第一篇　基　础

第一章　第四代语言的产生

社会现代化，关键在于科学技术现代化，而计算机的广泛应用则是科学技术现代化的前提和标志。

利用计算机解决社会上的各类问题，不仅需要有计算机硬件，同时也需要有指挥硬件工作的计算机软件。计算机硬件和软件密切配合协同工作，共同构成计算机系统。如果用人来比喻计算机系统，硬件就好像是身体，软件则相当于大脑。也有人把硬件比喻为人的肉体，把软件比喻为人的灵魂（精神），这种比喻方法不仅形象通俗地说明了软、硬件功能的不同，同时也表明了它们在物理形态方面的区别：硬件是看得见模得着的物理部件，软件是看不见模不着的逻辑部件。

随着计算机应用的日益普及和深化，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，而且现代软件通常规模十分庞大，逻辑非常复杂。由于微电子学技术的进步，计算机硬件的性能/价格比平均每十年提高两个数量级，硬件质量也在稳步提高；与此相反，计算机软件成本却在逐步上升，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瓶颈。

人们把软件开发和维护过程中所遇到的一系列严重问题统称为“软件危机”，并且从60年代后期开始认真研究消除软件危机的方法，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）。软件工程传统的基本途径，是采用生存周期方法学来开发与维护软件。

对于某些类型的软件来说，生存周期方法学确实有效，实践中也曾取得过巨大的成功。但是，人们在实践中也逐步认识到，对于另外一些类型的软件来说，生存周期方法学并不适用。也就是说，生存周期方法学并不是软件工程所应采取的唯一途径。在某些类型的计算机应用领域中，一种更新型的开发途径正在取代经典的生存周期方法学，新途径的特点是快速地构造原型系统，因此，软件工程的这种新途径通常称为快速原型法。为适应原型法的需要，出现了一类新的软件工具，通常统称为计算机第四代语言（简称为第四代语言，缩写为4GL）。

1.1　软件危机

1.1.1　计算机系统的发展过程

过去人们往往按照计算机硬件的演变来划分计算机系统发展的时期，然而，为了了解计算机软件发展演变的过程，特别是为了了解软件危机是怎样产生的，又是如何加剧的，从而

探索解决软件危机的途径，应该更全面地回顾计算机系统发展的简短历史，按照计算机应用领域的演变而不仅仅是根据硬件特点的演变来划分计算机系统发展的时期。

从世界上出现第一台计算机到60年代中期是计算机系统发展的早期时代，包括第一代和第二代计算机。在这个时期人们用很大力气研究和发展计算机硬件，经历了从电子管计算机到晶体管计算机的变革；可是，人们对计算机软件的研究和发展却不够重视，认为软件开发是有了计算机硬件后才需要考虑的问题，基本上没有系统化的软件开发方法。软件开发在那个时期只是根据应用的需要写出能够运行的机器语言程序。由于硬件价格昂贵，运算速度低，内存容量少，所以当时的程序员非常强调"程序设计技巧"，把缩短每一微秒CPU时间和节省每一个二进制存储单元，作为程序设计（也就是那个时期的软件开发）的重要目标。

在计算机系统发展的早期时代，大多数系统采用批处理方式工作，只有个别系统是交互式系统。当时大多数软件的开发者和使用者是同一个（或同一组）人。编写程序的人要设法使程序在机器上运行，并且负责使用这个程序解决预定的问题，如果使用过程中程序发生错误也得由这个人设法修改。由于这种个体化的软件环境，使得软件设计通常是在人们头脑里进行的一个隐含的过程，而且除了程序清单和上机说明书之外，一般没有其它文档资料保存。

从60年代中期到70年代初期是计算机系统发展的第三代时期。在这个时期硬件经历了从晶体管计算机到集成电路计算机的变革，CPU速度和内存容量都有了很大提高，从而为计算机在众多领域中的应用提供了潜在的可能性。为了更有效地利用硬件的性能，计算机系统普遍采用多道程序和多用户分时的工作方式；能够从多个数据源采集数据，高速进行处理，并在很短的时间内（几毫秒）产生输出信息，从而控制实际过程进行的实时系统，开辟了计算机应用的一个新领域；联机辅助存储设备的进展导致出现了第一代数据库管理系统。

这个时期的另一个重要特征是出现了"软件作坊"，广泛使用商品软件。这是因为随着计算机应用的普及和深化，需要的软件往往规模相当庞大，以致单个用户无力开发；此外，许多不同的部门和企业往往需要相同的或类似的软件，各自开发就会使人力浪费很大。在这种形势下"软件作坊"就应运而生了，许多用户不再是自己开发软件而是购买或定做软件。不过，在这个"软件作坊"时期，开发软件的方法，基本上仍然沿用早期时代形成的个体手工业式的开发方法。

