高等学校教材

软件开发环境

林琪超 编

上海交通大学出版社

软 件 开 发 环 境

林琪超 编

上海交通大学出版社

内 容 简 介

本书较全面系统地介绍了各种软件开发环境的作用、一般原理、关键技术和基本实现方法。全书共分九章，第一章是引论、第二章是人机界面、第三～第八章是各种环境的特色部分、第九章是环境数据库。书中还适当介绍了人工智能技术在软件开发环境中的应用。本书的特色是既注意了最基本的原理和方法，又尽可能地反映一些最新的重要的研究成果；既注重科学性，又注重实用性；既有原理性的论述，又有适当的实例与之配合。

本书可作为高等院校计算机专业本科生或研究生的教材，亦可作为有关科技人员的参考书。

软 件 开 发 环 境

出 版：上海交通大学出版社
（淮海中路 1984 弄 19 号）
发 行：新华书店上海发行所
印 刷：江 苏 太 仓 印 刷 厂
开 本：787×1092(毫米)1/16
印 张：14
字 数：3400000
版 次：1991 年 5 月 第一版
印 次：1991 年 6 月 第一次
印 数：1—2200
科 目：246—317
ISBN 7—313—00861—9/TP・39
定 价：3.65 元

出 版 说 明

根据国务院关于高等学校教材工作分工的规定，我部承担了全国高等学校、中等专业学校工科电子类专业教材的编审、出版的组织工作。由于各有关院校及参与编审工作的广大教师共同努力，有关出版社的紧密配合，从1978年至1985年，已编审、出版了两轮教材，正在陆续供给高等学校和中等专业学校教学使用。

为了使工科电子类专业教材能更好地适应“三个面向”的需要，贯彻“努力提高教材质量，逐步实现教材多样化，增加不同品种、不同层次、不学学术观点、不同风格、不同改革试验的教材”的精神，我部所属的七个高等学校教材编审委员会和两个中等专业学校教材编审委员会，在总结前两轮教材工作的基础上，结合教育形势的发展和教育改革的需要，制订了1986～1990年的“七五”（第三轮）教材编审出版规划。列入规划的教料、实验教材、教学参考书等近400种选题。这批教材的评选推荐和编写工作由各编委会直接组织进行。

这批教材的书稿，是从通过教学实践、师生反映较好的讲议中经院校推荐，由编审委员会（小组）评选择优产生出来的。广大编审者、各编审委员会和有关出版社为保证教材的出版和提高教材的质量，作出了不懈的努力。

限于水平和经验，这批教材的编审、出版工作还会有缺点和不足之处，希望使用教材的单位，广大教师和同学积极提出批评建议，共同为不断提高工科电子类专业教材的质量而努力。

机械电子工业部教材办公室

前　言

本书系按电子工业部制订的工科电子类专业教材1986～1990年编审出版规划，由计算机与自动控制教材编审委员会计算机编审小组征稿、评选、推荐出版的。

软件开发环境是一门新兴的学科，目前正在迅速发展之中。它对提高软件产品的质量和生产率有着重要的作用。本书力图较全面系统地介绍各种软件开发环境的作用、一般原理、关键技术和基本实现方法，并且尽可能地反映这个领域中一些最新的和最重要的研究成果。

第一章引论，介绍了软件加工模型、软件开发方法、软件开发环境的分类、现状及研究方向。第二章人机界面，介绍了它的硬件和软件基础、设计原则和开发工具。第三章～第八章介绍了各种软件开发环境的特色部分，其中第三～第六章涉及支持瀑布型软件加工模型的程序设计环境、系统合成环境、项目管理环境和维护环境.第七、八章分别介绍支持新型软件加工模型的软件原型环境和程序生成环境。第九章介绍了环境数据库，它是软件开发环境的核心部件。本书还适当介绍了人工智能技术在软件开发环境中的应用，它对增强软件开发环境的能力方面有着极为重要的作用。

由于本课程内容涉及面广，是系统软件中的上层建筑，因此，在学习本门课程之前，学生必须先修编译原理和数据库原理。本书可作为计算机专业本科生或研究生的教材。选用本教材，最好能开些使用现有环境或实现基础技术的实验，以获得感性认识、增强环境的使用和开发的能力。

