# 实用软件工程环境导论

吴广茂　贾大强　主编

国防工业出版社

# 实用软件工程环境导论

吴广茂　贾大强　主编







国防工业出版社

（京）新登字 106 号

图书在版编目(CIP)数据

实用软件工程环境导论/吴广茂，贾大强主编．—北京：国防工业出版社，1994
ISBN 7—118—01137—1

Ⅰ．实…
Ⅱ．①吴…②贾…
Ⅲ．软件工程—环境因素
Ⅳ．TP311.5

实用软件工程环境导论
吴广茂　贾大强　主编
*
国防工业出版社出版发行
（北京市海淀区紫竹院南路 23 号）
（邮政编码 100044）
新华书店经售
北京孙史山胶印厂印刷
开本 787×1092　1/16　印张 13½　305 千字
1994 年 6 月第 1 版　1994 年 6 月北京第 1 次印刷　印数：1—1000 册
ISBN 7—118—01137—1/TP・148　　定价：12.80 元

# 前　言

自从“软件工程”(Software engineering)作为一个专门的学科问世以来，已有20多年。作为软件工程学科当今最活跃的分支之一，“软件工程环境”(Software engineering environment)已经在世界范围内引起了越来越广泛的关注。

软件开发，尤其是大型软件开发的复杂性，带来了一系列组织与技术问题，这些问题的解决需要合适的方法、规程与工具。为使软件生产进程合理化并改进软件最终产品的质量，人们付出了不懈的努力。探讨了多种方法，制订了许多标准、规范和规程，研制了大量软件工具，软件开发在理论研究和工程实践方面都取得了长足的进展。但时至今日，人们仍未完全摆脱“软件危机”(Software crisis)的困扰，软件生产效率，软件本身质量，远未达到使人满意的程度。究其原因，有下列两个方面：

(1) 当前普遍使用的方法、规程和工具，只对软件生存周期中孤立阶段，单个活动提供支持，而对相关的开发阶段和相关的开发活动则无法提供必要的辅助；

(2) 针对软件开发在技术上、管理上提出的方法零零星星，但这些方法都只是作为一种补救措施，而缺乏一套一体化的协调一致的解决办法。

在软件工程进入第三个十年之际，软件工程环境已经成为一个热门课题。建造一个支持软件全生存周期，适合不同层次软件开发人员使用，支持软件生产各方面活动的集成化软件工程环境是软件工程界的一个主攻方向。世界发达国家为此耗费巨资，投入大量人力，如美国的 STARS，日本的 Σ，英国的 ALVEY，欧洲的 ESPRIT 等，这些耗资巨大的项目，近年来都取得了一些令人瞩目的结果。

人们寄于厚望的建造软件工程环境的目标得以实现，将会对现存的软件工程方法学产生重大影响，甚至引起一场变革。人们将最终抛弃软件开发的落后的手工方式，步入一个软件生产方式的崭新天地。

为使读者对软件工程环境有一个较为全面的了解，我们选取了国外较为有名的有关软件工程环境方面的工作，做深入的介绍。本书共分为八章，其内容是这样组织的：第一章是本书内容的一个概括，给出了软件工程环境的概念，采用根据内在性质和结构形式将软件工程环境分类的提法，介绍了这种分类方法下各类环境的特点。对全书其他各章的内容，在第一章中也有一个提纲挈领的提示。第二章介绍了 Ada 程序设计支持环境(APSE)的层次结构、设计背景、基本原理和设计指南，分析了 APSE 的需求。环境数据库是 APSE 的核心部分，存放着软件生存期中与开发有关的一切信息，界面管理提供环境数据库和工具集的界面控制语言，工具集中包括了用于程序设计、程序维护既相互独立又相互联系的工具组。第三章通过介绍 CAIS (通用 Ada 程序设计支持环境接口集)的技术基础，讨论了 CAIS 的13个技术特征，研究了 CAIS 的实现及其在软件集成中的作用。论述了 CAIS 中嵌入的一组集成化程度很高的概念，依据这些概念协调集成工具集的