随着计算机系统数量的不断增加，计算机软件库就开始膨胀。但是，在程序运行中发现错误时必须设法改正；当用户的要求有改变时也必须相应地修改程序；当买进新硬件或操作系统更新版本时通常必须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"出现了！1968年北大西洋公约组织的计算机科学家曾在当时的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出现了。

计算机系统发展的第四代从70年代初期开始，一直沿续到80年代。这个时期硬件发展的特点是从集成电路计算机进步到大规模和超大规模集成电路计算机，高性能低成本的微处理机大量涌现，发展日新月异。

这个时期计算机应用又有了进一步的普及和深化，开辟了许多新的应用领域。分布式系

统使用多台计算机共同解决一个问题，各台计算机并行工作分别完成各自的子任务，同时各台计算机之间进行必要的通信，增加了计算机系统的复杂程度；利用计算机硬件提供的简单算术运算和逻辑运算能力，模拟人类复杂的思维过程的人工智能系统，更需要十分复杂的计算机软件才能实现。

硬件的迅速发展已经远远超出人们所能提供的软件支撑能力，然而，硬件只提供了潜在的计算和逻辑能力，如果没有软件来驾驭和开发这种能力，人类并不能有效地使用计算机。在计算机系统发展的第四代，软件危机进一步加剧了。软件维护费用占数据处理总预算的50%以上，软件开发生产率远远满足不了对于新系统的需求。为了对付日益严重的软件危机，计算机科学家和软件工程师开始认真研究和采用软件工程学。

那么，什么是软件危机的含义呢？

1.1.2 软件危机的含义

软件危机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，以满足对软件日益增长的需求；如何维护数量不断膨胀的已有软件，以延长这些软件的使用寿命。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不准确。实际成本比估计成本高出一个数量级，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. 软件产品的质量不稳定。软件可靠性和质量保证的确切的定量概念才提出不久，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的和必不可少的。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。

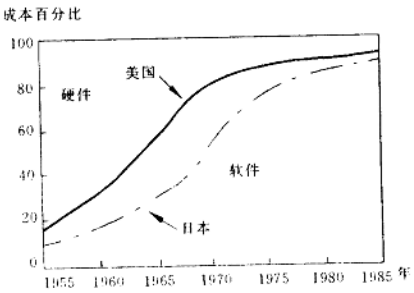

图 1-1 美、日两国计算机硬件和软件成本的变化趋势

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，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。图 1-1 描绘了美、日两国计算机硬件成本和软件成本在计算机系统总成本中所占比例逐年变化的情况，可以看出在 1985 年软件成本已占总成本的 90%左右。

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"的现象使得人类不能充分利用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。那么，为什么会发生软件危机呢？

1.1.3 软件危机的原因

在软件开发和维护的过程中存在的许多严重问题，一方面与软件的特点有关，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，则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，并根据修改后的设计重新编写有关的程序，这就在客观上使得软件较难维护。

软件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过程中，也确实积累和总结了许多成功的经验。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；可是，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做法的形成，可以归因于在计算机系统发展的早期时代软件开发的个体化特点，今天这些错误认识仍然迷惑着不少软件人员。因此，为树立正确的概念来指导软件开发和维护工作，有必要对若干主要的错误认识做一番剖析，在下面的叙述中引号内是典型的错误认识，在每个错误认识后面的论述是对它的剖析。

"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，对用户需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，对用户需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。

“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，通常把软件经历的这个时期称为生存周期。正如上面讲过的，软件开发最初的工作是问题定义，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一致的看法。经过上述软件定义时期的准备工作后才能进入开发时期，在开发时期首先需要对软件进行设计（通常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，程序编完之后还必须经过大量的调试工作（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一般只占软件开发全部工作量的 20%左右。

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，应该肃清只重视程序而忽视软件配置的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，因此付出的代价增加很快；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付出的代价高 2～3 个数量级。图 1-2 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 1-3 是美国贝尔实验室统计得出的定量结果。

“软件投入生产性运行以后需要的维护工作并不多，而且维护是一种很容易做的简单工作。”

由于有这种看法，给维护分配的预算往往比较少，配备的人员也比较弱；然而，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，并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

