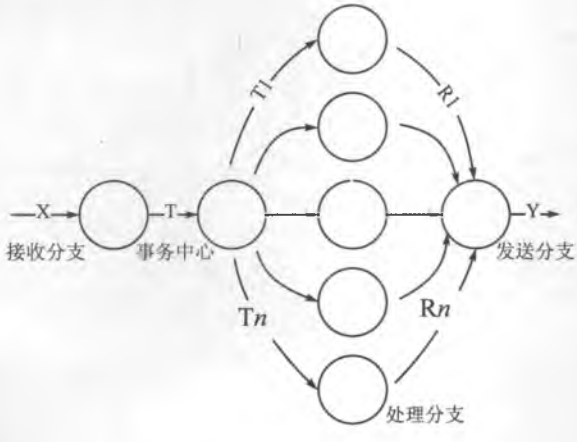

图 2-19　事务型问题

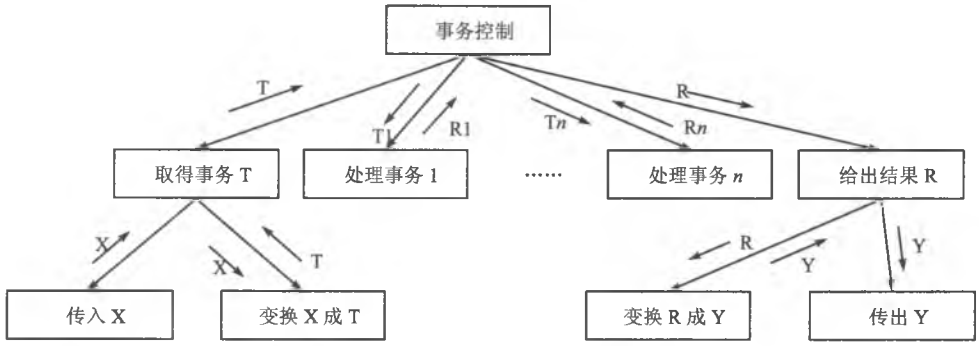

图 2-20　事务型问题结构图

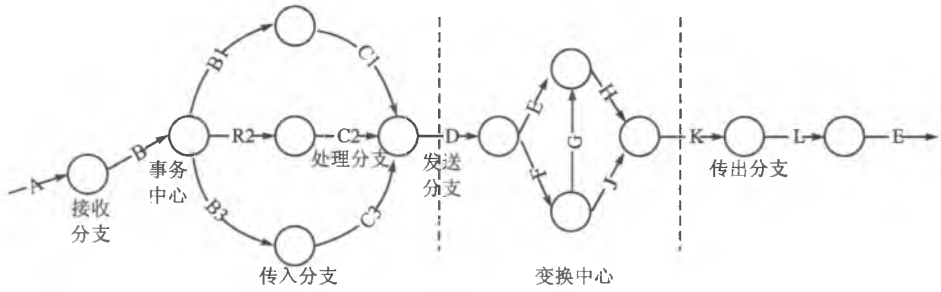

图 2-21　混合型问题数据流图

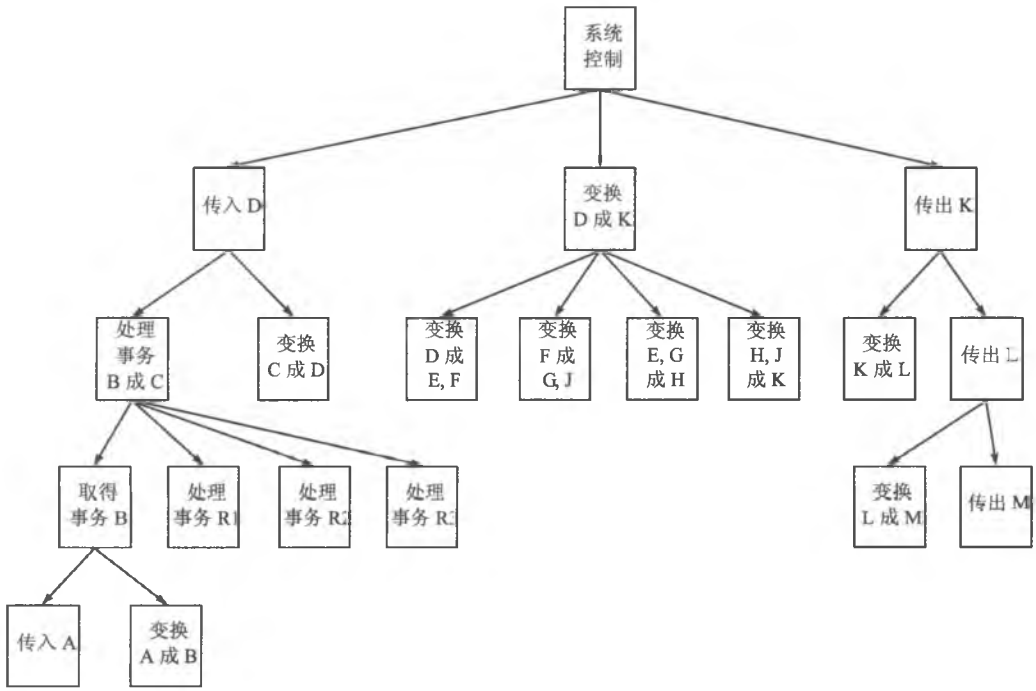

图 2-22　混合型问题结构图

对于这种混合型问题，一般以变换型问题为主，首先找出变换中心，设计出结构图的上层；然后根据数据流图的各部分具体类型分别映射得到它们的结构图。例如，对于如图 2-22 所示混合型问题，从整体上可以将其看成是一个从 A 到 M 的变换型问题，从 D 到 K 之间的变换是变换中心；从 A 到 D 是传入分支，具有事务型问题的特点；从 K 到 M 是传出分支。因此，该混合型问题结构图的上层可以由“传入 D”模块、“变换 D 成 K”模块和“传出 K”模块组成，“传入 D”模块的下层结构图由从传入分支映射得到的事务型问题结构图组成，“变换 D 成 K”模块和“传出 K”模块的下层结构图可以按通常的变换型问题映射方法获得。

2.2.3　用户界面设计

像人类追求心灵美和外表美那样，软件系统也追求（内在的）功能强大和（外表的）界面友好。

美的界面能消除用户由感觉引起的乏味、紧张和疲劳（情绪低落），大大提高用户的工作效率，从而进一步为发挥用户技能和完成任务做出贡献。从人-机界面发展历史与趋势上可以看出人们对界面美的需求，以及美在界面设计中的导向作用。

界面设计已经经历了两个界限分明的时代。第一代是以文本为基础的简单交互，如常见的命令行、字符菜单等。由于第一代界面考虑人的因素太少，用户兴趣不高。随着技术的发展，出现了第二代直接操纵的界面。它大量使用图形、语音和其他交互媒介，充分地考虑了人对美的需求。直接操纵的界面使用视听、触摸等技术，让人可以凭借生活常识、经历和推理来操纵软件，愉快地完成任务。更高层次的界面甚至模拟了人的生活空间，例如，虚拟现实环境。

