计算机软件工程丛书

软件工程导论

（修订版）

张海藩

清华大学出版社

软件工程导论

（修订版）

张海藩　编著

清华大学出版社

内 容 提 要

本书第一版系作者根据在北京大学校内外多次讲授“软件工程概论”课程所用的讲义改写而成。为适应新的发展情况，作者总结四年来的教学和科研的经验，对原书内容作了调整和增删，变动约65%，成为修订版。书中较全面系统地介绍了软件工程的概念、原理以及典型的技术方法。本书的特点是既注重系统性和科学性，又注重实用性；既有广度（全面概括地介绍了各种常用方法），又有深度（具体详尽地讲述了一种方法）；既有原理性论述，又有丰富的实例与之配合，特别是正文后面的附录B，比较完整地讲述了用软件工程的方法开发一个实际软件的过程，对读者深入理解软件工程学很有帮助，也是上机实习的好材料。本书正文共十章，第一章是概论，第二章至第九章顺序讲述软件生存周期各阶段的任务、过程、方法和工具，第十章集中讨论软件工程使用的管理技术。

本书可作为高等院校计算机系“软件工程”课程的教材或教学参考书，可供有一定实际经验的软件工作人员和需要开发应用软件的广大计算机用户阅读参考。

（京）新登字158号

软件工程导论（修订版）

张海藩 编著

责任编辑 贾仲良

☆

清华大学出版社出版

北京 清华园

国防工业出版社印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16 印张：19.25 字数：454千字

1992年6月第2版 1992年6月第1次印刷

印数：00001—10000

ISBN 7-302-00972-4/TP·360

定价：9.30元

前　言

计算机工业发达国家在发展软件的过程中曾经走过不少弯路，受过许多挫折，至今仍然经受着“软件危机”的困扰。人们开发优质软件的能力大大落后于计算机硬件日新月异的进展和社会对计算机软件不断增长的需求，这种状况已经严重妨碍了计算技术的进步。

为了摆脱软件危机的困扰，一门研究软件开发与维护的普遍原理和技术的工程学科——软件工程学——从六十年代末期开始迅速发展起来了，现在它已经成为计算机科学技术的一个重要分支，一个异常活跃的研究领域。严格遵循软件工程方法论可以大大提高软件开发的成功率，能够显著减少软件开发和维护中的问题。

软件工程学（通常简称软件工程）研究的范围非常广泛，包括技术方法、工具和管理等许多方面，软件工程又是一门迅速发展的新兴学科，新的技术方法和工具不断涌现，真可谓五花八门层出不穷。因此，在一本书中不可能包含软件工程的全部内容。本书《软件工程导论》是软件工程的入门介绍，着重从实用角度讲述软件工程的基本原理、概念和技术方法，同时也尽量注意书的全面性和系统性。希望本书既能对实际的软件开发工作有所帮助，又能为读者在今后深入研究这门学科奠定良好的基础。

本书正文共十章。第一章概括介绍软件工程学产生的历史背景以及它的基本原理、概念和方法。第二章到第九章按软件生存周期的顺序介绍各个阶段的任务、过程、方法和工具。第十章着重讨论软件工程的管理技术。正文后面有两个附录，比较完整地介绍了两个实际软件的开发。附录A着重讲述从问题定义到实现的过程，把这个具体例子和课文前几章的内容结合起来学习，有助于加深对一些基本概念和方法的理解。附录B讲述一个交互式正文编辑程序的设计和描述，它是上机实习的好材料（例如，可以实习把设计翻译成程序、设计测试方案以及维护的方法）。

《软件工程导论》可以为多种读者服务。本书通俗易懂，实例丰富，既有对多种常见方法的全面概括介绍，又有对一种典型方法的深入详尽介绍，很适合于有一定实践经验的软件工作者和广大计算机用户参考或自学；对于高等院校计算机系高年级本科生和研究生来说，本书可以做为软件工程课程的教材。