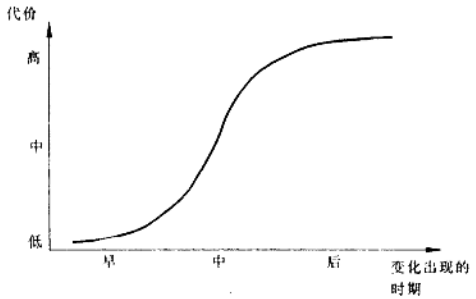

图 1-2 同一变动需付出的代价随时间变化的趋势

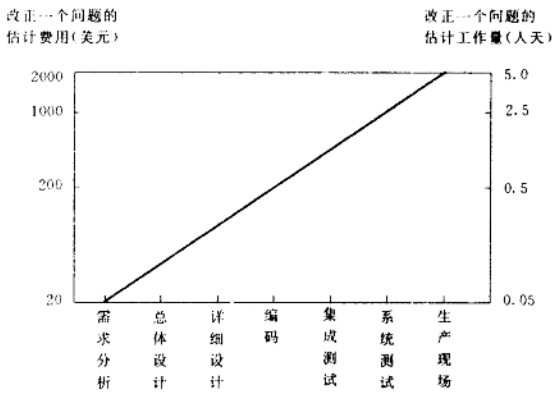

图 1-3 在不同时期改正同一问题需付出的代价

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，还仅仅是消除软件危机的开始，要全面消除软件危机必须采取一系列的综合措施。

1.1.4 消除软件危机的途径

首先必须认识到，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。必须充分吸收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(例如，结构程序设计技术)，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组合成一个整体，支持软件开发的全过程，则称为软件工程支撑环境。

总之，为了消除软件危机，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科学。采用工程的概念、原理、技术和方法来开发和维护软件，把经过时间考验而证明是正确的管理技术和当前能够得到的最好的技术方法相结合，这就是软件工程。

1.2.1 软件工程的基本原理

自从 1968 年在联邦德国召开的国际会议上正式提出并使用了“软件工程”这个术语以来，研究软件工程的专家学者们陆续提出了一百多条关于软件工程的准则或“信条”。著名的软件工程专家 B. W. Boehm 综合这些学者们的意见并总结了 TRW 公司多年开发软件的经验，于 1983 年在一篇论文中提出了软件工程的七条基本原理。他认为这七条原理是确保软件产品质量和开发效率的原理的最小集合。这七条原理是互相独立的，其中任意六条原理的组合都不能代替另一条原理，因此，它们是缺一不可的最小集合，然而这七条原理又是相当完备的，人们虽然不能用数字方法严格证明它们是一个完备的集合，但是，可以证明在此之前已经提出的一百多条软件工程原理都可以由这七条原理的任意组合来蕴含或派生。

下面简要介绍软件工程的七条基本原理：

1. 用分阶段的生存周期计划严格管理

有人经统计发现，在不成功的软件项目中有一半左右是由于计划不周造成的，可见把建立完善的计划作为第一条基本原理是吸取了前人的教训而提出来的。

在软件开发与维护的漫长的生存周期中，需要完成许多性质各异的工作。这条基本原理意味着，应该把软件生存周期划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件的开发与维护工作进行管理。Boehm 认为，在软件的整个生存周期中应该制定并严格执行六类计划，它们是项目概要计划，里程碑计划，项目控制计划，产品控制计划，验证计划，运行维护计划。

不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划。

2. 坚持进行阶段评审

当时已经认识到，软件的质量保证工作不能等到编码阶段结束之后再进行。这样说至少有两个理由：第一，大部分错误是在编码之前造成的，例如，根据 Boehm 等人的统计，设计错误占软件错误的 63%，编码错误仅占 37%；第二，错误发现与改正得越晚，所需付出的代价也越高(参见图 1.2 和图 1.3)。因此，在每个阶段都须进行严格的评审，以便尽早发现在

软件开发过程中所犯的错误，是一条必须遵循的重要原则。

3. 实行严格的产品控制

在软件开发过程中不应随意改变需求，因为改变一项需求往往需要付出较高的代价。但是，在软件开发过程中改变需求又是难免的，由于外部环境的变化，相应地改变用户需求是一种客观需要，虽然不能硬性禁止客户提出改变需求的要求，而只能依靠科学的产品控制技术来顺应这种要求。也就是说，当改变需求时，为了保持软件各个配置成分的一致性，必须实行严格的产品控制，其中主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分（各个阶段产生的文档或程序代码）。基准配置管理也称为变动控制：一切有关修改软件的建议，特别是涉及到对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。绝对不能谁想修改软件（包括尚在开发过程中的软件），就随意进行修改。

4. 采用现代程序设计技术

从提出软件工程的概念开始，人们一直把主要精力用于研究各种新的程序设计技术。60年代末提出的结构程序设计技术，已经成为绝大多数人公认的先进的程序设计技术。以后又进一步发展出各种结构分析（SA）与结构设计（SD）技术。实践表明，采用先进的技术既可提高软件开发的效率，又可提高软件维护的效率。

5. 结果应能清楚地审查

软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员（或开发小组）的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评价和管理。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查。

6. 开发小组的人员应该少而精

这条基本原理的含义是，软件开发小组的组成人员的素质应该好，而人数则不宜过多。开发小组人员的素质和数量是影响软件产品质量和开发效率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为N时，可能的通信路径有 $N(N-1)/2$ 条，可见随着人数N的增大，通信开销将急剧增加。因此，组成少而精的开发小组是软件工程的一条基本原理。

