软件工程技术与实践

刘志峰 主编

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY

软件工程技术与实践

刘志峰　主编

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书在注重实用的前提下，汇集了近年来国内外在软件工程开发和管理领域的主流技术和实用工具，内容丰富、系统严谨。

本书主要介绍了软件工程领域的组织模式、项目管理模式、人员管理模式及质量管理模式。以理论和实践相结合的方式论述了软件开发管理策略和分析工具、可行性研究方法、需求分析方法、系统设计模型、编码、测试、维护等，并提供了大量已实际应用的范例。

本书可作为高等院校高年级学生及研究生的教学参考书，同时对软件组织和单位的开发者、组织者、管理者来说也是一本实用性很强的参考书。

图书在版编目（CIP）数据

软件工程技术与实践/刘志峰主编. —北京：电子工业出版社，2004.8
ISBN 7-121-00192-6

Ⅰ.软… Ⅱ.刘… Ⅲ.软件工程 Ⅳ.TP311.5

中国版本图书馆 CIP 数据核字（2004）第 077811 号

责任编辑：李 洁
印 刷：北京智力达印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：13.5 字数：344 千字
印 次：2004 年 8 月第 1 次印刷
印 数：4 000 册 定价：20.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　言

软件是新技术革命的核心部分，软件产业已成为一个具有独立形态的包含高附加值的优势产业，而软件工程正是软件产业健康发展的关键技术之一。

软件工程经过30多年的发展，已成为一门成熟的专业学科。软件工程的目标就是提高软件的质量和生产率，最终达到工业化生产。像传统工业一样，软件开发也需要众多的相关技术支持，因此项目管理、质量管理、CMM等与软件工程的结合越来越紧密。本书汇集了近年来国内外在软件工程开发和管理领域的主流技术和实用工具，结合大量实例讨论了软件开发的流程建设和各种相关技术方法，它有助于科研学术研究与公司企业实际应用的结合。将理论实例化、将应用规范化、将资源最优化是本书编写的目的。

全书由刘志峰主编，共分10章。第1章介绍了软件危机、软件工程学现状、软件开发模型和策略、软件行业概况等内容，由刘志峰博士、联想电脑公司张琦撰写；第2章介绍了软件开发相关知识，包括项目管理、CMM、ISO 9000等，由王蕾博士撰写；第3章介绍了软件开发人力资源管理，并提供了软件系统集成公司组织范例，由刘志峰博士、吴喜文副教授、王建华副教授撰写；第4章介绍了项目计划管理方法、软件规模估算方法和分析工具，由刘志峰博士、李富平副教授、彭映辉讲师撰写；第5章介绍了可行性分析模型、方法、步骤，并提供了两个实际案例，由杨文通教授、王哲讲师撰写；第6章介绍了需求分析方法、过程、原则及相应案例，由杨文通教授、王蕾博士撰写；第7章介绍了系统设计和软件配置管理，第8章介绍了软件的编码、测试与改错部分，并提供了使用表格和执行规范，第9章介绍了软件维护常识、维护评价与再生工程等，第10章以某销售系统软件为例，介绍了按照软件工程体系进行开发的过程，帮助读者实际体验软件工程管理技术的规范；第7、8、9、10章由刘志峰博士撰写。

北京工业大学刘志峰博士负责全书的统稿。

东北大学丁津原教授于百忙之中审阅了本书并提出宝贵的修改意见和建议，在此特别感谢。

由于编者水平和时间限制，疏漏和不准确之处在所难免，敬请读者批评指正。

刘志峰

2004年7月于北京

目　录

第 1 章　概述 …… (1)

1.1　软件开发的历史与软件危机… (1)
1.1.1　个体手工方式时期…… (1)
1.1.2　软件作坊时期…… (2)
1.1.3　程序设计时期…… (2)
1.1.4　软件危机实例…… (2)
1.1.5　软件危机…… (3)
1.1.6　软件危机的医治…… (5)
1.2　软件工程概述…… (5)
1.2.1　软件工程学科…… (5)
1.2.2　软件生命周期…… (6)
1.3　软件工程学发展现状…… (9)
1.3.1　新的软件开发模式…… (9)
1.3.2　计算机辅助软件工程 CASE …… (9)
1.3.3　软件自动生成器…… (9)
1.3.4　软件工程与人工智能… (9)
1.4　软件工程的目标与常用模型 …… (10)
1.4.1　软件工程的目标 …… (10)
1.4.2　软件工程的基本模型 …… (11)
1.5　软件开发的基本策略 …… (12)
1.5.1　抽象与模型方法 …… (13)
1.5.2　懒汉方法——复用 … (14)
1.5.3　模块化方法——分解 …… (15)
1.5.4　聪明方法——平衡 … (16)
1.6　软件工程基本观念 …… (17)
1.6.1　一些不正确的观念 … (17)
1.6.2　一些有争议不正确的观念 …… (17)
1.6.3　软件项目常见错误 … (18)
1.7　软件行业概况 …… (19)
1.7.1　政府管理者 …… (19)
1.7.2　行业管理者 …… (19)
1.7.3　中国软件产业现状 … (20)
1.7.4　相关信息 …… (20)

第 2 章　软件开发相关知识…… (22)