本书根据编者最近几年在北京大学校内外多次讲授“软件工程概论”课所用的讲义改写而成，改写时充分考虑了在校内外讲授这门课程时广大学员的建议和要求，并且增加了最近收集到的大量新材料。

编者在美国马里蓝大学进修期间，听过该校朱耀汉（Yaohan Chu）教授讲授的软件工程课，并且在朱教授指导下从事过一些软件开发工作，这些都为本书的编写奠定了基础。本书编写前征求了复旦大学计算机系副教授潘锦平同志对写作大纲的意见，初稿写出后又征求了潘锦平和北大计算机系副系主任许卓群等同志对内容编排的意见，承蒙他们给

予真诚的鼓励并且提出了许多宝贵的具体建议。此外，本书编写过程中董士海副教授曾经提供了许多很有价值的材料。谨在此向他们表示衷心的感谢！

本书不当之处敬请广大读者不吝赐教。

编者　1986 年

修订版前言

本书第一版自1987年6月出版以来，颇受读者欢迎，不少高校用其作为“软件工程概论”课的教材。然而第一版的内容主要取材于1985年以前出版的书籍或文章和著者的实践总结，经过五年多的时间，这一学科又有了不少新的进展，为了跟踪学科发展方向，更好地为广大读者服务，编者对原书作了认真修订。

这次修订的指导思想是，保持原书系统性较强，内容比较全面，有丰富的实例与原理性论述紧密配合的特点，增加了反映学科最新发展方向的新内容，同时又适当压缩全书的篇幅，以降低成本，减轻读者负担。为此在修订时对书的取材作了十分慎重的斟酌，删掉了前后重复的内容，去掉了对理解本书的内容不是十分必要的图表和文字，占篇幅比较多的原附录A也被割爱了；增加了近年来比较流行的原型法、面向对象的程序设计和软件再用技术的介绍，强调了软件工程与人工智能相互结合、互相促进的发展趋势。对附录B介绍的编辑程序的功能作了扩充，不仅增加了编辑命令，还增加了输入和编辑汉字的功能。

此次修订还适当加强了对日本软件工程方法的介绍，例如，第五章增加了对日立公司发明的PAD图的介绍，第八章讲述了日立预测法的内容。

另一个比较重要的改动，是使书中使用的术语与近年来国内的习惯用法保持一致，对其他文字也作了进一步的加工和润色。

为便于教学，此次修订在每章后面都附上了适当数量的习题。

编者　1990年11月

于北京信息工程学院

目　录

第一章 概 论

在工业发达的国家中计算机系统已经经历了三个不同的发展时期。随着计算机应用日益普及和深化，正在运行使用着的计算机软件的数量以惊人的速度急剧膨胀，而且现代软件的规模往往十分庞大，包含数百万行代码，耗资几十亿美元，花费几千人年的劳动才开发出来的软件产品，在70年代已经屡见不鲜了。

由于微电子学技术的进步，计算机硬件性能/价格比平均每十年提高二个数量级，而且质量稳步提高；与此同时，计算机软件成本却在逐年上升，质量没有可靠的保证，软件开发的生产率也远远跟不上普及计算机应用的要求。软件已经成为限制计算机系统发展的关键因素。

在计算机系统发展的早期时代所形成的一些错误概念和做法，已经严重地阻碍了计算机软件的开发，更严重的是，用错误方法开发出来的许多大型软件几乎根本无法维护，只好提前报废，造成大量人力、物力的浪费。西方计算机科学家把软件开发和维护过程中遇到的一系列严重问题统称为“软件危机”，并且在60年代后期开始认真研究解决软件危机的方法，从而逐步形成了计算机科学技术领域中的一门新兴的学科——计算机软件工程学，通常简称为软件工程。