7. 承认不断改进软件工程实践的必要性

遵循上述六条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产，但是，仅有上述六条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步。因此，Boehm 提出应把承认不断改进软件工程实践的必要性作为软件工程的第七条基本原理。按照这条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验，例如，收集进度和资源耗费数据，收集出错类型和问题报告数据等等。这些数据不仅可以用来评价新的软件技术的效果，而且可以用来指明必须着重开发的软件工具和应该优先研究的技术。

1.2.2 生存周期方法学

在软件开发与维护的实践过程中如何落实上述的软件工程基本原理呢？软件工程传统的基本途径是采用生存周期方法学，生存周期方法学的基本内容可概述如下：

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70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战略就是"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战略。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检查。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有明显提高。

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，以有利于软件开发工程的组织管理。粗略地说，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。下面的论述主要是针对应用软件，对系统软件也基本适用(对系统软件的需求往往较易确定)。

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。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。通常把软件定义时期所完成的工作称为系统分析，系统分析工作一般由高层次的技术人员——系统分析员承担。

开发时期具体设计和实现在前一个时期所定义的软件，这个时期通常由下述四个阶段组成：总体设计（也称为初步设计或概要设计），详细设计，编码与单元调试，综合调试。其中前两个阶段又称为系统设计，后两个阶段又称为系统实现。

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。

1.2.3 生存周期阶段的典型划分

前面讲过，软件生存周期阶段的划分方法应该根据软件项目的具体情况决定。下面介绍的生存周期阶段典型划分方法，是根据在整个生存周期中应该完成的任务的性质来划分阶段的。根据任务性质，可以把软件生存周期划分成八个阶段：

1. 问题定义

这个阶段应该完成的基本任务，是通过调研搞清楚用户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，然而在实践中它却可能是最常被忽视的一个步骤。

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，澄清含糊不清的地方，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。

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决，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更多的人力物力，及时终止不值得投资的工程项目，可以避免更大的浪费。

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的要求，更不知道怎样利用计算机解决他们的问题；软件开发人员知道怎样用软件实现人们的要求，但是，对特定用户的具体要求并不完全清楚。因此，系统分析员在需求分析阶段必须和用户密切配合，充分交流信息，以得出经过用户确认的系统逻辑模型。通常用数据流图、数据字典和简要的算法描述表达系统的逻辑模型。

在需求分析阶段确定的系统逻辑模型是以后设计和实现目标系统的基础，必须准确完整地体现用户的要求。系统分析员通常都是计算机软件专家，技术专家一般都喜欢很快着手进行具体设计，然而，一旦分析员开始谈论程序设计的细节，就会脱离用户，使他们不能继续提出他们的要求和建议。软件工程使用的结构分析设计的方法为每个阶段都规定了特定的结束标准，需求分析阶段必须提出完整准确的系统逻辑模型，经过用户确认之后才能进入下一个阶段，这就可以有效地防止和克服急于着手进行具体设计的倾向。

4. 总体设计

总体设计也叫概要设计或初步设计，这个阶段必须回答的关键问题是：“概括地说，应该如何解决这个问题？”

首先，应该考虑几种可能的解决方案。例如，目标系统的一些主要功能是用计算机自动完成还是用人工完成；如果使用计算机，那么是使用批处理方式还是人机交互方式；信息存储使用传统的文件系统还是数据库；……等等。通常至少应该考虑下述几类可能的方案：

(1) 低成本的解决方案。这类系统只能完成必须的工作，不能多做一点额外的工作。

(2) 中等成本的解决方案。这样的系统不仅能够很好地完成预定的任务，使用起来很方便，而且可能还具有用户没有具体指定的某些功能和特点。虽然用户没有提出这些具体要求，但是系统分析员根据自己的知识和经验断定，这些附加的能力在实践中将证明是很有价值的。

(3) 高成本的“十全十美”系统。此系统具有用户可能希望有的所有功能和特点。

系统分析员应该使用系统流程图或其他工具描述每种可能的系统，估计每种方案的成本和效益，还应该在充分权衡各种方案的利弊的基础上，推荐一个较好的系统（最佳方案），并且制定实现所推荐的系统的详细计划。如果用户接受分析员推荐的系统，则可以着手完成本阶段的另一项主要工作。

上面的工作确定了解决问题的策略以及目标系统需要哪些程序，可是，怎样设计这些程序呢？结构设计的一条基本原理就是程序应该模块化，也就是一个大程序应该由许多规模适中的模块按合理的层次结构组织而成（模块就是程序对象的有名字的集合，例如，子程序、函数、宏等）。因此，总体设计的第二项主要任务就是设计软件的结构，也就是确定程序由哪些模块组成以及模块间的关系。通常用层次图或结构图描绘软件的结构。