本书由上海工业大学林琪超编写，由西安电子科技大学蔡希尧教授主审。

本书编写过程中还得到了重庆大学梁光春副教授和上海工业大学张吉锋副教授的热情鼓励和支持，还有我的同行和研究生们也提供了许多有价值的资料和帮助，他们是上海计算机软件技术开发中心的朱三元研究员和周庆隆助理研究员，上海工业大学的郑衍衡教授、孙振飞副教授、王令宏、陈素云、嵇文广、蔡方方和赵琼瑛等同志，在此向他们表示衷心的感谢。

由于该领域正在发展和完善之中，尚未见系统介绍这一领域的书籍，因此本书是在搜集、整理和综合大量文献资料的基础上编写而成的。由于编者水平有限，无论在取材、编排方面，还是内容理解方面，难免存在不妥之处，殷切希望广大读者批评指正。

编　者

1990年6月

目　　录

第一章 引 论

1.1 软件开发环境

软件开发环境 (saftware development environment, SDE) 是一组相关的软件工具的集合，它们组织在一起支持某种软件开发方法或者与某种软件加工模型相适应。软件工具是指这样一类计算机程序，它们可用来帮助开发、测试、分析或维护另一计算机程序和它的文件编制。例如，自动设计工具、编辑程序、编译程序、测试工具和维护工具等。软件开发环境有很多别名，例如，在欧洲的流行叫法是集成式项目支援环境 (integrated project support environment, IPSE)。

软件开发环境的主要组成成分是软件工具。为了提高软件本身的质量和软件开发的生产率，人们开发了不少工具为软件开发服务。例如，最基本的文本编辑程序、编译程序、调试程序和连接程序；进一步还有数据流分析程序、测试复盖分析程序和配置管理系统等自动化工具。面对形形式式的工具，开发人员会感到眼花瞭乱，难于熟练地一一使用它们。也就是说，从用户的角度考虑，不仅需要有众多的工具来辅助软件的开发，还希望它们能有一个统一的界面，以便于掌握和使用。另外，从提高工具之间信息传递的效率来考虑，希望对共享的信息能有一个统一的内部结构，并且存放在一个信息库中，以便于各个工具去存取。因此，软件开发环境的基本组成有三个部分：会话系统、工具集和环境数据库，如图 1.1 所示。

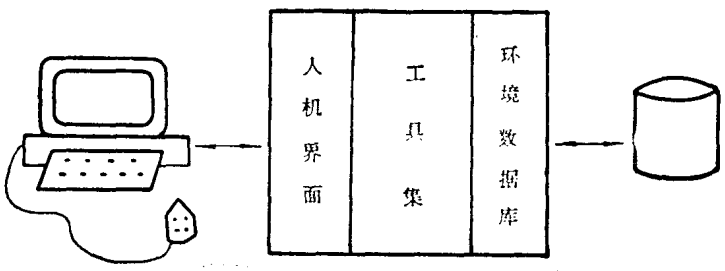


图 1.1 软件开发环境的基本组成

人机界面是软件开发环境与用户之间的一个统一的交互式对话系统。多窗口屏幕显示、弹出型 (pop-up) 菜单驱动方式和鼠标器 (mouse) 控制等新型的软件和硬件技术支持了一个友好的用户接口。它是软件开发环境的重要质量标志。

存储各种软件工具加工产生的软件产品或半成品(如源代码、测试数据和各种文档等) 的软件环境数据库是软件开发环境的核心。工具之间的联系和相互理解都是通过存储在信息库中的共享数据得以实现的。由于软件信息的形式繁多，如有正文、图形、源代码等，现有的数据库模型并不能理想地满足它的要求，因此各国计算机软件专家正在这方面进行各种理论上和实践上的探索。另外，考虑人工智能技术的应用，还希望它是一个知识库。

软件工具在软件开发环境中已不是各自封闭和分离的了，而是以综合、一致和整体连贯的形态来支持软件的开发，它们是与某种软件开发方法或者与某种软件加工模型相适应的，是

本书的主要内容。

软件开发环境的具体组成可能千姿百态，但都包含上述三个基本组成，并应具备下列特性（或者说遵守下列准则）：

（1）可用性——用户友好性、易学、对项目工作人员的实际支持等。

（2）自动化程度——在软件开发过程中，对用户所进行的平凡的、耗时的或困难的活动提供自动化辅助的程度。

（3）公用性——复盖各种类型的用户，如程序员、设计人员、项目经理和质量保证工作人员等。或者说复盖软件开发过程中的各种活动，如体系结构设计、程序设计、测试和维护等。