1.1 软件危机

在计算机系统发展的早期时代(60年代中期以前)，通用硬件相当普遍，软件却是为每个具体应用而专门编写的。这时的软件通常是规模较小的程序，编写者和使用者往往是同一个(或同一组)人。这种个体化的软件环境，使得软件设计通常是在人们头脑中进行的一个隐含的过程，除了程序清单之外，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。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，许多程序的个体化特性使得它们最终成为不可维护的。“软件危机”开始出现了！1968年北大西洋公约组织的计算机科学家在联邦德国召开国际会议讨论软件危机问题，在这次会议上正式提出并使用了“软件工程”这个名词，一门新兴的工程学科就此诞生了。

1.1.1 什么是软件危机

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：如何开发软件，怎样满足对软件的日益增长的需求；如何维护数量不断膨胀的已有软件。具体地说，软件危机主要有下述一些表现：

1. 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有可能高出一个数量级，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。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满。

2. 用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求只有模糊的了解，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。软件开发人员和用户之间的信息交流往往很不充分，“闭门造车”必然导致最终的产品不符合用户的实际需要。

3. 软件产品的质量往往靠不住。软件可靠性和质量保证的确切的定量概念刚刚出现不久，软件质量保证技术（审查、复审和测试）还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

4. 软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可再用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

5. 软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是“最新式的”（即和程序代码完全一致的）。软件开发组织的管理人员可以使用这些文档资料作为“里程碑”，来管理和评价软件开发工程的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息；对于软件维护人员而言，这些文档资料更是至关重要必不可少的。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。

6. 软件成本在计算机系统总成本中所占的比例逐年上升。由于微电子学技术的进步和生产自动化程度不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在1985年软件成本大约已占计算机系统总成本的90%。

7. 软件开发生产率提高的速度远远跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象使人类不能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。

1.1.2 产生软件危机的原因

在软件开发和维护的过程中存在这么多严重问题，一方面与软件本身的特点有关，另一方面也和软件开发与维护的方法不正确有关。

软件不同于硬件，它是计算机系统中的逻辑部件而不是物理部件。在写出程序代码并在计算机上试运行之前，软件开发过程的进展情况较难衡量，软件开发的质量也较难评价，因此，管理和控制软件开发过程相当困难。此外，软件在运行过程中不会因为使用时间过长而被"用坏"，如果运行中发现错误，很可能是遇到了一个在开发时期引入的在测试阶段没能检测出来的故障。因此，软件维护通常意味着改正或修改原来的设计，这就在客观上使得软件较难维护。

软件不同于一般程序，它的一个显著特点是规模庞大。例如，美国四代宇宙飞船的软件规模呈指数增长，70年代末穿梭号宇宙飞船的软件包含4000万行目标代码。假设一个人一年可以开发出一个一万行的程序，为了开发一个4000万行的软件，是否集中4000人的力量一年就可以完成呢？绝对做不到！因为代码长度增加了4000倍，程序复杂程度的增加远远超过4000倍。而且如何保证每个人完成的工作合在一起确实能构成一个高质量的大型软件系统，更是一个极端复杂困难的问题，不仅涉及许多技术问题，诸如分析方法、设计方法、形式说明方法、版本控制等，更重要的是必须有严格而科学的管理。

软件本身独有的特点确实给开发和维护带来一些客观困难，但是人们在开发和使用计算机系统的长期实践中，也确实积累和总结出了许多成功的经验。如果坚持不懈地使用经过实践考验证明是正确的方法，许多困难是完全可以克服的，过去也确实有一些成功的范例[3]。但是，目前相当多的软件专业人员对软件开发和维护还有不少糊涂观念，在实践过程中或多或少地采用了错误的方法和技术，这可能是使软件问题发展成软件危机的主要原因。

与软件开发和维护有关的许多错误认识和作法的形成，可以归因于在计算机系统发展的早期软件开发的个体化特点。错误认识和作法主要表现为忽视软件需求分析的重要性，认为软件开发就是写程序并设法使之运行，轻视软件维护等。

