教育部高职高专规划教材
Jiaoyubu Gaozhi Gaozhuan Guihua Jiaocai

高职高专 现代信息技术系列教材

软件工程

张海藩 编著

人民邮电出版社
POSTS & TELECOM PRESS

教育部高职高专规划教材

高职高专现代信息技术系列教材

软 件 工 程

张海藩 编著

图书在版编目（CIP）数据

软件工程/张海藩编著. —北京：人民邮电出版社，2003.7
ISBN 7-115-11258-4

Ⅰ.软… Ⅱ.张… Ⅲ.软件工程-高等学校：技术学校-教材 Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字（2003）第 034405 号

内 容 提 要

本书总结了编者多年来从事软件工程教学与研究的经验，并吸取了国内外众多同类教科书的精华。

本书共 7 章。第 1 章概述软件工程与软件过程；第 2 章讲述结构化分析的任务、过程、方法和工具；第 3 章讲述结构化设计的任务、准则、方法和工具；第 4 章着重介绍几种常用的测试技术；第 5 章讲述面向对象的概念、模型、分析、设计与实现；第 6 章讲述软件维护；第 7 章讲述软件项目的计划、组织和质量保证，并简要地介绍了能力成熟度模型。

本书的特点是：讲解深入浅出，通俗易懂，便于自学；把丰富的实例与原理性论述紧密配合，着重讲透基本的概念、原理、技术和方法；特别注重实用性，用几个综合性实例概括了本书的主要内容。认真阅读这些实例，不仅对读者深入理解软件工程很有帮助，而且有助于读者学会把软件工程的理论与技术运用到实际工作中去，这些实例还可作为上机实习的材料。

本书可作为大学专科或高等职业技术学院软件工程课程教材，也可作为大学本科相应课程的教学参考书。

教育部高职高专规划教材
高职高专现代信息技术系列教材

软件工程

◆ 编　　著　张海藩
责任编辑　潘春燕

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
读者热线　010-67194042
北京汉魂图文设计有限公司制作
北京朝阳展望印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：20.5
字数：493 千字　　2003 年 7 月第 1 版
印数：1-6 000 册　　2003 年 7 月北京第 1 次印刷

ISBN 7-115-11258-4/TP・3443

定价：27.00 元

本书如有印装质量问题，请与本社联系　电话：（010）67129223

高职高专现代信息技术系列教材

编 委 会 名 单

丛书前言

江泽民总书记在十五大报告中提出了培养数以亿计高素质的劳动者和数以千万计专门人才的要求，指明了高等教育的发展方向。只有培养出大量高素质的劳动者，才能把我国的人数优势转化为人力优势，提高全民族的竞争力。因此，我国近年来十分重视高等职业教育，把高等职业教育作为高等教育的重要组成部分，并以法律形式加以约束与保证。高等职业教育由此进入了蓬勃发展时期，驶入了高速发展的快车道。

高等职业教育有其自身的特点。正如教育部“面向21世纪教育振兴行动计划”所指出的那样，“高等职业教育必须面向地区经济建设和社会发展，适应就业市场的实际需要，培养生产、管理、服务第一线需要的实用人才，真正办出特色。”因此，不能以本科压缩和变形的形式组织高等职业教育，必须按照高等职业教育的自身规律组织教学体系。为此，我们根据高等职业教育的特点及社会对教材的普遍需求，组织高等职业学校有丰富教学经验的老师，编写了这套《高职高专现代信息技术系列教材》。本套书已纳入教育部高职高专规划教材。

本套教材充分考虑了高等职业教育的培养目标、教学现状和发展方向，在编写中突出了实用性。本套教材重点讲述目前在信息技术行业实践中不可缺少的、广泛使用的、从业人员必须掌握的实用技术。即便是必要的理论基础，也从实用的角度、结合具体实践加以讲述。大量具体操作步骤、许多实践应用技巧、接近实际的实训材料保证了本套教材的实用性。

在本套教材编写大纲的制定过程中，广泛收集了高等职业学院的教学计划，调研了多个省市高等职业教育的实际，反复讨论和修改，使得编写大纲能最大限度地符合我国高等职业教育的要求，切合高等职业教育实际。