2.1　项目管理 …… (22)
2.1.1　项目管理概述 …… (22)
2.1.2　项目阶段和项目生命周期 …… (23)
2.1.3　项目属性与目标 …… (24)
2.1.4　项目管理知识体系 … (25)
2.1.5　项目管理软件介绍 … (26)
2.1.6　软件项目管理 …… (28)
2.1.7　项目管理实施案例 … (29)
2.1.8　IT 公司项目管理规范 …… (32)
2.2　CMM …… (38)
2.2.1　CMM 概述 …… (38)
2.2.2　CMM 体系结构 …… (41)
2.2.3　初始级（等级 1） … (41)
2.2.4　可重复级（等级 2）…(42)
2.2.5　定义级（等级 3） … (43)
2.2.6　管理级（等级 4） … (44)
2.2.7　优化级（等级 5） … (44)
2.2.8　CMM 改进与实施 … (45)
2.2.9　CMM 实施案例 …… (46)
2.3　ISO 9000 …… (50)
2.3.1　ISO 9000 概述 …… (50)
2.3.2　ISO 9000 软件相关规则 …… (50)

2.3.3 软件行业推行 ISO 9000 的典型步骤与注意事项 …… (52)
2.3.4 CMM 和 ISO 9000 对比 …… (52)
2.3.5 ISO 9000 范例 …… (54)
2.4 印度软件开发概况 …… (54)
2.4.1 软件企业的组织结构 …… (55)
2.4.2 项目组结构 …… (55)
2.4.3 项目计划 …… (56)
2.4.4 印度软件项目管理 … (56)

第 3 章 软件开发组织及人力资源管理 …… (59)

3.1 软件开发组织 …… (59)
3.1.1 软件开发组织模型 … (59)
3.1.2 软件开发团队策略 … (61)
3.1.3 软件开发人员的组织与分工 …… (63)
3.1.4 范例：J2EE 项目中开发团队的组建 …… (64)
3.2 程序员 …… (65)
3.3 程序经理 …… (66)
3.3.1 程序经理素质 …… (66)
3.3.2 程序经理技能 …… (67)
3.3.3 程序员等级制度 …… (68)
3.4 软件组织人力资源管理 …… (69)
3.4.1 组织规划（管理规划） …… (70)
3.4.2 人员组织 …… (71)
3.4.3 团队发展 …… (73)
3.5 软件集成公司组织结构范例（开发维护部分） …… (77)
3.5.1 产品开发立项 …… (77)
3.5.2 设计实现阶段 …… (77)
3.5.3 测试阶段 …… (78)
3.5.4 产品推广阶段 …… (78)
3.5.5 项目实施与维护阶段…(79)

第 4 章 项目计划和项目工具 …… (80)

4.1 项目计划 …… (80)
4.1.1 项目计划概述 …… (80)
4.1.2 项目计划过程 …… (80)
4.1.3 项目规划技巧 …… (82)
4.1.4 软件项目开发计划编写规范 …… (83)
4.2 软件项目规模估算方法 …… (85)
4.2.1 软件估算的基础与影响 …… (85)
4.2.2 进度计划方法 …… (88)
4.2.3 成本估算方法 …… (91)
4.3 项目计划方法 …… (92)
4.3.1 甘特图（Gantt Chart） …… (92)
4.3.2 网络计划方法 …… (93)
4.3.3 活动排序的工具和方法…… (100)
4.3.4 工作分解结构（WBS） …… (101)
4.3.5 责任矩阵…… (103)
4.3.6 工期压缩方法——时间/成本平衡法 …… (104)

第 5 章 可行性分析 …… (107)

5.1 可行性分析定义…… (107)
5.2 可行性分析的要素…… (107)
5.2.1 经济…… (107)
5.2.2 技术…… (109)
5.2.3 社会环境…… (110)
5.3 可行性研究步骤…… (110)
5.4 可行性分析范例…… (112)
5.4.1 某航空机票预订系统可行性分析报告…… (112)
5.4.2 ×××虚拟主机管理系统可行性分析…… (115)

第6章 需求分析 …… (121)

6.1 需求分析基础…… (121)
6.1.1 需求分析概念…… (121)
6.1.2 需求工程难点分析… (122)
6.1.3 需求风险…… (124)
6.2 需求分析原则…… (125)
6.3 需求分析方法…… (127)
6.3.1 未雨绸缪…… (127)
6.3.2 灵活应变…… (128)
6.3.3 参考法…… (129)
6.3.4 需求控制方法…… (129)
6.3.5 原型化方法…… (129)
6.4 需求分析人员与工具…… (131)
6.4.1 需求分析人员…… (131)
6.4.2 需求分析工具…… (131)
6.5 需求分析过程…… (134)
6.5.1 需求类型…… (134)
6.5.2 需求获取与表达…… (134)
6.5.3 需求评审…… (135)
6.5.4 需求确认与变更…… (137)
6.5.5 需求跟踪…… (139)
6.5.6 需求验证…… (139)
6.6 需求分析规范…… (140)
6.7 需求分析案例…… (143)
6.7.1 需求分析示例1——销售系统…… (143)
6.7.2 需求分析示例2——校园学籍管理系统…… (144)

第7章 系统设计 …… (149)

