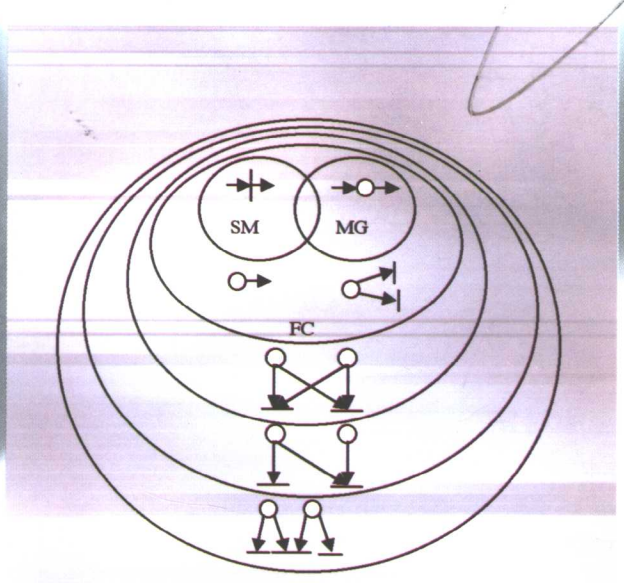
高等学校研究生系列教材

软件开发的形式化方法

Formal Methods of Software Development

古天龙



高等教育出版社
HIGHER EDUCATION PRESS

高等学校研究生系列教材

软件开发的形式化方法

古天龙

高等教育出版社

内容提要

形式化方法是建立在严格数学基础上、具有精确数学语义的开发方法。从广义角度,形式化方法是软件开发过程中分析、设计及实现的系统工程方法。狭义地,形式化方法是软件规格和验证的方法。本书对软件开发的形式化方法进行了介绍和讨论,内容涵盖了 SE2004 的 SEEK 中关于"软件的形式化方法"的知识点,主要包括:有限状态机、Statecharts、Petri 网、通信顺序进程、通信系统演算、一阶逻辑、程序正确性证明、时态逻辑、模型检验、Z、VDM、Larch 等。

本书可作为计算机、软件工程等专业高年级本科生或研究生的教学用书,也可供相关领域的研究人员和工程技术人员参考。

图书在版编目(CIP)数据

软件开发的形式化方法/古天龙. —北京:高等教育出版社,2005.1

ISBN 7-04-016079-X

Ⅰ.软... Ⅱ.古... Ⅲ.软件开发-方法

Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字(2005)第 000772 号

策划编辑 倪文慧 **责任编辑** 孙惠丽 **市场策划** 陈 振

封面设计 王凌波 **责任印制** 孔 源

出版发行	高等教育出版社	**购书热线**	010-58581118
社 址	北京市西城区德外大街 4 号	**免费咨询**	800-810-0598
邮政编码	100011	**网 址**	http://www.hep.edu.cn
总 机	010-58581000		http://www.hep.com.cn
经 销	北京蓝色畅想图书发行有限公司	**网上订购**	http://www.landraco.com
印 刷	潮河印业有限公司		http://www.landraco.com.cn
开 本	787×1092 1/16	**版 次**	2005 年 1 月第 1 版
印 张	17	**印 次**	2005 年 1 月第 1 次印刷
字 数	370 000	**定 价**	26.00 元

物料号:16079-00

前　言

软件是计算机应用系统中不可分割的一个重要组成部分。在商务数据库管理、宇宙飞船、机器人、飞机控制、通信系统、核电站控制、制造自动化等系统中，软件发挥着不可替代的作用。但在软件设计和开发过程中会遇到不少困难和问题，严谨的软件开发和设计方法——形式化方法，为解决或部分解决这些困难提供了可行途径。

形式化方法是基于严密的、数学上的形式机制的系统研究方法。客观地讲，有了数学的应用，就有了形式化方法。但是，一般认为形式化方法是始于20世纪60年代末的Floyd、Hoare和Manna等在程序正确性证明方面的研究，当时由于“软件危机”，人们试图用数学方法证明程序的正确性而发展成为了各种程序验证方法，但是受程序规模的限制，这些方法并未达到预期的应用效果。从20世纪80年代末开始，由于在硬件设计领域形式化方法的工业应用的结果，掀起了形式化方法和技术的学术研究和工业应用的热潮。在建立软件设计和开发的形式化理论和技术基础方面，研究人员已开展了大量的工作，建立了一些较为成熟并初步进入应用的方法和语言，例如：Statecharts、Petri网、通信顺序进程、通信系统演算、程序正确性证明、时态逻辑、模型检验、Z、VDM、Larch等。