在选择作者时，我们特意挑选了在高等职业教育一线的优秀骨干教师。他们熟悉高等职业教育的教学实际，并有多年的教学经验；其中许多是“双师型”教师，既是教授、副教授，同时又是高级工程师、认证高级设计师；他们既有坚实的理论知识，很强的实践能力，又有较多的写作经验及较好的文字水平。

目前我国许多行业开始实行劳动准入制度和职业资格制度，为此，本套教材也兼顾了一些证书考试(如计算机等级考试)，并提供了一些具有较强针对性的训练题目。

对于本套教材我们将提供教学支持(如提供电子教案等)，同时注意收集本套教材的使用情况，不断修改和完善。

本套教材是高等职业学院、高等技术学院、高等专科学院教材。适用于信息技术的相关专业，如计算机应用、计算机网络、信息管理、电子商务、计算机科学技术、会计电算化等。也可供优秀职高学校选作教材。对于那些要提高自己应用技能或参加一些证书考试的读者，本套教材也不失为一套较好的参考书。

最后，恳请广大读者将本套教材的使用情况及各种意见、建设及时反馈给我们，以便我们在今后的工作中，不断改进和完善。

编者的话

软件工程是指导计算机软件开发与维护的一门工程学科。经过30多年的研究与发展，软件工程正逐步走向成熟，应用日益广泛。目前，业内人士已经普遍认识到，如果一个软件项目不用软件工程来指导，则必然会受到实践的惩罚。

计算机专业的学生毕业之后，无论从事软件开发、维护还是销售，都离不开软件工程的知识。可以说，软件工程概论课是他们参加工作后马上就要直接应用的一门专业课。但是，在校学生由于缺乏从事软件项目的实践经验，往往认识不到软件工程的重要性，更难于掌握软件工程的精髓。

为帮助读者更好地学习、掌握软件工程，本书首先从具体的例子谈起，强调指出开发软件不等于编写程序，学习和运用软件工程非常必要。然后用丰富的实例与原理性论述紧密配合，讲透软件工程的基本概念、原理、技术和方法，在讲完每个重点专题之后，都用一个综合性的实例概括全章内容。认真阅读这些综合性实例，不仅对读者深入理解软件工程很有帮助，而且有助于读者学会把软件工程的理论与技术运用到实际工作中去。

本书共7章。第1章概述，讲述软件生命周期各阶段的基本任务，全面概括地介绍软件工程这门学科以及典型的软件过程模型。第2章结构化分析，讲述结构化分析的任务、过程、方法和工具，最后讲述了一个实际软件(工资支付系统)的结构化分析过程。第3章结构化设计，讲述软件设计的任务、准则和启发规则，介绍典型的设计方法和工具，最后讲述了一个实际软件(汉字行编辑程序)的结构化设计过程。第4章结构化实现，讲述编码和测试，重点介绍了几种常用的测试技术。第5章面向对象方法学导论，首先通过一个面向对象程序设计实例引入面向对象的基本概念和方法，然后系统地讲述面向对象的概念、方法和模型，并用一个综合性的实例(C++类库管理系统的面向对象分析与设计)概括了本章的主要内容，最后简要地介绍面向对象实现及面向对象方法学的主要优点。第6章软件维护，讲述软件维护的定义、特点和过程，讨论影响软件可维护性的主要因素，并简要地介绍预防性维护。第7章软件项目管理，讲述软件项目的计划、组织和质量保证，并简要地介绍国际上流行的能力成熟度模型。

在本书的编写过程中，牟永敏博士曾对本书编写大纲提出过一些有益的建议，张劲松和张展新用VC++实现了书中讲述的C++类库管理系统，张雯和张杰为本书的出版做了许多具体工作，谨在此向他们表示感谢。

编者

2003年3月

目　录

第 1 章　概述

开发软件是不是就是编写程序？怎样才能以较低的成本开发出高质量的软件？为了开发出一个符合用户需要的软件，应该完成哪些工作？本章将详细地讨论并概括地回答上述这些问题。

