计算机辅助软件工程

CASE

基础与技术

苏金树 寇国华 吕梅 编著

人民邮电出版社

计算机辅助软件工程
CASE 基础与技术

苏金树　寇国华　吕梅　编著

人民邮电出版社

内 容 提 要

本书是一本介绍计算机辅助软件工程CASE这一高新技术的书籍。它向广大计算机专业人员，特别是软件开发人员详细讲述了软件系统设计的基本理论与方法、软件工程中文档的产生、CASE工具与CASE环境、典型应用系统原型设计、面向对象程序设计……等一系列相关的基础知识与应用技术，并通过实例讲解了CASE环境的应用。理论与实际相结合，使广大读者更易于了解和掌握这一高技术。

计算机辅助软件工程CASE基础与技术

Jisuanji Fuzhu Ruanjian Gongcheng CASE Jichu yu Jishu

◆ 编　　著：苏金树　寇国华　吕梅

责任编辑：张瑞喜

◆ 人民邮电出版社出版发行　北京崇文区夕照寺街14号

北京市密云春雷印刷厂印刷

新华书店总店北京发行所经销

◆ 开本：787×1092　1/16

印张：14.25

字数：347千字　　1997年6月第1版

印数：1—5 000册　　1997年6月北京第1次印刷

ISBN7-115-06332-X/TP·380

定价：19.00元

前　言

近二十年来，随着微型计算机的发展与普及，国内外研制开发了数量庞大、种类丰富的软件，如各种各样的操作系统、开发语言、开发平台以及应用系统。这些软件的推出与应用又大大促进了微型计算机的发展与推广。但是，尽管如此，由于诸多原因，人们仍在惊叹软件的发展远远跟不上硬件的发展，惊呼全世界计算机领域内出现的软件危机。

软件危机的出现与深化，促进了软件开发理论与技术的研究与发展。软件工程理论与技术、数据库理论与实现、面向对象方法与实践都使人们看到了解决软件危机的光明前景。

在诸多的解决软件危机理论技术中，计算机辅助软件工程 CASE 的实现与发展日益引起人们的高度重视，并在国外得到了广泛的应用。我国对 CASE 也非常重视，投入了大量经费，组织了大量人力进行了这方面的研究和开发，并且推出了几个较为成熟的产品。

为介绍 CASE 的理论与技术，我们在实践的基础上编写了这本书。

本书共分为九章。第一章概要介绍了软件危机及 CASE 的历史与发展。第二章讲述了软件系统设计基本理论与方法。第三章说明了软件工程的各个阶段及其应产生的文档。第四章说明了 CASE 工具与 CASE 环境。第五章讲解了在开发中大型软件系统中常用的快速原型法。第六章说明了典型应用系统原型设计。第七章介绍了目前十分流行的面向对象程序设计。第八章进一步说明了 CASE 工具及相关技术。第九章通过两个实例介绍了实用的 CASE 环境。

目　录

第一章 概 述

1.1 计算机软件的基本特性

计算机是近代工业发展的主要技术之一，计算机硬件与计算机软件的迅速发展，并广泛应用于国防、工业、农业、科学研究、教育、文化艺术等领域。同时计算机随着软件的数量不断增长，软件的规模也迅速地扩大。软件已在工业控制系统、商业信息系统、视频处理系统、通信网络系统、运输系统乃至我们日常生活的各种系统中也在发挥着极其重要的作用。

软件是程序清单、图表，以及相关文档的集合。更一般地讲，软件是计算机程序、过程、规则以及相关的文档和所处理的数据。对用户而言，主要关心软件产品的形式及其自身质量。应用软件主要由向用户发行的与软件系统相关的程序包、文档以及必要的附加说明组成，但是不包括开发者的测试实例、内部文档，以及用于开发应用的其它软件工具。

软件作为过程比较难以准确定义，这是因为到目前为止，软件开发尚没有完善的开发理论或者公理基础。就象古代建筑师不懂牛顿定律，依然可以建起古代文明一样，软件开发人员在长期从事软件开发过程中积累了大量的丰富经验。从 60 年代末开始，西方计算机界从日益严重的软件危机中吸取了经验教训，认真研究了解决软件危机的软件开发的理论和方法。在此基础上逐步总结，并以系统的观点对待软件的开发、运行、维护等，由此逐步形成了软件工程这门计算机学科的一个崭新分支。

