第 1 章
认识软件工程

现如今各类应用软件已经普遍深入商业、文化和日常生活等各个方面。软件在当今信息社会中占有重要的地位。然而，如何在有限的时间内，利用有限的资金开发出高质量的软件，一直是软件开发者面临的难题。软件工程的目的就是通过适当的成本控制能够高效地开发出满足用户需求的软件产品。

1.1 软件

一个完整的计算机系统由硬件和软件两部分组成。本节主要介绍软件的相关概念。

1.1.1 软件的定义

《信息技术软件工程术语》(GB/T 11457—2006)国家标准中对计算机软件的定义为：与计算机系统的操作有关的计算机程序、规程、规则，以及可能有的文件、文档及数据。

在理解软件的含义时，不能片面地认为软件就是计算机程序。必须认识到，程序只是完整的软件产品的一个组成部分。在软件开发的每个阶段，都要得出最终产品的一个或几个组成部分(可能是半成品)，这些组成部分或者是文档资料，或者是程序，或者是数据。也就是说，一个软件产品必须由一个完整的配置组成，其中主要包括程序、文档和数据等成分。必须摒弃只重视程序而忽略文档和数据的错误观念。

因此，软件的正确含义应该包括以下内容。

(1) 指令的集合(计算机程序)。通过执行指令能够满足预期的特征、功能和性能需求。

(2) 数据结构。程序处理的原材料，经处理能变成所需的信息。

(3) 文档。用以描述程序功能需求以及程序如何操作和使用。

简单地说：软件＝程序＋数据＋文档。

从广义上讲，社会结构中的管理系统、思想意识形态、法律法规等也属于软件范畴。这里所说的软件，特指计算机软件。

1.1.2 软件的特点

与其他硬件一样(如机械器件)，软件也是一种产品，只不过是一种逻辑产品而非物理产品。

与硬件相比，软件具有以下特点。

1. 生产方式不同

软件可进行开发，不是传统意义上的硬件制造。

(1) 软件的指令表现为代码，也可以说软件的“原材料”是代码，而硬件的原材料是物理材料。因此，硬件制造阶段中的质量问题对于软件来说是不存在的或者易于纠正的。但是，程序是由代码组合而成的，实现同样的功能，不同的设计人员会有不同的设计方案。所以，软件开发不是传统意义上的硬件制造，而是一种创新、开发过程。

(2) 二者都需要构建产品，但是构建方法不同。在硬件设计中，多采用标准件，构件复用是工程进程中通用的方法，而在软件设计中，大规模的复用才刚刚开始尝试。

(3) 硬件试制成功后，批量生产需要建生产线，投入大量的人力、物力和资金，生产过程中要进行产品的质量控制，对每件产品进行严格的检验；而软件由于存储在光、电、磁等介质上，所以开发成功之后，只需对原版软件进行复制即可。

软件产品成本主要在于开发设计，因此不能像管理制造项目那样管理软件开发项目。

2. 失效率不同

硬件在早期具有相对较高的失效率(这种失效率通常来自设计或生产缺陷)。当缺陷被逐个纠正之后，失效率也降低，并在一段时间内保持平稳(理想情况下很低)。然而，随着时间推移，因为灰尘、震动、使用不当、温度超限以及其他环境问题所累积的硬件组件损耗将再次抬高失效率。简而言之，硬件开始“磨损”了。图 1.1 描述了以时间为变量的硬件的失效率。

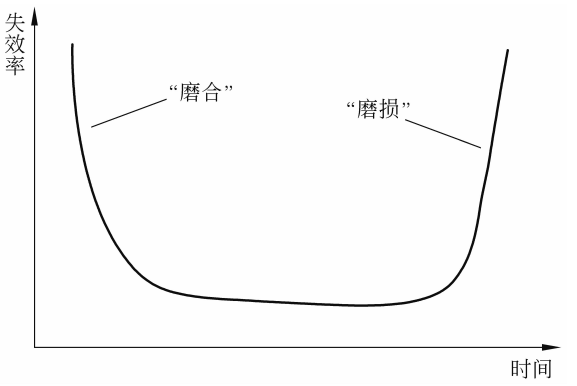

图 1.1　硬件失效率曲线

软件是不会受到此类问题影响的。因此，从理论上说，软件的失效率曲线应该呈现为图 1.2 所示的“理想曲线”。在软件生命周期的前期，未知的缺陷将造成高失效率。然而随着错误被纠正(理想情况下，不会引入新的错误)，曲线将如图 1.2 所示，趋于平缓。