5. 详细设计

总体设计阶段以比较抽象概括的方式提出了解决问题的办法，详细设计阶段的任务就是把解法具体化，也就是回答下面这个关键问题：“应该怎样具体地实现这个系统呢”？

这个阶段的任务还不是编写程序，而是设计出程序的详细规格说明。这种规格说明的作用很类似于其它工程领域中工程师经常使用的工程蓝图，它们应该包含必要的细节，程序

员可以根据它们写出实际的程序代码。

通常用 HIPO 图(层次图加输入/处理/输出图)或 PDL 语言(过程设计语言)描述详细设计的结果。

6. 编码和单元调试

这个阶段的关键任务是写出正确的容易理解容易维护的程序模块。

程序员应该根据目标系统的性质和实际环境,选取一种适当的高级程序设计语言(必要时用汇编语言),把详细设计的结果翻译成用选定的语言书写的程序,并且仔细调试编写出的每一个模块。

7. 综合测试

这个阶段的关键任务是通过各种类型的测试(及相应的调试)使软件达到预定的要求。

最基本的测试是集成测试和验收测试。集成测试是根据设计的软件结构,把经过单元调试的模块按某种选定的策略装配,在装配过程中对程序进行必要的测试检验。所谓验收测试则是按照规格说明书的规定(通常在需求分析阶段确定),由用户(或在用户积极参加下)对目标系统进行验收。

必要时还可以再通过现场测试或平行运行等方法对目标系统进一步测试检验。

为了使用户能够积极参加验收测试,并且在系统投入生产性运行以后能够正确有效地使用这个系统,通常需要以正式的或非正式的方式对用户进行培训。

通过对软件测试结果的分析可以预测软件的可靠性;反之,根据对软件可靠性的要求也可以决定测试和调试过程什么时候可以结束。

应该用正式的文档资料把测试计划、详细测试方案以及实际测试结果保存,作为软件配置的一个组成成分。

8. 软件维护

维护阶段的关键任务是,通过各种必要的维护活动使系统持久地满足用户的需要。

通常有四类维护活动:改正性维护,也就是诊断和改正在使用过程中发现的软件错误;适应性维护,即修改软件以适应环境的变化;完善性维护,即根据用户的要求改进或扩充软件使它更完善;预防性维护,即修改软件为将来的维护活动预先做准备。

虽然没有把维护阶段进一步划分成更小的阶段,但是,每一项维护活动都应该经过提出维护要求(或报告问题),分析维护要求,提出维护方案,审批维护方案,确定维护计划,修改软件设计,修改程序,测试程序,复查验收等一系列步骤,因此,维护活动实质上是经历了一次压缩和简化了的软件定义和开发的全过程。

每一项维护活动都应该准确地记录,作为正式的文档资料加以保存。

1.2.4 瀑布模型

前面两小节分别介绍了生存周期方法学的要点和典型的阶段划分方法,同时也简要地叙述了生存周期每个阶段应完成的基本任务。把生存周期划分为阶段的目的与实质是:便于控制开发工作的复杂性;通过一定的有限步骤,把用户需要解决的问题从抽象的逻辑概念逐步转化为具体的物理实现。

从前面的论述可以知道,按照传统的生存周期方法学开发软件,各个阶段的工作自顶向下从抽象到具体顺序展开,好象奔流不息的瀑布,总是从高处的台阶依次流向低处的台

阶。因此，传统的生存周期方法学可以用瀑布模型（Waterfall model）来模拟，如图 1-4 所示。

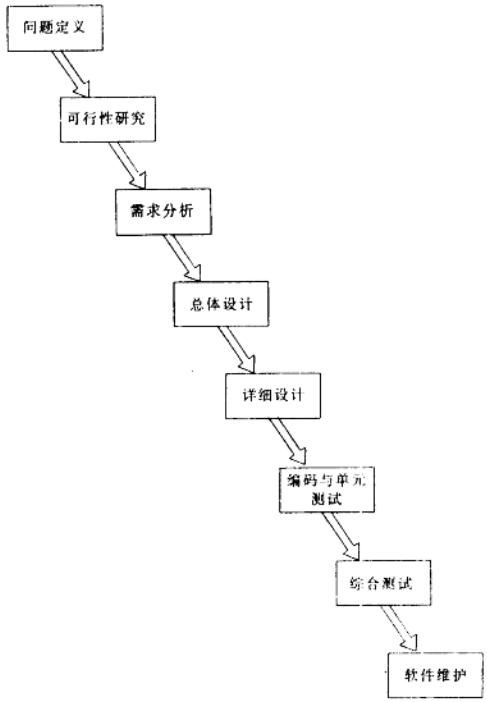

图 1-4　瀑布模型

按照传统的瀑布模型来开发软件，有如下几个特点：

1. 阶段间具有顺序性和依赖性

这个特点有两重含义：第一，必须等前一阶段的工作完成之后，才能开始后一阶段的工作；第二，前一阶段的输出文档就是后一阶段的输入文档，因此，只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。可见，万一在生存周期某一阶段发现了问题，很可能需要追溯到在它之前的一些阶段，必要时还要修改前面已经完成的文档。然而，在生存周期后期改正早期阶段造成的问题，需要付出很高的代价。这就好象水已经从瀑布顶部流泻到底部，再想使它返回到高处需要付出很大能量一样。