（4）集成化程度——用户接口一致性和信息共享的程度。

（5）适应性——环境被定制、剪裁或扩展时符合用户要求的程度。对定制而言，是指环境符合项目的特性、过程和/或各个用户的爱好等的程度。对剪裁而言，是指提供有效能力的程度。对扩展而言，是指适合改变后的需求的程度。

（6）价值——得益和成本的比率。得益是指生产率的增长，产品质量的提高、目标应用开发时间/成本的降低等。成本是指投资、开发所需的时间，培训使用人员到一定水平所需要的时间等。

1.2 软件加工模型

软件开发的过程是指把用户需要转化为软件需求，把软件需求转化为设计，用代码来实现设计，对代码进行测试，进行文件编制，并签署确认它可以投入运行性使用的过程。考虑到软件在运行过程中还需要进行适应性、增强性等维护，这类活动常常也是由开发者来完成的，因此也可以把结束时间延长到该产品不再需要开发人员来进行维护为止，甚至延长到包括维护和退役阶段。这个过程也可称为软件开发周期（software development cycle）。

如何组织这样一个过程，这是有不同的模型的，称为软件加工模型（software process mvodel）。较著名的软件加工模型有四种：瀑布型、快速原型、螺旋线型和自动程序设计型。

1.2.1 瀑 布 型

瀑布型模型（waterfall model）是最常用的传统的软件加工模型。下面按照我国制订的《计算机软件开发规范》的国家标准（GB 8566—88）来介绍。GB 8566—88 将软件加工模型称为软件开发流程，共分为八个阶段。

1. 可行性研究和计划

这个阶段的任务是了解客户的要求及现实环境，从技术、经济和社会因素等方面研究并论证本软件项目的可行性，编写可行性研究报告，制订初步项目开发计划。

作为这个阶段完成的标志，应交付二个文件：可行性研究报告和初步的项目开发计划。前者应对成本/效益分析提供几种可供选择的解答，后者应具有明确的、可检查的标志。

2. 需求分析

这个阶段的任务是确定被开发软件的运行环境、功能和性能要求，编写用户手册概要和确认测试准则，为概要设计提供需求说明书。

作为这个阶段完成的标志，应交付的文件有：软件需求说明书；修改后的项目开发计划；用户手册概要；确认测试计划；数据要求说明书。

3. 概要设计

这个阶段的任务是根据软件需求说明，建立目标系统的总体结构和模块间的关系，定义各功能模块的接口控制特征，设计全局数据库/数据结构，规定设计限制，制订测试计划。

作为这个阶段完成的标志是：所有已定义的软件需求均被所设计的系统覆盖；建立了系统结构，明确指出系统各模块的功能，模块间的层次关系及接口控制特征；并交付经过验证的文件，包括概要设计说明书、数据库/数据结构设计说明书和组装测试计划。

4. 详细设计

对概要设计中产生的功能模块逐步细化，形成若干个可编程模块；采用某种详细设计表示方法对各个程序模块进行过程描述，包括模块内的算法及数据结构；确定各程序模块之间的详细接口信息，包括参数的形式和传送方式。

这个阶段要交付的文件有详细设计说明书和模块开发卷宗。

5. 实现

实现阶段的任务是将详细设计说明转化为所要求的程序设计语言或数据库语言书写的源程序。并对编写好的源程序进行程序单元测试，验证程序模块接口与详细设计说明的一致性。这个阶段的文件编制要求是在模块开发卷宗中填写相应于本阶段的内容。

这个阶段是与详细设计阶段相呼应的，实现详细设计阶段规定的程序模块，并通过单元测试加以验证。

6. 组装测试

这个阶段是与概要设计阶段相呼应的，即根据概要设计中各功能模块的说明及制订的组装测试计划，将经过单元测试的模块逐步进行组装和测试。如果测试结果符合要求，并交付了可运行的软件系统源程序清单和组装测试分析报告，就可以认为这个阶段已经完成了。如果测试结果不符合要求，将按组装测试分析报告，回到合适的阶段去改正错误。

7. 确认测试

这个阶段是与需求说明阶段相呼应的，即根据软件需求说明书中定义的全部功能和性能要求以及确认测试计划来测试整个软件系统是否达到了要求。对用户手册和操作手册要使用，以证实其实用性和有效性，并改正其中的错误。这个阶段应交付的文件有：确认测试分析报告；最终用户手册和操作手册；以及项目开发总结报告。这个阶段必须邀请客户一起参加。