建造者与用户之间的需求，以获得工具的可移植性与集成以及数据的可移植性。CAIS 设计的许多方面可以应用到一般环境的构造与集成上。第四章介绍了欧洲联合开发的一种标准的软件系统 PCTE（可移植公共工具环境）的目标、原理和实际设施，讨论了 PCTE 的基本机制，包括执行、通信、进程间通信、对象管理系统、并发性和完整性控制、分布机制和仿真设施，重点介绍了 PCTE 的用户界面组成基本原理和实现。PCTE 的出现是软件工程环境走向标准化的重要一步，它提出的观点已经为许多软件工程环境广泛采用。第五章介绍了德国开发的一个较为典型的软件工程环境 EPOS（面向工程设计和项目管理的开发环境），通过对 EPOS—R、EPOS—S、EPOS—P 三种规格化语言和 EPOS 工具的分析，论述了 EPOS 在软件项目中作为所有开发人员的得力助手，对软件生存期的各阶段提供支持，把他们从纷乱繁杂的重复劳动中解放出来，去从事更具创造性的工作。第六章介绍了 IBM 系统应用结构 SAA，IBM 称 SAA 为一种体系结构，定义了一致的一组接口、约定和规程，跨越了 IBM 的三个主要的计算机系统环境，为用户提供一个跨越这些环境的一致的视图。只要按照 SAA 规范开发应用程序，便可方便地从一个系统移植到另一个系统，不必做修改或只做少量的修改，这就基本上使用户摆脱了移植程序的苦恼。本章论述了 SAA 的构成、公共程序设计界面、公共用户界面、公共通信支持以及如何在 SAA 的基础上建立应用软件。第七章介绍了建立在 VAX 计算机上的 VAX set，这是一组软件工具的集合，是支持软件生存周期活动的一组强有力的工具，为软件开发和维护提供了坚实的基础。本章分别给出了六个 VAX set 工具的基本思想、组织结构、作用、功能及使用特点，通过工具之间的联合使用可发挥 VAX set 的更大潜力。第八章探讨了嵌入式系统软件开发的特点，介绍了构造以仿真器为核心的嵌入式系统开发环境，分析了 SDE、ICE 的设计思想，论述了以仿真器为核心构造嵌入式系统开发环境的可行性，使应用十分广泛的嵌入式系统的软件开发得到强有力的支持。

此书是在航空航天工业部软件工程化工作小组“软件工程环境”项目的进展中完成的。其中第一章由吴广茂、王玉玺、贾大强和汤幼宁编写，第二章由廖彬山和高仲仪编写，第三章由童长忠和周伯生编写，第四章由张祺和李声远编写，第五章由郭广旭、贾大强和吴广茂编写，第六章由李凌雪、潘文星和吴广茂编写，第七章由袁志禄编写，第八章由牛文生、周耀荣和彭德文编写。软件工程环境研究领域毕竟仍是一个新的领域，我们仍缺少更多的实践。在此我们对在本书的写作中曾给与我们帮助的同志表示谢意，并向所引用参考文献的作者表示感谢。

书中错误在所难免，希望软件工程界同仁及广大读者指正，我们将不胜感激。

吴广茂　贾大强

# 目　录

# 第一章　软件开发环境及本书的内容综述

## 1.1　引　　言

“环境”(Environment)这个术语，目前国内尚无统一定义，甚至连“IEEE 软件工程术语标准词汇”(IEEE Std 729—1983)中也未列出明确条目。可是，这个词在软件界已使用得相当广泛。通常，把“环境”看成是软件开发者建立软件系统所使用的硬件工具和软件工具的集合。随着技术的发展和用户期望的提高，环境的功能也随之“水涨船高”。

“编程环境”(Programming environment)和“软件开发环境”(Software development environment)这两个术语产生的时间有先后，但有时被混杂使用，其实，它们的词义不尽相同，其内涵区别很大。编程环境系指仅支持软件开发期中编码阶段的环境；软件开发环境则指支持软件开发期中各阶段全部活动的环境。

软件工程环境(Software engineering environment)比“编程环境”、“软件开发环境”的内函更丰富，它应支持整个软件生存周期(包括初始期、开发期、运行和维护期)中开发和管理的全部活动。这正是软件界莘莘学子孜孜以求的目标，但直至目前，仍是壮志难酬。

环境有别于操作系统及其某些服务程序(例如：存储管理、数据管理、多道程序管理等)，环境比操作系统及其基本服务程序具有更强的功能。这些功能通过一些典型的工具予以实现。如窗口管理程序、浏览程序(Browser)、配置管理程序、任务管理程序等。