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。

一个软件从定义、开发、使用和维护，直到最终被废弃，要经历一个漫长的时期，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生存周期。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，在上述软件生存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)。Boehm曾经指出："软件是程序以及开发、使用和维护程序需要的所有文档。"这也就是对软件的定义。所以，一个软件产品必须由一个完整的配置组成，应该清除只重视程序而忽视软件配置其余成分的糊涂观念。

作好软件定义时期的工作，是降低软件成本提高软件质量的关键。如果软件开发人员在定义时期没有正确全面地理解用户需求，直到测试阶段或软件交付使用后才发现"已完成的"软件不完全符合用户的需要，这时再修改就为时已晚了。

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，因此付出的代价剧增；在软件"已经完成"时再引入变动，当然需要付出更高得多的代价。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一个变动需要付出的代价的变化趋势。图1.2是美国贝尔实验室统计得出的定量结果。

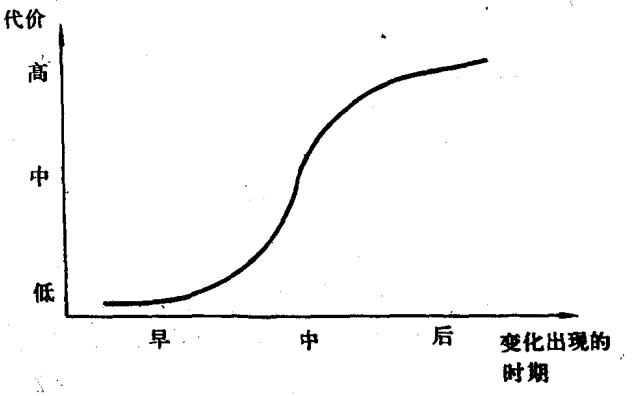

图1.1　引入同一变动付出的代价随时间变化的趋势

通过上面的论述不难认识到，轻视维护是一个最大的错误。许多软件产品的使用寿命长达十年甚至二十年，在这样漫长的时期中不仅必须改正使用过程中发现的每一个潜伏的错误，而且当环境变化时(例如硬件或系统软件更新换代)还必须相应地修改软件以适应新的环境，特别是必须经常改进或扩充原来的软件以满足用户不断变化的需要。所有这些改动都属于维护工作，而且是在软件已经完成之后进行的，因此维护是极端艰巨复杂的工作，需要花费很大代价。统计数据表明，实际上用于软件维护的费用占软件总费用的55～70%。软件工程学的一个重要目标就是提高软件的可维护性，减少软件维护的代价。

了解产生软件危机的原因，澄清错误认识，建立起关于软件开发和维护的正确概念，

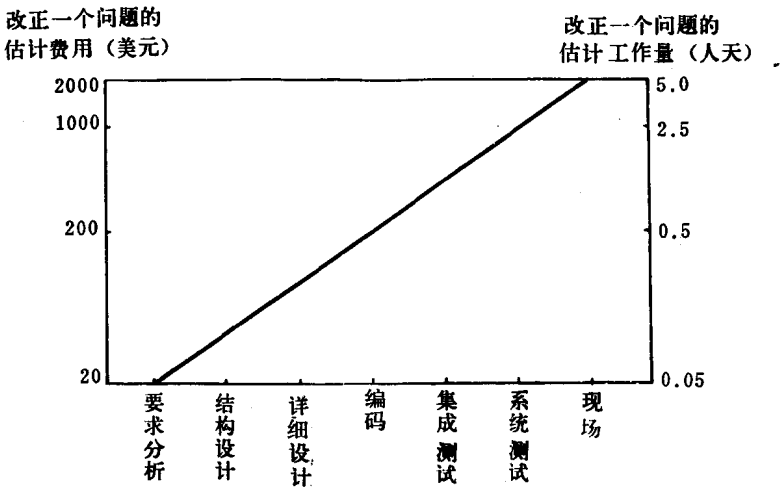

图 1.2　改正一个问题需要付出的代价(越在早期代价越小)