7.1 系统设计概述…… (149)
7.1.1 系统设计概念与重要性…… (149)
7.1.2 系统设计的任务与策略…… (149)
7.2 软件配置管理…… (150)
7.2.1 配置标示/配置项 … (150)
7.2.2 版本管理…… (151)
7.2.3 变更管理…… (151)
7.2.4 配置审核…… (152)
7.3 概要设计…… (153)
7.3.1 软件体系结构设计 … (154)
7.3.2 数据设计…… (156)
7.4 详细设计…… (157)
7.4.1 模块设计…… (158)
7.4.2 用户界面设计…… (159)
7.4.3 详细设计规范…… (162)
7.5 系统设计范例…… (163)

第8章 编码、测试与改错 …… (169)

8.1 编码…… (169)
8.1.1 程序语言特性…… (169)
8.1.2 程序设计语言的选择…… (169)
8.1.3 编码规范范本…… (170)
8.2 软件测试…… (175)
8.2.1 软件测试的教训…… (175)
8.2.2 软件测试目的与概念…… (175)
8.2.3 软件测试分类…… (176)
8.2.4 软件测试原则与完成准则…… (178)
8.2.5 软件测试过程…… (178)
8.2.6 软件测试规范…… (180)
8.3 改错…… (186)
8.3.1 改错步骤…… (186)
8.3.2 调试方法…… (186)
8.3.3 Bug跟踪与管理 …… (186)

第9章 软件维护 …… (188)

9.1 软件维护常识…… (188)
9.1.1 维护类型…… (188)
9.1.2 维护比重…… (188)
9.2 软件维护的代价及其影响因素…… (189)

9.2.1 软件维护成本…… (189)
9.2.2 软件维护影响因素… (189)
9.2.3 软件维护工作量…… (190)
9.2.4 软件维护的策略…… (190)
9.3 软件维护流程…… (191)
9.4 软件维护档案…… (191)
9.4.1 软件维护相关报告… (191)
9.4.2 软件维护记录…… (192)
9.5 维护评价与总结…… (192)
9.6 提高可维护性的方法…… (193)
9.7 逆向工程和再生工程…… (194)
9.7.1 逆向工程…… (194)
9.7.2 再生工程…… (195)
第 10 章 软件项目开发实例…… (196)
10.1 系统结构 …… (196)
10.2 数据库建立 …… (197)
10.3 系统功能设计 …… (199)
参考文献 …… (207)

第1章 概述

1.1 软件开发的历史与软件危机

从第一台电子计算机问世以来，计算机的数量由少到多，存储容量不断扩大，运算速度不断提高，体积微型化，从单机运行到互联成网。计算机硬件技术的每次突破，都为软件技术的发展提供了更加广阔的空间，开拓了更新、更为广阔的应用领域。20 世纪 60 年代以来，由于新的微电子元器件的出现，计算机硬件的性能和质量逐年提高，计算机硬件的价格大幅度下降，使得计算机的应用几乎遍及社会的各个领域和部门，成为人们日常工作和生活中不可缺少的工具。人们对软件的需求也急剧增加，软件系统也从简单发展到复杂，从小型发展到大型，由封闭系统发展成开放的系统。在这个不断演进的过程中，软件开发从注意技巧发展为注重管理，软件开发过程从目标管理转向过程管理，力图在可接受的性能价格比条件下，不断改进个人和软件开发组织的开发过程，强调在各自条件下追求软件过程的改进。

应该看到，由于最初计算机软件的开发技术远远没有跟上硬件技术的发展，因此使得软件开发的成本逐年剧增，软件的质量也没有可靠的保证，软件开发技术已经成为影响计算机系统发展的“瓶颈”。在软件发展初期，程序设计是少数智者才能做的事。他们有着非凡的智力与技能，可以编写出复杂而曲折的程序来控制“弱智”的电脑（计算机）。那个时期程序员把程序代码的集合称做软件。人们曾经把程序设计看做是一种任人发挥创造才能的技术领域，只要程序能得到正确的结果，程序的写法可以不受任何约束。人们认为好的程序应该是运用了许多与众不同的技巧和窍门的程序。这种观点好像把编程序当成了一种艺术，因为艺术总是要宣扬个性，喜欢与众不同。（实际上，编程序的确可以看做是一种艺术，不过这种艺术是在遵循一定范式和准则这个前提下的艺术。）

在软件发展初期，同时也产生了一堆问题：程序质量低下，错误频出，进度延误，费用剧增……这些问题严重阻碍了计算机软件的开发，导致了“软件危机”，但同时也产生了一门新学科——软件工程。

要知道什么是软件工程，当然得先知道什么是软件。软件就是计算机系统中与硬件相互依存的另一部分，它是包括程序、数据和相关文档的完整组合。程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发、维护和使用有关的图文材料。

注意：软件不仅包括程序，还包括文档。所以做软件也不仅仅是编程序，还需要写文档，近些年来人们认识到后者应该提到与前者并重的地位。让我们先来看一下软件发展的历史和一些相关数据。

1.1.1 个体手工方式时期

从 1947 年到 20 世纪 60 年代初，是计算机软件发展的初期。这个时期，人们最关心的是计算机能否可靠、持续地运行等问题。对于软件，还没有充分认识到它在发挥计算机效能上所占的重要位置，仅仅是把它当做在计算机上求解某一问题或帮助人脑进行计算而已。