界面的美充分体现了人-机交互作用中人的特性与意图，越来越多的用户将通过具有吸引力而令人愉快的人-机界面与计算机打交道。

一个好的用户界面应具有以下特点。

1．可使用性

（1）使用的简单性；
（2）用户界面中所用术语的标准化和一致性；
（3）拥有帮助功能；
（4）快速的系统响应和低的系统成本；
（5）用户界面应具有容错能力。

2．灵活性

（1）考虑用户的特点、能力、知识水平，应当使用户界面能够满足不同用户的要求；
（2）用户可以根据需要制订和修改界面方式；

（3）系统能够满足用户的希望和需要；

（4）与其他软件系统应有标准的接口。

3．复杂性和可靠性

（1）用户界面的规模和组织的复杂程度就是界面的复杂性；

（2）用户界面的可靠性是指无故障使用的间隔时间。

2.2.4　设计评审

在开发时期的每个阶段，特别是设计阶段结束时都要进行严格的技术评审，尽量不让错误传播到下一个阶段。设计评审一般采用评审会议的形式来进行。

软件设计评审流程如图2-23所示。

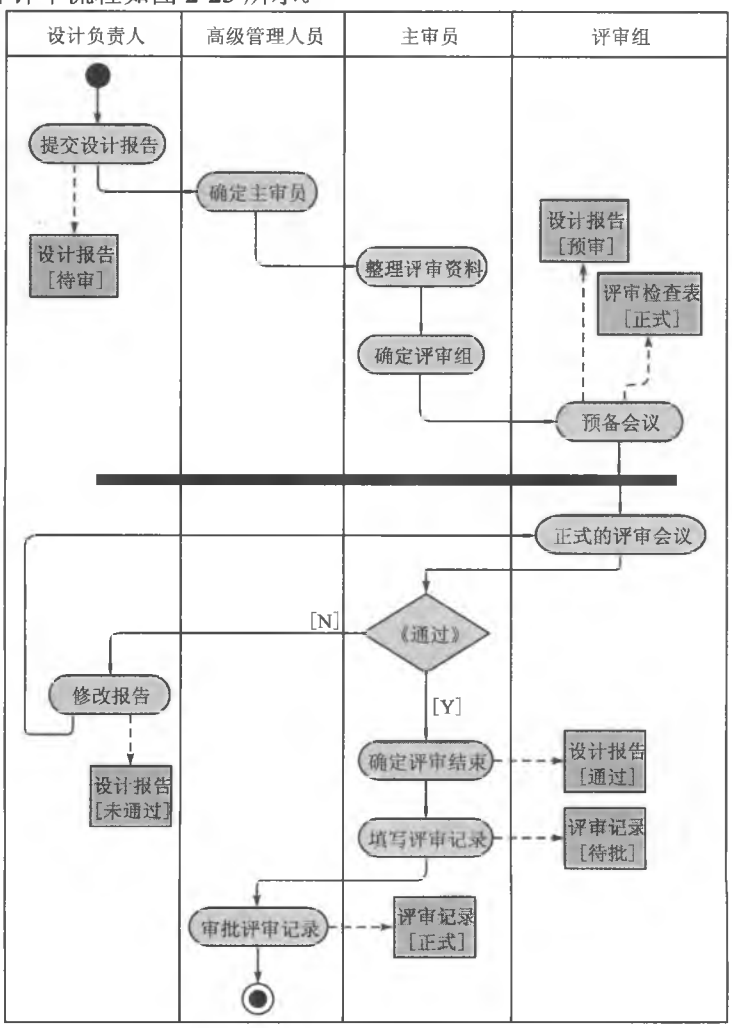

图2-23　软件设计评审流程图

- 设计负责人职责：一般工程设计均由软件公司选派设计负责人，设计负责人承担该项工程的全部设计管理任务；对设计质量、进度及各单项设计间的组织协调等全面负责，对各单项工程之间的衔接、协调和总体方案质量负主要责任，并负责编写总说明，汇编总概算。
- 高级管理人员职责：确定主审员、审批评审记录。
- 主审员职责：在评审会前提出项目的书面评审意见、确定评审组、确定评审结果并填写评审记录。
- 评审组职责：专业评审组评委表决通过项目初评结论并报综合评审会议，通过设计报告。

2.3　软件测试

软件测试是为了发现错误而执行程序的过程，是根据程序开发阶段的规格说明及程序内部结构而精心设计的一批测试用例（输入数据及其预期结果的集合），并利用这些测试用例去运行程序，以发现程序错误的过程。

从软件开发者的角度出发，则希望软件测试成为表明软件产品中不存在错误的过程，验证该软件已正确地实现了用户的要求，确立人们对软件质量的信心。从用户的角度出发，普遍希望通过软件测试暴露软件中隐藏的错误和缺陷，以考虑是否可接受该产品。

应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭；测试用例应当由测试输入数据和对应的预期输出结果这两部分组成；程序员应避免检查自己的程序；在设计测试用例时，应包括合理的输入条件和不合理的输入条件；充分注意测试中的群集现象。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目成正比。严格执行测试计划，排除测试的随意性；应当对每一个测试结果做全面检查；妥善保存测试计划、测试用例、出错统计和最终分析报告，为软件维护提供方便。

软件测试并不等于程序测试。软件测试应贯穿于软件定义与开发的整个期间。需求分析、概要设计、详细设计，以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明，以及源程序，都应成为软件测试的对象。

2.3.1　测试用例设计

测试用例是为特定目标开发的测试输入、执行条件和预期结果的集合。设计测试用例通常有两种常用的测试方法：黑盒测试和白盒测试。

1．黑盒测试

黑盒测试把测试对象看做一个空盒子，不考虑程序的内部逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明，又称为功能测试或数据驱动测试。

黑盒测试方法主要是在程序的接口上进行测试，主要是为了发现以下错误。

- 是否有不正确或遗漏了的功能；在接口上，能否正确的接收输入，能否输出正确的结果；
- 是否有数据结构错误或外部信息访问错误；性能上是否能够满足要求；是否有初始化或终止性错误；
- 黑盒测试需要在所有可能的输入条件和输出条件中确定测试数据，以检查程序是否都能产生正确的输出；有时测试数据量太大，是不现实的。

黑盒测试的测试用例设计方法主要有如下几种。

（1）等价类划分。等价类划分是一种典型的黑盒测试方法，使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。该方法把所有可能的输入数据即程序的输入域划分为若干个部分，然后从每一部分中选取少数有代表性的数据作为测试用例。

使用这一方法设计测试用例要经历划分等价类（列出等价类表）和选取测试用例两步。

- 第一步首先划分等价类。等价类是指某个输入域的子集合。在该子集合中，各个输入数据对揭露程序中的错误都是等效的。测试某等价类的代表值就等价于对这一类其他值的测试。等价类的划分有两种不同的情况：有效等价类是指对于程序的规格说明来说，是合理的、有意义的输入数据构成的集合；无效等价类是指对于程序的规格说明来说，是不合理的、无意义的输入数据构成的集合。
- 第二步再从划分出的等价类中按以下原则选择测试用例。为每一个等价类规定一个唯一编号；设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

（2）边界值分析。边界值分析也是一种黑盒测试方法，是对等价类划分方法的补充。人们从长期的测试工作经验得知，大量的错误是发生在输入或输出范围的边界上，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。使用边界值方法设计测试用例，应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。

（3）错误推测法。人们也可以靠经验和直觉推测程序中可能存在的各种错误，从而有针对性地编写检查这些错误的例子。其基本思想是：列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。

（4）因果图。如果在测试时必须考虑输入条件的各种组合，可使用一种适于描述多种条件的组合，相应产生多个动作的形式来设计测试用例，这就需要利用因果图。这种方法最终生成的就是判定表。它适合于检查程序输入条件的各种组合情况。用因果图生成测试用例的基本步骤是：

- 分析软件规格说明描述中，哪些是原因（即输入条件或输入条件的等价类），哪些是结果（输出条件），并给每个原因和结果赋予一个标志符；
- 分析软件规格说明描述中的语义，找出原因与结果之间，原因与原因之间对应的是什么关系？根据这些关系，画出因果图；
- 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特殊情况，在因果图上用一些记号标明约束或限制条件；
- 把因果图转换成判定表；
- 把判定表的每一列拿出来作为依据，设计测试用例。

2. 白盒测试

白盒测试把测试对象看做一个透明的盒子，它允许测试人员利用程序内部的逻辑结构和有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序的状态，确定实际的状态是否与预期的状态一致，又称为结构测试或逻辑驱动测试。

白盒测试主要对程序模块进行如下检查：

- 对程序模块的所有独立的执行路径至少测试一次；
- 对所有的逻辑判定，取“真”与取“假”的两种情况都至少测试一次；
- 在循环的边界和运行界限内执行循环体；
- 测试内部数据结构的有效性等。

3. 逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础的设计用例的技术。它属白盒测试，包括语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、条件组合覆盖、路径覆盖等。