近年来，人们把注意力转向了对集成软件工程环境〔(ISEE，Integrated Software Engineering Environment)〕的研究。1989 年 5 月召开了第一届 ISEE 研讨会，半年之后，于同年 12 月份召开了第二届 ISEE 研讨会，在 1990 年 5 月召开了第三届 ISEE 研讨会，会议如此频繁，可见对这个课题兴趣之浓厚。1990 年研讨的重点是 ISEE 的需求以及接口技术；1991 年的目光已放在确定一个通用的相对完善的软件工程环境的结构模型上。

## 1.2　软件开发环境的分类

问世的软件开发环境，已见诸报刊杂志的不下数十种，正在研制的恐为数更多。这些环境规模各异、对象不同，所依附的计算机操作系统五花八门。为了对环境的现状有一个概括的了解，我们根据环境的内在性质和结构形式，将环境分为四类。

(1) 以语言为中心的环境(Language—centered environments)。这种环境是围绕着一种语言建立的，提供了一套适合于此种语言的工具，这种环境的交互水平很高，而且

支持编写大型程序。

(2) 面向结构的环境 (Structure—oriented environments)。这种环境的特点是用户可对结构进行直接操作。所用的技术与语言无关，对不同的语言，环境可提供不同的生成器，以便将结构设计转换成某种语言描述的代码。

(3) 工具箱环境 (Toolkit environments)。顾名思义，这种环境犹如一个工具箱，集有许多软件工具。这些工具用途相当广泛，其中有些工具可支持编写大型程序，且与所使用的语言无关，如配置管理和版本控制工具。个别环境还能对工具的用途进行控制和管理。

(4) 基于方法的环境 (Method—based environments)。这种环境支持软件开发过程中的多种活动，其中包括开发组的任务和项目管理的任务。在这些环境中，也为特殊的规格说明和设计方法提供了一些工具。

把环境分为上述四类，其方法是仅按环境的发展方向予以划分，而不是把若干环境罗列出来再进行分类。实际上，某一种环境可能同时具有上述四类中某几类的特征，这就表明这种环境力求适合更广泛的领域，可以在这些领域中收到更多的反馈信息，促进其进一步的研究和发展。

对环境的发展趋势，不同的专家持有不同的观点。若从一位“软件工具制造者”的观点来看，环境的发展应把重点放在工具的集成上；若从一位“专家系统构造者”的观点来看，环境的发展应力求使软件开发过程自动化，也就是说要用到知识库方面的技术；若从一位“环境用户”的观点来看，环境的发展应满足用户多方面的需求。不同专业、不同技术层次的用户，对环境的发展有着不同的需求，其涉及面相当广泛。但无论如何环境至少应考虑下述几项：

(1) 从时间上，应支持整个软件生存周期中各个阶段上（从需求分析到设计、编码、测试乃至维护）软件的开发和管理，有跟踪各阶段上信息的能力；

(2) 从广度上，应支持用户开发不同规模软件，也就是说，既支持单用户编小程序，也要支持多用户编大程序；

(3) 从灵活性上，应支持用户对环境的剪裁和扩充。环境具有开放性，为使环境适应新设备及新技术，用户可以向环境增加新的软件工具。

为了说明每一类环境的特点，下面将适当引用某些具体的环境，但不对其细节作过多说明，也不对这些环境作褒贬性的评论。这是因为，用户属于不同的专业领域和技术层次，他们有不同的需求，不可能对各种环境定出一个统一的衡量标准。而且这些环境要在不同的硬件和操作系统上运行，也会因之具有某些相应的特色。就目前的情况看，还没有哪一个环境能使所有的用户都满意。

### 1.2.1 以语言为中心的环境

所谓以语言为中心的环境，是指那些专为支持某一种语言而建立的操作系统和工具集。目前在国际上著名的以语言为中心的环境颇多，例如，用于 LISP 语言的 Interlisp，用于 Mesa/Cedar 语言的 Cedar，用于 Smalltalk 语言的 Smalltalk 和 Smalltalk/V，以及用于 Ada 语言的 Rational environment。Lisp 环境建立得最早，可追溯到 60 年代后期。在 70 年代中期，研究者们同时做了 Cedar，Smalltalk 和 Interlisp 环境。对 Lisp 研究的鼎盛

时期是80年代初期，在这个时期定义了COMMON Lisp语言，诞生了Rational environment环境，1980年Xerox研究小组正式推出了Smalltalk—80系统，Digitalk公司1986年在IBM PC机上推出了Smalltalk/V，这是一个集成化的、图形交互式的、开放式的、既可剪裁又可扩充的编程环境。