“理想曲线”只是软件实际失效模型的粗略简化。实际上，软件虽然不会磨损，但是退化的确存在。在完整的生命周期里，软件的每次变更都可能引入新的错误，使失效率曲线陡然上升。在曲线回到最初的稳定状态前，新的变更会引起曲线又一次上升。就这样，最小的失效率水平逐渐上升，可以说，不断的变更是软件逐步失效的根本原因。

磨损的另一方面同样说明了软硬件的不同。磨损的硬件部件可以用备用的构件替换，而软件却不存在备用构件。每个软件的错误都暗示了设计的缺陷或者在从设计转化为机器

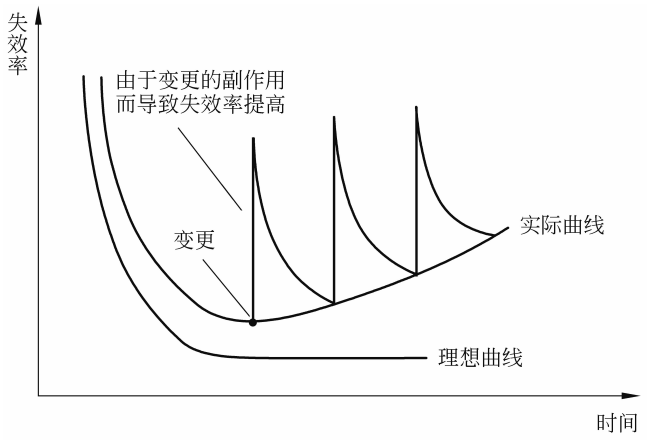

图 1.2　软件失效的曲线

可执行代码的过程中产生的错误。因此，软件维护比硬件维护更为复杂，所需的工作量非常大。一般来说，大型软件的维护成本高达开发总成本的 4 倍左右。目前，软件开发组织把 60％以上的工作量用于软件维护上。

3. 复杂性不同

软件是一种逻辑实体，开发软件在技术、管理等多方面都比硬件复杂得多，需要投入大量、高强度的脑力劳动，成本高、风险大。

4. 精度不同

硬件产品允许有误差，而软件产品却不允许有误差，程序的每个语句的语法和语义必须完全的正确。

5. 要求不同

软件不仅要满足客户的功能需求，还要受限于诸多社会因素。许多软件的开发和运行涉及软件客户的机构设置、体制以及管理方式等，甚至涉及人的观念和心理，还涉及软件知识产权及法律等问题。

6. 依附性不同

硬件产品能够独立存在和工作。但是，软件的开发和运行必须依附于软件环境。软件环境由计算机系统硬件、通信网络、支撑软件及所服务的业务领域等要素构成。软件的依附性决定了人们在软件开发过程中必须考虑软件环境对软件的制约和影响。

用户是软件产品的使用者，客户是软件产品的购买者。有时候两者是一致的，客户就是用户；但有时候，两者并不一致。

1.1.3　软件的分类

计算机软件的分类方法很多，本书仅从软件层次、存储方式和规模 3 个方面来划分软件的类别。

1. 按层次划分

(1) 系统软件

系统软件是计算机系统中最靠近硬件的一层，如操作系统、编译器、设备驱动程序、数据库管理系统、远程通信处理器等。

① 系统软件与计算机硬件紧密配合在一起并大量地进行交互。

② 使计算机系统各个部件、相关软件和数据协调、高效地工作。

③ 其他软件一般都通过系统软件发挥作用。

④ 系统软件与具体的应用领域无关。

(2) 支撑软件

支撑软件位于系统软件和应用软件之间，是辅助和支持开发人员开发与维护应用软件的工具软件，以及可复用软件资源等。其中既包括帮助程序人员开发软件产品的工具，也包括帮助管理人员控制开发进程的工具，如软件开发环境、中间件等。