- 语句覆盖：就是设计若干个测试用例，运行被测程序，使每一可执行语句至少执行一次。
- 判定覆盖：设计若干个测试用例，运行被测程序，使程序中每个判断的取真分支和取假分支至少经历一次，又称为分支覆盖。
- 条件覆盖：设计若干个测试用例，运行被测程序，使程序中每个判断的每个条件的可能取值至少执行一次。
- 判定-条件覆盖：设计足够的测试用例，使判断中每个条件的所有可能取值至少执行一次，每个判断中的每个条件的可能取值至少执行一次。
- 条件组合覆盖：设计足够的测试用例，运行被测程序，使每个判断的所有可能的条件取值组合至少执行一次。
- 路径覆盖：设计足够的测试用例，覆盖程序中所有可能的路径。

2.3.2 软件测试策略

从测试实际的前后过程来看，软件测试是由一系列不同的测试所组成，这些软件测试的步骤分为：单元测试、集成测试（又称组装测试）、确认测试和系统测试。软件开发的过程是自顶向下的，测试则正好相反，以上这些过程就是自底向上，逐步集成的。

1. 单元测试

单元测试也称为模块测试，是针对每个模块进行的测试，可从程序的内部结构出发设计测试用例，多个模块可以平行地对立地测试。通常在编码阶段进行，必要的时候要制作驱动模块和桩模块。

驱动模块是指在单元测试和集成测试中，协调输入和输出的测试程序；桩模块指模拟被调用单元的程序。

单元测试可以测试模块接口、局域数据结构、独立路径、错误处理路径和边界条件五个方面的内容。

- 模块接口测试主要包括：调用本模块的输入参数是否正确；本模块调用子模块时输入给子模块的参数是否正确；全局量的定义在各模块中是否一致；在做内外存交换时要考虑文件属性是否正确；Open 与 Close 语句是否正确；缓冲区容量与记录长度是否匹配；在进行读写操作之前是否打开了文件；在结束文件处理时是否关闭了文件；正文书写/输入错误；I/O 错误是否做了检查并做了处理等。
- 局域数据结构测试包括：不正确或不一致的数据类型说明；使用尚未赋值或尚未初始化的变量；错误的初始值或错误的默认值；变量名拼写错或书写错；不一致的数据类型；全局数据对模块的影响等。
- 独立路径测试包括：选择适当的测试用例，对模块中重要的执行路径进行测试；应当设计测试用例查找由于错误的计算；不正确的比较或不正常的控制流而导致的错误；对基本执行路径和循环进行测试可以发现大量的路径错误。
- 错误处理测试：出错的描述是否难以理解；出错的描述是否能够对错误定位；显示的错误与实际的错误是否相符；对错误条件的处理正确与否；在对错误进行处理之前，错误条件是否已经引起系统的干预等。
- 边界条件测试：注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。如果对模块运行时间有要求的话，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。

2. 集成测试

在单元测试的基础上，将所有模块按照设计要求组装成系统，必须精心计划，应提交集成测试计划、集成测试规格说明和集成测试分析报告。

这时需要考虑的问题是：在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；一个模块的功能是否会对另一个模块的功能产生不利的影响；各个子功能组合起来，能否达到预期要求的父功能；全局数据结构是否有问题；单个模块的误差累积起来，是否会放大，从而达到不能接受的程度。

把模块组装为系统的方式有两种：一次性组装方式和增殖式组装方式。在组装测试时，应当确定关键模块，对这些关键模块及早进行测试，这些关键模块具有如下特征：满足某些软件需求，在程序的模块结构中位于较高的层次，较复杂易发生错误，有明确定义的性能要求。

3．确认测试

确认测试验证软件的功能、性能及其他特性是否与用户的要求一致。

对软件的功能和性能要求在软件需求规格说明书中已经明确规定。它包含的信息就是软件确认测试的基础。确认测试的主要步骤如下。

首先进行有效性测试：在模拟的环境（可能就在开发的环境）下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求，通过制订确认测试计划和执行测试计划确定软件的功能和性能等特性是否与需求相符、所有的文档都是正确且便于使用的，同时对于其他软件需求（可移植性、兼容性、出错自动恢复、可维护性等）都要进行测试。

其次进行软件配置复查，保证做到：软件配置的所有成分都齐全；各方面的质量均符合要求；具有维护阶段所必需的细节，而且已经编排好分类的目录。

最后进行验收测试：验收测试是以用户为主的测试，软件开发人员和质量保证人员也应参加，由用户参加设计测试用例，使用生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

确认测试应交付的文档有：确认测试分析报告；最终的用户手册和操作手册；项目开发总结报告。

4．系统测试

系统测试是将软件放在整个计算机环境下，包括软硬件平台、某些支持软件、数据和人员等，在实际运行环境下进行一系列的测试。系统测试的目的是通过与系统的需求定义做比较，发现软件与系统的定义不符合或与之矛盾的地方。

5．α测试和β测试

在软件交付使用之后，用户将如何实际使用程序，对于开发者来说是不知道的。通常在软件发布上市之前需要进行α测试和β测试。

α测试是由一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。α测试的目的是评价软件产品的FLURPS（功能、局域化、可使用性、可靠性、性能和支持）。尤其注重产品的界面和特色。

α测试可以从软件产品编码结束之时开始，或者在模块（子系统）测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。

β测试是由软件的多个用户在实际使用环境下进行的测试。这些用户返回有关错误信息给开发者。

由于β测试时，开发者通常并不在测试现场。因而，β测试是在开发者无法控制的环境下进行的软件现场应用。在β测试中，由用户记下遇到的所有问题，包括真实的，以及主观认定的，定期向开发者报告。β测试主要衡量产品的FLURPS，着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当α测试达到一定的可靠程度时，才能开始β测试。它处在整个测试的最后阶段。同时，产品的所有手册文本也应该在此阶段完全定稿。

2.3.3　软件测试类型

软件测试由一系列不同的测试组成。主要目的是对以计算机为基础的系统进行充分的测试。软件测试大致可以分为如下几大类。

（1）功能测试。功能测试是在规定的一段时间内运行软件系统的所有功能，以验证这个软件系统有无严重错误。

（2）可靠性测试。如果系统需求说明书中有对可靠性的要求，则需进行可靠性测试。

可靠性测试的评价指标主要有平均故障间隔时间 MTBF 和平均故障修复时间 MTTR。平均失效间隔时间 MTBF 是否超过规定时限，因故障而停机的时间 MTTR 在一年中应不超过多少时间。

（3）强度测试。强度测试是要检查在系统运行环境不正常乃至发生故障的情况下，系统可以运行到何种程度的测试。例如：把输入数据速率提高一个数量级，确定输入功能将如何响应；设计需要占用最大存储量或其他资源的测试用例进行测试；设计出在虚拟存储管理机制中引起“颠簸”的测试用例进行测试；设计出会对磁盘常驻内存的数据过度访问的测试用例进行测试。强度测试的一个变种就是敏感性测试。敏感性测试是指在程序有效数据界限内一个小范围内的一组数据可能引起极端的或不平稳的错误处理出现，或者导致极度的性能下降的情况发生。此测试用以发现可能引起这种不稳定性或不正常处理的某些数据组合。

（4）性能测试。性能测试是要检查系统是否满足在需求说明书中规定的性能，特别是对实时系统或嵌入式系统。性能测试常常需要与强度测试结合起来进行，并常常要

求同时进行硬件和软件检测。通常，对软件性能的检测表现在以下几个方面：响应时间、吞吐量、辅助存储区。例如，缓冲区、工作区的大小、数据处理精度等。

（5）恢复测试。恢复测试是要证实在克服硬件故障（包括掉电、硬件或网络出错等）后，系统能否正常地继续进行工作，并不对系统造成任何损害。为此，可采用各种人工干预的手段，如模拟硬件故障，故意造成软件出错等。并由此检查错误探测功能：系统能否发现硬件失效与故障；能否切换或启动备用的硬件；在故障发生时能否保护正在运行的作业和系统状态；在系统恢复后能否从最后记录下来的无错误状态开始继续执行作业，等等。掉电测试：其目的是测试软件系统在发生电源中断时能否保护当时的状态且不毁坏数据，然后在电源恢复时从保留的断点处重新进行操作。

