21世纪高等学校规划教材

RUANJIAN TIXI JIEGOU LILUN YU SHIJIAN

软件体系结构理论与实践

张春祥 主 编
高雪瑶 副主编

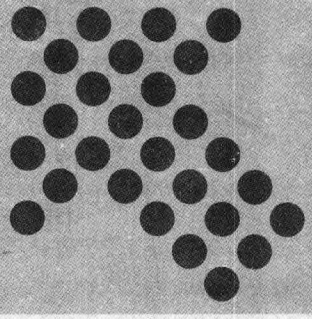
中国电力出版社
CHINA ELECTRIC POWER PRESS

21世纪高等学校规划教材

RUANJIAN TIXI JIEGOU LILUN YU SHIJIAN

软件体系结构理论与实践

主　编　张春祥
副主编　高雪瑶
主　审　卢志茂



中国电力出版社
http://jc.cepp.com.cn

内 容 提 要

本书为21世纪高等学校规划教材。

软件体系结构是从软件设计发展出来的一门新兴学科，目前已经成为软件工程的一个重要研究领域。软件体系结构的目标是为软件开发者提供统一、精确、高度抽象和易于分析的系统信息，合理的框架结构是应用系统开发的重要基础和保障。本书将系统地介绍软件体系结构的基本原理，对软件体系结构的理论知识、发展状况和应用实践进行细致的分析。本书主要包括软件体系结构的研究背景、形式化描述、体系结构风格、评估方法、框架结构的动态演化和产品线开发等内容。本书将采用案例、数据、图示和其他相关材料对知识点进行讲解。通过学习本书的相关内容，读者将对软件体系结构的概念和知识有一个全面的了解。

本书可作为高等院校本科生、研究生及工程硕士相关课程的教材，也可作为软件开发人员的参考书籍。

图书在版编目（CIP）数据

软件体系结构理论与实践 / 张春祥主编. —北京：中国电力出版社，2011.7

21世纪高等学校规划教材

ISBN 978-7-5123-1925-7

Ⅰ. ①软… Ⅱ. ①张… Ⅲ. ①软件－系统结构－高等学校－教材 Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字（2011）第141099号

中国电力出版社出版、发行

（北京市东城区北京站西街19号 100005 http://www.cepp.sgcc.com.cn）

航远印刷有限公司印刷

各地新华书店经售

*

2011年8月第一版 2011年8月北京第一次印刷

787毫米×1092毫米 16开本 17.625印张 427千字

定价 **30.00** 元

前　言

在计算机学科和软件工程学科中，软件体系结构是一个非常重要的研究领域。自从20世纪60年代以来，人们就开始对系统的框架结构进行探索，取得了一些成果，并将其应用于软件开发过程中。然而，计算机和软件正在快速地发展，相关理论也在不断完善，这就需要更新软件体系结构教材的内容，以反映最新的软件开发理论和框架实现技术。

本书比较系统地介绍了软件体系结构的理论知识和实现技术，既兼顾传统的和实用的软件框架开发方法，又包含软件体系结构的最新研究成果，其特点介绍如下。

其一，本书的理论部分，以知识点的形式对重点内容进行了分析和总结，使枯燥的理论内容变得醒目、易于理解。

其二，针对每一种设计模式，给出了一个案例，在学习过程中，可以将理论知识和应用实践紧密地结合起来。

其三，配有电子教案，可以免费从网站 http://jc.cepp.sgcc.com.cn 获取。

全书共分10章，第1章软件体系结构概论，介绍了软件体系结构的概念、发展趋势和应用现状；第2章软件体系结构建模，讨论了体系结构的模型和描述方法；第3章软件体系结构风格，讲解了几种常用软件的框架结构；第4章特定领域的软件体系结构，介绍了DSSA的基本概念、领域工程、应用工程和开发过程；第5章Web服务体系结构，讨论了Web服务技术、面向服务的体系结构、企业服务总线和网格体系结构；第6章软件演化技术，讲解了演化的基本概念、静态演化技术、动态演化技术和演化软件的设计原则；第7章软件产品线，介绍了软件产品线的起源、定义、开发模型和组织结构；第8章设计模式，讨论了创建型设计模式、结构型设计模式和行为型设计模式；第9章软件体系结构评估，讲解了基于场景的评估方法和基于度量的评估方法；第10章云计算，介绍了云计算的概念、体系结构、核心技术、安全保障和应用实例。

本书由多年来一直从事软件体系结构教学工作的教师亲自编写，第1～4章由哈尔滨理工大学张春祥编写，第5～7章由哈尔滨理工大学高雪瑶编写，第8～10章由哈尔滨理工大学李金刚编写。张春祥组织本书的编写，负责全书的策划、编审和统稿。