从广义角度，形式化方法是软件开发过程中规格、设计及实现的系统工程方法。狭义地，形式化方法是软件规格和验证的方法，因此，形式化方法又分为形式化规格方法和形式化验证方法。形式化规格就是通过数学符号对系统及行为进行精确、简洁的描述。任何大型系统开发的前提都是需求规格。没有这样的规格，系统开发人员就没有一个系统用户的确切描述，这些用户就不得不从多个方面承担不明确规格带来的误错后果。精确的需求规格已被大多数工程学科所接受，计算机系统在精确性方面并不比任何其他工程任务差。然而，不幸的是，当今的软件工业实践在很大程度上还依赖于非形式文本和图形。形式化的另一个方面是设计的验证。程序是数学化的文本，这样就有可能解释程序和它们规格之间的形式关系。基于形式化方法，可以建立软件或程序在各种情形下性质及行为的描述。系统开发的步骤可以基于形式化规格，也可以针对形式化规格来验证。这样，在开发过程中就可以采用形式化验证以及时检查出误错。开发过程中尽早地消除误错是改善软件开发过程质量的关键之一，也是工业应用中采用形式化方法的一个重要理由。

近年来，形式化方法越来越受到国外计算机教育界的重视。2004年8月，IEEE－CS和ACM联合任务组提交了SE2004（Software Engineering 2004）最终报告，在该报告给出的SEEK（Software Engineering Education Knowledge）中，“软件的形式化方法（Formal Methods in Software Engineering）”被单列为一门必修课程（序列号为SE313）。SE2004最终报告的推

出，必将对我国软件工程等相关专业的本科和研究生教育产生重要影响。

本书涵盖了SEEK中“软件的形式化方法”的内容，如形式化的数学基础、形式化建模、形式模型的验证、形式化分析与设计，以及程序变换等。全书内容共有9章。第1章对软件及其开发进行介绍，包括：软件开发的历史、软件危机、软件工程、软件开发的形式化方法等；第2章阐述了软件开发中的有限状态机及其扩展方法，包括有限状态机、Statecharts等；第3章对Petri网方法相关的基本概念、性质、分析技术、高级Petri网等进行讨论；第4章对通信顺序进程、通信系统演算等进行了介绍；第5章阐述了命题逻辑、一阶谓词逻辑、程序正确性证明、程序变换等；第6章讨论了命题线性时态逻辑、一阶线性时态逻辑、计算树逻辑、模型检验等；第7章阐述了Z语言及其应用；第8章对VDM及其应用进行了介绍；第9章讨论了Larch及其应用。本书可作为计算机、软件工程等专业高年级本科生或研究生的教学用书，也可供相关领域的研究人员和工程技术人员参考。

在本书出版过程中，北京航空航天大学杨文龙教授、北京大学袁崇义教授提出了许多宝贵的建议并审阅了书稿，在此表示衷心的感谢。

由于作者水平有限，本书错漏和不妥之处在所难免，恳请广大读者批评指正。

作 者

2005.01

目　录

第1章　软件及其开发概述

软件或者程序是计算机及其应用系统的一个重要组成部分。伴随着计算机的诞生和发展，软件及其开发经历了半个多世纪的历程。本章简要回顾了软件及其开发的历史；阐述了软件发展过程中的软件危机、软件工程、软件开发的形式化方法等。

1.1　软件开发的历史

第一台通用电子数字计算机 ENIAC 于 1946 年 2 月 15 日诞生，与之相伴，出现了计算机软件（早期称为程序）。软件是计算机系统中与硬件相互依存的另一部分，它与硬件合为一体完成系统功能。在计算机发展的早期阶段（1946 年到 20 世纪 60 年代中期），计算机系统还是以硬件为主，软件费用只占总费用的 20% 左右；到了计算机发展的第二个阶段（20 世纪 60 年代中期到 20 世纪 80 年代初期），在硬件费用成倍下降的同时，软件费用却迅速上升，达到了总费用的 60% 以上；之后软件费用一直持续上升，其在计算机系统总费用中的比例一直上升到今天的 80% 以上。

所谓软件开发，实际上就是把现实世界的需求反映成软件的模型化并予以实现的过程。在计算机发展的不同时期，人们对软件的认识不同，相应地，所进行的软件开发也具有不同的特点。软件及其开发大体经历了如下三个阶段：