1.1　开发软件不等于编写程序

1.1.1　开发软件应该完成的工作远远多于编写程序应该完成的工作

开发软件是否就是编写程序呢？为了回答这个大家关心的问题，让我们来考察两个具体例子。

假设讲授 C 语言程序设计课的老师给学生布置了一道作业："编写一个程序，读入圆半径、三角形的底边长和高、矩形的两条边长，计算圆形、三角形和矩形的面积。"怎样编写这个程序呢？因为在中学数学课上已经学过计算上述几种图形面积的公式，这道作业看起来比较容易完成。一些同学可能先在主函数 main 中说明几个浮点型的变量，分别用于保存圆半径、三角形底边长和高以及矩形两条边长的值，然后用输入语句读入原始数据，最后用已知的公式计算并用输出语句输出圆形、三角形和矩形的面积值。而一些对结构程序设计技术掌握得比较好的同学，可能先定义三个函数，它们的功能分别是计算圆面积、计算三角形面积和计算矩形面积，然后在主函数 main 中调用这三个函数。一些细心的同学还可能想到，在计算面积之前应该先对输入的原始数据进行校核，如果是非法数据（例如，负数），则提醒用户重新输入。

从上面这个例子可以看出，由于对程序的需求很明确，编写程序所应完成的工作主要是设计算法（即完成指定功能的步骤），然后用程序设计语言（例如，C 语言）表达该算法。

下面让我们再通过一个具体例子考察开发软件所应完成的工作。

小王是计算机应用专业的一名毕业生，毕业后在一所职业高中工作，分配给他的工作是为该校开发一个工资支付软件，以便每月运行该软件生成当月的工资明细表和相应的财务报表。怎样完成这项软件开发任务呢？

在校期间，小王的 C 语言程序设计课是全班学得最好的，为了尽快完成任务，他想立即动手用 C 语言编写工资支付程序。但是，一旦开始编写程序，他才发现问题并不像他想象的那样简单。怎样计算每个人的工资呢？教师的工资怎样计算？干部的工资怎样计算？工人的工资怎样计算？工资明细表中还应该包含哪些内容？会计科需要哪些财务报表？每张报表中包含哪些条目？……上述种种问题小王都不能确切地回答。然而不能准确地回答这些问题就根本无法编写程序。为了搞清楚这些问题，必须向工资支付系统的用户（即会计）虚心请教，

也就是说，必须不厌其烦地做深入的调查研究工作，以便准确具体地了解用户对软件的需求。通常把通过调查研究向用户了解需求的工作过程称为需求分析。

是否知道了用户对软件的需求之后就可以立即动手编写程序了呢？回答仍然是否定的。为了开发一个软件，所需编写的程序规模通常都很大，必须先经过精心的设计过程，依据设计好的方案才能编写出合格的程序。这就好像为了盖一栋大楼必须先经过设计过程，画出蓝图，然后施工人员严格按照蓝图施工才能盖出高质量的大楼。盖大楼与搭一个简易的小棚子完全不同，搭小棚子可能无须设计，一边搭一边想也可以，盖大楼则必须先进行精心的设计，否则必垮无疑。同样，编写一个小程序可以边干边想，而编写一个大型程序则必须先进行设计，边干边想必然陷入混乱而不可收拾。

事实上，为了盖一栋大楼，工程师必须首先知道大楼的用途，并且事先进行地质勘探，以探明该地的地质条件是否适合盖这么高的大楼，与此同时有关部门则从环保、消防等方方面面审查是否可以在该地区盖这样的大楼，这些都属于可行性研究的范畴。如果在该地区可以盖这样的大楼，工程师则进一步深入地了解用户对该大楼的具体需求，以便以这些需求为依据进行设计。类似地，为了开发一个软件，软件工程师必须首先弄清楚要用这个软件解决用户的什么问题（这个过程称为问题定义），以便明确该软件的“主攻”方向。接下来往往需要进行可行性研究，以便弄清楚用户的问题是否有行得通的解决办法，如果没有可行的解决方案就鲁莽地着手开发软件，必然造成大量资源的浪费。通过了可行性研究之后，还必须进行需求分析和设计，然后才能编写程序。

众所周知，大楼盖好之后需要经过严格的检验过程，仅当大楼确实符合用户需求而且质量合格时，才能交付用户使用。软件开发与此类似，程序编写出来之后，也必须经过严格的检验过程（称为测试），软件符合用户需求而且质量合格，才能交付给用户使用。