虽然我们经过精心的准备和调研，也进行了多次修改，但书中还难免存在不足和错误，希望使用本教材的教师和学生提出批评和建议，以便改进我们的工作，提高教材的质量。

编　者

2010年9月

主编联系方式：z6c6x6@yahoo.com.cn

目　录

第1章　软件体系结构概论

1.1　软　件　危　机

20世纪60年代初期，计算机刚刚投入实际应用。由于机器的计算性能与存储性能都很低，因而当时的软件系统往往是针对某个特定应用和特定的计算机进行设计和开发的。此时，所采用的开发技术是与计算机硬件系统密切相关的机器代码或汇编语言。这一时期的软件规模相对较小，很少使用系统化的开发方法，整个开发缺乏必要的管理过程，通常也不遗留任何技术性文档资料。这种软件设计方法往往等同于编制程序，基本上是个人使用、个人设计、个人操作和自给自足式的私人化软件生产方式。人们经常称其为“手工作坊”式的软件开发。

20世纪60年代中期，随着大容量和高速度计算机的出现，计算机的应用范围也随之迅速扩大，其普及程度也不断地提高。此时，人们对软件的需求急剧地增长。操作系统的发展引起了计算机使用方式的革命性变化；高级语言的出现可以使程序员在编程时不再考虑硬件平台；第一代数据库管理系统的诞生可以使人们对大规模数据进行加工与处理。这一时期，软件系统的规模越来越大，复杂程度越来越高，软件可靠性问题也越来越突出。原来的个人设计、个人使用的方式不再能满足要求，软件危机开始爆发。整个软件开发行业迫切需要通过改变软件生产方式来提高软件生产率。

软件危机具体表现在以下几个方面。

（1）对开发成本和开发进度难以进行准确的估计。在开发一个新的软件产品时，经常会出现实际成本远远高于估计成本的情况，同时产品发布时间比预期进度要拖延几个月甚至几年。一旦出现这种现象，将极大地降低软件开发企业的信誉度。许多公司采取了某些权宜之计，来节约开发成本和追赶开发进度，但这将导致软件产品质量下降，产品质量的降低不可避免地会引起用户的不满。

（2）用户对软件产品不满意。软件开发人员和用户之间的交流很不充分，常常在对用户需求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就仓促上阵、匆忙着手编写程序。很多人错误地认为代码和程序就是软件产品，系统功能的划分不是开发人员和用户之间交流的结果，而仅仅是开发人员的奇思妙想。这种闭门造车的开发模式必然会导致最终的软件产品不符合用户的实际需要。

（3）软件产品的质量难以保证。在这一时期，由于软件质量保证技术，如审查、复审和测试，还没有应用到软件开发过程中，因而造成了所开发的软件产品质量较低，软件产品的可靠性较差。

（4）软件产品维护非常困难。由于整个开发过程缺乏必要的管理而且可遵循的文档资料极为有限，因而难以修复应用程序中的错误。修复软件产品缺陷的工作量和软件产品开发的工作量几乎相等。在这种开发模式下，不可能对应用程序进行简单地修改后就将其移植到新的硬件环境中，也不能在原有程序中增加新的功能以满足用户不断变化的需求。

（5）软件产品没有适当的文档资料。软件产品不仅仅是程序，还应该包括一整套文档资

料。这些文档资料是在软件开发过程中产生的，应该是最新的，与软件代码完全一致。这些文档资料对软件开发过程、软件产品的维护及二次开发提供了必要的指导信息，以提高开发与维护人员的工作效率与工作质量。缺乏文档必然给软件的开发与维护带来严重的问题和困难。

（6）软件成本在计算机系统总成本中所占的比例逐年上升。随着微电子技术的快速发展和生产自动化程度的日益提高，计算机硬件成本正在逐年下降。而人们对软件产品的需求、软件产品的规模及软件开发成本却在不断地扩大，而且正在逐年上升。美国在1995年的调查表明，软件成本大约已经占到计算机系统总成本的90%。

（7）软件开发效率的提高远远跟不上计算机应用快速普及的趋势。

通过对软件危机的具体表现进行分析，可以将其成因归纳为以下几个方面：

（1）硬件生产效率的快速提高。目前，计算机的发展已进入一个新的历史阶段，硬件产品已系列化和标准化，实现了即插即用。硬件产品的生产可以采用高精尖的现代化工具和手段，实现了自动成批生产。与过去相比，硬件的生产效率提高了几百万倍，生产能力过剩。这一切都使得高性能计算机的价格有了大幅度地下降，使高性能计算机的普及成为可能。高性能计算机的普及与推广，使人们能够开发规模更为庞大、复杂度更高的软件应用程序来服务用户。

（2）软件产品生产效率较低。伴随着高性能计算机的快速普及，整个社会对应用软件产品的需求也越来越大。在这一时期，软件开发还沿用着“手工作坊”的生产方式，虽然高级程序设计语言和可视化开发工具的出现使得生产效率有了较大的提高，但仍然不能满足用户的需求，生产能力极其低下。

（3）软件供需失衡。随着高性能计算机的普及，整个社会对大规模高质量软件产品的需求也在扩张。由于软件开发过程难以控制，导致生产效率下降、生产成本提高。这二者导致了软件生产的恶性循环，由此产生了软件危机。