（6）启动/停止测试。这类测试的目的是验证在机器启动及关机阶段，软件系统正确处理的能力。这类测试包括：反复启动软件系统（例如操作系统自举、网络的启动、应用程序的调用等），在尽可能多的情况下关机。

（7）配置测试。这类测试是要检查计算机系统内各个设备或各种资源之间的相互联结和功能分配中的错误。它主要包括以下几种：配置命令测试即验证全部配置命令的可操作性（有效性），特别是对最大配置和最小配置要进行测试，软件配置和硬件配置都要测试；循环配置测试即证明对每个设备物理与逻辑的，逻辑与功能的每次循环置换都能正常工作；修复测试即检查每种配置状态及哪个设备是坏的，并用自动或手工的方式进行配置状态间的转换。

（8）安全性测试。安全性测试是要检验在系统中已经存在的系统安全性、保密性措施是否发挥作用，有无漏洞。力图破坏系统的保护机构以进入系统的主要方法有以下几种：正面攻击或从侧面、背面攻击系统中易受损坏的那些部分；以系统输入为突破口，利用输入的容错性进行正面攻击；申请和占用过多的资源使系统崩溃，以破坏安全措施，从而进入系统；故意使系统出错，利用系统恢复的过程，窃取用户口令及其他有用的信息；通过浏览残留在计算机各种资源中的垃圾（无用信息），以获取如口令、安全码、译码关键字等信息；浏览全局数据，期望从中找到进入系统的关键字；浏览那些逻辑上不存在，但物理上还存在的各种记录和资料等。

（9）可使用性测试。可使用性测试主要从使用的合理性和方便性等角度对软件系统进行检查，发现人为因素或使用上的问题。要保证在足够详细的程度下，用户界面便于使用；输入量可容错、响应时间和响应方式合理可行；输出信息有意义、正确并前后一致；出错信息能够引导用户去解决问题；软件文档全面、正规、确切。

（10）安装测试。安装测试的目的不是查找软件错误，而是查找安装错误。在安装软件系统时，会有多种选择，例如，要分配和装入文件与程序库、布置适用的硬件配置，以及进行程序的联结；而安装测试就是要找出在这些安装过程中出现的错误。安装测试是在系统安装之后进行测试。它要检验：用户选择的一套任选方案是否相容，系统的每一部分是否都齐全，所有文件是否都已产生并确有所需要的内容，硬件的配置是否合理，等等。

（11）过程测试。在一些大型的系统中，部分工作由软件自动完成，其他工作则需由各种人员，包括操作员、数据库管理员、终端用户等，按一定规程同计算机配合，靠人工来完成。指定由人工完成的过程也需要经过仔细的检查，这就是所谓的过程测试。

（12）容量测试。容量测试是要检验系统的能力最高能达到什么程度，例如，对于编译程序，让它处理特别长的源程序；对于操作系统，让它的作业队列“满员”；对于信息检索系统，让它使用频率达到最大。在使系统的全部资源达到“满负荷”的情况下，测试系统的承受能力。

（13）文档测试。这种文档测试是检查用户文档（如用户手册）的清晰性和精确性。

（14）兼容性测试。这类测试主要想验证软件产品在不同版本之间的兼容性。有两类基本的兼容性测试：向下兼容和交错兼容。

2.3.4　面向对象的软件测试

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。针对面向对象的开发模型，结合传统的测试步骤划分，提出一种面向对象的测试模型。该模型包括 OOA Test、OOD Test、OOP Test、面向对象单元测试、集成测试和系统测试。

OOA Test 和 OOD Test 分别是对分析结果和设计结果的测试，主要是对分析和设计产生的文本进行测试，是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代码实现进行测试，其主要测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 C++ 语言实现，主要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用，类间的消息传递等。

1．面向对象分析的测试

传统的面向过程分析是一个功能分解的过程，把一个系统看成可以分解的功能的集合。这种传统的功能分解法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析是把 E-R 图和语义网络模型，即信息造型中的概念，与面向对象程序语言中的重要概念结合在一起而形成的分析方法，最后通常得到的是问题空间的图表的形式描述。

OOA 直接映射问题空间，全面地将问题空间中实现功能的现实抽象化。将问题空间中的实例抽象为对象，用对象的结构反映问题空间的复杂实例和复杂关系，用属性和

服务表示实例的特性和行为。对一个系统而言，与传统分析方法产生的结果相反，行为相对稳定，结构则相对不稳定，这更充分反映了现实的特性。由于 OOA 的结果是为后面阶段类的选定和实现，类层次结构的组织和实现提供平台。

因此，OOA 对问题空间分析抽象的不完整，最终会影响软件的功能实现，导致软件开发后期大量可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。由此可见，对 OOA 的测试重点应该放在完整性和冗余性方面。OOA 阶段的测试划分为以下五个方面：对认定的对象的测试；对认定的结构的测试；对认定的主题的测试；对定义的属性和实例关联的测试；对定义的服务和消息关联的测试。

2．面向对象设计的测试

通常结构化的设计方法是用面向作业的设计方法，它把系统分解后，提出一组作业，这些作业是以过程实现系统的基础构造，把问题域的分析转化为求解域的设计，分析的结果是设计阶段的输入。

而面向对象设计（OOD）采用“造型的观点”，以 OOA 为基础归纳类，并建立类结构或进一步构造成类库，实现分析结果对问题空间的抽象。OOD 归纳的类可以是对象简单的延续，也可以是不同对象的相同或相似的服务。因为 OOD 是 OOA 的进一步细化和更高层的抽象。所以，OOD 与 OOA 的界限通常难以区分。由于 OOD 确定类和类结构不仅能满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，能方便实现功能的重用和扩充，以不断适应用户的要求。因此，对 OOD 的测试，建议针对功能的实现和重用，以及对 OOA 结果的扩展，从如下三方面考虑：

- 对认定的类的测试；
- 对构造的类层次结构的测试；
- 对类库的支持的测试。

其中，对构造的类层次结构的测试通常基于 OOA 中产生的分类结构的原则来组织，着重体现父类和子类间的一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，测试如下几个方面：

- 类层次结构是否包含了所有定义的类；
- 是否能体现 OOA 中所定义的实例关联；
- 是否能实现 OOA 中所定义的消息关联；
- 子类是否具有父类没有的新特性；
- 子类间的共同特性是否完全在父类中得以体现。

3．面向对象编程的测试

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继

承使传统测试遇到了一个难题，即对继承的代码究竟如何测试，多态性使得面向对象程序对外呈现出强大的处理能力，但同时却使程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现封装在类中，能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误能精确地确定在某一具体的类。在面向对象编程（OOP）阶段，将测试集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面：

（1）数据成员是否满足数据封装的要求；

（2）类是否实现了要求的功能。

4．面向对象的单元测试

传统的单元测试是针对程序的函数、过程或完成某一定功能的程序块。沿用单元测试的概念，实际测试类成员函数。传统的单元测试方法在面向对象的单元测试中都可以使用。

面向对象编程的特性使得对成员函数的测试，又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性，使子类继承或过载的父类成员函数出现了传统测试中未遇见的问题。需做以下的考虑。

（1）继承的成员函数是否都不需要测试。对父类中已经测试过的成员函数，下面两种情况需要在子类中重新测试：继承的成员函数在子类中做了改动；成员函数调用了改动过的成员函数的部分。

（2）对父类的测试是否能照搬到子类。只需在父类测试要求和测试用例上添加对子类函数的新的测试要求和增补相应的测试用例。

5．面向对象的集成测试

面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，由于程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于黑盒子的集成测试。

面向对象的集成测试能够检测出，相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注系统的结构和内部的相互作用。面向对象的集成测试可以先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求，提供一种成为“可逆性工程”的功能，即通过原程序得到类关系图和函数功能调用关系图，将“可逆性工程”得到的结果与OOD的结果相比较，检测程序结构和实现是否有缺陷，即通过这种方法检测OOP是否达到了设计要求。