软件工程实际上就是软件作为过程的实践，该理论注重于软件开发过程，也就是将用户的需要转换为软件需求，再将软件需求转换为概要设计和详细设计。设计完成后，在进入编程阶段，将设计的结果转换为代码并对代码加以测试、运行以及维护，在这一过程中同时产生相应的文档。

1.2 软件的自然属性及内含分类

在对软件开发进行深层次探讨之前，有必要先充分理解软件开发的难度，从软件的特性可以了解其内在的失败风险。

1.2.1 软件自然属性

软件开发的难度涉及许多因素，主要有：软件复杂性、软件精确性、软件的不可见性、软件的变迁性等。这些因素或大或小地在某种程度上引起对问题理解、说明、设计、实现及其管理等多方面的难度及复杂性。

1. 软件复杂性

软件是相当复杂的。即使数百行的小程序也可以复杂到难以穷尽所有测试路径的程度。随着系统增大，复杂性不断增加，系统内部相互作用也随之增加，从而很难孤立地测试各个部分。加上软件内部相互作用的方式多样性，可能遇到的问题更加难以预测，使得测试全系统将更加困难。许多软件产品虽然声明已经经过非常严格的测试，但在实际运行过程中仍然发现有重大问题，并屡见不鲜。

正是由于软件的复杂性，所以充分理解一个软件系统十分困难。即使精确地用文档描述一个小型的软件系统，也需要费以时日。

软件开发人员总是试图寻找一些方法，以减少软件系统的复杂性。如采用有向图或其它特殊符号，使得其表示易于被人们所理解，并以此设计、使用复杂的系统。不过，减少复杂性意味着承担过于简化的风险，并由此可能造成某些关键概念被忽略或者被误解。

2. 软件精确性

软件是精确而具有逻辑的，这与容许一定差错的传统工程概念不同。传统工程中在一定的误差之内仍可以满足要求，而程序设计中没有与容错相似或对应的概念(这里所指容错的概念与计算机系统的容错概念不同)。程序中单一的错误，不只使软件性能的降低，而且可能引起软件系统的彻底失败(用计算机的术语来讲，称为系统崩溃)。程序必须符合程序设计的标准，语言的实现必须符合内在硬件的要求。系统相互通信时，必须采用统一的协议标准信息，即定义通信协议。

3. 软件不可见性

软件是不可见的。软件系统的内部工作机制是隐含的，只能根据它们与外部世界的接口来想象。这样就难以衡量软件开发工程的进展及其质量。例如，不能准确地说系统的85%是完整的。由于软件的不可见性，即使规范地编写文档，也难以全面地反映系统的全部真实性。软件不是一种物理的产品，难以用软件之外的事物模拟。

由于系统表示的困难性，使得其在某些关键的概念上难以与其他人交流，由于缺少观察系统的手段，所以软件开发人员所开发的系统必然面临着不能满足客户需求的风险。

4. 软件变迁性

软件易于改变，无论其功能或复杂性如何，都存在功能的改进和故障的排除问题。从某种意义上讲，一万行程序修改一行和十行程序修改一行没有太多的区别。共同的一点是，修改后的软件系统并不意味着依然能够正确执行。随着条件的变化，修改(或称维护)软件也是不可避免的。基于这些变化难以控制的原因，往往需要引入变化控制机制。

软件开发不是一个封闭的系统，驱动变化的外部力量总是存在的。并非所有变化都是人们希望的，有些必须坚持原有方案，不能改变，否则将导致不良后果。有些却是不可忽略的，因为视而不见，不加修改就有可能引起系统过时的风险。