8. 使用和维护

对投入运行后的软件系统进行修改，以改正在开发阶段产生、在测试阶段又未发现的错误，使软件系统能适应外界环境的改变，并实现软件系统的功能扩充和性能改善。这个阶段填写的文件有软件问题报告和软件修改报告。这个阶段将与软件生命同时终止。

综上所述，可知软件的开发都要经过上面八个阶段，每个阶段结束时都应编制、评审和交付详细的文档资料，作为下一阶段或更后面的某个阶段的工作依据。如同瀑布飞流拾级而下一样，若在某个阶段中发现了错误，将在它之前的某个阶段中进行修正，然后重新经历修正后的各个阶段。而越是后阶段发现的错误常常要在越是前面的阶段中去改正，即返工的代价越是昂贵。瀑布型模型可用图 1.2 示意。

图 1.2 中如瀑布拾级而下的顺序是软件开发的正常流程，旁边标的是相应阶段产生的主要文件。

为了评定和改进在软件开发和修改期间所产生的产品及半成品的质量，需要实施验证和确认。验证 (verification) 是确定软件开发周期中一个给定阶段的产品是否达到前阶段确立

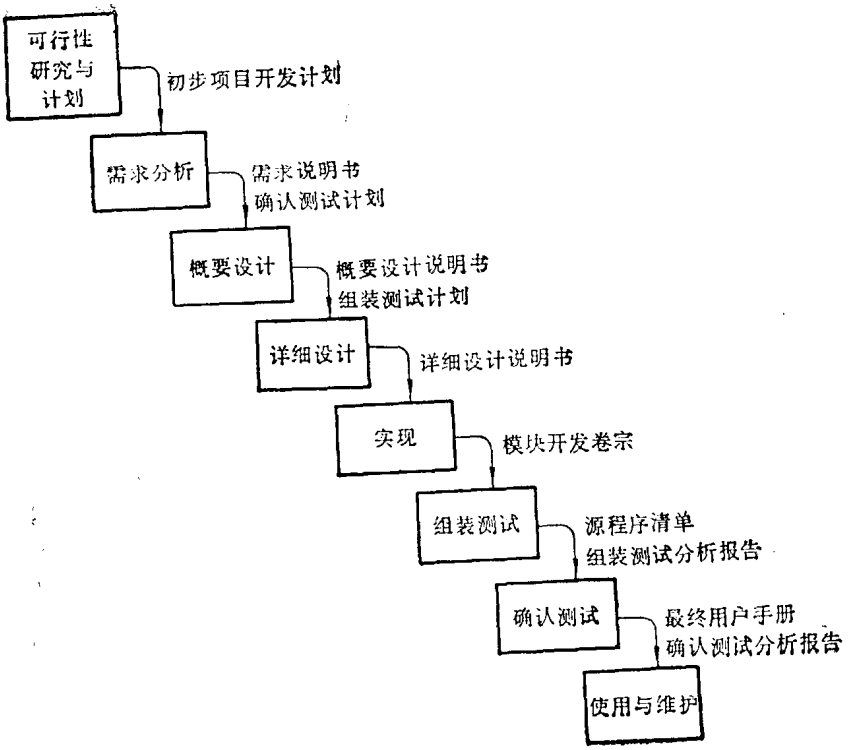


图 1.2 瀑布型软件加工模型

的需求的过程，即解决“我们正在构造的产品正确吗？”这样的问题。确认（validation）是软件开发过程结束时对软件进行评价，以确认它和软件需求相一致的过程，即解决“我们正在构造正确的产品吗？”这样的问题。

验证有二种：水平验证和垂直验证。水平验证是指每一阶段的目标表示的自一致性。垂直验证是论证所派生的表示确实是基础表示的结果。当使用转换机制（如编译程序）时，垂直验证是自动达到的，另外，也可用形式化的正确性证明技术。

一般，确认不像验证那么精确和可计算，它必须涉及判断。如果原始的应用概念可以用某种域模型（domain model）表示，对于开发过程的每一步有一个域子模型（domain submodel）相对应，那么每一步的表示与域子模型比较应该在某种意义上是等效的。