动态测试设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定的覆盖标准。测试所要达到的覆盖标准可以是达到类所有的

服务要求或服务提供的一定覆盖率，依据类间传递的消息，达到对所有执行线程的一定覆盖率，达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。值得注意的是设计测试用例时，不但要设计确认类功能满足的输入，还应该有意识地设计一些被禁止的例子，确认类是否有不合法的行为产生，如发送与类状态不相适应的消息，要求不相适应的服务等。

6．面向对象的系统测试

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA 分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全“再现”问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再认识。系统测试需要对被测的软件结合需求分析做仔细的测试分析，建立测试用例。

2.4　软件维护

软件正式交付用户以后，即进入漫长的维护期。在这一阶段，基本任务是保证软件在这段相当长的时期内能够正常运行。软件维护需要的工作量很大，随着时间的推移，软件维护对开发商带来的成本压力也越来越大。现在许多软件开发商要把 70%的工作量用在维护已有的软件上。平均来说，大型软件的维护成本高达开发成本的 4 倍左右。

2.4.1　软件的可维护性

软件的维护活动是基于软件是可维护的这一前提，软件的可维护性考虑应该贯穿于软件开发的各个阶段。

1．软件具有可维护性

原则上，任何人造系统，都是可维护的。可见的实体，如桌椅板凳、车子、房子、计算机等，因为我们知道如何组装它们，知道它们的结构，所以可以维护。软件虽然是不可见的，我们在开发的时候，是非常清楚它们是如何组成的，如何活动的，不管是多么复杂的软件，开发者同样了解它，可以维护它。

依据软件本身的特点，软件具有可维护性主要由以下三个因素决定：

（1）可理解性。软件通常是沿着“子系统—模块—功能—内部处理过程”这样的思路逐步细化的，软件维护人员通过了解同样的思路，可以理解软件。

（2）可测试性。借助人工的经验或者先进的测试工具，维护人员可以对运行的软件进行测试，找出问题所在。

（3）可修改性。任何软件都是通过某种开发工具来完成的，都有源码，运用同样的工具，维护人员可以对现有软件进行修改、编译，使之重新运行起来。

2．采用软件工程提高软件的可维护性

软件危机从某种意义上可以看成是软件维护的危机，由此诞生的软件工程也可以看成是提高软件可维护性的工程。软件工程的采用，使得软件开发过程进一步规范，强制性产生了一系列的文档，因为这些文档从各个阶段、各个方面对软件结构、原理加以说明，极大地丰富了维护所需的资源，增加了软件的可维护性。所以，对于维护人员来说文档比程序源码更为重要。

软件系统的文档可分为用户文档和系统文档两大类。

用户文档主要是描述软件功能和使用方法。至少应包括以下几个方面。

（1）功能说明：说明系统能做什么。

（2）安装文档：说明系统安装过程及所需软硬件配置。

（3）用户使用手册：通过详尽的例子向终端用户介绍如何使用这个系统。

（4）参考手册：从技术的角度对系统进行形式化的描述。

（5）管理员指南：说明系统管理人员如何处理使用中出现的各种情况。

系统文档则关心实现细节，描述系统需求、设计、实现和测试等各个方面。

软件开发过程中的绝大多数文档都属于系统文档，如需求分析报告、系统设计报告、源码、测试计划等。

3．注重可维护性的开发过程

随着软件规模的不断增大，维护活动耗费的成本急剧攀升，甚至有人预言，假如再不重视软件的可维护性，总有一天，开发商将不得不投入所有的资源进行软件维护，而无力开发新的软件。这虽然有些危言耸听，但在需求分析、设计、编码各阶段加入灵活应变因素的做法还是得到越来越多的开发者的认可，如此一来，既可以降低维护难度，从而降低维护成本，又可以提高软件的可维护性。

要在开发过程中提高软件的可维护性，必须从如何使软件易于理解、易于测试、易于修改出发进行考虑。具体如下：

- 在需求分析阶段，应该对将来要改进的和可能会修改的部分加以明确说明；对软件的跨平台可移植性进行讨论，形成解决方案。
- 在设计阶段，应该尽量遵循“高内聚低耦合”的模块设计原则，对将来可能要修改的地方，采用灵活的易于扩充的设计方案；考虑跨平台可移植性的设计；加大可重用构件的设计力度；如果各方面条件都满足，应该使用面向对象的设计方法。
- 在编码阶段，应该采用科学的代码规范，强化注释的力度，保证注释的质量，这一点对于将来的维护非常重要；加大可重用构件的使用力度。同样，如果各方面条件都满足，应该使用面向对象的编码工具。

- 在测试阶段，一方面，测试的目的本质上是为了减少各种维护的工作量，尤其是纠错型维护。也就是说，测试做好了，以后的维护量就少了；另一方面，测试相关的文档（包括测试大纲、用例设计、测试报告等）是维护后的回归测试的基础，测试阶段要是文档不全，维护后几乎无法进行回归测试，维护的质量也就无法保证，最糟糕的情况莫过于改了旧错，又引入了新错，进入恶性循环。
- 在维护阶段，要有严格的配置管理，每一次维护工作之后，都要按照配置关联，同步更新维护有关的系统文档（包括维护需求、源代码、设计文档、测试文档）和用户文档（包括用户使用手册等），保证系统的一致性；同时，在大的维护之后，交付之前，一定要及时做好用户的培训工作，否则就会使用户因为受挫折而产生不满，实际工作中，有些所谓的“错误”可能只是由于用户使用手册描述不清楚而造成的。

4．可维护性的度量

我们从软件的内部和外部两个方面来度量可维护性。

在软件外部，可以用平均修复时间（Mean Time To Repair，MTTR）来度量软件的可维护性，它是指处理一个有错误的软件组件需要花费的平均时间。如果我们用 M 表示可维护指标，那么

$M = 1/(1+\text{MTTR})$

为此，我们需要每一个问题详细记录以下信息：

- 分析问题需要的时间；
- 确定改动方案的时间；
- 执行改动花费的时间；
- 测试改动花费的时间；
- 其他管理浪费的时间。

在软件内部，可以通过度量软件的复杂性来间接度量可维护性。与软件复杂性相关的因素有以下几个方面。

（1）环路数。这是 McCabe 于 1976 年提出的。环路数可以反映源代码结构的复杂度。环路数的计算过程是：分析源码，绘制出等效的控制流图；运用图论的知识计算出控制流图的线性无关路径数，用这个数度量和比较复杂程度。通过观察环路数的增长幅度，我们可以比较多种维护方案的优劣，选择出对环路数影响最小的作为最优的方案。

（2）软件规模。通常，可以认为软件包含的组件越多，软件就越复杂，可维护性就越差；

（3）其他因素。包括嵌套深度、系统用户数等。

对于软件内部的可维护性，迄今还没有一个突出的、全面的、通用的模型，需要结合不同开发组织自身的经验，建立合适的经验模型。一般情况下，可以按如下步骤建立经验模型：

- 首先，确定影响可维护性的若干主要因素，并为其制订尺度；
- 其次，用大量的现有案例，计算出一个包含影响因素及其权值的多项式模型；
- 再次，利用这个经验模型分析新的软件，再根据实际的维护活动的感受进行校准；
- 重复这一校准过程，就能逐步建立起逼近实际的基本稳定的模型。

这个模型最常用的是其比较特性，也就是说，可以据此对问题的维护难度进行排序。

2.4.2　软件维护的分类

软件的维护从性质上分为：纠错型维护、适应型维护、预防型和完善型维护。

尽管经过严格的测试，但并不能保证软件中彻底没有错误，随着运行时间的延续，数据量的积累，各种应用环境的变化，错误仍会顽固地暴露出来，此时就要进行纠错型维护。

伴随着计算机硬件的新产品、操作系统的新版本的不断推出，正在运行的软件必须进行适应型维护。

用户逐渐熟悉软件以后，会提出一些改进需求，为了满足这些需求，必须进行完善型维护，这样的维护几乎占到维护工作量的一半以上。比如，打印格式的调整、统计口径的增加、业务流程的完善等。