Lisp环境对开发以语言为中心的环境最有影响。它引伸出了一种探测方式（Exploratory style）编程的概念，显示了让用户知道语义信息的益处。此后，相继开发的那些针对其他几种语言的环境，扩充了这种概念，这样，环境可以支持编写大型软件。

以语言为中心的环境，鼓励用户使用探测方式编写程序，使用这种方式可以加快程序的编写、执行、测试、排错和修改。由于这些环境在实现方法上考虑到应用程序与环境相匹配的问题，因此这些环境的所有工具都适合于用户开发应用程序。

以语言为中心的环境结合了特定语言的实现方法。例如，为了支持用户快速生成软件原型，Interlisp使用了一个很大的虚存空间。而它的工具在寻址空间中作为一个完整系统，应用程序嵌入到同一个寻址空间。

应用程序嵌入到环境中，使得应用程序的编写者在建立一个应用程序时，可以使用环境中的全部工具。由于环境和应用程序共享同一种语言，所以应用程序具有该环境的全部特征，就好像是环境的一个建筑块一样。例如，Interlisp是一个“驻留系统”，Lisp程序就作为一个数据结构驻留在运行系统中。这样，可以使用户利用复用程序（看作是环境的一部分）快速建成一个应用程序原型。用同样的方法，用户还可以扩充环境的工具，以满足某些特殊需要。这种方法，在Smalltalk中尤为明显。

为了生成这些信息，可能要用一些诸如浏览程序之类的工具。这种环境帮助用户理解编码在开发过程中的状态和结构。浏览还有这样一层意思，即沿着程序块进行“航测”，并对目标块及其之间的关系提出询问。Interlisp环境中的工具Masterscope就属于首批的浏览程序，它产生出用户在开发程序中要用的语义信息。浏览程序作为一项基本工具，在探测方式编程的发展中起了重要作用。其实，在程序维护中，它也具有很大潜力。维护者通常不是程序的开发者，维护工作只能依靠源程序和某些文档资料。在对一个大型程序进行修改之前，维护者常常要花很长时间去理解程序的结构及其内部的连接关系，浏览程序作为一项工具，恰好可以帮助维护者了解程序的结构，标出内部连接关系。从而可以明确地看出当程序的某一部分发生修改时，它所波及和影响的范围如何。有些浏览程序可让软件维护人员交互式检查程序结构，查询修改后的波及和影响范围。

高级编程语言不完全支持建造大型系统的全部活动。例如，Ada提供了一些程序包，这些包支持模块化，但不对已完成的各种版本进行控制。鉴于这种情况，在以语言为中心的环境中添加了一些工具，以支持编写大型程序系统。例如，对模块多种版本的控制和管理技术，目前已有了很大发展。这些技术为版本控制和配置管理提出了更形式化的定义。

由于实现这些环境时使用了依赖于语言的某些特殊技术，所以这些环境通常不支持别种语言，而且也不便于应用程序的移植。

目前，商品化软件系统的开发者们正苦心精炼软件系统的实现技术，旨在改善软件系统的性能。他们也正为命令式语言（如C和Modula—2）建立以语言为中心的环境，并希望把这些环境升级以支持设计阶段，还力图把项目管理的一些技术也结合进去。研究界正致力于把以语言为中心的某些技术，用到像Prolog那样的语言中去，也想用到规格说

明语言中去。

### 1.2.2 面向结构的环境

面向结构，指的是面向程序的结构。面向结构的环境，其本来的思想是为用户建立一种交互式工具，即一种语法制导编辑程序。后来它被扩充成一种编程环境。这种环境支持以交互方式进行语义分析、程序调试等工作。通过这种环境的中心部件——编辑程序，用户可以交互式操作所有结构，进而继续完善，使之能支持编写小程序和大型程序，并支持那些诸如历史记录和存取控制表之类的结构。这样，开始时使用的术语“语法制导”，就慢慢地被“面向结构”所代替了。

面向结构的环境在几方面为环境技术作出了重要贡献：对程序结构提供了直接操作；从同一程序结构生成几种程序图；用户可对静态语义和语义信息进行逐步检查；最重要的是，为了形式化描述一种语言的语法和静态语义，生成了一种结构化编辑程序。

面向结构的环境，可以使用户直接交互式地操作程序结构，而不必记忆其详细的语法。程序的正文显示在屏幕上，用户可以直接修改。早期的环境（如 Emily）使用程序结构分析树的方法。近年来，大部分环境使用抽象语法树的方法，这种方法是由 Mentor 环境首先引进的。