（1）程序设计阶段（1946 年 ~ 1956 年）

自 1946 年 2 月 15 日第一台通用电子数字计算机 ENIAC 宣告研制成功，到 1956 年高级语言的诞生，是软件发展的第一个阶段。在这一阶段，计算机主要用于科学计算与工程计算，处理对象是数值数据；程序的设计和编制工作主要以个体手工的方式进行，往往是谁使用就由谁设计、编制和维护；程序的设计也没有什么系统的方法可以遵循，往往是在某个人的头脑中完成的一个隐藏的过程；编制程序所使用的工具是汇编语言及机器语言，程序的设计和编制工作复杂、繁琐、费时和易出差错；开发技术和手段主要是采用了子程序和程序库；开发程序在方法上主要是追求编程技巧和程序运行效率，很少考虑到结构的清晰性、易读性和易维护性；程序的质量完全取决于个人的编程技术。在这期间，人们对与程序有关的文档的重要性认识不足，重点考虑程序本身，“软件”一词尚未出现。

（2）程序系统阶段（1956 年 ~ 1968 年）

1956 年，由美国计算机科学家 J. Backus 领导研制的 FORTRAN 语言问世，标志着高级语

言的诞生，同时也标志着软件的发展进入了第二个阶段。在这一阶段，计算机的应用领域不断扩大，出现了大量的数据处理问题，其性质与科学计算有明显区别，涉及到了非数值数据；软件的需求也不断增长，同时软件的规模也越来越大，软件的开发逐渐采用小组开发的方式；此时，编制程序所使用的工具是高级语言，效率得到了很大的提高；在开发方法上出现了结构化程序设计方法，但个人技巧依然占很大的比重；软件的质量不再完全取决于个人的编程技术，而是与整个开发小组的技术水平相关。在20世纪50年代后期，人们逐渐认识到与程序有关的文档的重要性，从而到了20世纪60年代初期，出现了“软件”一词，此时所谓的软件指的是程序及其说明书。随着软件的复杂程度迅速提高，其开发周期迅速变长，同时正确性难以得到保证，可靠性问题相当突出。到了20世纪60年代中期，出现了难以控制的局面，即所谓的软件危机。

（3）软件工程阶段（1968年以来）

1968年秋季，在当时的联邦德国召开的讨论软件危机问题的NATO会议上正式提出了“软件工程”这一术语，标志着一门新的工程科学的诞生，同时也标志着软件的发展进入了第三个阶段。在这一阶段，软件的应用领域急剧扩大，软件的需求者不再是少数用户，而是一个庞大的市场用户群体；在这种需求下，以软件的产品化、系列化、工程化、标准化为特征的软件产业发展了起来；软件开发方式逐步由个体合作方式转向工程方式，采用工程的原理、技术以及方法来指导软件的开发；软件的质量取决于开发团队的管理水平；软件的开发除了应用高级程序设计语言之外，还采用了需求定义语言、软件功能语言、软件设计语言等，统称为软件语言，覆盖了整个软件生命周期；同时，软件开发技术取得了很大的进步，出现了面向对象开发、网络及分布式开发等技术，对于以智能化、自动化、集成化、并行化，以及自然化为标志的软件开发新技术的研究也取得了较大的进展。在这个阶段，人们对软件有了更为深刻的认识，现代软件工程将软件定义为：

① 当它被执行时能够提供所要求的功能和性能的指令或计算机程序；

② 使得该程序能够满意地处理信息的数据结构；

③ 描述程序的功能需求以及程序的操作和使用文档。

时至今日，虽然在软件开发的技术、管理、工具等方面都取得了很大的进步，但仍未获得实质的突破性进展，正如Frederick P. Brooks所说的“没有银弹”。以成本高、进度缓慢甚至延迟，以及质量差为表现的软件危机依然困扰着人们。

1.2　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。1968年北大西洋公约组织的计算机科学家在联邦德国召开的国际学术会议上第一次提出了“软件危

机”(Software Crisis)这个词。概括地说,软件危机包含两方面问题:

(1) 如何开发软件,以满足不断增长,日趋复杂的需求;

(2) 如何维护数量不断膨胀的软件产品。

具体来看,软件危机主要表现在如下几个方面:

(1) 软件成本日益增长

在计算机发展的早期,计算机系统还是以硬件为主,软件费用只占总费用的20%左右;随着计算机的不断发展,计算机硬件的成本不断下降,而软件的成本却日益增长,软件在计算机系统中的成本上升到今天的80%以上。一组来自美国空军计算机系统的数据体现了这一情况:1955年,软件费用约占总费用的18%,1970年达到60%,1975年达到72%,1980年达到80%,1985年达到85%左右。