从上述论述可知，开发软件并非就是编写程序，事实上编写程序仅仅是开发软件所应完成的工作的一部分，而且只占一小部分。为了开发出一个符合用户需要、质量合格的软件，软件工程师必须首先弄清楚用户面临的问题是什么，也就是要明确软件的“主攻”方向；接下来应该进行可行性研究，分析用户面临的问题是否有行得通的解决方案，为避免浪费资源，仅在该软件的开发是可行的前提下，才进行实质性的开发工作；然后应该进行需求分析工作，通过与用户的反复交流，搞清楚用户对该软件的具体需求，这些需求是进行软件设计的依据；在编写程序之前需要先进行设计，通常，大型软件的设计工作又分成两个阶段进行，先进行总体设计（又称为概要设计），再进行详细设计；编写程序实质上是把设计结果翻译成用某种程序设计语言书写的程序；程序编写出来之后，还需要经过严格的测试过程，软件确实符合用户需求而且质量合格，才能交付给用户使用。

此外，规模较大的软件都是由多人分工协作开发出来的，为了使开发人员相互之间能准确地交换信息，同时也为了在软件交付给用户使用期间能比较容易地修改或扩充，必须把软件开发全过程中各个阶段的工作成果用文字、图表等适宜的形式准确地记录下来（通常把这些记录称为文档）。事实上，软件是由程序、数据和相关的文档组成的。从软件的构成可以看出，程序仅仅是组成软件的一个成分，从这个角度说，开发软件也绝不等于编写程序。

1.1.2 错误做法导致软件危机

刚才我们通过两个具体例子说明了开发软件不等于编写程序，但是，迄今为止，仍然有

不少人错误地认为开发软件就是编写程序，或者认为开发软件主要就是编写程序。人们之所以有错误的认识并在开发软件时采用了错误的做法，主要可归因于在计算机系统发展的早期阶段“开发软件”的个体化特点。

在计算机系统发展的早期（20 世纪 60 年代中期以前），通用硬件已经相当普遍，软件却是为每个具体应用而专门编写的，大多数人认为开发软件是不需要预先计划的事情。这时的“软件”实际上就是规模较小的程序，程序的编写者和使用者往往是同一个（或同一组）人。由于规模小，程序编写起来相当容易，当时既没有什么系统化的软件开发方法，也没有对软件开发工作进行任何管理。这种个体化的软件环境，使得软件设计往往只是在人们头脑中隐含进行的一个模糊过程，除了程序清单之外，根本没有其他文档资料保存下来。

从 60 年代中期到 70 年代中期是计算机系统发展的第二代时期，这个时期的一个重要特征是出现了“软件作坊”，广泛使用产品软件。但是，“软件作坊”基本上仍然沿用早期形成的个体化软件开发方法。随着计算机应用的日益普及，软件数量急剧膨胀。在程序运行时发现的错误必须设法改正；用户有了新的需求时必须相应地修改程序；硬件或操作系统更新时，通常需要修改程序以适应新的环境。在软件已经交付给用户使用之后，为了改正错误或满足新的需求而修改软件的过程，称为软件维护。上述种种软件维护工作，以令人吃惊的比例耗费资源。更严重的是，许多软件的个体化特性使得它们最终成为不可维护的。“软件危机”就这样开始出现了！

所谓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正常运行的软件才具有的，实际上，几乎所有软件都不同程度地存在这些问题。

概括地说，软件危机包含下述两方面的问题：怎样开发软件，以满足人们对软件日益增长的需求；如何维护数量不断增加的已有软件。

具体说来，软件危机主要有以下一些典型表现：

- 对软件开发成本和进度的估计常常很不准确。实际成本比估计成本高出几倍甚至十几倍，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。
- 用户对“已完成的”软件系统不满意甚至拒绝接受的现象经常发生。
- 软件产品的质量往往靠不住。
- 软件常常是不可维护的。很多程序中的错误都非常难以改正，实际上不可能使这些程序适应新的运行环境，也不能根据用户的需要在原有程序中增加一些新的功能。
- 软件通常没有适当的文档资料。缺乏必要的文档资料或者文档资料不合格，必然给软件开发和维护带来许多严重的困难和问题。
- 软件成本在计算机系统总成本中所占的比例逐年上升，例如，美国在 1985 年软件成本大约已占计算机系统总成本的 90%
- 软件开发生产率提高的速度，既跟不上硬件的发展速度，也远远跟不上计算机应用迅速普及的趋势。软件产品“供不应求”的现象使得人类无法充分利用现代计算机硬件提供的巨大潜力。