当时，硬件通用化程度已经较高了，但软件程序的规模很小，程序的开发者和用户往往又是

同一个人，开发与使用是一个“内循环”，或者是一个极小范围的循环。这种个体化的开发环境，使得程序的开发结果，只有程序框图和源程序清单可以留下来。同时由于硬件体积大，存储容量小，运算速度慢，因此十分讲究编程技巧，以解决计算机内存容量不够和运算速度太低的矛盾。由于过分追求编程技巧，因此编出的程序除程序作者之外，其他人很难读懂。所以这个时期，也称为个体手工方式时期。

1.1.2 软件作坊时期

20 世纪 60 年代初到 60 年代末，是计算机软件发展的第二个时期。此时，计算机硬件技术有了较大的发展，稳定性与可靠性也有了极大的提高。通道技术、中断技术的出现，外存储设备、人机交互设备的改进，为计算机应用领域的扩大奠定了基础。计算机从单一的科学计算，扩展到数据处理、实时控制等方面。同时，人们为摆脱机器码编程的困难，相继研制出了一批高级程序设计语言，如 ALGOL、BASIC、FORTRAN 等，计算机公司也投入了大量的人力、物力从事系统软件和支撑软件的开发研究。此时的软件概念仅仅局限于“软件 = 程序 + 说明”。

该时期软件开发的特征主要表现在以下三个方面：

（1）因程序的规模增大，程序的设计需要多人分工协作，软件开发的方式由“个体生产”发展到“软件作坊”。

（2）程序的运行、维护、使用也不再由一个人承担。

（3）程序已不再是硬件的附属品，而是计算机系统中与硬件相互依存、共同发挥作用的不可缺少的部分。

这个时期，软件规模相当大，软件产业已经开始萌芽，软件产品广泛销售，软件数量急剧增加。“软件作坊”基本沿用了软件发展早期所形成的个体化开发方式。该时期软件开发与维护难度很大，软件运行出错，必须修改软件；用户需求改变，必须修改软件；硬件或操作系统更新，也需要修改软件以适应新的环境。上述种种情况，使得软件的开发与维护费用以惊人的速度递增。更为严重的是，由于软件的个体化开发特征，使得许多软件产品不可维护，最终导致严重的“软件危机”。

1.1.3 程序设计时期

从 1968 年至今，是计算机软件发展的第三个时期。这个时期软件产品已经兴起，软件作坊已经发展为软件公司，甚至是跨国公司。软件的开发是以工程化的思想为指导，用工程化的原则、方法和标准来开发和维护软件。软件开发的成功率大大提高，软件的质量也有了很大的保证。软件也开始产品化、系列化、标准化、工程化。这个时期，称为软件设计时期。

1.1.4 软件危机实例

软件本身是一个逻辑实体，开发过程又是一个“渐进的思考”过程，很难进行管理。开发人员根据自己的喜好和习惯来写程序，没有统一的标准和规范可以遵循。因而，在软件的开发过程中，人们遇到了极大的困难。有的软件开发彻底失败了，有的软件虽然开发出来了，但运行的结果极不理想，如程序中包含着许多错误，每次错误修改之后又会有一批新的错误取而代之。这些软件有的因无法维护而不能满足用户的新要求，最终失败了；有的虽然完成了，但比原计划推迟了好几年，而且成本大大超出了预算。

软件所具有的独特性，使其不能完全按照工业品的方式来制造，其特点如下：

（1）软件是逻辑实体，而不是物理实体；

（2）没有标准统一的加工过程；

（3）没有磨损、老化问题；

（4）软件的开发和运行常受到计算机系统的限制，对计算机系统有着不同程度的依赖性；

（5）软件的开发至今尚未完全摆脱手工开发方式；

（6）由于实际问题的复杂性和程序逻辑结构的复杂性，导致软件本身复杂；

（7）软件成本相当昂贵；

（8）相当多的软件工作涉及社会因素。

随着软件的规模越来越大，人们在实践中发现随心所欲编写的程序给维护、修改带来了很大的麻烦。程序晦涩难懂，不同程序员，不同时期编写的模块难以接口。因此，软件开发遇到了很大的困难，往往投入很大，收获甚微。开发的软件漏洞百出，或无人使用。下面是有关软件开发遇到的一些统计数字。

大约70%的软件开发项目超出了估算的时间，大型项目平均超出计划交付时间20%到50%，90%以上的软件项目开发费用超出预算，并且项目越大，超出项目计划的程度越高。美国政府审计局统计数字：只有不到2%的合同定购软件在发布时具有可用性——98%以上的项目都失败了。

20世纪末的软件危机中最棘手的莫过于千年虫问题，千年虫如同一个定时炸弹，十几年前就有人提出了预警，但是无人注意，直到日期到来的前两年，才引起了恐慌，于是各种补救、测试手段蜂拥而至，所以我们期望的在我们的银行卡中增加几个零的预想，并未成为现实，软件专家们把千年虫这个圣诞老人的礼物在21世纪到来之前全都给消灭了。