面向结构的环境可用几种方法操作结构，其中之一是纯结构编辑，它可以看成是初级的模板驱动编辑（Template—driven editing）。Aloe 编辑程序就是该方法之例。

另一种方法是混合模式操作（Mixed—mode operation），这种方法已用在几种商品化的面向结构的环境中。例如在 Rational environment 中，用户可以对正文表示法进行操作，也可以对结构进行操作。用户读一段程序，就像读正文一样，然后由环境对它进行处理。环境把这一段正文用渐增式方法转换为程序的结构。用户可以使用那些对程序结构和程序正文使用的命令，对该程序进行编辑。

面向结构的环境可以把一个程序的结构生成其正文表示法，从同一个结构可以生成不同的表示法。因此，用户就可以利用面向结构的环境从详略不同程度上观察程序。当浏览一个大型程序的结构时，可以开设不同的窗口观察不同详略的程序内容。在研究性的环境 Gandalf 中有这种功能，在商品化的环境 Rational environment 中也有这种功能。

当人们建立了语法制导编辑程序之后，就会发现正确的语法只是支持程序员的一个方面而已，为了给程序员更多的帮助，还需把静态语义分析程序加到编辑程序中。用户可以存取语义信息（诸如标识符的定义、使用它们的位置等），然后用编辑程序把这些信息打印出来。所以，一个面向结构的环境不限于向用户立即反馈语法错误，也要报告静态语义错误。这些工作要在用户结束对程序的编辑之前做。这就意味着，环境必须跟踪用户对程序的修改，而且分析那些受影响的部分。

渐增处理的方法，在 LOIPE 环境中用作编码生成和连编。当中要解决的一个矛盾是，处理尽可能小的单元以及减少算法复杂性这二者之间的折衷问题。在不同的级别上可用不同的处理类型。例如，若在程序单元级上检查静态语义，则在语言结构级上处理语法，并且在模块级上生成编码。

面向结构的环境一个主要的贡献是，它们可以支持对程序结构以一种与语言无关的方式进行操作。环境的开发者为了达到这一点，封装一种语言的语法性质和语义性质。

Aloe 编辑程序具有描述一种语言语法的能力。Conell program synthesizer 是支持描述语义性质的一个著名的环境。

面向结构的环境已较为成熟，在市场上已有商业化产品出售。环境的构造者对不同的语言可以不用付诸很大的力量就能生成某些结构编辑程序。许多情况下，这样生成的编辑程序往往仅限于语法检查，而语义分析程序还是要人花时间构造的。

面向结构的环境，开始是作为教学辅助工具而问世的，仅支持编码阶段并且只对编写小程序有效。后来人们为扩大面向结构的环境的应用范围，曾作了多方面的努力，以求支持编写大型程序以及编写较困难的程序。尽管如此，目前许多面向结构的环境所使用的技术，仍有不少弱点，比如编写大型程序的效率还不很高，而且在协调多用户的并发存取方面也存在一些问题。

为嵌入式软件建立开发环境，是非常重要的课题。为用于飞行器计算机提出的集成化系统设计环境（SDE，System Design Environment）颇具代表性。嵌入式系统具有许多特点，其最明显之处是：

（1）嵌入式系统的软件密切依赖于硬件，这就要求支持软件开发的工具必须同支持硬件开发的工具集成在一起；

（2）嵌入式系统几乎总是实时系统，这就要求支持这种系统的设计工具必须分析时间分配，以确定这个系统是否满足运算需求；

（3）嵌入式系统通常用于安全性要求很高的项目（比如飞机、卫星等）中，这就要求在嵌入到实际系统之前，必须用模拟环境支持设计验证。

据称 SDE 能满足这些需求，它支持嵌入式系统的设计、实现、集成和测试各个阶段。SDE 的工具中包括：①设计搜集工具（Design capture tools），帮助开发者把系统按层次构形，进行模块分解，确保系统的功能与需求一致；②设计实现工具和编译工具，包括语言的编译程序、连编程序等；③设计模拟工具，为系统的运行提供了一种软件模拟过程。

SED 系统通过设计过程管理系统（DPMS，Design Process Management System）集成了包括上述工具在内的多种工具。DPMS 提供了一个友好的公共用户界面和一个设计数据管理系统。DPMS 除了集成各种设计工具之外，还为用户提供了许多信息，以帮助用户使用各个工具。