(3) 应用软件

在支撑软件之上，应用软件是为特定应用领域、行业服务的一类软件，协助业务操作、管理或技术决策，在日常工作和生活中大量使用。除了传统数据处理的应用程序外，应用软件也常被用于业务领域的实时控制(例如，销售点的销售处理、实时制造过程控制等)。

以上 3 类软件的层次关系如图 1.3 所示。

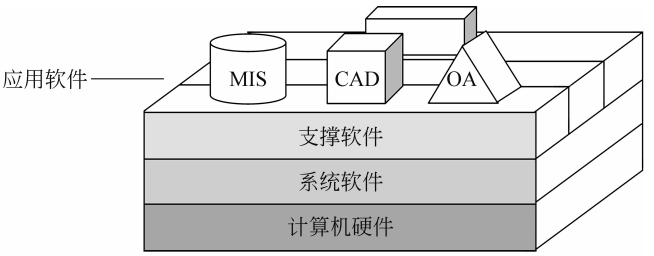

图 1.3　软件的层次关系

2. 按存储方式划分

(1) 一般性软件

一般性软件往往依赖操作系统，存储于磁盘等载体中，如通常在 PC 上使用的各种软件。

(2) 嵌入式软件

嵌入式软件可能不依赖操作系统，也可能依赖特定的嵌入式操作系统。这类软件将微型操作系统与应用软件嵌入在 ROM 或 FLASH 存储器中。嵌入式软件有着广泛的应用，如日常生活中使用的数字电视机、机顶盒、数码相机、DVD、音响设备、可视电话、家庭网络设备、洗衣机、电冰箱、智能玩具等。

3. 按规模划分

软件按规模(人力、时间、源程序行数)划分可分为 6 类，如表 1.1 所示。

表 1.1　软件按规模划分

类　别	参 加 人 数	研 制 期 限	产品规模(源代码行)
微型	1	1～4 周	500 行
小型	1	1～6 月	1000～2000 行
中型	2～5	1～2 年	5000～50 000 行
大型	5～20	2～3 年	50 000～100 000 行
甚大型	100～1000	4～5 年	1 000 000 行
超大型	2000～5000	5～10 年	1 000 000～10 000 000 行

1.2　软件危机催生软件工程

软件危机催生了软件工程,剖析软件发展史上曾经发生的软件危机,有助于我们理解软件工程的思想和方法。

1.2.1　什么是软件危机

早期的软件通常规模较小,使用的是个体化软件开发方法。随着计算机应用日益普及和深化,软件数量急剧膨胀,软件规模也日趋庞大。但软件开发技术、生产率、质量可靠性、成本控制难以跟上硬件技术的进步,导致了“软件危机”。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,并正式提出“软件工程”的概念。

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。这些问题绝不仅限存在于不能正常运行的软件,实际上,几乎所有软件中都不同程度地存在。由于软件危机的存在,软件的设计、开发和维护人员不得不直面两个问题,即如何开发软件,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。

鉴于软件危机所导致的严重后果和其本身的隐蔽性,近年来有人建议把软件危机更名为“软件萧条(Depression)”或“软件困扰(Affliction)”。不过“软件危机”这个词强调了问题的严重性,而且也已为绝大多数软件工作者所熟悉,所以本书仍沿用它。

1.2.2　软件危机的主要表现

归纳起来,软件危机主要有以下表现。

(1) 软件开发生产率很低,导致软件产品开发时间长。

(2) 对软件开发成本和进度估计不准确,导致实际成本比估计成本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。

根据美国国防部 2003 年报告,仅有 35%的软件项目能够按计划在预算内完成,有 20%以上的软件项目以失败告终。

(3) 软件成本占计算机系统总成本比例逐年上升,导致软件价格十分昂贵。

由于微电子技术的进步和生产自动化程度的不断提高,硬件成本逐年下降。但是,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。美国空军 1955 年所购买软件在整个计算机系统中的费用占比为 18%,1970 年占比为 60%,1985 年则达到 85%。