(2) 开发进度难以控制

许多软件开发项目一再延迟,甚至最终被取消。1995年,美国共取消了810亿美元的软件项目,其中31%的项目未做完就取消了,53%的软件项目进度通常要延长一半的时间,通常只有9%的软件项目能够及时交付并且费用也不超支。1998年,美国企业应用项目不成功的比率高达75%,其中28%的项目最终被撤销,40%的项目周期被无限拖长或资金超出预算。

(3) 软件质量差

所开发出来的软件质量往往达不到用户的要求,软件的错误率经常使用户不满和缺乏信赖感。尤其是在一些比较关键的系统中,软件的缺陷往往造成重大经济损失和社会影响。例如,1985年11月21日,由于计算机软件的错误,造成纽约银行与美联储电子结算系统收支失衡,发生了超额支付,而这个问题一直到晚上才被发现,纽约银行当日账务出现了230亿的短款。再一个比较典型的案例是美国Florida州的福利救济系统,该系统用于处理数百万受抚养儿童、食品券、医疗援助等受资助家庭接受者的资格认证,它基于巨型机系统,支持84个数据库、1 390个程序、12 000多个终端和个人计算机。1992年,该系统的错误使得成千上万的人收到了他们无权收到的救济,而其他成千上万急需食品券的人却排着长队等待了好几天。据后来统计,该错误导致了多支付2.6亿美元以及少支付5 800万美元医疗补助的后果。2002年,美国商务部的国立标准技术研究所(NIST)发表了一个有关软件缺陷的损失的调查报告。该报告指出:“据推测,由于软件缺陷而引起的损失额每年高达595亿美元。这一数字相当于美国国内生产总值的0.6%”。对于一些安全攸关的系统,软件的缺陷更是造成了灾难性的后果。例如,1996年,欧洲航天局阿丽亚娜5型(Ariane5)火箭在发射40秒钟后发生爆炸,发射基地上2名法国士兵当场死亡,损耗资产达10亿美元之巨,历时9年的航天计划因此严重受挫。事后,专家的调查分析报告指出,爆炸原因在于惯性导航系统软件技术要求和设计的错误。再一个比较典型的案例是“Therac-25事件”,Therac-25是加拿大原子能公司(AECL)和一家法国公司CGR联合开发的一种医疗设备,它产生的高能光束

或电子流能够杀死人体毒瘤而不会伤害毒瘤附近健康的人体组织。该设备于1982年正式投入生产和使用，在1985年6月到1987年1月的不到两年的时间里，因为软件缺陷引发了6起由于电子流或X－光束的过量使用造成的医疗事故，造成了4人死亡，2人重伤的严重后果。

（4）软件维护困难

在运行软件的过程中，由于新的错误的发现、运行环境的改变、或者用户提出了新的要求等原因，需要对软件不断进行修改。但是，由于在软件设计和开发过程中没有遵循严格的标准，没有完整的真实反映系统状况的记录文档，给软件维护造成了巨大的困难。

造成软件危机的原因是多方面的，但其根本原因在于软件本身所具有的复杂性。软件的这种复杂性主要体现在以下几个方面：

（1）软件规模的复杂性

随着计算机应用的日益广泛，需要开发的软件规模越来越庞大。以美国宇航局的软件系统为例：1963年，水星计划的软件系统约有200万条指令；1967年，双子星座计划系统约为400万条指令；1973年，阿波罗计划系统达到1 000万条指令；1979年，哥伦比亚航天飞机系统更是达到了4 000万条指令。软件庞大的规模是引起技术上和心理上的挫折的一个重要因素；此外，规模的复杂性引起了大量学习和理解上的负担。由于在需求分析及生成规格的阶段需要搜集和分析的信息数量非常巨大，从而可能会使得信息不正确或不完整，并且在审查阶段也未能检查出来。正如N. G. Leveson所认为的：几乎所有与计算机过程控制系统有关的事故都是源于这类由软件规模因素所引起的错误。

（2）结构的复杂性

结构复杂性体现在管理和技术两个方面。在管理方面，开发小组用来组织和管理开发活动时所采用的层次的宽度和深度，决定了用来管理系统的结构的复杂性；此外，软件开发机构内部的惯例和制度可能会改变各小组之间的信息流动，从而增加了结构的复杂性。在技术方面，软件系统的模块结构愈加复杂，模块之间复杂的调用关系以及接口信息往往超过了人们所能接受的程度。这种结构的复杂性可以用模块之间的耦合度来衡量，耦合度反映了在需求变化的情况下，相应所需修改的模块的数量。