综上所述，可用图 1.3 表示软件开发过程中的验证与确认验证（V & V）。

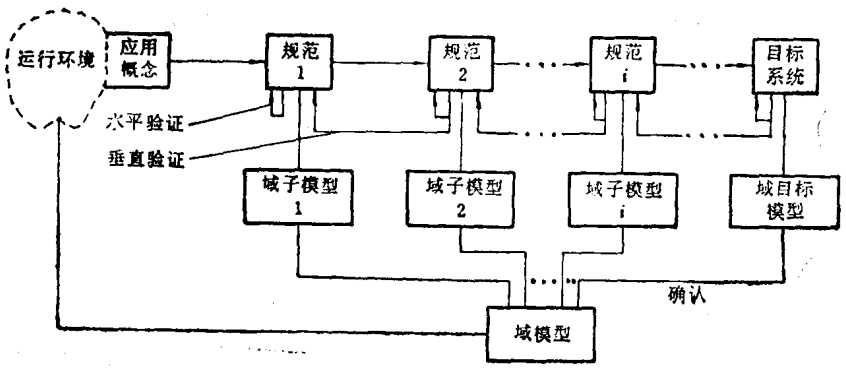


图 1.3 验证与确认

验证和确认都属于质量保证措施。常用的技术有下列一些：对分析和设计阶段有走查和审查；对源代码有走查和审查、静态分析、符号执行、单元测试、系统的组装测试。形式化验证技术是一种严格的方式，可用来证明源程序与它的需求规格说明是否一致。

近 20 年来，人们按照瀑布型软件加工模型制作软件产品。它对于现代软件的开发起了重要的推动作用。但是它还存在一些缺陷，如它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发现，这样的返工显然要付出昂贵的代价；软件维护和扩充十分困难，因为它们必然要涉及到低级的经过优化的源程序上，而源程序中常常体现了程序编制者的技巧和知识，这些情况常常因缺乏记录而失传，如果再由于波动作用而传播到别的部分的话，那么会使维护和扩充更加困难；各种规格说明书是手工制作的，通常是非形式化的、不完整的和冗长的，这不仅使编制和使用这些说明书要消耗大量的精力，而且也是引起软件故障的重要原因之一；另外，重视软件开发流程各阶段的文件编制可能产生副作用，会使整个项目开发趋于更快地产生文档，而不是仔细地考虑关键的问题，有人甚至称之为文件驱动方法。

由于瀑布型的缺陷，促使人们探索新的加工模型，80 年代相继出现了下列几种主要的新模型。

1.2.2 快 速 原 型

由于返工造成大量重复劳动的一个主要原因是软件需求说明书是在对任务或用户界面的需求不甚了解的基础上编制的。当向用户调查他们对软件的需求时，他们常常会说："我不能准确地告诉你我想干什么，当我看到它的时候我就知道了。" 快速原型模型 (rapid prototyping model) 能改善这种情况，同时对开发风险的估计也有帮助。

快速原型模型把软件开发流程分成二个大的阶段：原型开发阶段和目标软件开发阶段，如图 1.4 所示。

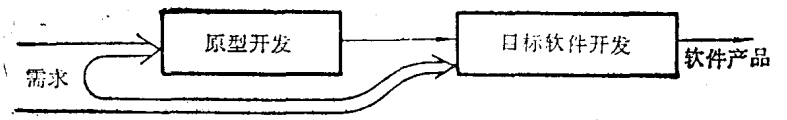


图 1.4 快速原型模型

第一阶段的任务是快速开发一个原型，它应该包含目标软件的关键问题和反映目标软件的大致面貌。从这样的原型，可以反映出开发人员是否准确理解了用户对系统功能的期望，也可以反映出目标软件开发的可行性，即判定开发目标软件的风险程度，从而可以正确作出下一步的决策。第二阶段才是实际系统的开发。这种模型对开发小型的或中型的软件是很适宜的。对大型软件来说，原型可能就非常复杂而难以快速形成了。