设计数据管理系统的功能是进行配置管理和版本更新管理。这是通过一个面向对象的设计数据库而实现的。设计数据库中的信息与一个规则库结合在一起，以控制各个独立工具的操作。DPMS 可以很灵活地被用户修改和扩充。用户可以修改规则库，也可以建立新的规则库，从而集成自己所需要的新的工具。

### 1.2.3 工具箱环境

工具箱环境就像一个“箱子”，装着若干软件工具。工具可以扩充，也便于移植。早期的工具箱环境只是想提供一些工具支持软件生存周期中的编码阶段，对这些工具也没有什么严格的控制和管理，说穿了就是操作系统加编码工具。这些工具有编译程序、编辑程序、汇编程序、连编程序和调试程序，还有一些支持大型软件开发的工具（例如版本控制和配置管理程序等）。工具箱方法以用户需求为驱动，独立于语言，支持开发大型软件

项目。这些特点促使工具箱环境迅速发展，各个计算机厂商竞相在自己的机器上推出适合用户需求的工具箱环境。如在市场上流行的有 Unix/PWB (Programmer's Work Bench)，DEC 公司的 VMS VAX set，Apollo domain 软件工程环境 DSEE，可移植的公共工具环境 (PCTE，Portable Common Tool Environment)，以及 Ada 的公共 APSE 接口集 (CAIS Common APSE Interface Set)。

在本书的后面几章对 VAX set，PCTE，CAIS，EPOS 等环境有详细介绍。在这里我们只概述工具箱环境的几个特点。

Unix 这种操作系统鼓励用户对它进行扩充。因为它的数据组织方式很简单（用 ASCII 字节流方式），工具之间的数据交换和数据存储都很方便，所以用户容易裁剪 Unix 环境。例如，增加一些新的工具，修改若干已有的工具。Unix/PWB 对什么时候使用工具，如何使用工具等问题，似乎未加约束。这种模式的好处是为用户使用这些工具提供了灵活性，而不是像有的工具那样，靠“一致性”和“自动管理用户活动”，给用户增加了限制。这些工具的简单性及其交互能力，使之易于移植到类似的环境。

DSEE 有些工具嵌入到它的文件系统中，提供一种版本控制机制，使得环境的用户可以透明地使用软件工具。而像 PCTE 和 CAIS 这样的环境就支持类的对象，因为它有一套可扩充的对象属性。

VAX set 所蕴藏的可供用户广为使用的工具，几乎贯穿软件生存周期的各个阶段（如图1.1）。

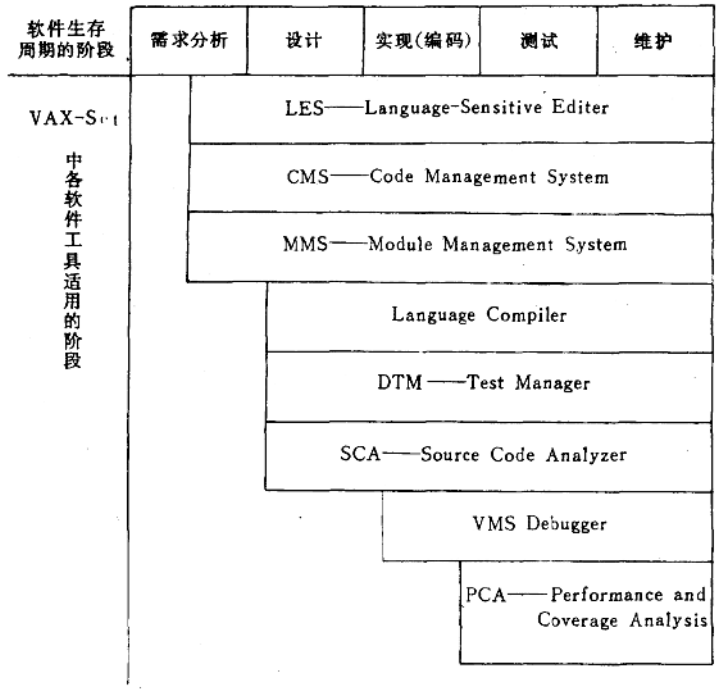


图1.1 VAX set 中的软件工具与软件生存周期各阶段的关系

为了提供更多的环境所定义的控制工具，开发商品化软件环境的设计师们就使用了一种方法，即把更高层的接口放在一般操作系统中用户命令接口的顶层。像项目管理用的那些环境，就能使用户用该环境中高层功能进行工作。同时也可以跳回到原来操作系统（诸如 Unix 和 VWS）的命令级上。