2. 推迟实现的观点

缺乏软件工程实践经验的软件开发人员，接到软件开发任务以后常常急于求成，总想尽早开始编写程序。但是，实践表明，对于规模较大的软件项目来说，往往编码开始得越早最终完成开发工作所需要的时间反而越长。这是因为，前面阶段的工作没做或做得不扎实，过早地考虑程序实现往往导致大量返工，有时甚至发生无法弥补的问题，带来灾难性后果。

瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本

任务规定，在这两个阶段只考虑目标系统的逻辑模型，不涉及软件的物理实现。因此，直到设计阶段结束，所完成的文档都是独立于设备的抽象的逻辑模型。

清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。

3. 质量保证的观点

软件工程的基本目标是优质、高产。为了保证所开发的软件的质量，在瀑布模型的每个阶段都应坚持两个重要做法：

第一，每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。完整、准确的合格文档不仅是软件开发时期各类人员之间相互通信的媒介，也是运行时期对软件进行维护的重要依据。

第二，每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。事实上，越是早期阶段犯下的错误，暴露出来的时间就越晚，排除故障改正错误所需付出的代价也越高。因此，及时审查，是保证软件质量，降低软件成本的重要措施。

以上三点是隐含在软件生存周期各阶段后面的观点和指导思想，是比具体任务更重要更根本的东西。只有掌握了这些指导思想，才能在软件开发中更自觉更主动，才能更好地运用软件工程方法学完成软件开发与维护的艰巨工作。

1.3 变革软件开发方法

为了对付日益加剧的软件危机，在本世纪 60 年代末开始出现了一门新兴的工程学科——软件工程。软件工程的传统途径是生存周期方法学，这个方法学的本质是在实质性的软件开发工作开始之前，通过需求分析预先定义软件需求，然后一个阶段接着一个阶段有条不紊地开发用户所要求的软件，实现预先定义的软件需求。生存周期方法学相对于早期时代的只重视程序设计，轻视对用户需求的了解和分析，最终产品只有程序代码没有相应的文档资料的个体化的"软件"开发方法来说，是一个巨大的进步，对实现软件生产的工程化曾经起了重要的促进作用，部分地缓解了软件危机。

使用生存周期方法学指导软件开发工作，确实有许多成功的典型范例。例如，1971 年 IBM 公司在纽约时报信息库管理系统和稍后在美国宇航局空间实验室飞行模拟系统的开发过程中，使用结构程序设计技术和主程序员组的组织方式，获得圆满成功。这两个系统都相当庞大，前一个系统包含 8.3 行高级语言的源程序，后一个系统包含 40 万行源程序，而且在开发过程中用户的要求曾经有过很多变化，然而这两个系统的开发工作都按预定时间高质量地完成了。一些规模更庞大，包含几百万行甚至几千万行源程序，工作量达几千人年的巨型、超巨型的软件系统，在生存周期方法学的指导下也成功地开发出来了。国内也有许多用软件工程方法成功地开发软件的范例。

但是，实践表明，传统的生存周期方法学并不能完全消除软件危机。生存周期方法学仍然有许多不足之处，对某些类型的软件比较适用，对另一些类型的软件则完全不适用。软件工程这门新兴学科需要不断发展和完善，正如 Boehm 提出的软件工程第七条原理所指出的那样，必须承认不断改进软件工程实践的必要性。近年来软件工程的新途径——快速原型法——应用得越来越广泛了。

1.3.1 生存周期方法学的不足

生存周期方法学有许多不足之处，最主要的不足之处可归纳为以下几点：

1. 开发效率提高的幅度远远不能满足需要

生存周期方法学强调需求分析的重要性，强调在每个阶段结束之前必须进行评审，从而提高了软件开发的成功率减少了重大返工的次数；开发过程中实行严格的质量管理，采用先进的技术方法和软件工具，也都加快了软件开发的速度。采用生存周期方法学确实提高了许多软件的开发效率，但是，实践表明开发效率的提高仍然很有限，提高的幅度远远赶不上对软件产品的需要。例如，一份统计资料表明，如果以 1955 年美国的软件生产率为 1，则 1965 年的软件生产率增长到 2.0，1975 年增长到 2.7，1985 年增长到 3.6。也就是说，经过 30 年美国的软件生产率只翻了将近两番。

但是 随着计算机应用的不断普及，人们对计算机软件的需要量急剧增加。仅仅考虑随着计算机硬件数量不断增加，为了使这些计算机充分运行所需增加的软件数量就是十分惊人的。J. Martin 在他编著的关于第四代语言的书中曾经做过如下的粗略估计：