（3）环境的复杂性

首先，运行中的软件总是受其所处环境的影响，在接收到外界环境的触发事件时，软件应该做出正确的响应。为了保证软件的可靠性，原则上必须对其所处环境有很好的理解，对外界环境可能产生的所有事件进行考虑，但这往往是难以办到的。其次，对于许多软件系统来说，人们往往缺乏对其所运行的环境特性的认识。许多系统只有当成功地运行于其环境时，才能对其环境进行很好的理解。再次，软件运行环境的多样性和异构性给软件开发者带来了更大的挑战。

（4）应用领域的复杂性

软件中所操作的对象仅仅是对应用领域真实对象的模拟，因而软件开发者需要从现实世界中抽象出软件模型所需的部分，并以其为基础构建软件。但是对于有的应用领域来说，这些模拟只能是近似的。其原因可能是由于对应用领域对象的认识不完全，或者是由于该模型所具有的苛刻条件限制，或者两者兼而有之。对于一个应用工程来说，它所使用的模型应该是具有合理的科学理论支持，并且经过良好测试的。然而在某些软件应用领域中，并不存在可以认知的模型，或者没有准确的模型来描绘相应的物理对象的几何、拓扑以及其他特征。在这种缺乏准确认识的情况下，应用领域的某些方面很可能在软件中不能体现出来。同时，由于有的软件是根据近似的模型来构建的，因而这些模型不一定能反映事实情况。

(5) 交流的复杂性

由于软件庞大的规模、大量的内部结构，以及应用领域的不同属性等因素，在开发一个大型软件系统时所参与的不仅仅是单个的人，而是一个团队。该团队里的每个人参与开发过程中的一个或多个活动。此时，对于参与不同开发阶段的人来说，彼此之间明确的交流非常重要。但是，一方面，由于结构的复杂性以及开发团队的庞大等原因，团队成员之间的交流非常困难；另一方面，成员之间在进行交流时使用的媒介往往是自然语言、图、表等非形式化的方式，这些媒介很难准确地描述出所开发的产品的基本属性，并且，由于这些媒介本身所具有的二义性，往往会使开发人员产生错误的理解，这种错误将会随着开发过程的进行而逐渐蔓延开来，最终导致灾难性的后果。

为了克服软件危机，人们从多个方面进行了探索，提出了种种解决方案。这些方案归纳起来主要有两类：一类是采用工程的方法来组织和管理软件的开发，另一类是从理论上探讨程序正确性和软件可靠性问题。前者导致了软件工程的出现和发展，后者则推动了对形式化方法的深入研究。

1.3 软件工程

1968 年秋季，北大西洋公约组织(NATO)科技委员会在当时的联邦德国召集了近 50 名一流的编程人员、计算机科学家和工业界巨头，讨论和制定摆脱软件危机的对策。在这次会议上首次提出了软件工程这个术语和概念。

软件工程的基本思想是用现代工程的概念、原理、技术和方法来进行软件的开发、管理和维护。IEEE 对软件工程的定义为：软件工程是

(1) 运用系统的、规范的和可定量的方法来开发、运行和维护软件，即将工程化应用于软件；

(2) 对(1)中所涉及方法的研究。

软件工程的目标是成功地生产具有正确性、可用性以及开销合宜的产品。其中，正确性

指软件产品达到预期功能的程度；可用性指软件基本结构、实现及文档为用户可用的程度；开销合宜是指软件开发、运行和维护的整个开销满足用户要求的程度。概括地说，软件工程包括三个要素：方法、工具和过程。其中，方法提供了如何构造软件的技术，是完成软件工程项目的技术手段；工具为软件工程方法提供了自动化或半自动化的支持；软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。具体来看，软件工程研究的内容主要有：软件工程过程、软件生存周期、软件开发模型、软件开发方法、软件工具、软件开发环境、计算机辅助软件工程，以及软件工程经济学等。

（1）软件工程过程

软件工程过程规定了获取、供应、开发、操作以及维护软件时，所需要实施的过程、活动和任务。其目的是为各种人员提供一个公共的框架，以便用相同的语言进行交流。各种组织和开发机构可以根据具体情况进行选择和剪裁，以采用不同的过程、活动和任务。具体地说，软件工程过程包括开发过程、管理过程、供应过程、获取过程、操作过程、维护过程以及支持过程。其中，开发过程是开发者和机构为了定义和开发软件所需的活动，包括需求分析、设计、编码、集成、测试、安装和验收等活动；管理过程是指软件工程过程中的各项管理活动，包括项目开始和范围定义、项目管理计划、实施和控制、评审和评价，以及项目完成等；供应过程是供应方按照合同向需求方提供合同中的系统、软件产品或服务所需的活动；获取过程是需求方按合同要求获取一个系统、软件产品或服务的活动；操作过程是操作者和机构为了在规定的运行环境中为其用户运行一个计算机系统所需要的活动；维护过程是维护者和机构为了管理软件的修改，使其处于良好运行状态所需要的活动；支持过程对项目的生命周期过程给予支持，它有助于项目的成功并能提高项目的质量。