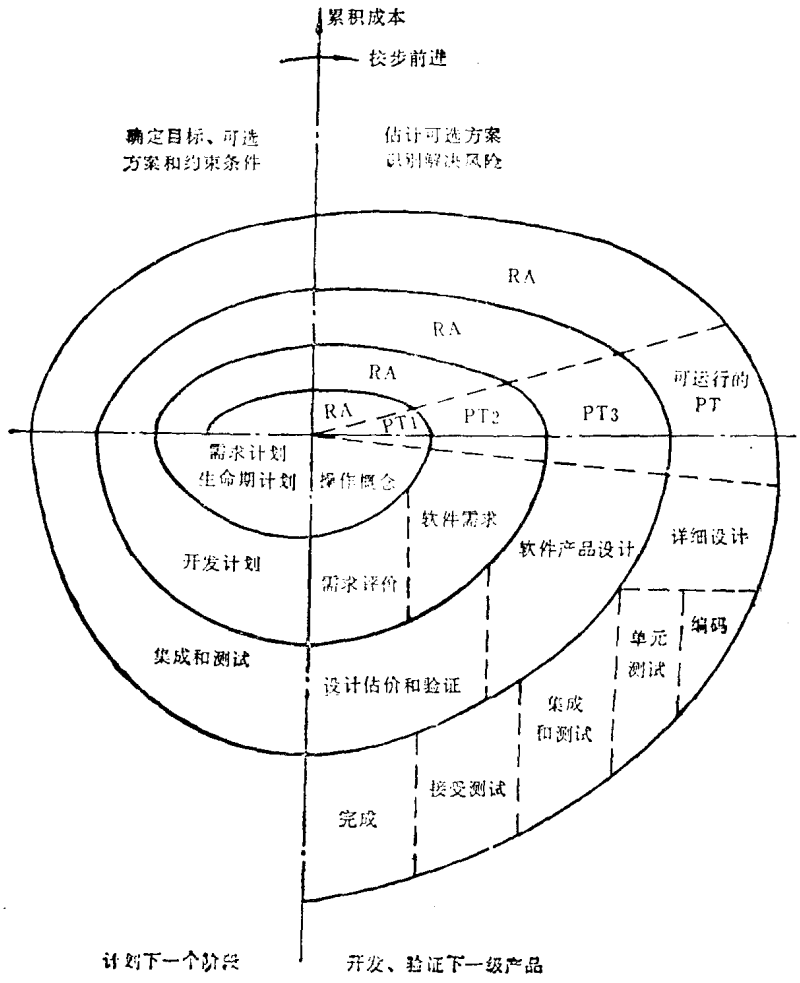
原型开发和目标软件的开发都还是需要经历瀑布型模型的某些阶段的。

1.2.3 螺 旋 线 型

螺旋线型模型 (spiral model) 是在快速原型的基础上扩展而成的。这种模型把整个软件开发流程分成了多个由风险分析和原型开发组成的阶段。因此也有人把这种模型归作快速原型模型。

螺旋线型模型的每个阶段都从确定阶段目标，制定可选方案和约束条件开始；然后对可选

方案进行估价，对风险进行识别和制定解决办法，并开发出一个原型；在此基础上开发和验证下一级产品，并计划下一个阶段。图 1.5 是这种模型的示意图。



其中 RA：风险分析(Risk analysis)
PT：原型(Prototype)

图 1.5 螺旋线型软件开发模型

根据阶段目标和约束条件去估价可选方案时，常常会觉得难以确定，这是项目风险的主要原因。如果陷入这种境地，可采用原型、模拟、征求用户意见，分解模型或这些技术和别的风险消除技术的组合等技术，以求作出正确的估价。