假设需要为之编写新程序的计算机的数量每年递增 25%，在未来十年中计算机的运行速度将提高 40 倍，则十年后一年应编写的程序量比现在多 372 倍左右。现在美国全国约有 40 万名程序员，如果十年后他们仍然以同样方式编写程序，生产率仍然保持今天的水平，则到那时美国需要 1.48 亿名程序员，也就是说，每一个美国人不论男女老少都要当程序员，才能使每台计算机都充分利用。

尽管 J. Martin 的估算十分粗略，所得到的数字与实际情况可能相差甚远，但它所反映出的问题和所揭示的趋势却是正确的，是应该高度重视的。

由于人们现在开发软件的效率仍然很低，所以，当用户需要计算机应用系统时，他们并不能及时得到所需要的系统。从用户提出要求到他们最后得到所需要的目标系统，往往需要经过几年的延迟时间。

计算机是迄今为止人类所发明的最灵活的机器，适用于极其广泛的应用领域。计算机的成本正在逐年迅速下降，它的性能不断提高，应该尽可能充分地使用计算机的能力，以提高各行各业的工作效率。但是，当今计算机的巨大潜力并未得到充分利用，企业经理不能从他们的计算机系统及时获得所需要的信息，许多应当在计算机辅助下做出的决策事实上仍然是用人工方法根据很不充分甚至是很不适当的信息做出的。在激烈的市场竞争中，企业还没能在辅助决策、管理、设计、制造等诸多方面充分发挥计算机的巨大潜力。

问题不是由于计算机硬件的缺陷造成的，而是因为在建立应用系统的方法不当造成的。传统的“应用系统开发生存周期”是缓慢的和僵硬的。事实表明，这种开发方法对提高软件开发效率的作用是有限的。即使在管理得最好的企业中，对新应用系统的需求的增长数量也超过了数据处理部门所能提供的数量。由于供求之间的不平衡越来越严重，最终用户所需要的应用系统许多都不能及时进行开发，积压的等待开发的应用系统的数量越来越多，据统计，美国绝大多数公司登记在册的待开发的应用系统的积压时间已达两至四年，某些积压最严重的公司积压时间长达七年。除非找到并使用了更好的开发应用系统的方法，否则随着计算机硬件价格下降和越来越多的潜在用户有了可以使用的计算机，这种现象还会变得更加严重。

积压期过长和计算中心等数据处理部门不能迅速响应最终用户需求的现象，使得最终用户灰心丧气，对应用计算机解决他们的问题失掉信心和耐心，许多最终用户甚至不再向数据处理部门提出开发他们所急需的很有价值的应用系统的要求，以至许多企业中还存在许多没有记录在册的不可见的积压。不可见的积压的数量很难度量，通常比文档资料中已经记录在册的积压的数量还要多得多。

最终用户急需的应用系统不能及时进行开发，通常被积压数年；即使已经着手开发，由于需要经过漫长的开发期，又会再延迟很长时间才能得到可在计算机上运行的产品。这时，外部环境和市场状况早已发生了巨大变化，最终用户得到的应用系统自然不能完全满足现在的需要，从而使得企业在激烈的市场竞争中丧失许多宝贵的机遇（关于这个问题，下面还要详细阐述）。因此，软件开发效率低是一个十分严重的问题，必须认真对待，努力设法解决。

2. 用传统方法开发出的应用系统很难维护

传统的生存周期方法学强调文档资料的重要性，要求最终的软件产品由完整、一致的配置成分组成；在开发过程中使用结构分析、结构设计和结构程序设计等先进技术，强调软件的可读性、可修改性和可测试性应是软件的重要质量目标。因此，对用生存周期方法学开发出软件所进行的维护属于结构化维护的范畴，可维护性较之用早期个体化方法开发出的软件有明显提高，使得软件从不能维护变成可以维护。

但是，即使是用生存周期方法学开发出的软件，仍然是很难维护的，维护成本仍然很高。统计数字表明，软件维护的生产率大约比软件开发的生产率低 40 倍。

为什么用传统的生存周期方法学开发出的软件仍然很难维护呢？这是因为，传统的瀑布型软件开发方式本质上是自顶向下线性进行的，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 100 至 1000 倍（参见图 1-3）。

维护问题使得数据处理部门和软件开发组织所面临的问题更严重、所处的状况更糟。维护就是调整和重写已经存在的系统，以便适应新的需求或使得它们能与变化了的系统资源一起工作。由于分别开发的程序不能协同工作，或因为在把数据从一个系统传送给另一个系统时存在接口问题，因此，常常需要重新进行程序设计。在一个程序中所做的变化，通常会引起一串程序做相应的修改。因此，维护量将随着已有程序量的增长而呈上升的趋势。除非精心控制，否则程序之间的相互作用大约与程序数目的平方成比例地增长。另一个严重的维护问题，是需要花费许多时间去查找并改正软件中存在的故障。这些故障是在开发过程造成的，在生产运行过程中才暴露出来。