工具箱环境为编写大型程序提供了许多支持工具，这些工具不依赖于编程语言。这些工具的目的是使程序员易于管理源程序，对源程序的版本号提供了一种管理机制。Unix/PWB和 VMS VAX set 提供了一些相当著名的工具。例如，Unix 中的源程序控制系统 SCCS (Source Code Control System)，VMS 上的源程序管理系统 CMS，名字稍异，功能雷同，都可以实现源程序的版本控制。这些工具除记录一套完整的源程序外，对其它的版本，只记录源程序中改动的部分，极大地节省了存储空间。而当用户取用某一版本的程序时，它又产生该版本下的完整程序。

工具箱环境使用操作系统的某些功能，再“粘连”上一些工具，构成了一个集合体，旨在形成一个不依赖于语言的环境。它用某些适当的工具支持多种语言。这种环境具有高度的可剪裁性，但对软件工具集合体的管理和控制，提供的能力较弱。所以用户必须建立一些管理措施，以保证正确使用这些工具。由于这些工具的可剪裁性和可移植性，使这些工具得到了广泛应用。尽管如此，工具箱环境对大型软件系统的维护仍不能提供很强的辅助能力。

### 1.2.4 基于方法的环境

每一个基于方法的环境，都支持研制软件的一种特定的方法。我们可以把研制方法划分成两个大的方面：开发方法和管理方法。开发方法是指软件开发周期中各阶段上适用的开发软件的方法；管理方法是指管理开发过程的方法。这两种方法的用户也有区别，开发方法是由软件开发者在不同的阶段上使用；而管理方法由管理人员使用，对人员和活动进行管理，使得众多的开发者所开发的产品能有次序地协调一致地进展。所谓基于方法的环境就是支持开发方法和管理方法。

（一）对开发方法的支持

开发方法支持软件开发周期中的各个阶段。不同的方法其形式化程度不同：有的方法可能是非形式化的，如写正文；有的方法是半形式化的，像用检查工具对文本和图形的描述进行有限度的验证；还有的方法是形式化的，它用一种基本理论模型，而不仅是一种可以验证的描述。

适用于规格说明阶段和设计阶段的有一些半形式化方法，如 SREM、IORL、CORE、SDL、SADT、PSL/PSA，各种形式的数据流图和控制流图，以及实体—关系（E—R）图等。在规格说明阶段上用的更形式化的方法有 Petri 网，有限状态机，以及几种规格说明语言，诸如 GIST，Refine，VDM 和 Anna 等。从70年代中期开始，几种半形式化方法在实践中有了不同程度的应用。同一时期，形式化方法还很少用。

起初，开发方法的各种工具是在主机上提供的，而且所使用的也是字符终端或特殊的图形终端。像 TAGS 和 DCDS 就属于这种系统。由于 PC 机和工作站提供了多种图形设备，从而激励人们开发大量的商品化工具（尤其是对模式设计的半形式化方法），这些工具常常称为计算机辅助软件工程工具（或简记为 CASE）。这方面的产品不乏其例，如，In-

dex Technology 公司的 Excelerator，Nastec 公司的 CASE 2000，Cadre 公司的 Teamwork，以及交互式开发环境 Software through Pictures。上述这些工具支持用户交互式地进行图形设计或修改图形设计，还帮助用户把设计抽象成一个分级的模型，而且还能进行一定程度的一致性检查。用户也能使用某些分析程序（诸如层次检查程序），以检查所用名字的一致性，设计层之间的关联性。分析程序产生的交叉引用信息放在数据字典中，也可交互查询。

研制 CASE 的许多厂商已实现了支持几种设计方法的工具，而且用户也可以修改这些工具，使之适用于用户自己使用的那些设计方法。最近推出的一些软件工具，支持用户使用图形编辑器定义自己的图形符号，也支持编一些分析程序。在有些情况下，一种工具对不同的方法可以生成不同的适用版本，所谓对不同方法的集成，就是只集成选单中可以选到的那些图形编辑器和分析程序的适用版本。对有些方法，所不同的只是符号的形状（例如，Merise 流程图和 ER—Chen 流程图就是这样），一种工具可以用二套符号集中的任一种进行设计。这种工具的各种适用版本的交叉引用信息都存放在同一个数据字典中。

EPOS (Engineering and Project—management Oriented Support system) 是一个较为典型的软件工程环境。西德乃至欧洲广为使用。EPOS 是一个集成的开发环境，这个系统的特点是：