（4）用户需求不明确。在软件开发过程中，用户需求不明确的问题主要体现在以下四个方面。在软件开发出来之前，用户自己还不清楚自己到底需要一个什么样的产品；用户对软件开发需求的描述不精确，可能有遗漏、有二义性、甚至有错误。在软件开发过程中，用户不断地提出修改软件产品功能、界面和支撑环境等要求；软件开发人员对计算机专业知识较为熟悉，对应用领域的专业知识了解很少，而用户通常不具有计算机专业知识，却掌握了大量的与应用领域相关的专业知识；由于软件开发人员和用户交流的不充分及交流过程中存在的某些误解，通常会导致开发人员对用户需求的理解与用户本来的愿望之间有一定的差异。

（5）整个软件开发过程缺乏正确的理论指导。软件开发不同于其他工业产品的生产，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。在“手工作坊”式的软件开发过程中，缺乏有力的方法学和工具方面的支持。在软件开发过程中，由于过分地依靠程序设计人员的技巧和创造性，因而加剧了软件产品的个性化，这也是引发软件危机的一个重要原因。

（6）软件产品的规模越来越大。随着软件应用范围的扩大，软件产品的规模也变得越来越大。以美国宇航局的软件系统为例：1963年，水星计划系统包括200万条指令；1967年，双子星座计划系统达到400万条指令；1973年，阿波罗计划系统包含1000万条指令；1979年，哥伦比亚航天飞机系统达到4000万条指令。开发大型软件项目需要组织一定的人力来共

同完成，多数管理人员缺乏开发大型软件系统的经验，各类人员信息交流不及时、不准确，有时还会产生误解，开发人员不能有效地、独立自主地处理大型软件开发的全部关系和各个分支，因此容易产生疏漏和错误，从而导致开发效率下降，难以保证软件产品的质量。

（7）软件产品开发的复杂度越来越高。很明显，随着规模的增加，系统模块与模块之间的关系也变得越来越复杂。无论是规模的增大还是复杂度的提高都会制约软件产品的开发与维护。软件产品的特殊性和人类智力的局限性，导致了人们无力处理“复杂问题”。所谓“复杂问题”的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具来提高软件开发的效率和能力，新的、更大的及更复杂的问题又会摆在人们面前。

1.2 软 件 复 用

复用就是指“利用现成的东西去构造新的事物”，文人称之为“拿来主义”。被复用的对象可以是有形的具体物体，也可以是无形的研究成果。复用不是人类懒惰的表现而是智慧的结晶。因为人类总是在继承了前人成果的基础上，不断地加以利用、改进或创新后才会进步。复用的内涵包括了提高质量与生产率两个方面。由经验可知，在一个新系统中，大部分的内容是成熟的，只有小部分内容是创新的。一般地可以认为，成熟的东西经过成千上万次的修正，总是比较可靠的，具有较高的质量，而大量成熟的工作可以通过复用来快速地实现，以提高生产效率。在作具体工作时，勤劳并且聪明的人应该把大部分的时间用在小比例的创新性工作上，而把小部分的时间用在大比例的成熟工作中，这样才能把工作做得又快又好。这种时间分配方法有利于提高工作效率，便于高质量地完成任务。

将复用的思想应用于软件开发的全过程，便形成了软件复用技术，有时也被称为软件重用技术。目前，世界上已经有 1000 多亿行程序代码，据统计无数功能被重写了成千上万次，这真是极大的浪费。面向对象（Object Oriented）学者经常说的一句口头禅就是“请不要再发明相同的车轮子了”。

所谓软件复用是指利用现有的软件资源来开发新应用系统的过程。其中的软件资源可能是已经存在的软件，也可能是专门用于开发设计且可复用的软件构件。软件复用就是要利用已开发的且对应用有贡献的软件元素来构建新软件系统。它是一项完整的活动，而不仅仅是一个对象。

软件复用不仅要使自己拿来方便，还要让别人拿去方便，即所谓的“拿来拿去主义”。目前，面向对象方法和 Microsoft 公司的 COM 规范都能很好地用于实现大规模的软件复用，是软件复用技术的两个典型代表。

可复用的软件资源即复用成分，是软件复用技术的核心与基础。整个软件复用过程都紧紧地围绕着可复用的软件资源展开。因此，实现软件复用需要解决三个基本问题，一是必须有可以复用的对象；二是所复用的对象须是可用的；三是复用者要知道怎样去使用被复用的对象。相应地，软件复用也包括：软件对象的开发、软件对象的理解和软件对象的复用三个相关的基本过程。复用对象的获取、管理和利用构成了软件复用技术的三个基本要素。复用成分的获取有两层含义：一是将现有的软件成分抽象成为可复用的；二是从复用成分库中选取复用对象，以应用于某个具体问题。