由于必须花费大量的人力物力去维护已有的软件，自然极大地削弱了人们开发新软件的能力。维护已有软件的沉重负担，已经严重地妨碍了用户所需要的新应用系统的实现，使应用系统开发积压的现象进一步恶化。在许多数据处理部门，维护活动消耗了高达 75% 的人力资源。据报导，80 年代美国一年所花费的软件维护费用高达 300 亿美元。

更严重的问题是，用传统方法和常规程序设计语言（例如，COBOL）所开发的应用系统修改非常困难，以至不能及时快速地修改已有应用系统以适应商业环境和企业经营策略的不断变化，不能在激烈的市场竞争中充分发挥计算机在辅助决策和经营管理中的作用，这自然会使企业的经济利益受到严重损害，必然引起企业高层领导人的高度重视。

3. 用传统方法开发出的应用系统常常不能真正满足用户的需要

传统的生存周期方法学有一个致命的弱点，那就是在许多情况下传统的软件开发过程

并不能取得成功，所开发出的应用系统常常不能真正满足用户的需要。

有些经过几年艰苦努力开发出来的应用系统，当把它们交付给用户使用时，用户说这并不是他们所需要的系统；或者最终用户试用开发出的应用系统一小段时间之后就不再使用该系统，把它丢弃到一边了。也有一些新开发出的应用系统，用户使用几个星期后提出许多意见，认为他们需要的是与此不同的系统。据报导，在美国所开发的软件系统中，真正符合用户需要并投入使用的约占总数的 1/4，另外 1/4 中途夭折，剩下的一半虽然完成了开发过程，但用户并未使用，在这约占半数的舍弃不用的软件中，很大一部分是因为它们未能真正满足用户的需求。

为什么会出现这种可悲的结果呢？前面我们已经反复讲过，生存周期方法学本质上是自顶向下一个阶段一个阶段地顺序进行软件开发，前一个阶段任务的正确完成是开始进行后一个阶段工作的前提和基础。特别是，只有当系统分析员能够正确地完成需求分析工作，提出完整准确的需求时，软件开发才有可能得出预期的正确结果。传统的生存周期方法学实质上是预先定义需求的方法，在实际进行软件开发之前必须预先对需求进行严格定义，准确地规定好所要开发的是一个什么样的系统，然后按照对目标系统的定义实现该目标系统。如果对目标系统定义不正确，即使开发工作进行得再好，最终实现的系统也不是用户所需要的系统。

预先定义软件需求的目的是为了提高软件开发的成功率，与只重视编程忽视需求分析的早期个体化方法相比是一个重大进步。但是，实践表明，在系统建立起来之前往往很难仅仅依靠分析就确定出一套完整、准确、一致、有效的应用需要，这种预先定义需求的方法更不能适应用户需求不断变化的情况：

(1) 某些类型的系统其需求很难预先定义

遵循传统的生存周期方法学开发软件，正确地定义需求是系统成功的关键。预先定义需求的策略是假设最终用户没有在计算机上实际使用软件的工作经验，就能预先精确地提出全部系统需求。对于某些类型的软件系统，例如，操作系统和编译系统等系统软件，或者导弹控制系统和生产控制系统等控制软件，其需求经仔细分析之后可以预先指定；对于数量占多数的更常用的一些应用系统，其需求往往很难预先准确地指定。也就是说上述假设只对某些软件成立，对于多数软件并不成立。许多用户对他们的需求最初只有模糊笼统的概念，要求一个只有初步设想的人准确无误地说出全部需求显然是不切实际的。人们为了充实和细化他们的设想，通常需要经过在某个能运行的系统上的实践过程。

在商业数据处理领域，经常遇到需求不能预先定义的情况。例如，管理信息系统(MIS)是一种最重要、最常用的数据处理系统，然而这类系统的需求是很难预先定义的。

使用传统的开发途径，MIS 的开发者应该向企业经理询问他们需要什么信息，然而，花费相当长时间之后他们发现绝大多数经理实际上并不能准确地说出他们需要哪些信息。为了确切地说出需要什么信息，一位总经理必须知道他现在应该做出的以及将来将要做出的每种类型的决策，以及怎样做出决策。有些总经理为了保险起见，要求 MIS 系统能够向他提供一切信息，为了满足这类总经理的要求，某些开发者曾经开发出可以在庞大的报表中提供每一条信息的系统，但是，这样做的结果恰恰是掩盖了而不是揭示出了为做出决策所需要的少量关键信息。

有时，某些个性很强的经理很肯定地提出了他或他主管的部门所需要的信息，系统分