螺旋线型模型也可以与其他软件开发模型适当混合起来。考虑的因素主要是程序风险的相对重要性和各种风险消除办法的相对有效性。

螺旋线型模型的风险驱动原理使开发人员把重点放在识别和尽早解决高风险的问题上，这能有效地减少重复工作，对非常大的、复杂的软件系统特别适用。

1.2.4 自动程序设计型

自动程序设计型模型 (automatic programming model) 把软件开发流程分成二个基

本的阶段：软件规格说明开发和在机器辅助下将它转换成代码。

该开发模型以形式化的软件规格说明为中心。软件的整个开发过程由规格说明开发、规格说明确认、交互式转换和维护等活动组成。如图 1.6 所示。

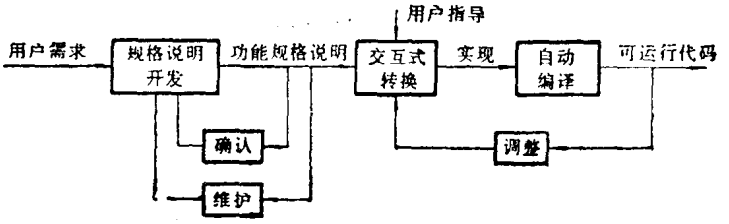


图 1.6　自动程序设计模型

规格说明开发是按用户需求生成软件规格说明。规格说明确认是验证软件规格说明是否满足用户的需求。交互式转换阶段是从形式的“做什么”的功能规格说明推导出“如何做”的具体实现。目前的编译技术还无法将功能规格说明完全自动地生成可运行的代码，其中某些决策和优化还必须由人去指导。交互式转换产生可自动生成代码的实现。实现的正确性将由规格说明的正确性和规则及其作用的正确性而得以保证。这样，由于机器的辅助，实现工作大为简化，从根本上改变了过去手工实现的方式。维护也只需修改规格说明，有时需修改推导决策，然后重演（replay）一下实现的推导过程就可自动完成，这就改变了传统的在优化了的源代码上进行“添补”的维护方式。

自动程序设计型与瀑布型相比，设计、实现和测试阶段都在用户适当指导下由机器翻译自动完成；维护也只施于软件规格说明上，而不是在需求分析、设计和实现等众多的阶段上。

1.2.5　软件加工模型比较

为了加深对上述软件开发模型的理解，我们用图 1.7 来示意它们与用户需要之间的差距，以作比较。

图中横坐标表示时间，纵坐标表示软件功能性。用户需要所标的斜线表示随着时间的推移用户对软件功能的要求越来越高。当然，这种需求增长可能是非线性的和不连续的，但作为示意一种趋势而用了一条斜线。传统模型的曲线是表示：t_0 时用户需要某种功能性的软件，由于开发周期的延迟，到 t_1 时才有了可运行的软件，而且由于种种原因，还没有达到用户在 t_0 时所要求的功能。以后对该软件进行维护，使它朝用户需要的目标完善和改进，但总是跟不上用户的需要。到 t_3 时觉得老在已有的软件上修修补补划不来，决定开发一个新软件。因此 t_3 到 t_4 这段时间曲线停在一个水平上，到 t_4 时新软件投入运行，还是不能达到 t_3 时用户所要求的功能，需要维护。以此类推，可见从某一功能水平来看，实际达到的时间总是较多地滞后于用户的要求；对某一时间来讲，实际达到的功能与用户的要求总有差距。快速原型能较快地使一个功能精简的样品投入运行，这由贴近 t_0 的一段短线来表示。由于有一个样品给用户试用，因此双方可取得较好的理解，可望用比传统方法较少的时间做出功能较强的软件，t_1 前的一段竖线就表达了

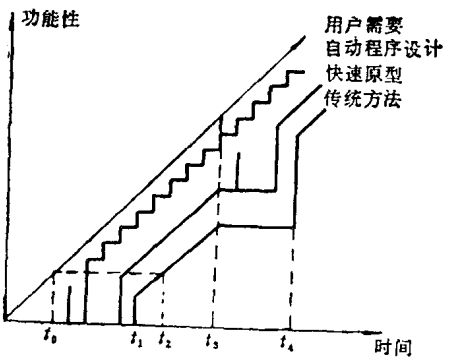


图 1.7　软件开发模型比较

这种情况。当然投入运行后的软件还是需要维护的。总的来看，这根曲线比传统方法更靠近用户的需要。自动程序设计模型由于有机器较多的辅助，开发人员理解了用户的需求和用宽域语言（wide-spectrum languages）写下这些需求，就可自动综合出一个系统了。一方面开发时间大大减少，另一方面开发成本也大大降低，因此第一个软件完成的时间紧靠 t_0，达到的功能也相当接近用户的要求，而且以后需要的功能增强时，重新合成一个也方便，可以频繁地修改需求和重新生成新软件，使曲线成为阶梯形，以求尽可能满足用户的需求。可见，解决软件危机的出路在自动化，但目前自动化程序设计能做到的程度有限，特别是对大规模的程序，还无能为力。随着知识工程在软件开发环境中的应用逐步广泛和深入，自动化的程度必将逐步提高。

1.3 软件开发方法

软件开发是这样一个过程：从对用户的需求进行分析开始，经过设计和实现，最后产生可以正确执行的代码和有关使用和维护的各种文档。软件开发方法就是完成这个过程的方法。现有的开发方法主要有三种类型。