(4) 软件开发人员在对客户要求没有确切认识的情况下匆忙编写程序，导致最终产品不符合客户的实际需要。

(5) 软件质量保证技术(审查、复审、程序正确性证明和测试)没有坚持应用到软件开发的全过程中，导致软件质量不可靠。

因软件质量不可靠而产生的教训很多。20 世纪 80 年代欧洲“阿丽亚娜”火箭因软件错误发射失败；1996 年美国“阿特拉斯”火箭也因软件故障发射失败；英国 1986 年开发的办公室信息系统 Folios 历经 4 年，因性能达不到要求于 1989 年该项目被取消；日本第 5 代计算机项目因为软件问题在投入 50 亿美元后于 1993 年下马。

(6) 软件没有适当的文档资料，导致软件不可维护。

美国 IBM 公司在 1963 年至 1966 年为 IBM360 机器开发的操作系统，花了大约 5000 人·年的工作量，最多时有 1000 人投入开发工作，写出了近 100 万行源程序。尽管投入了这么多的人力和物力，得到的结果却极其糟糕。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 1000 个程序错误而修正的结果。

该项目的负责人 F. D. Brooks 曾无比沉痛地说：“……正像一个逃亡的野兽落到泥潭中做垂死的挣扎，越是挣扎，陷得越深，最后无法逃脱灭顶的灾难。……程序设计工作正像这样一个泥潭，……一批批程序员被迫在泥潭中拼命挣扎，……谁也没有料到问题竟会陷入这样的困境……”IBM360 机器的操作系统的历史教训已成为软件开发项目的典型事例而被载入史册。

以上列举的仅仅是软件危机的一些典型的表现，与软件开发和维护有关的问题远远不止这些。虽然软件危机的表现各不相同，往往可归结为成本、质量和生产率 3 个方面的问题。

1.2.3 产生软件危机的原因

要消除软件危机，必须找出产生软件危机的原因。从开发人员的角度看，软件危机的原因可以归结为以下两个方面。

1. 客观方面——软件本身的特点

(1) 软件是逻辑实体，质量难以控制

由于软件缺乏“可见性”，软件开发过程的进展情况较难衡量，软件的质量也较难评价，因此，管理和控制软件开发过程相当困难。

(2) 软件不会“用坏”，但软件维护更困难

软件在运行过程中不会因使用时间过长而被“用坏”，如果运行中发现了错误，很可能是在开发时期产生的错误。因此，软件维护通常意味着改正或修改原来的设计，这就在客观上使软件较难维护。

(3) 软件规模庞大，管理比较困难

软件规模庞大，需要许多人合作完成。开发一个高质量的大型软件系统，不仅涉及许多技术问题，更涉及科学而严密的管理。

2. 主观方面——错误观点导致错误做法

(1) 忽视需求分析，软件“文不对题”

错误认识之一：有一个对目标的概括描述就足以着手编写程序了，细节可以在以后再

补充。不少失败的软件，就是因为往往在没有完全搞清楚客户需求之前就仓促上阵所造成的。这正如写命题作文，题意未明就匆匆下笔，结果必然是“下笔千言，离题万里”。

(2) 认为“软件＝程序”，给交流、使用、维护带来极大困难

错误认识之二：软件开发就是编写程序并设法使它运行。

软件生产是一个有序的过程，软件产品不仅包含程序，还包含数据和文档。在软件生产的每个阶段都要产生最终产品的一个或几个组成部分。也就是说，一个软件产品必须由一个完整的配置组成，软件配置主要包括程序、文档和数据等成分。一个没有文档的软件，会给开发人员交流、用户使用、软件维护等带来极大的困难。

(3) 轻视软件维护，付出更大代价

错误认识之三：客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。

应该认识到，软件变动发生在不同阶段需要付出的代价是不相同的。前期的工作相对于后期的工作更抽象、更重要，一旦变动会“牵一发而动全身”。早期变动涉及的面较小，因而代价也比较小；但在中后期，软件配置的许多成分已经完成，此时变动涉及的面大，逻辑复杂，因此付出的代价会更大。软件变动需要付出的代价的变化趋势，如图 1.4 所示。