（2）软件生命周期

软件生命周期借用了工程产品中产品生命周期的概念，是指与软件产品相关的一系列活动的全周期。具体地说，软件生命周期包括五个活动：需求分析和规格、软件设计、程序编写、软件测试，以及运行和维护。其中，需求分析和规格是指对软件系统的需求进行分析，建立起系统的逻辑模型，并给出详细的需求规格；软件设计的根本任务是将分析活动得到的逻辑模型设计成具体的计算机软件方案，分为对软件总体结构的设计和对具体模块的实现算法的设计；程序编写也就是把软件设计结果转换成用某种语言描述的计算机可以接受的程序代码，在程序编写的过程中应该遵循一系列的准则，以形成一个良好的程序设计风格；软件测试是保证软件质量的重要手段，其目的是通过各种类型的调试和测试使软件达到预定的要求，分为模块测试、组装测试以及确认测试；软件运行过程中可能由于各方面的原因，需要对其进行修改，相应的活动便称为维护，维护是软件生命周期中持续时间最长、工作量最大、费用最高的一项活动。

（3）软件开发模型

为了组织软件生命周期内的各种活动，人们提出了多种模型，称为软件生命周期模型，

也称为软件开发模型。软件开发模型是从软件项目需求定义开始，直至软件被淘汰为止，跨越整个生存期的系统开发、运行和维护所实施的全部过程、活动和任务的结构框架。它确立了软件开发中各阶段的次序以及各阶段的任务和活动准则，给出了开发过程中所应遵守的规定和限制，以便于各种活动的协调以及各种人员的有效配置和交流。

Winston Royce 于 1970 年提出的瀑布模型(Waterfall Model)是在 20 世纪 80 年代之前惟一被广泛采用的生命周期模型。瀑布模型遵循软件生命周期的划分，将开发过程分为六个阶段，并明确规定了每个阶段的任务。上一阶段完成确定的任务后就产生一定格式的文档交给下一阶段，下一阶段以其作为输入来进行相应的活动，活动的结果又作为输出传给再下一个阶段。其中，在完成了每一阶段的任务时需要对本阶段的工作及结果进行评审，若评审通过，则继续进行下一阶段的活动，否则将返回前一阶段，甚至更前面的阶段。采用瀑布模型进行开发的具体过程为：首先在软件计划阶段对项目开发的工作范围进行理解，生成计划任务书；接下来在需求分析阶段对用户需求进行理解和记录，生成详细、准确的需求规格；在设计阶段则根据需求规格来建立软件系统的体系结构、数据结构、过程细节和接口性质等，生成设计规格；在编码阶段把设计规格转换成用某种语言书写的程序；接下来在测试阶段对程序进行测试，以发现和排除错误，最终得到一个可运行的系统，将其交付给用户；在运行软件的过程中，由于新的错误的发现、运行环境的改变、或者用户提出了新的要求等原因，需要对软件不断进行修改，与之对应的就是维护阶段。上述各不同阶段的任务往往由不同的人员完成。瀑布模型主要缺陷在于：

(1) 软件需求分析的准确性很难确定，甚至是不可能和不现实的；

(2) 不能及时提供反馈信息，需求分析阶段的错误可能要到目标程序完成后才能发现，这时的返工需要付出昂贵的代价；

(3) 用户和软件项目负责人在得到实际有一个可供使用的系统之前，可能不得不等待很长时间。

原型模型(Prototype Model)是在考虑到瀑布模型的缺陷性的基础上建立的。其主要思想是在项目的早期尽快地生产一个便宜和简化的系统原型版本，将这个原型版本作为用户和开发人员学习的一种工具，通过原型反馈来加深对系统的理解。采用原型模型开发的具体过程为：首先获得一组基本需求说明，并以其为基础快速地构造出相应的原型，在构造原型时应注意要着眼于预期的评估，而不是为了正规的长期使用；接下来便可以运行原型并对其进行评价，此时的关键是要注意来自用户的反馈信息；根据评价的结果对需求进行修改之后，需要再次开发相应的原型；经过反复多次进行原型开发和原型运行评价，最终得到经过确认的需求规格；以最终的需求规格作为基础，便可开发出最终的软件产品。在实现最终的软件产品的过程中，原来的原型既可以全部或部分地成为最终系统的组成部分，也可以将其丢弃而重新进行开发，具体情况视原型实现的特点和环境而定。原型模型开发存在的问题是：