1．传统的非形式化化开发方法

它的基本思想是引入一些非形式的图形和文字记号，如数据流程图、结构图、结构化英语和判定表等，提供一定的设计原则，协助开发人员按照一定的步骤，比较明确和简练地书写设计文档，用人工开发出所要的软件。

这种方法有以数据流为基础的结构设计方法，以数据结构为基础的 Jackson 方法等。这些方法在软件工程类的书上都有介绍，这里不再赘述了，但常用的标记方法我们将在第三章进行介绍。

2．严格的形式化开发方法

它的基本思想是对系统建立一个数学模型，研究和提供一种基于数学的或形式语义学的软件规格说明语言，用这种语言严格地描述所开发的软件功能，并由计算机完全机械化地将它转换成可以执行的代码。

维也纳开发方法（vienna development method，简称 VDM）就属这种方法。VDM 是在 1973 年由奥地利的 IBM 维也纳实验室首先提出的一套开发大型软件系统的方法。它以指称语义为基础，充分利用了 60～70 年代发展起来的以数学为基础的形式化技术，使软件开发过程具有高度的严密性和系统性。它是早期用于程序设计语言定义及其编译，后来成为应用于操作系统、数据库、办公室自动化等软件领域的通用方法。

近几年，国外研制了许多 VDM 支援工具，如 UNIX 环境下的词法分析程序生成器 Lex 和语法分析程序生成器 Yacc 等。但由于 VDM 要求专门的数学基础，因此使用范围还远不及传统的非形式化方法广泛。本书也不再对它们作深入的介绍。

3．逐步从非形式化向形式化过渡的开发方法

这是一种介于形式化和非形式化之间的开发方法，即从非形式化规格说明逐渐向形式化源代码过渡的方法。过渡过程中的各种中间文档用非形式化语言和形式化语言的混合体来描述。随着设计的深化，文档中的形式化部分逐渐增多，而非形式化部分逐渐减少，直至完全变成形式化的可执行的源代码。

表 1.1 对非形式化和形式化开发方法作了简明的比较，指出了两者之间的重要差别。

表1.1 非形式化和形式化开发方法比较

	非形式化方法	形式化方法
软件规格说明	非形式的。	用严格的形式化语言描述。
软件原型	可有可无。若需要，人工编制，软件完成后不再有用。	说明本身就可认为是原型（因为它是可执行的），以后仍有用。
软件确认	对源程序进行。	对说明进行。
编　程	手工实现。	机械执行或计算机辅助完成。如果需要人工干预，决策过程被记下。
测　试	对源程序进行。	理想情况下可不进行
维　护	对源程序进行。	对说明进行，用"重演"方法生成新的源程序。
优化决策	不作记录，软件完成后就失传了。	作记录，"重演"时，仍起作用。

从以上列举的特点可见，传统方法由于采用不严格的非形式化的方法来描述软件规格说明，所以很难得到计算机系统的有效支持和实现自动化，这对提高软件质量和生产率显然是不利的。反之，形式化方法易于实现代码生成自动化。因此，世界各国的专家对形式化方法给予高度的重视。如美国国防部的STARS计划认为这是20世纪90年代软件开发的新模式，是革命性的方法。欧洲共同体的ESPRIT计划的研究重点也是形式化开发方法。由于计算机的广泛应用，软件系统的功能千差万别，如何用一种包罗万象的形式化语言把它们完全严格地描述出来呢？看来研究任务是十分艰巨的。

1.4 软体开发环境的分类

软件开发环境的分类方法很多。这里介绍三种：一种是按解决软件开发中不同类型的问题来分的；一种是按现有的软件开发环境的演变趋向来分的；还有一种是按集成化程度来分的。

1.4.1 按解决的问题分类

软件开发中遇到的问题主要出现在三个级别上：程序设计级、系统合成级和项目管理级。软件开发环境也应该在这三个级别上给予支持。

1. 程序设计环境

程序设计环境主要解决一个相对他人独立工作的程序员如何把规范说明转变成可工作的程序，即属于局部编程（programming-in-the-small）的范畴。这个过程包括两个重要部分：方法和工具。其中方法（例如"结构化编码技术"）可能是更重要的部分，因为对于设计和编码都很差的程序而言，再好的工具也不会是灵丹妙药。但作为软件开发环境而言，我们以后将把重点放在工具上。

2. 系统合成环境

系统合成环境主要考虑把很多子系统集中成一个大系统的问题，即属于全局编程（pro-