以上三种维护都是用户驱动的，用户是维护需求的提出者，而开发商“为了明天的需要，把今天的方法应用到昨天的系统中”，目的是为了使旧系统焕发新活力，这样的维护是预防型维护，这种维护所占的比例很小，因为它耗资巨大。

Lientz 和 Swanson 调查发现（1980年），完善性维护约占50%，适应性维护约占25%，纠错性维护约占21%，其他维护只占4%。这个调查已是二十几年前的事了，具体的比例数我们不必深究，现在的情况并没有多大变化。

2.4.3　软件维护的工作量

维护活动可以分成生产类（比如，确认维护需求、设计、编码、测试、培训等）和非生产类（比如，熟悉原有软件的代码，理解原有软件的结构等）。

维护工作量可以用这样一个模型表示：

$$M=P+K^{c-d}$$

其中：M 是维护用的总工作量，P 是生产类活动的工作量，K 是经验常数，c 是软件的复杂程度，d 是维护人员对软件的熟悉程度。

对于一次具体的维护，确认需求和设计的工作量与问题的难易程度和大小有关，相对来说，工作量比较稳定。编码工作则与软件本身的质量有很大的关系，如果原来的编码格式混乱，注释不清，就会使生产类活动的工作量 P 增大；在软件的复杂度 c 一定

的前提下，维护人员对软件的熟悉程度 d 越低，则维护工作量呈指数规律增加；同样，如果由于开发混乱，导致软件复杂度 c 增加，从而使维护人员理解软件的难度增加，对软件的熟悉程度 d 也降低，那么维护工作量会以更快的速度上升。

除了原有软件本身的质量（包括设计质量、代码质量、文档质量和测试质量）对维护工作量有影响外，还有如下一些因素也会影响到维护工作量：

（1）维护工作本身是否规范，是否按软件工程的正确方法进行，对后续的维护工作量的影响同样不可忽视。如果维护工作不规范，代码修改与文档修改不同步，会导致维护后的软件更加复杂，更加难以理解，难以熟悉，维护工作量也会以指数速度增加。

（2）软件系统的类型不同，维护工作量也有区别。通常，一个系统越依赖于真实世界，就越可能发生变化，也就需要更大的维护工作量。按照对真实世界的依赖程度，软件系统可以分为：抽象系统、近似系统和模拟系统。抽象系统描述的问题具有形式化的精确定义，比如涉及标准数值计算方法的计算软件；近似系统是对于真实世界的一个简化的近似方案的描述，比如围棋软件，虽然围棋的规则是精确的，但由于由此衍生的走步方案和对弈模拟则几乎是无穷尽的，因此一般的设计都有计算深度的限定；模拟系统则包括广泛的行业应用软件，系统本身就承担着全部或部分业务，是嵌入真实世界中运行的，比如管理信息系统、ERP 系统、计费系统等。因此可见，抽象系统的维护工作量最小，模拟系统的维护工作量最大。

（3）硬件因素。不可靠的硬件系统会使软件系统产生一些令人恼火的随机性的问题，使追踪问题的根源变得更加困难。

上面所讲的都是客观因素，还有维护人员本身的因素，基本上在软件开发队伍中，维护被认为是出力不讨好的工作，维护人员作为软件企业中长期面对用户的角色，经常得承担来自其他开发人员和用户的双重压力，情绪低落，热情不高；同时部分软件企业对维护不够重视，认为维护是个无底洞，因而在人力和成本的投入上采取敷衍的态度，安排进行维护的人员大多是技术不够扎实的员工，甚至把维护作为培训新员工的渠道，导致维护人员技术素质一般，又缺乏热情，维护工作量岂有不大之理。

好在这种局面正在改观，随着软件企业的数量的激增，导致竞争加剧，在某些领域甚至到了白热化的程度，巩固已有的客户，开发已有的客户，从已有的客户身上寻找新的商机变得越来越重要，维护人员作为服务的窗口，逐渐被重视，因为只有维护做好了，才能巩固已有客户。

2.4.4　软件维护作业的实施和管理

不管是哪种类型的维护，都需要类似开发的过程，本质上说，维护过程是修改和压缩了的开发过程。

1. 建立维护组织

有实力的软件企业应该建立专业的维护部门，对大多数的中小软件企业，虽然没有专门的维护组织，但非正式的明确责任却是非常必要的，需要指定维护负责人，组建临时的维护小组，明确维护流程及各人的职责，可以专设维护配置员，也可由维护负责人兼任。如果同时有多个系统需要维护，可再设维护管理员，监督协调各维护小组，结构大致如图 2-24 所示。

对于大型软件系统的维护，设置维护管理员是非常必要的，由其协调和同步各维护小组的工作。

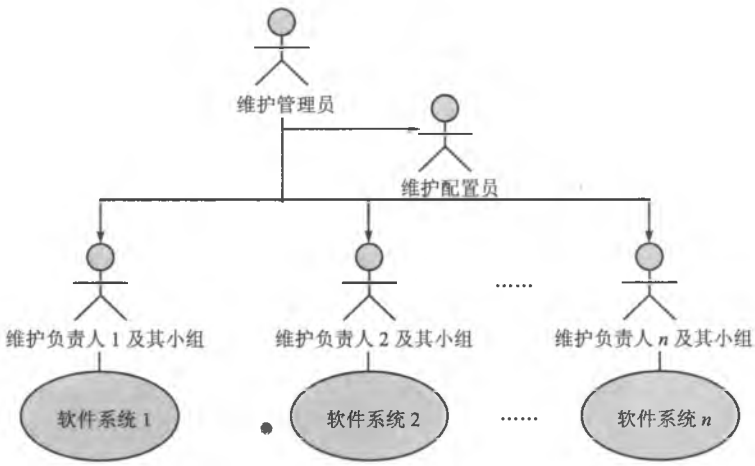

图 2-24　维护的组织结构

维护配置员则对维护活动需要的资源，以及维护过程中产生的各种文档进行标志和配置。由于维护必然会带来变动，因此需要维护配置员至少做好如下的变动控制：

- 变动的时间；
- 执行变动的人员；
- 变动的内容；
- 变动的结果；
- 批准变动的人员；
- 变动的通知范围；
- 变动的级别。

2. 提出维护需求

维护需求通常由用户提出，维护人员应该给用户提供空白的软件问题报告，并用具体的例子向用户说明如何记录碰到的软件问题，以便规范用户完整描述其维护需求。

软件问题报告通常包括如下内容：

- 问题发现人姓名及其所在的部门；
- 发现问题的时间；
- 发生问题的子系统名称；
- 问题产生前进行的操作；
- 问题的界面或描述；
- 问题的后果描述，比如退出当前操作，退出应用系统还是导致死机。

用户把填好的软件问题报告提交给维护管理员，维护管理员再根据各维护小组的分工，把软件问题报告转给相应的维护小组，由其负责人组织实施维护作业，同时由维护配置员进行维护资源配置管理。

如果同时有多个维护需求，维护管理员则应根据对用户的影响程度，首先确定维护的优先顺序，再依次安排维护活动。

3. 实施维护作业

每一次维护活动的实施都要经历如下的步骤：

- 确认维护需求；
- 制订维护计划；
- 编码；
- 测试；
- 交付用户。

维护作业中各阶段的参与人及其活动如图 2-25 所示。

图 2-25 描述的是一个符合软件工程思想和规范的维护流程。其中，维护计划包括维护的人员、时间安排、维护需求描述、维护设计及相关测试文档。另外，制订维护风险控制计划对保证维护按时保质的完成非常重要。

虽然维护所带来的副作用是客观存在的，不可能完全消除。但这样的工作流程可以大幅降低因管理因素产生的维护副作用（例如，因为配置混乱造成的文档更新不及时等），保证每一次维护活动都有据可查。