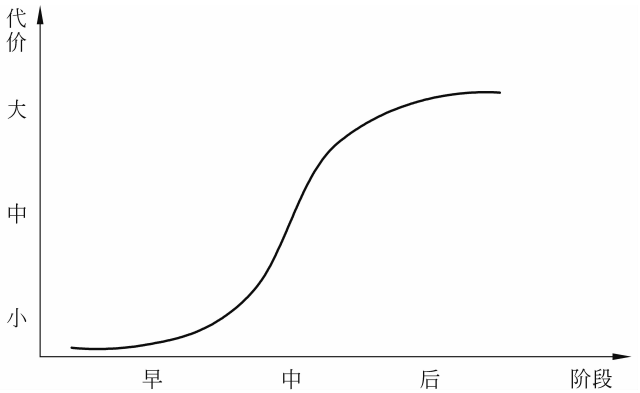

图 1.4　软件变动需要付出的代价的变化趋势

由于观点错误，在软件开发过程中采用了错误的方法和技术，是形成软件危机的主要原因。

1.3　理解软件工程的定义及概念

通过归纳产生软件危机原因的主观因素，并借鉴其他工程领域的成功经验，人们认识到“摆脱软件危机的出路在于软件开发的标准化和工程化”。因此，在 1968 年北大西洋公约组织讨论软件危机问题的国际会议上，计算机科学家提出并使用了“软件工程”这个名词，于是出现了一个新的学科——软件工程学。

1.3.1　软件工程的定义

人们给软件工程下过许多定义，下面列举 3 个典型定义。

- 1968 年 NATO(北大西洋公约组织)在德国召开的学术会议上，Fritz Bauer 给出的最早定义："软件工程是为了经济地获得可靠的和能在实际机器上有效运行的软件，而确立和使用的健全的工程原理。"
- 1983 年 IEEE(美国电气和电子工程师协会)的软件工程定义："软件工程是开发、运行、维护和修复软件的系统方法。"
- 1993 年 IEEE 进一步给出的定义："软件工程是把系统化的、规范的、可量化的方法应用于软件开发、运行和维护过程，也就是把工程化方法应用于软件，并研究相关的途径。"

归纳这些定义，可以概括为软件工程是指导计算机软件开发和维护的一门工程学科。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，以便经济地获得可靠的、高效运行的软件。

简而言之：软件工程＝工程化原理＋管理方法＋技术方法。

1.3.2 理解软件工程的概念

可以从以下 4 个方面理解软件工程的概念。

1. 有系统工程的共性

软件工程与其他领域的工程一样，都统属于系统工程，都采用工程学的思想和方法，具有系统工程的共性。

一般工程学科要探讨和解决的问题有工程原理、工程对象、工程过程、工程方法、工程技术、工程组织和管理、工程质量等。把工程学的理论和方法应用到软件开发中，就形成了软件工程学科。

工程化方法包括过程规范化、文档标准化、方法系统化、管理科学化(量化、高效管理)和开发平台自动化。

软件工程将工程化方法应用于软件开发，使软件开发像硬件一样采用流水线生产、工业化批量生产。

2. 有自己的特性

由于软件是逻辑元素而非物理元素，因此软件工程具有自己的鲜明特性。

软件开发的工程化，较之于机械、纺织、化工、建筑等传统工程更复杂、更困难。实际上软件的工程化到目前远未达到成熟程度。在软件开发过程中，人的能力、技巧、水平等个体差异仍然在很大程度上决定着所开发软件的质量，软件开发还不能完全摆脱人的个体差异和手动方式的制约。

3. 目标是提高效率、降低成本、保证质量

软件工程的目标是在给定成本、进度的前提下，开发出满足客户需求的高质量(可修改性、有效性、可靠性、可理解性、可维护性、可重用性、可适应性、可移植性、可追踪性和可互操作性)的软件产品。

4. 途径是管理＋技术

实现工程化方法开发软件，有以下两条途径。

(1) 管理措施

软件开发不是某种个体劳动的神秘技巧,而是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目。

美国国防部的一项研究表明,65%的项目由于管理不善导致难以控制进度、成本和质量;进一步的研究发现,管理是影响软件项目成功开发的全局性因素,而技术只是局部性因素。

如果软件开发组织不能对软件项目进行有效管理,就不能充分发挥软件开发方法和工具潜力,也就不能高效率地开发出高质量的软件产品。

这是工程化开发软件的主要途径。

(2) 技术措施

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。应该尽快消除在计算机系统早期发展阶段形成的一些错误观点和做法,研究探索更好更有效的方法和技术,开发和使用更好的软件工具。这是工程化开发软件的另一条途径。