软件复用的前提是存在可复用的软件成分，如果没有复用成分，软件复用也就无从谈起。

在可复用成分中，可复用的软件对象及其说明书是最普通的一种存在形式，也是人们理解软件复用的基础。为了得到可复用的软件对象，从已有系统中抽取和再工程是很自然的手段，也是一种很有发展前途的方式。从已有的应用系统中抽取可复用软件成分的过程，经常被称为“重用再工程”（Reuse Re-engineering）过程。

软件的逆向工程来源于硬件世界。硬件厂商总是想办法搞到竞争对手产品的设计方案和工艺流程，但是又得不到现成的档案。只好拆卸对手的产品并对其进行分析，企图从中获取有价值的东西。很多从事集成电路设计的技术人员，经常需要解剖来自国外的电路板，有时甚至不作分析就原封不动地复制该电路板的板图，然后投入实际生产，并美其名曰“反向设计”（Reverse Design）。从道理上讲，软件的逆向工程与硬件生产非常相似。但在很多时候，软件的逆向工程并不是针对竞争对手的，而是针对自己公司多年前的产品。期望从老产品中提取系统设计、需求说明等有价值的信息。逆工程（Reverse-engineering）是在软件维护过程中，对当前的软件系统进行理解，识别部件和部件之间关系的过程。再工程（Re-engineering）是在软件维护过程中，为了改善系统的性能，使其适应硬件和应用环境不断变化的需求，对原有系统进行再加工的过程。在对软件系统进行再工程之前，首先需要理解现有的应用系统，识别其中的部件和部件之间的关系。因此，再工程过程包括逆工程过程。在软件复用技术中，可复用成分的获取是一项极其重要的工作。通过重用再工程过程，不仅可以解决复用对象的来源问题，而且可以按照严格的筛选标准，获得高质量的可复用部件。重用再工程可以充分发挥已有系统的作用，可以充分挖掘现存系统的潜力。因此，重用再工程是一个很有发展前景的研究方向。

软件重用再工程可以被划分为五个阶段，即候选、选择、资格说明、分类和存储及查找和检索。

（1）在候选阶段，对程序源代码进行分析，根据功能和代码的内聚性，生成若干软件部件集合。每个集合中的元素就是可复用软件对象的候选者。

（2）在选择阶段，将软件部件集合和收集活动组织在一起，通过适当的降耦合、再工程和一般化处理，产生一组可复用软件对象。

（3）在资格说明阶段，对每一个可复用软件对象进行详细地描述，给出模块功能和界面说明书。

（4）在分类和存储阶段，按照分类标准对复用软件对象及其相关说明书进行分类组织，目的是确定中心仓库（Repository），并将生成的可复用软件对象及其说明书存入其中。

（5）在查找和检索阶段，建立用户界面使之与中心仓库进行交互，目的是使查找用户所需求的可复用软件对象的步骤尽可能简化。

1.2.1 软件复用技术的发展

软件复用技术自诞生之日起，其发展经历了以下几个阶段。

（1）1968～1978 年，萌芽和潜伏期。1968 年首次提出了软件复用的概念，希望通过代码复用来满足大规模开发软件系统的需求。设想可以根据通用性、性能和应用平台来对软件对象进行分类。像硬件设计与实现一样，通过组装标准的软件构件，就可以实现复杂的软件系统，这也是构件复用与软件复用的思想雏形。但是经过十几年的应用研究，软件复用技术并未取得实质性的进展。

（2）1979～1983 年，再发现期。1979 年 Lanergan 在论文中对一项软件复用项目进行了

详细地分析，发现设计部分和程序代码中有60%的冗余。如果对重复的部分进行标准化并在新项目的开发中被重新利用，将极大地提高软件开发的效率和软件产品的质量。这一发现使得复用技术重新引起了人们的关注。在这之后，其他学者也通过研究发现：商业、教育和金融等领域的软件系统的大部分逻辑结构和设计模式都属于编辑、维护和报表等类型的模块，可以通过重新设计这些模块并对其进行标准化来获得较高的复用率。

（3）1983～1994年，发展期。1983年，HedBiggerstuffA和AlanPetis在美国的Newport组织了第一次软件复用技术的研讨会，对软件复用的发展前景进行了分析与论证。随后在1984年和1987年，美国IEEE Transactions on Software Engineering和IEEE Software分别出版了有关软件复用的相关学术专集。1991年，在德国举行了第一次软件复用国际研讨会，在1993年又举行了第二次学术研讨会。

（4）1994年至今，成熟期。1994年，软件复用国际研讨会被改名为软件复用国际会议。此时，软件复用技术已经引起了计算机科学界的广泛重视，越来越多的人投入到这一技术的研究中。面向对象技术的崛起给软件复用带来了新的希望，出现了类库和构件等新的复用方式。目前，构件化和复用技术已经成为互联网时代软件开发的新趋势，同样也正被许多国际大型开发企业所采用。软件复用是解决“软件危机”的重要手段之一，也是软件开发技术发展的目的之一。无论是早期的结构化设计方法还是后来的面向对象设计方法都在努力地朝着这个方向发展。