（1）用户看到的是一个可运行的软件版本，但不知道这个原型是临时搭起来的，也不知道开发人员为了使其尽快运行还没有考虑软件的整体质量或今后的可维护性问题。

（2）为了使原型尽快投入运行，开发人员经常采用一些折中的解决办法。例如，采用一些效率不高的算法，仅仅用来证明方法是否可行。经过一段时间后，开发人员可能熟悉了这些选择，而忘掉了它们为什么不适当的原因，或者开发人员也不愿意再进行改动。

Barry W. Boehm 于 1988 年提出的螺旋模型（Spiral Model）综合了瀑布模型和原型模型的优点，并加入了风险分析。该模型适合用来指导大型软件项目的开发。螺旋模型将软件开发划分为制订计划、风险分析、实施工程和用户评估四类活动，这四类活动被分布在笛卡儿坐标的四个象限。开发过程沿着以笛卡儿坐标原点为起点的螺旋线自内向外旋转，依次经过四个象限。沿着螺旋线每转一圈，便开发出一个更为完善的新的软件版本；然后由客户对其进行评价，并给出修正建议；在此基础上再次计划，并进行风险分析。如果分析的结果是风险过大，开发机构和客户无法接受，则项目将有可能终止，否则将沿着螺旋线继续下去，直到开发出所期望的软件产品。螺旋模型一方面强调了风险分析，另一方面吸收了软件工程中的“演化”概念，适合于大型复杂软件系统的开发，对于具有高风险的软件开发是最为实际的方法。但在应用时仍存在不足：要求开发者具有相当丰富的风险评估经验和专业知识。如果项目风险较大，而又未能及时发现，则将会造成重大损失。

构件模型（Component Model）又称为“即插即用编程”方法，其思想是将已经开发好的构件集成到所设计好的框架中，从而得到满足需求的软件产品。构件模型融合了螺旋模型的特征，其本质上是演化的并且支持软件开发的迭代方法。从宏观上来看，构件模型与螺旋模型相似，主要区别在于实施工程的具体过程：首先，从候选构件的标识开始，一旦标识出候选构件，就可以搜索构件库，确认是否存在可用的构件；如果已经存在，则从构件库中提取出来复用；如果不存在，则采用面向对象方法来开发满足要求的构件，并将新的构件加入构件库中；之后，就可以对所提取出来的构件或新开发的构件进行组装，从而完成待开发软件的一次迭代。毫无疑问，基于构件的软件模型开发会大大地提高软件的可靠性和生产率，并且提高软件系统的灵活性和可维护性。

四代技术模型（4GT Model）是基于一系列软件工具的开发。4GT 所包含软件工具的共同特点是：能使软件开发人员在高层次上定义或规格软件的某些特征，然后工具根据开发人员的规格自动生成源代码。显然，软件在越高的级别上被规格，就能越快地生成程序。四代技术模型的核心在于规格软件的能力——使用特殊的语言形式或一种采用客户可以理解的术语描述待解决问题的图形符号体系。四代技术模型并不支持软件开发的全过程，只侧重于支持应用软件开发过程中的设计阶段和实现阶段，特别是支持界面以及与界面有关的处理过程。4GT 可以极大地降低软件开发时间从而提高软件的开发效率，但是 4GT 所生成的源代码的效率，以及软件的可维护性尚有不足，同时，在系统开发全过程中应用有限。

变换模型（Transformational Model）是基于形式化规格及程序变换的软件开发模型，又称

为自动程序设计模型，或者形式化方法模型。采用了形式化方法模型的软件开发以形式规格为中心，具有三个核心活动：规格的生成、形式化验证，以及计算机辅助下的程序求精变换。具体过程为：首先，根据用户需求生成软件需求规格；然后对该规格进行检查，判断其是否满足用户的需求以及其内部是否存在不一致、不完整等错误，同时对其进行形式化验证，判断其是否具有所期望的特性；之后，以规格为基础，对其进行逐步求精，每一步求精都降低了一些抽象程度或增加一些过程，最终得到可执行的程序代码。在求精的过程中需要注意保证各步求精所得结果之间的功能的一致性。理论上，只要有一个正确的满足用户需要的形式化规格，经过一系列正确的程序变换后，应该能生成正确的、可以接受的程序代码。形式化方法模型可以最大限度地提高软件开发的可靠性，尤其是安全关键系统软件开发的惟一可行途径，同时，可以改善软件的开发效率、有效控制开发进度。但是，形式化方法模型的应用中开发人员的培训是需要考虑的一个重要问题。