另一个经典的软件危机案例（Software Crisis Case），即IBM360操作系统的历史教训成为软件开发项目的典型事例而为人们所记取。

美国IBM公司在1963～1966年开发的IBM 360操作系统，这一项目花了5 000人一年的工作量，最多时有1 000人投入开发工作，写出了近100万行源程序，包含4 000多个模块，耗资达数亿美元。该系统投入运行后发现了2 000多个错误，据统计，这个操作系统每次发行的新版本都是从前一版本中找出1 000个程序错误而修正的结果……OS/360系统的负责人F.D.Brooms曾这样形象地描述了开发过程中的困难和混乱："……像巨兽在泥潭中作垂死挣扎，挣扎得越猛，泥浆就沾得越多，最后没有一个野兽能逃脱淹没在泥潭中的命运……程序设计就像是这样一个泥潭，一批批程序员在泥潭中挣扎……没人料到问题竟会这样棘手……"

1.1.5 软件危机

具体来讲，软件危机主要有以下几方面的表现：

（1）软件的复杂性越来越高，个体/作坊式开发已无法满足要求。计算机软件系统的规模和复杂程度，已经不是程序员各自为战的"手工作坊"式的生产方式可以解决的。软件开发的成本与进度严重估计不足，普遍的情况是实际成本远远高出计划成本，实际进度远远落后于理论进度。这不仅降低了软件组织/单位的信誉，而且，软件开发商为了赶进度与节约成本所采取的权宜之计，又常常会降低软件产品的质量，引起用户极大的不满。软件项目超出费用和进度表目标普遍存在的原因是：目标本身完全是错误的。

软件开发经验数据表明，计算机软件开发的周期与程序代码行是指数递增的关系，程序代码行与所需人年关系（1个人1年的工作量）如图1-1所示。

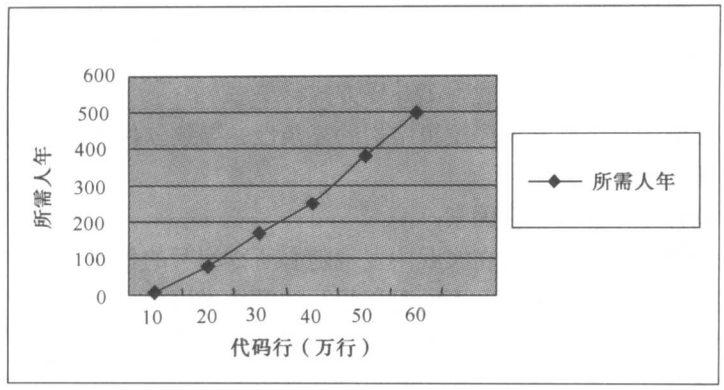

图 1-1　代码行与所需人年关系图

（2）软件成本在计算机系统中的比例持续上升。随着微电子技术的发展，计算机硬件的价格逐年下降，软件的价格却随着软件规模和数量的增加逐年攀升（美国审计局做过一个关于软件成本在计算机系统中成本的比例统计，如图 1-2 所示）。

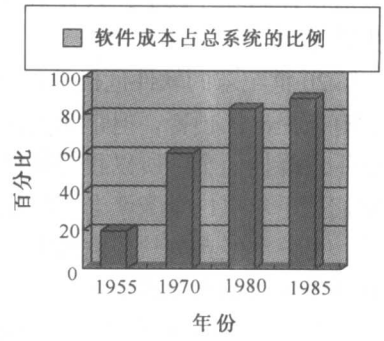

图 1-2　软件成本在系统成本中的比例

（3）软件维护工作量大。软件成本的 70％～80％是用于软件维护的，软件中的错误发现得越晚，修改错误所耗费的代价就越高，如同一个人生病一样，发现得早，癌症也可以治愈。数据表明，测试阶段改正一个错误所需的费用，大约是需求分析阶段的 100 倍。美国贝尔实验室对改正软件中的一个错误所需的费用统计如表 1-1 所示。

表 1-1　不同时期错误修正代价

问题修改阶段	改正一个问题所需的费用/美元
需求分析	20
详细设计	200
继承测试	1000
系统测试	2000

（4）软件＝程序，软件文档资源规范匮乏。计算机软件除了源代码外，还必须有一套完整的文档资料，这些资料记载了软件开发的全过程。软件组织的开发人员和管理人员，可以用这些资料来管理和评价软件开发过程的进度情况，并完成与客户的交流。这些资料作为通信工具，支持开发人员准确交流信息。同时，软件维护人员需要对这些资料进行正确的维护，文档的粗糙、遗失，甚至根本没有文档，加剧了软件系统的不可维护性。

（5）软件产品质量难以保证软件产品质量低劣、没有可靠的保证、出错率高。没有像工业产品那样，建立质量保证体系，无法真正满足客户的需要。软件系统的实用性/可用性较差，开发的关注点不是客户，一般用户是不懂计算机的专业人员，软件开发人员是不懂专业的计算机人员，两者之间缺乏准确而有效的沟通，开发人员在没有完全理解客户的真正需求和业务流程的情况下所开发出的软件，缺乏针对性，又无法按照客户的实际需要进行有效的修改，最终导致软件开发的失败。