1.2.2 软件复用的实现技术

软件复用技术的出发点是在开发应用系统时不再采用一切从零开始的模式，而是以已有工作为基础，充分利用过去应用开发过程中所积累的经验与知识，诸如：需求分析结果、设计方案、源代码、测试计划及测试案例等，将开发的重点集中于当前应用的特有部分。早期的软件复用主要停留在代码级复用，被复用的知识是程序，如子程序库和函数库等。随着技术的不断发展，复用的范围扩大到领域知识、体系结构、设计方案、技术文档及系统功能等。面向对象语言为软件复用提供了基本支持，并在软件复用领域中逐步发展为主流技术。根据复用对象的类型不同，软件复用可以分为代码复用、设计复用、分析复用及测试复用。

（1）代码复用可分为目标代码复用和源代码复用。其中，目标代码复用级别最低，历史也最久。目前，大部分编程语言的运行支持环境都提供了连接（Link）和绑定（Binding）功能来支持这种复用。源代码复用的级别略高于目标代码复用，程序员在编程时把一些想复用的代码段复制到自己的程序中。但是，这样做往往会产生一些新、旧代码不匹配的现象。大规模地实现源程序复用只能依靠构件库，构件库包含了大量的可复用构件，如对象链接及嵌入技术（OLE）。OLE既能够在源程序级别上定义构件以构造新系统，又能够使这些构件在目标代码级别上仍然是一些独立的可复用构件，能够在运行时刻被灵活地组合到各种不同的应用系统中。

（2）设计复用比源程序复用的级别更高。设计复用受实现环境的影响较小，从而使其被复用的机会更多，并且所需的修改更少。设计复用的途径有三种：第一种途径是从现有系统的设计结果中提取一些可复用的设计元素，并把这些元素应用于新系统的设计过程中；第二种途径是把现有系统的全部设计文档在新的软、硬件平台上重新实现，也就是把一个设计运用于多个具体实现；第三种途径是独立于任何具体应用，有计划地开发一些可复用的设计元素。

（3）分析复用要比设计复用的级别更高。可复用的分析成分是针对问题域的某些事物或某些问题所给出的具有普遍意义的解法。分析成分受设计技术及实现条件的影响较小，所以可复用的机会更大。分析复用的途径也有三种，从现有系统的分析结果中提取可复用的分析成分用于分析新系统；根据完整的分析文档来产生针对不同软、硬件平台和其他实现条件的多项设计结果；独立于具体应用问题，开发专门用于复用的分析成分。

（4）测试复用主要包括测试用例复用和测试过程复用。测试复用无法和分析复用、设计复用、代码复用进行级别上的比较，因为被复用的是一种信息。但是从这些信息的形态上看，大体与代码复用的级别相当。

根据信息复用的方式不同，可以将软件复用分为黑盒（Black-box）复用和白盒（White-box）复用。黑盒复用是指对已有软件构件不做任何修改，直接进行复用，这是最理想的复用方式。白盒复用是指已有软件构件并不能完全满足用户要求，需要根据用户需求进行适应性修改之后才可以使用，这是最常用的复用方式。

软件复用的实现技术一般包括组装和生成两种类型。

在组装技术中，软件构件是复用的基石。以抽象数据类型为理论基础，借用了集成电路中芯片的设计思想，将功能细节与数据结构隐藏封装在构件内部，并给出了精心设计的接口。构件在软件开发过程中类似于硬件中的芯片，通过组装可以形成更大的构件。构件是对某一函数、过程、子程序、数据类型及算法等可复用软件成分的抽象。一般来说，在对构件进行描述时，应该选择一种既不依赖于具体硬件平台也不依赖于具体编程语言的抽象描述语言，否则所开发的构件就会受到机器和语言的限制而使其复用性降低。例如，用某种高级语言编写的构件将会受到该语言数据结构和控制结构的限制，在复用时有可能在其他语言中找不到与之相对应的概念和结构。此外，对大量存在的构件必须采用数据库对其进行管理。利用构件构造软件系统，可以提高开发效率、缩短开发周期。构件不做任何修改只是一种理想的情形，在实际工作中，修改和增删构件是难免的。构件的组合方式主要有三种：连接，通常标准库中的函数是靠编译和连接程序与其他模块结合在一起的；消息传递和继承，在 Smalltalk 面向对象的程序结构体系中，通过消息传递和继承机制把对象和相关的其他对象联系在一起以合成一个新系统；管道机制，在 UNIX 系统中，用管道（Pipe）连接 Shell 命令，将前一命令的输出作为后一命令的输入，用管道机制把多个 Shell 命令连接在一起以实现更为复杂的功能。