还仅仅是解决软件危机的开始,全面解决软件危机需要一系列综合措施。

1.1.3　解决软件危机的途径

软件开发不是某种个体劳动的神秘技巧,而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。必须充分吸取和借鉴人类长期以来从事各种工程项目所积累的行之有效的原理、概念、技术和方法,特别要吸取几十年来人类从事计算机硬件研究和开发的经验教训。

应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索更好更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法。

应该开发和使用更好的软件工具。正如机械工具可以"放大"人类的体力一样,软件工具可以"放大"人类的智力。在软件开发的每个阶段都有许多繁琐重复的工作需要做,在适当的软件工具辅助下,开发人员可以把这类工作做得既快又好。如果把各个阶段使用的软件工具有机地集合成一个整体,支持软件开发的全过程,则称为软件工程支撑环境。

总之,为了解决软件危机,既要有技术措施(方法和工具),又要有必要的组织管理措施。软件工程正是从管理和技术两方面研究如何更好地开发和维护计算机软件的一门新兴学科。

1.2　软 件 工 程

软件工程是指导计算机软件开发和维护的工程学科。采用工程的概念、原理、技术和方法来开发与维护软件,把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来,这就是软件工程。

软件工程强调使用生存周期方法学和各种结构分析及结构设计技术。它们是在七十年代为了对付应用软件日益增长的复杂程度、漫长的开发周期以及用户对软件产品经常不满意的状况而发展起来的。人类解决复杂问题时普遍采用的一个策略就是"各个击破"，也就是对问题进行分解然后再分别解决各个子问题的策略。软件工程采用的生存周期方法学就是从时间角度对软件开发和维护的复杂问题进行分解，把软件生存的漫长周期依次划分为若干个阶段，每个阶段有相对独立的任务，然后逐步完成每个阶段的任务。采用软件工程方法论开发软件的时候，从对任务的抽象逻辑分析开始，一个阶段一个阶段地进行开发。前一个阶段任务的完成是开始进行后一个阶段工作的前提和基础，而后一阶段任务的完成通常是使前一阶段提出的解法更进一步具体化，加进了更多的物理细节。每一个阶段的开始和结束都有严格标准，对于任何两个相邻的阶段而言，前一阶段的结束标准就是后一阶段的开始标准。在每一个阶段结束之前都必须进行正式严格的技术审查和管理复审，从技术和管理两方面对这个阶段的开发成果进行检查，通过之后这个阶段才算结束；如果检查通不过，则必须进行必要的返工，并且返工后还要再经过审查。审查的一条主要标准就是每个阶段都应该交出"最新式的"(即和所开发的软件完全一致的)高质量的文档资料，从而保证在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。文档是通信的工具，它们清楚准确地说明了到这个时候为止，关于该项工程已经知道了什么，同时确立了下一步工作的基础。此外，文档也起备忘录的作用，如果文档不完整，那么一定是某些工作忘记做了，在进入生存周期的下一阶段之前，必须补足这些遗漏的细节。在完成生存周期每个阶段的任务时，应该采用适合该阶段任务特点的系统化的技术方法——结构分析或结构设计技术。

把软件生存周期划分成若干个阶段，每个阶段的任务相对独立，而且比较简单，便于不同人员分工协作，从而降低了整个软件开发工程的困难程度；在软件生存周期的每个阶段都采用科学的管理技术和良好的技术方法，而且在每个阶段结束之前都从技术和管理两个角度进行严格的审查，合格之后才开始下一阶段的工作，这就使软件开发工程的全过程以一种有条不紊的方式进行，保证了软件的质量，特别是提高了软件的可维护性。总之，采用软件工程方法论可以大大提高软件开发的成功率，软件开发的生产率也能明显提高。