1.1.6 软件危机的医治

软件危机更准确的说法即慢性痛苦（Chronic affliction）。

尽管忍受痛苦，但是软件依然在我们这个世界起着越来越重要的作用，但是如果能够医治痛苦，那么软件业将发展得更加健康。如何医治这种软件业的慢性痛苦？长期的实践，使人们意识到：软件编程，它有自己的生命周期（Life cycle）。大型软件系统的开发与其他工程项目（如建造桥梁、制造飞机、轮船等）的开发是同理的。于是，人们在茫然中，渐渐地抛弃了原先的观点，开始探索一种新的软件开发的思想。这就导致了软件工程的产生，程序也从按个人意图创造的“艺术品”转化为一种工程化的产品。

为了摆脱软件危机，北大西洋公约组织成员国在1968年和1969年于前联邦德国两度召开会议，一群程序员、计算机科学家与工业界人士聚集一起共商对策。会议总结了软件开发中失败的经验与教训，吸收了机械工程和土木工程设计中成熟而严密的工程设计思想，通过借鉴传统工业的成功做法，他们主张通过工程化的方法开发软件来解决软件危机，并冠以“软件工程”这一术语。30多年来，尽管软件的一些毛病像人类的感冒一样无法根治，但软件的发展速度超过了任何传统工业，其间并未出现真正的软件危机，如今软件工程已发展成了一门学科。

软件工程主要讲述软件开发的道理，基本上是软件实践者的成功经验和失败教训的总结。软件工程的观念、方法、策略和规范都是朴实无华的，平凡之人皆可领会，关键在于运用。我们应该活用软件工程的方法，未雨绸缪，尽量事先掌握，预料将要出现的问题，控制每个实践环节，并防患于未然。其中一些解决问题的想法如下：

（1）优秀的管理（Better management）；

（2）不同的团队组织模式（Different team organizations）；

（3）更好的语言和工具（Better languages & tools）；

（4）项目管理方法（Project management）；

（5）统一编程规范（Uniform coding conventions）等。

当“软件工程学”刚一问世，还处于学术研究阶段时，工程化的设计思想就对软件开发产生了巨大的影响。1971年，IBM公司首先运用软件工程技术，成功开发出了纽约时报情报库系统和空间实验室的飞行模拟系统。这两个系统规模都很大，开发过程中用户又提出了许多新的要求，但在减少了人力、物力，削减了经费的情况下，这两个软件项目还是成功地、高质量地完成了。

1.2 软件工程概述

1.2.1 软件工程学科

软件工程的定义很多，下面是不同的人和机构给出的定义。

（1）Boehm：运用现代科学技术知识来设计并构造计算机程序及为开发、运行和维护这些程序所必需的相关文件资料。

（2）IEEE：软件工程是开发、运行、维护和修复软件的系统方法。

（3）Fritz Bauer：建立并使用完善的工程化原则，以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法。

从科学角度来看，软件工程学科是指导计算机软件开发和维护的学科。它运用工程的概念、原理、方法、技术来开发和维护软件，软件工程包含的面很广，有基础的理论研究、应用研究，

也有实际开发；它还涉及管理学、数学、经济学、工程科学等学科。因此，可以说软件工程是一门指导计算机软件开发和维护的交叉学科，是信息化进程中的技术支持。与其他工程之间的主要区别是，软件工程的产品是程序，而不是其他什么有形的物质产品。

一般来说，软件工程可以看做是一种方法，一种思想，是指导大家如何更好地开发软件的方法和原则。软件工程也可看做是一种工具，是帮助大家分析、设计软件的工具，如：流程图，数据字典，以及一些辅助软件等。软件工程还可以看做是一种过程，就是软件开发的过程。

如同写作要有“时间、地点、人物”一样，软件工程也有三要素即方法、工具和过程。

（1）软件工程方法为软件开发提供了“如何做”的技术。

（2）软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。

（3）软件工程过程定义了：

- 方法使用的顺序；
- 要求交付的文档资料；
- 为保证质量和适应变化所需要的管理；
- 软件开发各个阶段完成的里程碑。

软件工程技术有两个明显的特点：

- 强调规范化。为了使有许多人共同开发的软件系统能准确无误地工作，开发人员必须遵守相同的约束规范，就是用统一的软件开发模型来规范软件开发步骤和应该进行的工作，用产品描述模型来规范文档格式，使其具有一致性和兼容性，规范化使软件生产摆脱了个人生产方式，进入了标准化、工程化的生产阶段。

- 强调文档化。一个复杂的软件要想让其他人员读懂，除程序代码外，还应用完备的设计文档来说明开发者的设计思想、设计过程和设计的具体实现技术等一系列相关信息。因此文档是十分重要的，是软件的重要组成部分。而且，开发人员按要求进度提交指定内容的文档，能使软件生产过程的不可见性变为部分可见，从而便于管理者对软件进度和软件开发过程进行管理。最后，可以通过对提交的文档进行技术审查和管理审查，保证软件的质量和有效的管理。

1.2.2 软件生命周期

软件产品从立项、开发、使用、维护到废止的整个时期，称为软件生命周期。在软件工程学中，将软件的生命周期分解为问题定义、可行性研究、需求分析、总体设计、详细设计、编码与单元测试、综合测试、运行与维护等几个阶段，每个阶段的任务都相对独立、简单，便于不同的人员分工协作，从而降低软件开发的难度。所以，应注意，当开始计划开发一个软件的时候，这个软件的生存期就已经开始了。

在软件生命周期的每个阶段都有明确的要求，严格的标准与规范，以及与开发软件完全一致的高质量的文档资料，从而保证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。