在生成技术中，由程序生成器完成对软件结构模式的复用。生成器导出的模式相当于种子，从中可生长出新的专用软件构件，如 Visual C++中的 Wizard 等。生成技术利用可复用模式，通过生成程序产生一个新的程序或程序段，产生的程序即是模式的实例。可复用的模式分为代码模式和规则模式。前者的一个示例是应用生成器，可复用的代码模式就存在于生成器中，通过特定的参数替换，生成抽象软件模块的具体实例。后者的一个示例是变换系统，它是变换规则的集合；在其变换方法中通常采用超高级的规格说明语言，形式化地给出软件需求规格说明，利用程序变换系统，把用超高级规格说明语言编写的程序转化为某种可执行语言书写的程序。这种超高级语言的抽象能力、逻辑性较强，形式化能力较好，便于软件开发者来维护。与构件复用相比，模式复用主要用于某些具体应用领域。构件方法支持自底向上的开发模式，相对应的构件应该具有平台独立性、软件独立性、可读性、可理解性及可修改性。变换方法更侧重于程序的推导方式和推理规则，支持自顶向下的软件开发模式。变换

方法因其形式化程度较高、抽象程度较强，一般的软件开发者不容易掌握这项技术，而且在对某些实际问题进行描述时也存在着巨大的困难。同时经过多次变换得到的可执行程序的执行效率较低，对时间和存储空间的需求也较高，这些都是需要迫切解决的问题。因此，在实际应用中可以将这两种方法结合使用，取长补短，同时吸收人工智能的研究成果，以知识库为辅助工具，促进复用技术的成熟、拓展与深化。

1.2.3 影响软件复用的相关因素

制约软件复用的关键性因素包括软件构件技术（Software Component Technology）、领域工程（Domain Engineering）、软件构架（Software Architecture）、软件再工程（Software Re-engineering）、开放系统（Open System）、软件过程（Software Process）、CASE 技术及各种非技术因素。这些因素互相联系、互相影响，共同影响着软件复用的实现。

软件构件技术是软件复用的核心与基础，是近几年来迅速发展并受到高度重视的一个学科分支。目前，国内外对于软件构件技术的研究已经取得了一定的成果，构件技术的研究正朝着深入和实用的方向发展。构件技术发展的趋势主要表现在从集中式的小粒度组件向分布式的大粒度组件发展和从用于界面制作的窗口组件向完成逻辑功能的业务组件发展这两个方面。

领域工程是为一组相似或相近的应用工程建立基本能力和必备基础的过程，它覆盖了建立可复用软件构件的所有活动。领域工程主要包括领域分析、领域设计和领域实现三个主要阶段。其产品是可复用的软件构件，包含领域模型、领域构架、领域特定语言、代码生成器和代码构件。软件构架是对系统整体设计结构的描述，包括组织结构、控制结构、构件之间的通信、同步和数据访问协议、设计元素之间的功能分配、物理设计、设计元素集成、设计方案的伸缩性和性能及设计选择等。在基于复用的软件开发中，软件构架可以作为一种大粒度、抽象级别较高的元素进行复用，而且为构件的组装提供了基础和上下文，对于成功的复用具有非常重要的意义。

软件再工程是一个过程，它将逆向工程、重用和正向工程组合起来，对现存的系统进行重新构造以获取新的应用系统。再工程的基础是系统理解，包括对运行系统、源代码、设计、分析，以及相应技术文档的全面理解。但在很多情况下，由于各类文档的丢失，只能对源代码进行理解，即所谓的程序理解。

开放系统技术是在系统的开发过程中使用接口的标准，同时使用了符合接口标准的相关实现技术。目前，分布对象技术是开放系统中的一项主流技术，其目标是解决异构环境中的互操作问题。该技术使符合接口标准的构件可以方便地以“即插即用”的方式组装到系统中，实现黑盒复用。

CASE 技术与软件复用密切相关，其主要研究内容包括在面向复用的软件开发过程中，抽取、描述、分类和存储可复用的构件；在基于复用的软件开发技术中，检索、提取、组装及度量可复用的构件等。

非技术因素包括机构组织、管理方法、开发人员的知识更新、知识产权等。

1.2.4 软件复用的意义

复用的最明显优势在于降低总体开发成本，不过成本缩减只是一个潜在的优势，软件复用的其他优势表现在如下几个方面：

（1）增加软件系统的可靠性。在运行的软件系统中，重复使用的构件可靠性比一个新开

发的构件要高。其主要原因是，这些重复使用的构件在各种环境中可能已经得到了实际使用和反复测试。在最初使用这些构件时，构件在设计和实现上的缺陷都已经暴露出来了，设计人员根据缺陷对其进行修正，因而在复用时构件的失败次数将会降低。

（2）降低了软件开发过程中的风险。与开发新构件相比，复用构件的成本不确定性要小得多。软件复用降低了项目成本估计中的不确定性，这是项目管理中的一项重要因素。在开发大型项目时，如果能够大规模地复用软件构件，则开发失败的风险会大大地降低。