1.4　软件工程方法学

软件工程方法学是用来指导软件开发过程的一个技术集合,也称范型。目前使用最广泛的范型是传统方法学和面向对象方法学。

1.4.1　传统方法学

传统方法学诞生于 20 世纪 60 年代末,是为克服软件危机而发展起来的。传统方法学主要有功能分解法、结构化方法和 Jackson 方法,前两者是面向过程(行为)的,后者是面向数据的。其中,结构化方法影响最广、使用最多。因此,传统方法学也称生命周期方法学或结构化范型。

1. 结构化范型的要点

(1) 把软件生命周期划分成若干个阶段

把软件生命周期划分成若干个阶段,每个阶段有相对独立的任务,然后顺序完成各个阶段的任务。

(2) 采用结构化方法

结构化方法的基本思想是把一个复杂问题的求解过程分阶段进行,而且这种分解是自顶向下,逐层分解,使每个阶段处理的问题都控制在人们容易理解和处理的范围内。在技术上采用自顶向下分析设计、自底向上编程和实现,具有极强的可操作性。

结构化方法由结构化分析、结构化设计和结构化实现 3 部分有机组合而成。

① 结构化分析——从对问题的抽象逻辑分析开始,对系统进行功能分解。

② 结构化设计——根据功能分解的结果,将程序结构模块化。

③ 结构化实现——过程设计避免 goto 语句,任何程序只有一个入口和一个出口,并且只用"顺序""选择"和"循环"3 种基本控制结构。

(3) 前一阶段的结束标准就是后一阶段的开始标准

每个阶段的开始和结束都有严格的标准,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。

(4) 进行严格的技术审查和管理复审

在每个阶段结束之前必须正式地进行严格的技术审查和管理复审，只有通过之后这个阶段才算结束。审查的主要标准是每个阶段都应该交出“最新式的”(和所开发的软件完全一致的)高质量的文档资料。

2. 结构化范型的本质

结构化范型从系统需求分析出发，导出系统的功能；再将功能逐层分解、细化，并映射为软件结构。软件结构中的每个模块对应一个功能分解项，最终实现为一个过程或子过程。因此，过程是构成系统的基本成分，结构化范型的本质是着眼点是程序的过程，即面向过程。

结构化范型在一定时期内缓解了软件危机。但是，随着软件的飞速发展，也逐渐暴露出一些缺陷，主要表现为程序的数据与操作相分离；程序的可重用性差；数据处理范围较小和通信机制几乎空白等。

1.4.2 面向对象方法学

面向对象方法学也称面向对象范型，诞生于 20 世纪 80 年代中期，是为弥补结构化范型的缺陷而发展起来的。

1. 面向对象方法学的要点

(1) 把数据及在数据上的操作融合在一个对象中

把对象作为融合了数据及在数据上的操作的软件构件。对象具有属性和行为，属性对应为数据，行为则是对数据的操作。

(2) 把所有对象都归并成类

把一组具有相同属性和行为的相似对象归并成一个类，对象则是类的一个实例。

(3) 按照父类与子类的关系，把若干个相关类组织成一个层次结构的系统

子类可以自动继承父类的属性和行为。利用继承可以实现软件重用。

(4) 对象彼此间仅能通过发送消息相互联系

系统的一个功能是通过多个对象配合完成的，对象之间的配合是通过发送消息来联系的，对象响应消息是由对象执行相应的操作实现的。

2. 面向对象方法学的本质

面向对象方法学把数据及在数据上的操作封装在一个对象中，对象是构成系统的基本成分。面向对象方法学的本质是着眼点是程序的对象，即面向对象。对象封装了“数据＋算法”。

结构化范型在开发软件项目时，着眼于软件的处理功能(过程)，没有将数据与处理融为一体；而面向对象方法学则着眼于软件的对象，对象封装了数据及在数据上的操作(处理)。

面向对象方法学尽量模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界、解决问题的方法与过程，从而使描述问题的问题空间(也称为问题域)与实现