目前划分软件生命周期的方法有很多，软件规模、种类、开发方式、开发环境及开发方法都会影响软件生命周期阶段的划分。划分软件生命周期阶段应遵循的一条基本原则是，各阶段的任务应尽可能相对独立，以降低每个阶段的复杂程度，简化不同阶段之间的联系，利于软件开发工程化管理。

软件生存期模型是用来表示软件生存期内各种活动是如何组织的，是从软件项目需求直到软件经使用后废弃为止，跨越整个生存期的系统开发、运作和维护所实施的全部过程及活动和任务的结构框架。一般情况下，软件生命周期由软件定义、软件开发、软件维护三个时期组成。每个时期又分为若干个阶段：

软件定义，又称为系统分析。这个时期的任务，是确定软件开发的总目标，确定软件开发工

程的可行性，确定实现工程目标应该采用的策略和必须完成的功能，估计完成该项工程需要的资源和成本，制定出工程进度表。软件定义，可进一步划分为三个阶段，即问题定义、可行性研究和需求分析。

软件开发，是实现前一个时期定义的软件。它包含四个阶段：总体设计、详细设计、编码与单元测试、综合测试。

软件维护的任务，是使软件能够持久地满足用户的需求。具体地说，当软件在使用过程中发现错误，应能及时地改正；当用户在使用过程中提出新要求，应能按要求进行更新；当系统环境改变，应能对软件进行修正，以适应新的环境。

1. 问题定义

问题定义阶段必须考虑的问题是“做什么”。

正确理解用户的真正需求，是系统开发成功的必要条件。软件开发人员与用户之间的沟通，必须通过系统分析员对用户进行访问调查，扼要地写出对问题的理解，并在有用户参加的会议上认真讨论，澄清含糊不清的地方，改正理解不正确的地方，最后得到一份双方都认可的文档。在文档中，系统分析员要写明问题的性质、工程的预期目标以及工程的规模。问题定义阶段是软件生命周期中最短的阶段。

2. 可行性研究

可行性研究要研究问题的范围，并探索这个问题是否值得去解决，以及是否有可行的解决办法。

可行性论证是分析员在收集资料的基础上，经过分析，明确软件项目的目标、问题域、主要功能和性能要求，确定应用软件的支撑环境，以及经济、制作和时间限制等方面的约束条件，并用高层逻辑模型（通常用数据流图）对各种可能方案进行可行性分析及成本/效益分析。如果该项目在技术和经济上均可行，可明确地写出开发任务的全面要求和细节，形成软件计划任务书，作为本阶段的工作总结。软件计划任务书，包括软件项目目标、主要功能、性能、系统的高层逻辑模型、系统界面、可供使用的资源、进度安排和成本预算。

可行性研究的结果是使用部门负责人乃至高层领导者做出是否继续这项工程决定的重要依据。

3. 需求分析

需求分析即系统分析，通常采用系统模型定义系统。这一阶段的工作相当于机械工程中的方案设计，土木工程中的建筑方案设计。

需求分析阶段的任务，主要是确定目标系统必须具备的功能。系统分析员和用户密切配合，充分交流信息，得出经过用户确认的系统逻辑模型。系统的逻辑模型，通常是用数据流图、数据词典和简要的描述表示系统的逻辑关系。

需求分析阶段仍不能具体地解决问题，只能在前一阶段的基础上，对几种可行的方案进一步分析，得出经用户确认的系统逻辑模型。根据该系统逻辑模型，准确地回答“为了解决这个问题，目标系统必须做什么”。

需求分析只是原理性方案的设计。在这一阶段的工作中，为清晰地揭示问题的本质，往往略去具体问题中的一些次要因素，只将功能关系抽象为反映该问题的系统模型。这种模型，在机器设计中用机构运动简图来表示，在土木设计中用建筑方案草图来表示。

系统逻辑模型是以后设计和实现目标系统的基础，必须准确而完整地体现用户的要求。这一阶段，特别要注意克服急于着手进行具体设计的倾向。一旦分析员开始谈论程序设计的细节，就意味着他已脱离用户，并妨碍用户继续提出要求和建议。

4. 总体设计

总体设计，也叫概要设计或初步设计。总体设计的目标是将需求分析阶段定义的系统模型转换成相应的软件结构，以规定软件的形态及各成分间的层次关系、界面及接口要求。这一阶段的工作，在机械设计中称为装配图设计。

总体设计中，由于开发者在划分系统模块时，对模块的功能、模块与模块之间的联系方式、模块内各元素之间的联系等问题处理不同，设计出的软件结构也会不同。机械设计与土木设计也是如此。在机械设计中，根据同一机构运动简图，不同的设计人员会设计出不同结构的机器。正因为如此，才会有风格各异的轿车及其他一些机器。

总体设计应遵循的一条主要原则，就是程序模块化的原则。总体设计的结果通常以层次图或结构图来表示。

5. 详细设计

总体设计阶段以比较抽象、概括的方式提出了问题的解决方法。详细设计阶段的任务是把解法具体化，也就是应该怎样具体地实现这个系统。

详细设计亦即模块设计。它是在算法设计和结构设计的基础上，针对每个模块的功能、接口和算法定义，设计模块内部的算法过程及程序的逻辑结构，并编写模块设计说明。

这个阶段的工作还不是编程序，而是设计出程序的工序图。程序工序图通常用程序流程图、盒图、PAD 图、IPO 图或 PDL 语言来描述。它的作用类似于机械工程中的零件图、土木工程中的施工图。它们包含必要的处理细节，程序员根据它可以写出实际的程序代码，正如机械工人根据零件图可以加工出机械零件，建筑工人根据施工图可以进行建筑施工。