（3）加快项目开发的速度。让系统尽快走向市场比降低开发成本要重要得多。复用构件不但可以缩短项目的开发周期而且还能减少产品的验证时间，从而促进了产品的快速成形。

（4）软件复用促进了标准的推广。针对某些标准，诸如用户界面标准，可以实现一组通用的构件。例如可以将用户界面中的菜单实现为一个可复用的构件，无论在开发任何应用系统时，都使用同一菜单构件，在用户面前呈现相同的菜单格式。当用户面对相同的界面时，出现错误的可能性会大大地降低，提高了标准用户界面的可靠性。

（5）专家知识的有效利用。不用专家在不同项目中做重复的工作，而是让他们开发可复用的构件，用这些构件来封装专家的知识与经验。

1.3 软件构件的组织与检索

只有当构件要达到一定的规模时，才能有效地支持软件复用，然而获取大量的构件需要有较高的投入和长期的积累。此外，当构件达到较大的规模时，使用者要想从中找到一个自己需要的构件，并判断其是否与自己的需求相符，却不是一件轻而易举的事情。如何借助已有的构件来有机地、高效地构造新的应用系统，是软件复用技术的一个重要研究课题。为了高效地利用软件构件，必须对其进行有效地分类组织和检索。

有效组织的构件是实现构件高效管理与检索的基础。合理地组织构件的关键在于如何对构件进行有效地分类，从而按照不同的分类模式将构件组织成不同的构件集合。有效地对构件进行分类存储将直接关系到构件库管理、构件查询效率、构件可理解程度及构件可维护性等多个方面。对软件构件进行恰当地分类不仅能简化构件库的管理，而且也能提供更加多样化的检索途径，以提高构件的检索效率。合理的构件分类体系是构件库有效构造、存储、管理及检索的基础，其最终目的是对获取的信息进行正确地分类组织。构件的高效检索将依赖于分类体系的有效建立与实施。

1.3.1 构件的分类

构件的分类体系有很多种，从构件的表示角度出发，可以分为人工智能方法、超文本方法和信息科学方法三大类。在软件复用技术中，信息科学方法是应用较为成功的一种。信息科学方法主要包括枚举、层次、关键词、属性值、刻面（Facet）和本体等几种常见的分类方式，其中刻面分类方法由于能够表达构件的丰富信息，为人们所关注。

枚举分类方法能够将某个领域划分为若干个不相交的子领域，并依次构成层次结构。这种方法能够对领域进行高度结构化的划分，易于理解和使用。由于该方法的分类标准过于严格，使得分类模式难以随着领域的变化而改变，因此所能表示的关系极其有限。此外，枚举结构的创建者必须具有完整的领域知识，因而建立合理的枚举结构是一件非常困难的事情。

层次分类方法的本质是构建类的层次关系。首先把所有的构件划分为一些大类，再对每

一个大类进行细化形成层次较低的小类，不断地重复划分过程，直到层次关系的最低层是构件为止。实际上，层次结构的中间层就是一组构件所属的类。在查询构件时，从高到低逐层判断自己要找的构件应该属于哪一类。使用层次分类方法，可以很快地收敛到所要寻找的目标。但这种方法所存在的主要问题是所采用的分类标准可能只适合一部分构件，而其余构件可能不适合这种分类标准。

在关键词分类方法中，每个构件以一组与之相关的关键词编目。使用主题来描述构件，主题词多为短语。每个主题下可有多个描述子，多为单词。查询者使用关键词来描述所需要的构件，通过关键词与主题词相匹配的手段来寻找目标构件。该方法能够从构件文档中自动地抽取术语，并补充到术语空间中，避免了人工抽取术语所付出的代价。但术语自动抽取的精度往往很低，其抽取结果难以在实际中使用。由于还没有找到准确的术语判别特征，因此自动抽取的结果中往往存在着大量的噪声，对自动抽取的术语需要进行人工辅助判断，工作量仍然很大。

在属性/值分类方法中，为所有构件定义了一组属性，每个构件都使用一组属性/值来进行描述。项目开发人员通过指定一组属性/值，来检索构件库以寻找自己所需要的构件。

在刻面分类方法中，将术语置于一定的语境中，从反映本质特性的视角去精确描述构件。管理者使刻面与术语相互对应，在构件之间建立起一种复杂的关联关系。与其他分类策略相比，刻面分类方法具有易于修改且富有弹性的优点。其主要原因是，在修改某一个刻面时，不会影响到其余的刻面。同时每个刻面对应于一个结构化的术语空间，这使关键词的管理更为方便和有序。

从表面上看，属性/值分类方法与刻面分类方法非常相似，但是二者之间仍然存在着一定的差异。其差异主要表现为：①刻面对应的术语空间通常是有限的不定空间，而属性的值域往往是无限的确定空间；②刻面的选择要比属性的选取更为慎重；③刻面一般不超过 7 个，而属性的数量却没有限制；④属性没有优先级，刻面则可以设置优先级；⑤在属性/值方法中，不能使用同义词，而在刻面分类法中，可以定义同义词关系。