软件开发与建筑工程设计建造有许多类似之处，都具有以下基本步骤：

· 工程概要分析与设计(房屋结构草图)；
· 工程造价及规划(工程预算)；
· 制定规范(建造计划)；
· 实现(建造过程)；
· 验收与提交(工程交工)。

1.2.2 应用软件的内在分类

一个应用软件与另一个应用软件可能有许多不同之处，但应用软件总是由一些典型的语句种类构成。应用软件的内在语句种类主要有以下九种：

(1) 模型计算。例如图形变换，应用模型计算、数据库计算等；

(2) 用户输入。与用户交互，以便得到用户的输入，可以包括简单的或复杂的交互过程，如输入数据的语法检查，语义检查，并将数据输入到程序定义的数据结构中；

(3) 输出。例如打印或显示计算结果；

(4) 控制。例如比较、循环、分支等程序逻辑；

(5) 帮助信息。如果用户请求帮助，根据用户的请求显示适当的帮助信息；

(6) 错误处理。例如在输入、输出、计算或通信过程中出现错误时，显示错误信息，并加以恢复；

(7) 数据移动。将数据从一个数据结构移到另一个数据结构。分类、搜索、格式化是准备把数据作进一步处理的移动动作；

(8) 数据说明。例如在 C 语言中，使用 struct 等说明数据结构；

(9) 注解。提供清楚、精确、说明性的注解。

粗略一看，读者可能产生这样一种想法，即上述各个部分可以分别设计并实现，然后再将其集成在一起形成一个程序。这个想法是好的，遗憾的是“设计”、“实现”及其“集成”是非常复杂的。系统的复杂性取决于多种因素。如用户界面、分支控制、语句嵌套层次、数据结构类型以及应用时难以理解等。

1.3 软件危机

1.3.1 什么是软件危机

从本世纪六十年代末开始，人们越来越多地发现软件落后于硬件的发展，这主要表现在：

· 硬件的生产手段已进入现代化、自动化、工程化、质量高、数量大的阶段，硬件产品不断问世。而软件的生产仍然使用落后的手工作坊式，以个人智力劳动为主的生产方式；

· 软件产品的供小于求。特别是高质量的软件更不能满足需要；

· 有关计算机硬件理论与技术的研究成果远远超过软件，这就形成了硬件更新换代的速度远远超过软件；

· 软件生产的成本相对于硬件的生产成本越来越高。在计算机发展初期，一个计算机系

统的投资百分之九十以上用于硬件。但是随着硬件生产的迅速发展，成本越来越低，而软件的投资则越来越高。前几年在发达国家，甚至达到了硬件投资的几倍之多！

·硬件的某一部件的故障一般不会影响整个计算机系统的运行，而数十万条指令中一条或多条指令的错误则可能导致整个系统的瘫痪。硬件发展使得其可靠性越来越高，而软件生产随着其规模的增大可靠性越来越低。由于计算机的普遍应用，特别是国家的军事、政府、安全、金融等部门都在使用计算机及其软件，需要非常可靠的计算机系统，而对软件开发而言，依据目前的技术是难以做到的。所以，国际上某些专家曾经悲观的认为，现代化从一定意义上来讲是计算机化，但这则意味着不可靠和不稳定。这些悲观论者甚至预言，人类发明了计算机，但导致人类毁灭的也将是计算机。这种论调虽然是杞人忧天，但至少从一个方面说明了软件是如何的脆弱和易受攻击。

正是由于上述现象，人们惊呼“软件危机”。所谓软件危机，可以归纳为：

(1) 软件生产的高成本、低效率；

(2) 软件自身的不可靠。

就一个具体的软件工程项目而言，软件危机表现在：

(1) 开发周期过长而且很难估计其中所有不稳定因素。当提出并论证一个软件工程项目时，人们往往看得过于简单，但是一旦开始实施，问题将会接踵而来。例如：选择哪一种设计方法和技术，选择哪一类型的硬件产品，如何将任务分解并落实到每个开发人员，如何保证项目如期完成等等。其结果是开发周期不断延长，投入经费不断增多，甚至很难有人能说清楚最终完成的时间及其实现的全部功能。更不会有人可以保证所开发的系统是天衣无缝，无懈可击的。

(2) 测试软件十分困难。如果说开发一个软件系统是困难的，那么测试一个软件系统更为困难。这是由于软件测试理论和技术非常落后。开发一个适应所有软件的测试系统几乎是不可能的。而开发通用软件测试系统的难度要比开发软件系统本身的难度大得多，这是由于测试系统不允许有任何错误和疏漏。所有的测试系统可以检测出软件的错误，但却不能断定经过测试的软件不再有错误。换言之，经过测试的软件也不是万无一失的软件。