程序员可根据详细设计的结果进行编码设计。

6. 编码设计与单元测试

这个阶段的任务是根据详细设计的结果，选择一种适合的程序设计语言（必要时可采用汇编语言），把详细设计的结果翻译成程序的源代码。这一步工作就相当于机械设计中根据零件图加工零件，土木建筑设计中根据施工图进行施工。

每编写完一个模块，都要对模块进行测试，即单元测试，以便尽早发现程序中的错误和缺陷。

7. 综合测试

模块编码及测试完成后，需要根据软件结构进行组装，并进行各种综合测试。这一阶段的工作，相当于机械设计中根据装配图将加工与测试好的零件进行装配，并进行各种各样的综合测试。

软件测试中，测试计划、测试方案、测试用例报告及测试结果，是软件配置的一部分，应以正式的文档形式保存下来。

综合测试的目标，是产生一个可用的软件文本，修订和确认软件的使用手册。

8. 软件运行与维护

软件产品同机械产品一样，必须经过不断地使用和维护，才能逐步地发现存在于产品中的错误或不完善的地方。只有不断地改正所发现的错误，并完善其功能，产品才能适应社会需求而得以生存和发展。

软件的使用与维护是密不可分的。维护对于确保软件正常使用、延长使用期、充分发挥其经济效益是非常重要的。对于大型软件项目，维护是一项工程，可安排专门的人员负责，依照一套专门的方法和技术，使用一组专门的维护工具，进行软件维护。维护阶段的任务，是通过各种必要的维护措施使软件系统能持久地满足用户的需要。

维护可分为四类：纠错性维护、适应性维护、完善性维护和预防性维护。纠错性维护是对软

件在使用过程发现的错误进行诊断和改正；适应性维护是为了让软件适应新的环境（如操作系统的改变、支撑环境的改变等）而进行的修改；完善性维护是为了改进和扩充软件的功能而进行的修改；预防性维护是为将来的维护活动所做的准备。

每一项维护都要以正式文档的形式记录下来，作为软件配置的一部分。

1.3 软件工程学发展现状

随着软件工程学的不断研究和广泛应用，出现了许多新的设计思想和方法，如新的软件开发模式、软件重用、计算机辅助软件工程、软件工程的人工智能方法、软件自动生成等。

1.3.1 新的软件开发模式

传统的软件开发模式存在着不少弊端。因此，人们一直在探索新的软件开发模式。M.A.Jackson 与 F.D.Brooks 提出了适于大型软件系统的原型开发模式，后来又有人在原型开发模式的基础上提出了智能原型模式，并预言将进一步提高软件的开发率。J.Martin 提出的战略数据库规划模型，从信息化社会和计算机企业化管理的角度来构造各行业的系统目标，从而避免行业陈旧，软件也随之报废的悲剧。这一模型，伴随着广域网络和全球网络的建成而发挥巨大的作用。

1.3.2 计算机辅助软件工程 CASE

计算机辅助软件工程 CASE（Computer-Aided Software Engineering），是 20 世纪 80 年代末期从计算机辅助编程工具 4GLS（4th Gerenation Languages）和绘图工具发展而来的大型综合计算机软件工程开发环境。软件过程的车间称为基层的项目支撑环境（Integrate project support environment），放置在车间中的工具称为 CASE。

早期的 CASE 是以工具和辅助开发环境面貌出现，以自动化的编程环境来取代原有的一些结构简单、功能较弱的开发工具。随着计算机技术水平的提高，CASE 的概念从具体的工具发展成为一门方法学。目前，CASE 已从支持结构化开发方法、原型方法、面向对象方法，发展到支持包括知识处理语言在内的大型综合软件开发环境。它已成为工具和方法相结合的产物。

CASE 跨越软件过程的每个步骤和那些贯穿整个过程的全程性活动，组合了针对数据、工具和人机交互的集成机制。数据集成可以通过直接交换、公共文件结构、通过数据共享或互操作来实现；工具集成可以由一起工作的厂商定制设计或通过作为中心库的一部分的管理软件来实现；人机集成通过流行的界面标准来实现。其中，中心库是关系型的或面向对象的数据库，该库是软件工程信息的积聚和存储中心。

目前，各家公司都有自己的 CASE 产品。比较有代表性的是 DEC 公司的集成化 CASE 和 ORACLE公司的 CASE 方法。

1.3.3 软件自动生成器

软件自动生成器可以不经过软件开发的漫长周期，直接自动生成软件需求的程序或目标。例如 APS 自动程序生成器，只要求开发人员做出算法的构造，就可以直接生成C++或 PASCAL 的目标程序，使软件的开发工作大大简化，工作量甚至可以降低 80%～95%。

1.3.4 软件工程与人工智能

人工智能，是计算机科学的一个重要分支，是在对人的智能活动的研究中找出用计算机来替