刻面分类法能够从不同角度去描述复杂对象，弥补了关键词分类策略的缺陷，结构上比属性/值分类方法更为合理，开销也小于枚举分类法。同时，刻面分类法非常容易扩展，具有枚举、属性/值和关键词几种分类方法的优点。刻面分类法已经被众多国际组织所采纳，目前是使用得最为广泛的一种构件分类体系。

本体（Ontology）的概念最初起源于哲学领域，它在哲学中的定义是“对世界上客观存在的事物的系统描述”。即存在论，是对客观存在的系统的解释与说明，关心的是客观现实的抽象本质。十几年来，本体的概念和方法已经被成功地应用于计算机领域，如人工智能领域、知识工程、软件复用、数字图书馆、Web 异构信息处理、Web 语义计算及信息检索等。

尽管本体的定义有很多种不同的形式，但是从内涵上来看，不同的研究者对本体的认识都是一致的，都认为本体是特定领域或更广范围内的不同主体（如人、机器或软件系统等）之间进行交流的语义基础。简单地说，本体就是用来描述客观概念之间所存在的关系。本体给出了一种明确定义的共识，该共识主要是为机器提供服务的。目前，计算机只能把文本看成字符串来进行处理，并不能像人类一样理解自然语言中所表达的语义。计算机能够利用本体概念对一类事物进行统一的处理。

Ontology 的作用是捕获相关的领域知识，提供对该领域知识的共同理解，确定该领域内

共同认可的词汇，并从不同层次上形式化定义词汇之间的相互关系。目前，本体的概念主要由概念类、关系、函数、公理和实例五个部分组成。在本体的概念中，最重要的是术语和术语之间的关系，以及组合术语和关系的规则。Ontology 为解决传统信息检索的语义问题和通用问题提出了一条新的途径。在本体概念的支持下，机器能够理解语义，而不是单纯地进行关键字匹配。本体规范了领域词汇的语义，避免了一词多义或多个词语表达同一语义所带来的麻烦。

目前，广泛被使用的 Ontology 资源包括 Wordnet、Framenet、GUM、SENSUS、Mikrokmos、CYC、TOVE（加拿大多伦多大学的研究项目）和 Enterprise Project（爱丁堡大学人工智能应用研究所）。Wordnet 是基于心理语言规则的英文词典，它以 Synsets 为单位来组织信息。所谓的 Synsets 是在特定上下文环境中可互换的同义词集合。Framenet 也是英文词典，采用 Frame Semantics 的描述框架，提供了很强的语义分析能力，目前已经发展为 FramenetII。GUM、SENSUS 和 Mikrokmos 都是面向自然语言处理的，GUM 支持多种语言处理，包含基本概念及独立于各种语言的概念组织方式；SENSUS 为机器翻译提供了概念结构，包含了七万多个概念。Mikrokmos 也支持多语种处理，采用了一种中间语言 TMR 来表示知识；TOVE 本体包括活动、组织、资源、产品、成品和质量等几个部分，可以将它们作为大型企业模型集成的基础。

本体包含了语义信息，使用本体来描述构件，可以让计算机知道用户所描述的词汇的含义。在本体描述的基础上，可以对构件库进行语义检索，检索出含义上相匹配的构件。用构件本体来描述构件的通用语义及构件之间的联系；用领域本体来描述领域相关的语义。这两个本体贯穿了整个检索系统的语义网络。在检索时，将构件本体和领域本体进行概念合成，便于本体库的管理与扩展，类似于关系数据库中多个表格连接形成一个视图。

1.3.2 软件构件的检索与匹配

构件检索是软件复用技术中较为重要的环节。目前，软件复用技术还没有得到广泛的推广和应用，其中的一个主要原因是缺乏有效的检索方法和支撑工具。一般地讲，根据某一查询要求得到的可复用构件往往不止一个。从构件库中选择适当的构件来适应新系统的需求并进行组装，离不开构件的检索与匹配技术。

目前，已有的检索算法大体上可以分为基于人工智能的检索算法、基于超文本的检索算法、图书馆科学和信息科学中所使用的检索算法及基于形式规约的检索算法四类。

在基于人工智能的检索方法中，将领域知识显式地存储在构件库中，构件匹配采用类似于人工智能学科里的推理手段。这种方法基本上还停留在实验室里，在实际应用中很难得到推广和利用。

（1）基于行为采样的构件检索技术。它的基本思想是，利用软件构件的执行能力来检索构件。用输入数据、输出数据及返回类型作为构件的一个采样。对每一个构件，开发者选用一些典型的实际数据作为输入，然后得到该构件的输出数据。

（2）基于知识的构件检索技术。该方法对构件的自然语言描述作词法、句法和语义分析，同时使用知识库来存储应用领域和自然语言的语义信息。基于知识的构件检索的基本原理是根据用户提出的要求，生成系统内部的提问形式，启动推理机获取用户需所求的构件，并以用户易读的形式来显示。常用的知识表示形式包括语义网和框架描述。概念依赖模型采用了框架来描述和检索构件，不过这种方法需要相当丰富的框架资源。Bertrand Ibrahim 等人开发