完善的维护设计文档，可以有效降低因修改数据结构带来的副作用，保证与该数据结构有关的代码段的修改不被遗漏。

回归测试则可以有效地查明修改程序代码对原有软件的影响。

当然，并非所有的维护活动都得完全按照上述流程进行。例如，如果需要维护的软件系统是支撑客户核心业务的系统，必须 7×24 小时运行，这样的系统一旦发生恶性软件问题（比如数据库崩溃等）时，维护组织就会以“救火队”的方式迅速展开维护，没有时间进行计划评审，此时软件开发商应派出技术过硬的维护小组，最大限度地降低风险。

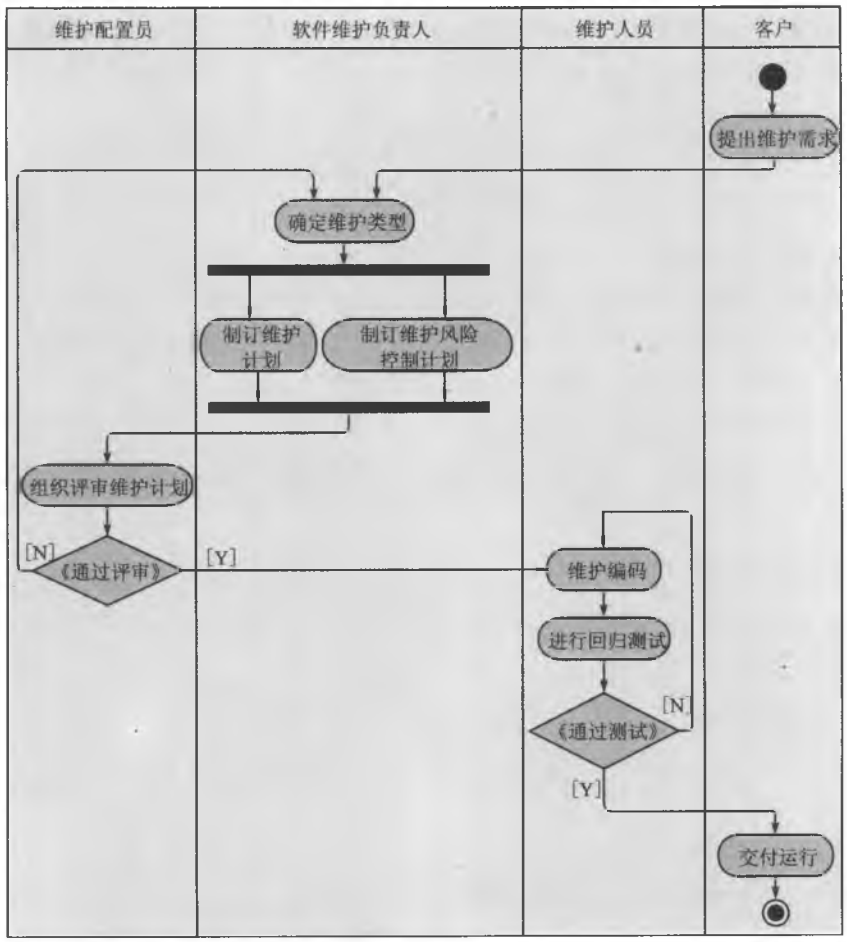

图 2-25　维护作业实施

4．记录维护要素

在维护活动中需要及时记录维护的有关信息，用以考查维护技术的有效性，估计软件的“优良”程度，确定维护的实际代价，同时这些记录将作为后续评价活动的依据。

那么需要记录哪些信息呢？Swanson 提出了信息内容，主要如下：

- 源码行数；
- 使用的开发语言；
- 程序的安装日期；
- 从安装至今程序运行的次数；
- 从安装至今程序失效的次数；
- 程序变动的标志；
- 因变动而增加的源码行数；
- 因变动而删除的源码行数；

- 每个改动耗费的人时数；
- 程序改动的日期；
- 修改者的姓名；
- 维护需求表的标志；
- 新维护需求是否源于以前的维护工作；
- 维护类型；
- 维护开始和完成日期；
- 累计用于维护的人时数；
- 已完成的维护创造的直接和间接的效益。

对于每项维护工作，都要收集上述数据，而且可以基于这些数据建立起维护数据库。

5. 评价维护活动

借助于维护记录，可以对维护工作做一些定量的统计。通常从以下几个方面进行度量：

- 以前维护工作引起的新维护占总维护需求的比例；
- 程序的平均失效次数；
- 每一类维护活动的总人时数及维护工作量占比；
- 维护需求表的平均周转时间；
- 维护活动中增加一条源码花费的平均人时数；
- 维护活动中删除一条源码花费的平均人时数；
- 按维护的程序、开发语言和维护类型统计程序变动数；
- 维护分类占比。

度量的结果可以作为以后调整维护工作的参考，对于合理规划维护工作量、优化资源分配、有针对性地强化对参与某类维护的人员的技术培训等方面可以起到积极的作用。

2.4.5　软件再生工程

软件的再生工程在软件开发现状中应用得很多，比如“单机版”改造为“网络版”、“客户-服务器模式（C/S）”改造为“浏览器-服务器模式（B/S）”、“非结构化”改造为“结构化”、“组件化改造”等软件行为，这些行为有的是全部再生，有的是局部再生。对于很久以前开发的软件，由于“未采用软件工程思想”等各种原因，导致文档缺失，甚至只剩下能运行的软件系统，这种情况大多考虑进行局部再生；对于近期采用软件工程思想开发的软件，由于文档相对齐全，则可以进行全部再生。

通常，软件的再生工程包括以下六类活动（可根据具体情况取舍）。

1. 筛选

这是局部再生工程中首先要进行的活动，需要选出现有的软件系统中需要进行再生的模块，通常应该着重对以下三类模块进行考查：

（1）确定将使用多年的；

（2）正在成功运行的；

（3）近期将做重大变更的。

2. 文档重构

对于早期的软件，这项活动非常耗费精力，应分轻重缓急分别对待：

（1）对于相对稳定的程序，暂且使其保持现状；

（2）对于当前正在修改的部分建立完整的文档，其他部分则在使用中逐步建立文档；

（3）对于支撑用户核心业务的程序，必须建立完整的文档，但也最好想方设法地降低建立文档的工作量。

3. 逆向工程

软件的逆向工程就是指分析一个程序的过程，最大程度地建立比源码更抽象的高级表达，它也是一个恢复设计结果的过程。逆向工程工具可以从现有的软件代码中抽取有关的数据、体系结构和处理过程的设计信息。

4. 代码重构

这是最常见的再生工程活动。首先用重构工具分析源码，标出非结构化的部分，然后自动重构问题代码，经过复审和测试生成最终的重构代码，同时更新有关文档。

5. 数据重构

指重构数据结构。这是一项全范围的活动。首先进行逆向工程，分解出当前的数据结构，再进行重构。数据结构是软件的基础信息，对其进行重构必然会导致代码重构。

6. 重新开发

基于上面几个活动的结果，应用软件工程的原理、概念、技术和方法重新开发现有系统。

2.5　软件开发环境

软件工程原理鼓励研制与采用各种先进的软件开发方法和工具，以便不断提高软件生产率。软件工程概念提出的初期，人们着重研究的是各种新的程序设计技术，这种程序设计方法和技术可使软件开发效率得到较大程度的改善。继程序设计技术、方法的改进之后，相继发展了许多适用于软件开发各阶段的方法。但是，这些软件方法和技术是

建立在以图表为工具的手工作业基础之上的。软件开发人员只为别人开发自动化的工具，而自己的生产活动中依靠的仍然是知识人的密集型脑力劳动，其非常突出的特点是高比重的重复劳动。显然这无法从根本上扭转软件生产率低的落后局面。

为了改变这种局面，一个直接而有效的途径是从软件人员的角度出发，在软件工程实践方面提供一整套开发与维护的支持，这就是 20 世纪 80 年代以来在国际上引起了广泛重视和研究的软件工具，进而发展成为日臻完善的软件开发环境。