(1) 不仅支持开发，也支持项目管理、质量保证和配置管理等活动；

(2) 支持项目开发中的各个阶段；

(3) 不仅支持软件项目，也支持软/硬件系统项目；

(4)支持多种设计方法。如，event—oriented，function—orien ted，module—oriented，data flow—oriented，data structure—oriented 等；

(5) 支持图形显示功能；

(6) 支持整个开发组。

EPOS 的组成部件及各部件之间的关系如图1.2所示。

·有三种规格说明语言，分别是：

EPOS—R (Requirement specification) ——需求规格说明语言，描述功能和方案；

EPOS—S (System specification) ——系统规格说明语言，描述硬件和软件设计；

EPOS—P (Project & configuration management) ——项目和配置管理规格说明语言，描述项目和配置管理。

·一个项目数据库，其中也有几个程序，能处理三种规格说明语言 (R，S，P) 所描述的信息；

·几个软件工具系统，处理 EPOS 数据库中的数据，例如：管理支持工具系统；对出错进行分析的工具系统；可自动生成全套文档的文档工具系统；设计方法支持工具系统；还有编码生成工具系统（从设计规格说明，按选定的语言，EPOS 自动生成源程序文件）。

·一个用户控制程序 (称为 EPOS—C)，使用户与 EPOS 之间进行通信。

图1.2所示的大致流程是，用户用 R，S，P 三种规格说明语言书写相应的内容，EPOS 进行分析，对其错误部分输出，反馈给用户，用户修改后重新输入。将正确的规格说明记入 EPOS 数据库，EPOS 的各种软件工具系统对 R，S，P 生成相应的内容。

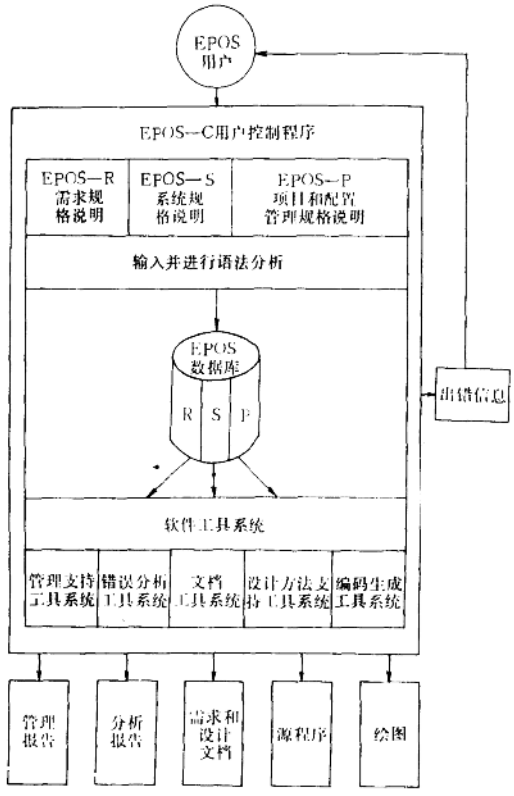


图1.2 EPOS 系统示意图

（二）对管理开发过程的支持

对软件开发进行管理，包含两层意思，一是管理产品，二是管理过程。说的更清楚些，即管理开发中的产品，以及管理开发和维护该产品的过程。

支持管理产品的工具包括版本管理和配置管理所用的那些工具；支持管理开发过程的工具包括项目管理（项目计划和控制）工具，任务管理（帮助开发者组织和跟踪任务）工具，通信管理（掌握和控制项目组织内的通信）工具等。

起初，专门支持管理的那些工具（诸如进度管理、成本估算、或对改变需求的控制管理等）都是孤立研制的。现在，人们试图把开发过程及其管理予以整体化和形式化。

EPOS 系统在项目管理和产品保证方面有丰富的支持工具，这是该系统的一个特点。EPOS 的管理工具系统具有丰富的程序，擅长进行项目计划、项目控制、配置管理和质量保证等。当系统的需求文档得到认可之后，EPOS 管理工具可生成工作任务分解图、项目组织结构图、职责矩阵图以及网络图。

基于方法的环境支持特定的开发方法，也支持对开发过程的管理。供单用户使用的许多工具已能支持半形式化的图形开发方法。现在环境的研制者们已不再开发那些针对特定方法的工具，而更强调工具的通用性。

从需求规格说明的开发到软件的实现，要把这个过程实现自动化，其先决条件是形