(4) 软件开发方法

软件开发方法是指在某种思想的指导下使用已经定义好的一系列技术和表示工具来组织软件开发过程的方法，它一般表述成一系列的步骤，每一步骤都与相应的技术和表示工具相关。应用某种软件开发方法的目标是有效地得到一个运行系统及其支持文档，并且满足有关的质量要求。典型的传统软件开发方法有结构化方法、面向数据结构方法、面向对象的开发方法等。

结构化方法 结构化方法是最早、最传统的软件开发方法。它是随着结构化程序设计(SP)方法的提出、结构化设计(SD)方法的出现、直到结构化分析(SA)方法的提出才逐渐形成的。所谓结构是指系统内各组成要素之间的相互联系、相互作用的框架。结构化方法强调所开发的软件的结构合理性，由此提出一组提高软件结构合理性的准则(如分解和抽象、模块的独立性、信息隐蔽等)，在这些准则的指导下使用一定的表示工具(如数据流图、控制流图、数据字典、判定表、判定树、结构化语言等)，按照特定的步骤来进行软件开发。针对不同的开发活动，结构化方法分为结构化分析(SA)、结构化设计(SD)以及结构化程序设计(SP)，其核心是结构化分析和结构化设计。

结构化分析实际上是一个建模过程。它应用分解和抽象的原则，以实际系统中数据处理的流程为基础，借助于实体－关系图(E－R图)、数据流图、控制流图、数据词典、结构化语言等表示工具，为所需解决的问题建立完整和准确的模型。进行结构化分析的大致步骤为：

① 创建实体－关系图；

② 创建数据流模型；

③ 创建控制流模型；

④ 生成控制规格；

⑤ 生成加工规格；

⑥ 建立完整的数据字典。

结构化设计是以结构化分析所产生的数据流图为基础，将数据流图按一定的步骤映射成软件模块结构，得到对软件模块结构进行描述的结构图(SC)。结构图描述了系统由哪些模块组成，并且给出了模块之间的调用关系。进行结构化设计的主要步骤为：

① 评审基本系统模型；

② 评审和完善软件的数据流图；

③ 确定数据流图的类型；

④ 根据确定的输入流和输出流的边界，找出变换中心或事务处理中心；

⑤ 实现第一层的分解；

⑥ 实现第二层的分解；

⑦ 根据优化准则对软件结构求精；

⑧ 描述模块功能、接口及全局数据结构。

完成上面的设计步骤后，将得到一个设计模型和一份完整的设计文档。以设计文档为蓝本就可以开发出用某种程序语言书写的程序代码。

面向数据结构方法　在许多应用领域中，问题的结构层次清楚，输入数据、输出数据和内部存储信息的数据结构都有一定的结构关系，这些内在的数据结构不但影响程序的结构，也影响着程序的处理过程。面向数据结构的方法就是利用这些结构作为软件开发的基础。

在众多的面向数据结构的方法中，比较典型的是 J. Warnier 提出的 Warnier 方法和 M. Jackson 提出的 Jackson 方法。Jackson 方法的发展可分为两个阶段，相应地有两种不同的方法。早期(20 世纪 70 年代)的 Jackson 方法的核心是面向数据结构的设计，称为 Jackson 结构程序设计方法(JSP)。从 20 世纪 80 年代初开始，Jackson 在 JSP 的基础上进行了扩展，提出了 Jackson 系统开发方法(JSD)。

Jackson 结构程序设计方法(JSP)按照输入、输出和内部信息的数据结构进行软件设计，把数据结构的描述映射成程序结构描述，从而得到问题的软件过程描述。若数据结构内有重复子结构，则对应程序一定有循环；若数据结构有选择性子结构，则对应程序一定有判定，以此揭示数据结构和程序结构之间的内在关系，设计出反映数据结构的程序结构。Jackson 结构程序设计方法主要由五个步骤组成：

① 分析并确定要处理的输入数据和输出数据的数据结构，并绘制出其 Jackson 结构图。

② 找出输入数据结构图和输出数据结构图中有对应关系的数据单元。

③ 应用下述三条规则从描绘输入/输出数据结构的 Jackson 结构图导出描绘程序结构的 Jackson 结构图：第一，对于上面找出的每对有对应关系的数据单元，如果它们分别在输入数据结构图和输出数据结构图中的层次相同，则按照该层次在程序结构图的相应位置画一个处理框；如果它们分别处于输入数据结构图和输出数据结构图中的不同层次，则按照较低的那个层次在程序结构图的相应位置画一个处理框；第二，对于输入数据结构图中剩余的每个数据单元，根据其所处的层次在程序结构图的相应层次画一个程序框；第三，对于输出数