目前划分软件生存周期阶段的方法有许多种，软件规模、种类、开发方式、开发环境以及开发时使用的方法论都影响软件生存周期阶段的划分。在划分软件生存周期的阶段时应该遵循的一条基本原则就是使各阶段的任务彼此间尽可能相对独立，同一阶段各项任务的性质尽可能相同，从而降低每个阶段任务的复杂程度，简化不同阶段之间的联系，有利于软件开发工程的组织管理。一般说来，软件生存周期由软件定义、软件开发和软件维护三个时期组成，每个时期又进一步划分成若干个阶段。下面的论述主要针对应用软件，对系统软件也基本适用。

软件定义时期的任务是确定软件开发工程必须完成的总目标；确定工程的可行性；导出实现工程目标应该采用的策略及系统必须完成的功能；估计完成该项工程需要的资源和成本，并且制定工程进度表。这个时期的工作通常又称为系统分析，由系统分析员负责完成。软件定义时期通常进一步划分成三个阶段，即问题定义、可行性研究和需求分析。

开发时期具体设计和实现在前一个时期定义的软件，它通常由下述四个阶段组成：总体设计，详细设计，编码和单元测试，综合测试。

维护时期的主要任务是使软件持久地满足用户的需要。具体地说，当软件在使用过程中发现错误时应该加以改正；当环境改变时应该修改软件以适应新的环境；当用户有新要求时应该及时改进软件以满足用户的新需要。通常对维护时期不再进一步划分阶段，但是每一次维护活动本质上都是一次压缩和简化了的定义和开发过程。

下面扼要介绍软件生存周期每个阶段的基本任务和结束标准。

1.2.1 问题定义

问题定义阶段必须回答的关键问题是："要解决的问题是什么？"如果不知道问题是什么就试图解决这个问题，显然是盲目的，只会白白浪费时间和金钱，最终得出的结果很可能是毫无意义的。尽管确切地定义问题的必要性是十分明显的，但是在实践中它却可能是最容易被忽视的一个步骤。

通过问题定义阶段的工作，系统分析员应该提出关于问题性质、工程目标和规模的书面报告。通过对系统的实际用户和使用部门负责人的访问调查，分析员扼要地写出他对问题的理解，并在用户和使用部门负责人的会议上认真讨论这份书面报告，澄清含糊不精的地方，改正理解不正确的地方，最后得出一份双方都满意的文档。

问题定义阶段是软件生存周期中最简短的阶段，一般只需要一天甚至更少的时间。

1.2.2 可行性研究

这个阶段要回答的关键问题是："对于上一个阶段所确定的问题有行得通的解决办法吗？"为了回答这个问题，系统分析员需要进行一次大大压缩和简化了的系统分析和设计的过程，也就是在较抽象的高层次上进行的分析和设计的过程。可行性研究应该比较简短，这个阶段的任务不是具体解决问题，而是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。

在问题定义阶段提出的对工程目标和规模的报告通常比较含糊。可行性研究阶段应该导出系统的高层逻辑模型（通常用数据流图表示），并且在此基础上更准确、更具体地确定工程规模和目标。然后分析员更准确地估计系统的成本和效益，对建议的系统进行仔细的成本/效益分析是这个阶段的主要任务之一。

可行性研究的结果是使用部门负责人做出是否继续进行这项工程的决定的重要依据，一般说来，只有投资可能取得较大效益的那些工程项目才值得继续进行下去。可行性研究以后的那些阶段将需要投入要多的人力物力。及时中止不值得投资的工程项目，可以避免更大的浪费。

1.2.3 需求分析

这个阶段的任务仍然不是具体地解决问题，而是准确地确定"为了解决这个问题，目标系统必须做什么"，主要是确定目标系统必须具备哪些功能。

用户了解他们所面对的问题，知道必须做什么，但是通常不能完整准确地表达出他们