(3) 项目的复杂程度与可靠程度成反比。可以想象，所开发的系统越复杂，其可靠程度就越低。假如平均一千个模块中有一个错误，那么十万个模块就可能有数百个错误，这是因为软件的错误可以蔓延。

(4) 软件开发人员的水平直接影响开发的效率以及可靠性。由于软件开发主要依靠每个软件开发人员的脑力劳动，软件开发人员的水平高低、责任心强弱将直接影响到整个系统的开发。

(5) 软件维护十分困难。所谓软件维护是指改进软件的功能，解决软件存在的错误或故障。

在系统投入运行一段时间后，增加一些功能，去掉一些功能是很正常的。而在系统运行过程中，会发现一些这样或那样的一些问题也是很正常的，因此，软件维护是必不可免的。但是：

·无论是改进功能还是纠正错误，都要依靠专业人员，尤其是参与该系统开发的人员。而这些人员往往在系统交付后又去承担其它项目；

·由于系统本身的复杂性，很难有人能够非常熟悉整个系统的所有软件，这就又可能造

成“动一发牵全身”。即一个问题尚未解决，又产生出十个新的问题。因而，软件维护又是十分困难的。

据国外统计就软件生命周期的各个阶段而言，在所投入的人力、物力的比例中，软件维护占到了 67%！

下面几个数字反映了软件发展所面临的重重困难：

- 软件费用在整个计算机应用系统中所占比例已超过百分之九十；
- 软件研制费用的百分之七十以上用在测试以及维护上；
- 软件费用的百分之六十七以上用在了产品的使用期(维护和支持)；
- 美国国防部 1979 年投资 6.8 百万美元的软件项目，但是能提交使用的不到百分之二十五。

下面几个实例可以看出软件的脆弱性，以及一旦出现问题后的严重后果：

- 1979 年第三次世界大战几乎爆发。当时一部失灵的计算机发出了攻击指令，并且差五分钟就要向原苏联发射导弹了；
- 美国科罗拉多河 1983 年出现灾难事件，当时计算机对堤坝的挡水水位计算错误，造成洪水泛滥，数以百万计的财物受损；
- 1989 年，美国政府官员尴尬地承认：计算机故障令 6.5 亿美元的学生贷款丢失；
- 1983 年，在美国旧金山海域一次军事演习中，计算机的错误指令使一门美国海军大炮差点击中一艘墨西哥货船；
- 八十年代初期，美国温哥华股市因计算机故障，导致近两年内每个交易日平均指数低报 1 点，累计损失 574 点，引致经济混乱；
- 1979 年新西兰一民航飞机失事，机上 257 人全部罹难，其原因是计算机的软件错误。

据专家统计，世界上平均每个月发生一起计算机引起的严重事故，而其中又绝大多数是因软件引起的！可见软件维护确实是十分困难的。

1.3.2 产生软件危机的原因

产生软件危机的原因主要有：

(1) 缺乏训练有素的人材。缺乏训练有素的人材可以归结为能力与技巧的缺乏，以及混乱的管理。典型的问题是：

- 缺乏良好的技能；
- 工程期间更换人员；
- 有些人岗位不适当；
- 人员目标与项目目标冲突；
- 小组成员无法沟通；
- 工程采用一些最容易得到的人，而不是采用具有适当技巧及技能的最合适人选。

(2) 不实际的进度及经费预算。所有项目的共同约束是进度及经费。进度及经费通常由主管领导来决定。如果确定了不切合实际的目标，工程管理人员必须让主管了解这一点。典型的问题有：

- 经费不足；
- 不实际的进度；

· 经费经常变动；

· 进度经常变动；

· 安排的项目缺乏优先次序。

(3) 开发错误的软件功能。如果用户的要求不能准确、清楚地得到理解，就有可能实现错误的软件功能。如果不尽早发现这些问题，大部分代码或全部代码最终都有可能作废。典型的问题有：