以上列举的仅仅是软件危机的一些明显的表现，与软件开发和维护有关的问题远远不止这些。

出现软件危机的主要原因是人们在开发软件时使用了错误的方法，而错误做法是在错误认识的指导下采用的。为了消除软件危机首先应该树立起对软件和软件开发的正确认识。因此下面我们着重批驳一些典型的似是而非的论调。

“有一个对目标的概括描述就足以着手编写程序了，许多细节可以在以后再补充。”

事实上，对用户要求没有完整准确的认识就匆忙着手编写程序是许多软件开发工程失败的主要原因之一。只有用户才真正了解他们自己的需要，但是许多用户在开始时并不能准确具体地叙述他们的需要，软件开发人员需要做大量深入细致的调查研究工作，反复多次地和用户交流信息，才能真正全面、准确、具体地了解用户的要求。对问题和目标的正确认识是解决任何问题的前提和出发点，软件开发同样也不例外。急于求成仓促上阵，对用户要求没有正确认识就匆忙着手编写程序，这就如同不打好地基就盖高楼一样，最终必然垮台。

“所谓软件开发就是编写程序并设法使它运行。”

一个软件从定义、开发、使用和维护，直到最终被废弃，经历了一个漫长的时期，这就如同一个人要经过胎儿、儿童、青年、中年、老年，直到最终死亡的漫长时期一样。通常把软件经历的这个漫长的时期称为软件生命周期（也称为生存周期）。正如我们在 1.1.1 小节中已经讲过的那样，软件开发最初的工作应是问题定义，也就是确定要求解决的问题是什么；然后要进行可行性研究，决定该问题是否存在一个可行的解决办法；接下来应该进行需求分析，也就是深入具体地了解用户的要求，在所要开发的系统（不妨称之为目标系统）必须做什么这个问题上和用户取得完全一致的看法。经过上述软件定义时期的准备工作才能进入开发时期，而在开发时期首先需要对软件进行设计（通常又分为总体设计和详细设计两个阶段），然后才能进入编写程序的阶段，程序编写完之后还必须经过大量的测试工作（需要的工作量通常占软件开发全部工作量的 40%～50%）才能最终交付使用。所以，编写程序只是软件开发过程中的一个阶段，而且在典型的软件开发工程中，编写程序所需的工作量只占软件开发全部工作量的 10%～20%。

另一方面还必须认识到程序只是完整的软件产品的一个组成部分，在上述软件生命周期的每个阶段都要得出最终产品的一个或几个组成部分（这些组成部分通常以文档资料的形式存在）。也就是说，一个软件产品必须由一个完整的配置组成，软件配置成分主要有程序、文档和数据。必须清除只重视程序而忽视软件配置其余成分的糊涂观念。

“用户对软件的要求不断变化，然而软件是柔软而灵活的，可以轻易地改动。”

确实，用户对软件的要求经常改变，特别是一个大型软件开发项目持续的时间往往相当长，在这段时间内由于外界环境变化以及人的认识不断深化，都会或多或少地改变对软件的要求。但是，必须看到也有相当多的改动不是由于用户要求的变化所造成的，而是由于软件开发人员在开发初期没有完全理解用户的要求，直到设计阶段甚至验收阶段才发现“已完成的”软件不完全符合用户的需要，从而必须进行修改。

严重的问题是，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，因此付出的代价较大；在软件“已经完成”时再引入变动，当然需要付出更高的代价。根据美国一些软件公司的统计资料，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。图 1.1 定性地描绘了在不同时期引入同一个变动需要付出的

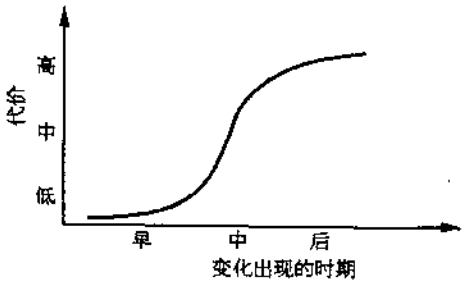

图 1.1 不同时期引入同一变动付出的代价