2.5.1　软件开发环境概述

软件开发环境（Software Development Environment，SDE）是一组相关软件工具的集合，它们组织在一起支持某种软件开发方法或者与某种软件开发模型相适应。SDE 在欧洲又叫集成式项目支援环境（Integrated Project Support Environment，IPSE）。

软件开发环境的主要组成成分是软件工具。人-机界面是软件开发环境与用户之间的一个统一的交互式对话系统，它是软件开发环境的重要质量标志。存储各种软件工具加工所产生的软件产品或半成品（如源代码、测试数据和各种文档资料等）的软件环境数据库是软件开发环境的核心。工具间的联系和相互理解都是通过存储在信息库中的共享数据得以实现的。

软件开发环境数据库是面向软件工作者的知识型信息数据库，其数据对象是多元化并具有智能性的。软件开发数据库用来支撑各种软件工具，尤其是自动设计工具、编译程序等的主动或被动的工作。

较初级的 SDE 数据库一般包含通用子程序库、可重组的程序加工信息库、模块描述与接口信息库、软件测试与纠错依据信息库等；较完整的 SDE 数据库还应包括可行性与需求信息档案、阶段设计详细档案、测试驱动数据库、软件维护档案等。更进一步的要求是面向软件规划到实现、维护全过程的自动进行，这要求 SDE 数据库系统是具有智能性的，其中比较基本的智能结果是软件编码的自动实现和优化、软件工程项目的多方面不同角度的自我分析与总结。这种智能结果还应主动地被重新改造、学习，以丰富 SDE 数据库的知识、信息和软件积累。这时候，软件开发环境在软件工程人员的恰当的外部控制或帮助下逐步向高度智能与自动化迈进。

软件实现的根据是计算机语言。时至今日，计算机语言发展为算法语言、数据库语言、智能模拟语言等多种门类，在几十种重要的算法语言中，C&C++语言日益成为广大计算机软件工作人员的亲密伙伴，这不仅因为它功能强大、构造灵活，更在于它提供了高度结构化的语法、简单而统一的软件构造方式，使得以它为主构造的 SDE 数据库的基础成分——子程序库的设计与建设显得异常方便。

事实上，以 C&C++为背景建立的 SDE 子程序库能为软件工作者提供比较有效、灵活、方便、友好的自动编码基础，尤其是 C++的封装等特性，更适合大项目的开发管理和维护。

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

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

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

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

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

工具集包括事务系统规划工具、项目管理工具、支撑工具、分析设计工具、程序设计工具、测试工具、原型建造工具、维护工具和框架工具等，所有这些工具可分为贯穿整个开发过程的工具（例如，软件项目管理工具）和解决软件生命周期中某一阶段问题的工具（例如，软件价格模型及估算工具）。

由于软件开发环境具有集成性、开放性、可裁减性、数据格式一致性、风格统一的用户界面等特性，因而能大幅度提高软件生产率。其中开放性是指允许其他的软件工具加入到软件开发环境之中。

2.5.2　软件开发环境的功能与分类

较完善的软件开发环境通常具有如下功能：

（1）软件开发的一致性及完整性维护；

（2）配置管理及版本控制；

（3）数据的多种表示形式及其在不同形式之间自动转换；

（4）信息的自动检索及更新；

（5）项目控制和管理；

（6）对方法学的支持。

软件开发环境可按以下几种角度分类。

按软件开发模型及开发方法分类，有支持瀑布模型、演化模型、螺旋模型、喷泉模型，以及结构化方法、信息模型方法、面向对象方法等不同模型及方法的软件开发环境。

按功能及结构特点分类，有单体型、协同型、分散型和并发型等多种类型的软件开发环境。

按应用范围分类，有通用型和专用型软件开发环境。其中专用型软件开发环境与应用领域有关，故又可称为应用型软件开发环境。

按开发阶段分类，有前端开发环境（支持系统规划、分析、设计等阶段的活动）、后端开发环境（支持编程、测试等阶段的活动）、软件维护环境和逆向工程环境等。此类环境往往可通过对功能较全的环境进行剪裁而得到。软件开发环境由工具集和集成机制两部分构成，工具集和集成机制间的关系犹如“插件”和“插槽”间的关系。

2.5.3　软件开发环境的结构

软件开发环境通常由以下几个部分组成。

（1）工具集。软件开发环境中的工具可包括：支持特定过程模型和开发方法的工具，如支持瀑布模型及数据流方法的分析工具、设计工具、编码工具、测试工具、维护工具，支持面向对象方法的 OOA 工具、OOD 工具和 OOP 工具等；独立于模型和方法的工具，如界面辅助生成工具和文档出版工具；亦可包括管理类工具和针对特定领域的应用类工具。

（2）集成机制。对工具的集成及用户软件的开发、维护及管理提供统一的支持。按功能可划分为环境信息库、过程控制及消息服务器、环境用户界面三个部分。

（3）环境信息库。环境信息库是软件开发环境的核心，用以储存与系统开发有关的信息并支持信息的交流与共享。库中储存两类信息，一类是开发过程中产生的有关被开发系统的信息，如分析文档、设计文档、测试报告等；另一类是环境提供的支持信息，如文档模板、系统配置、过程模型、可复用构件等。

（4）过程控制和消息服务器。过程控制和消息服务器是实现过程集成及控制集成的基础。过程集成是按照具体软件开发过程的要求进行工具的选择与组合，控制集成并行工具之间的通信和协同工作。

（5）环境用户界面。包括环境总界面和由它实行统一控制的各环境部件及工具的界面。统一的、具有一致的视感（Look & Feel）的用户界面是软件开发环境的重要特征，是充分发挥环境的优越性、高效地使用工具并减轻用户的学习负担的保证。

软件开发环境的结构为层次性结构，可分为四层。

（1）宿主层：包括基本宿主硬件和基本宿主软件。

（2）核心层：包括工具组、环境数据库和会话系统。

（3）基本层：包括至少一组工具，如编译工具、调试工具等。

（4）应用层：以基本层为基础补充某些工具，以适应应用软件的要求。

2.5.4　软件开发环境的发展

目前，随着软件开发工具的积累与自动化工具的增多，软件开发环境进入了第三代ICASE（Integrated Computer-Aided Software Engineering）。系统集成方式经历了从数据交换（早期CASE采用的集成方式：点到点的数据转换），到公共用户界面（第二代CASE：在一致的界面下调用众多不同的工具），再到目前的信息中心库方式。这是ICASE的主要集成方式。它不仅提供数据集成（1991年IEEE为工具互连提出了标准P1175）和控制集成（实现工具间的调用），还提供了一组用户界面管理设施和一大批工具，如垂直工具集（支持软件生存期各阶段，保证生成信息的完备性和一致性）、水平工具集（用于不同的软件开发方法），以及开放工具槽。

ICASE信息库是一组实现“数据-工具”，以及“数据-数据”集成的机制和数据结构，它提供了明显的数据库管理系统的功能。此外，中心库还完成了下面功能。

（1）数据完整性：包括确认中心库的数据项，保证相关对象间的一致性，以及当对一个对象的修改需要对其相关对象进行某些修改时自动完成层叠式修改等功能。

（2）信息共享：提供在多个开发者和多个开发工具间共享信息的机制，管理和控制对数据及加锁/解锁对象的多用户访问，以使得修改不会被相互间不经意地覆盖。

（3）数据-工具集成：建立可以被环境中所有工具访问的数据模型，控制对数据的访问，实现了配置管理功能。

（4）数据-数据集成：数据库管理系统建立数据对象间的关系，使其可以完成其他功能。

（5）方法学实施：存储在中心库中的数据的ER模型可能蕴含了特定的软件工程范型，至少关系和对象定义了一系列为了建立中心库的内容而必须进行的步骤。

（6）文档标准化：在数据库中对象的定义直接导致了创建软件工程文档的标准方法。

ICASE的最终目标是实现应用软件的全自动开发，即开发人员只要写好软件的需求规格说明书，软件开发环境就自动完成从需求分析开始的所有的软件开发工作，自动生成供用户直接使用的软件及有关文档。