· 系统要求含糊不清；

· 没有充分理解系统要求；

· 希望改变系统要求；

· 系统要求与系统设计冲突；

· 系统功能说明没有按时完成；

· 开发错误的用户界面。

(4) 实现用户不满意的用户界面。这就如同实现错误的软件功能一样，这是软件开发中经常遇到的问题。错误的用户界面对软件成功有潜在的危害。典型的问题是：

· 不友好的用户界面；

· 没有充分理解用户界面要求；

· 用户经常希望改变界面要求；

· 用户界面没有及时提出；

· 缺乏用户参与界面设计；

· 程序员使用的概念模型与用户的概念模型不同。

(5) 目标功能不严格。系统开发中往往出现易于理解的东西刻划得过细，或者程序员集中精力于感兴趣而与系统无关的事情上。典型的问题有：

· 用户提出的不必要功能；

· 开发者实现了不必要功能；

· 由于上述原因而遗漏了重要的功能；

· 系统要求性能低，而实现性能高。

(6) 需求不断变化。需求完全固定而没有变化是极少的，工程项目不考虑变化最终会导致系统过时，相反，如果工程天天都在变化，将导致工程无法正常进行。典型的问题有：

· 缺乏需求变化的基本准则；

· 变化过于频繁或者失控；

· 需求变化要求不严格；

· 对需求变化所带来的影响缺乏估计；

· 没有配置管理系统；

· 工程主管或上级主管中途换人，造成某些人为的变化。

(7) 缺少外部软件资源。软件工程项目中往往需要一些外部的软件资源，如工具、编译或其它商用软件。这些一般由第三方提供。由于不受项目内部的管理，所以可能出现不能满足要求的问题，主要问题是：

· 软件资源不能满足任务的要求；

· 未能按时得到商用软件；

· 外部软件资源性能、可靠性和功能较差；

· 缺乏商用软件的支持。

(8) 协作完成的任务不及时。协作任务如工程评价等。在软件完成后，文档及培训如果不及时或质量差都可能带来问题。如果协作任务出现在工程的关键路径上，由此而引起的问题或延迟可能造成后续任务的延迟，使整个工程有不能按时交工的风险。主要问题有：

· 不可预测的造价；

· 没有按确定的进度表完成任务；

· 任务质量达不到要求；

· 依赖于不存在的产品。

(9) 实时性能差。工程项目很少有严格的实时性能要求，对那些确实需要的项目，则不能有丝毫的怠慢。部分原因可能是开发者对硬件要求过低，部分原因可能是用户对性能要求过高。例如：

· 不能确定性能对设计的影响；

· 性能要求过多，没有进行投入产出分析；

· 没有充分理解苛刻的实时性能要求。

(10) 计算机科学的应用局限性。计算机在某些领域的应用是非常困难的，或者正在研究之中。除非绝对必要，否则不要涉足这些高风险领域。这些领域的应用软件特点是工程造价非常昂贵，例如：

· 分布式系统；

· 严格的实时系统；

· 高可靠性系统及容错系统；

· 严格的保密系统。

综上所述，产生软件危机的原因可以概括为：

(1) 落后的开发理论；

(2) 落后的开发技术；

(3) 落后的开发环境。

1.3.3 解决软件危机的途径

软件危机看起来是计算机应用及其发展过程中必然出现的一大问题，特别是计算机硬件技术发展远远超过了软件技术的发展，以及人们对高度可靠、高度复杂、高度灵活的软件需求日益迫切的情况下，软件危机显得更加突出。软件危机在发达国家十分普遍，同样在发展中国家已成为计算机应用的最大障碍。换言之，软件危机不是一个或几个国家的问题，而是一个国际问题，只不过在不同国家，不同地区的表现形式或严重程度有所区别罢了。

如前所述，产生软件危机的原因很多。从根本上来讲，是与计算机硬件更新换代的速度相比较，软件开发的理论与技术仍然是几十年一贯制。尽管有这样或那样的改进，但却无本质的变化，更谈不上更新换代。如果说，硬件生产早已进入现代化、自动化、工程化的阶段，那么软件生产仍然使用手工作坊式的以个人智力劳动为主的开发方式是不行的，人们认为，应当像生产硬件产品一样，研究软件生产的理论以及开发的方法。

具体来讲，解决软件危机的途经在于：