# 第一章　计算机辅助软件工程技术的现状

## 1.1　理想的软件开发环境

设想你正在自己专用的工作站上开发软件(见图 1.1)。这可能是一台带有彩色图形显示器和打印机的高档个人计算机。局域网把有关的工作站连在一起,网络上还有一台主机存放着公司的词典和数据库。工作站装有各种强有力的软件工具,因此,计算机能够辅助你完成软件开发过程中的各项工作,从帮助你跟踪正在执行中的任务到尽快地建立起正在开发的软件系统的原型。你可以用键盘或鼠标与工作站通信,检查所做的工作,访问由可重复使用的"软件芯片"组成的程序库,执行许多例行事务,这些例行事务是每个软件开发项目的一部分。

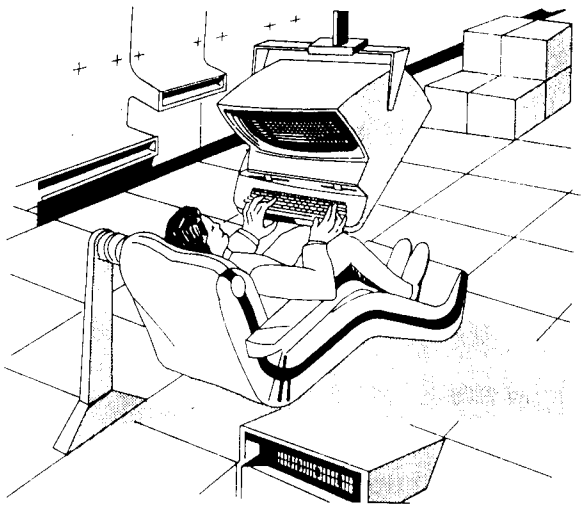

图 1.1　设想的专用软件开发工作站

虽然目前已经有许多强有力的软件工具,但和工作站相比,最主要的差别在于工作站能为你提供一个使用方便的集成软件工具包。由于使用了公共的用户接口,用户看不到每个工具之间的差别。换句话说,工作站提供了完整的硬件和软件集成环境,以支持软件的开发工作。这就大大地简化了软件工具的使用方法,减少了学习使用这些软件工具的难度,从而提高了软件开发效率。

尽管以上描绘的这幅图画还只是未来软件开发过程的幻想,然而,幸运的是,这里所说的已经接近成为现实了。

## 1.2 计算机辅助软件工程技术

目前，软件工具正在发生戏剧性的变化。许多用在微机上的软件工具正在建立，其目标是实现软件生存期各个环节的自动化。这些工具主要用于软件的分析和设计。使用这些工具，软件开发人员就能在个人计算机或工作站上，以对话的方式建立各种软件系统。

实时/嵌入式系统和管理信息系统部门的开发人员都非常欢迎这种变化，他们迫不及待地希望得到一种新的软件技术，以解决已经持续了二十多年未获解决的软件生产率问题。这种新技术就是计算机辅助软件工程技术。

计算机辅助软件工程技术可以简单地定义为软件开发的自动化，通常简称为 CASE 技术。这是一种先进的软件技术，它比以往的任一种软件技术都更加令人感到振奋和充满希望。它正在使软件开发人员逐渐忘掉了第三代、第四代甚至第五代的软件技术，因为 CASE 已经代替了第四代语言，正在成为一种最有力的软件开发技术。

## 1.3 软 件 自 动 化

CASE 对软件的生存期概念进行了新的探讨，这种探讨是建立在自动化基础上的。CASE 的实质是为软件开发人员提供一组优化集成的且能大量节省人力的软件开发工具，其目的是实现软件生存期各个环节的自动化并使之成为一个整体(见图 1.2)。

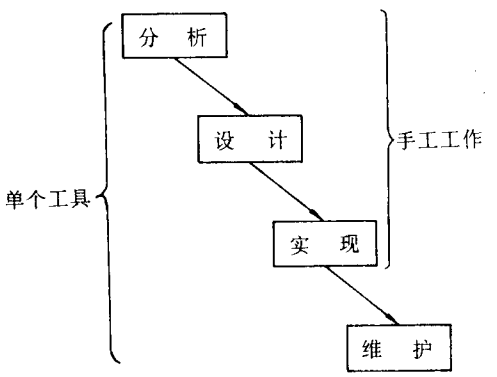


图 1.2 传统的软件开发方法：手工方法和单个工具结合

传统的软件技术有两种类型：工具与方法。软件工具包括各种第三代、第四代以及最近才出现的第五代工具，它们中的大多数是独立的、依赖于计算机的，而且主要集中在软件生存期的实现阶段。

软件方法包括手工的软件开发方法，如结构化分析、结构化设计和结构化程序设计。这些方法限定了软件开发的逐步规格化过程。

CASE 技术是软件工具和软件方法的结合。它不同于以前的软件技术，因为它强调了

解决整个软件开发过程的效率问题，而不仅仅是实现阶段。由于跨越了软件生存期的各个阶段，因此，CASE 也是一种最完满的软件技术。CASE 着眼于软件分析和设计以及程序实现和维护的自动化，从软件生存期的两端解决了软件生产率的问题。

由于手工的结构化方法实在太冗长乏味，而且需要花费很多人力，因此，在实际的开发过程中很少能够完全按照其要求去做，而 CASE 通过自动画出结构化图形、自动生成系统的文档，使手工的结构化方法得到实际的应用。

## 1.4 CASE 的作用

表 1.1 列出了 CASE 所能发挥的作用。该表说明 CASE 并不是一种全新的技术，而是建立在许多实际中已被证明是行之有效的技术和工具基础之上的。从这种意义上说，CASE 也可以被定义为“用一种新的手段对结构化概念和设计方法重新进行组装”，这一新手段就是自动化。

**表 1.1 CASE 的作用**

- 使结构化技术成为可行。
- 实施软件/信息工程技术。
- 通过自动检查提高软件的质量。
- 使原型的建立成为可行。
- 简化程序的维护工作。
- 加快软件的开发过程。
- 使开发人员的精力集中于开创性工作。
- 鼓励进化式和递增式的软件开发。
- 使软件部件可重复使用。

## 1.5 CASE 工 具

CASE 技术的核心是软件工具发展的必然结果。许多这类工具都是在微机上开发的，使用了强有力的图形功能以增强用户的接口，并将其加入到工作环境中去，使之能方便地被调用或互相调用。CASE 工具主要是为了提高专业软件人员的开发效率。有了工作站、局域网和基于 PC 机的软件工具，软件开发人员就可以在能快速响应的专用开发环境中工作，这一点在过去是不可想象的。

CASE 工具主要供专业的软件开发人员、而不是终端用户使用，因此，不能把 CASE 技术看成仅仅是对第四代软件工具的增强。另外，CASE 工具不同于第四代软件工具还体现在其它的两个方面：首先，许多 CASE 工具是立足于工作站的，而大部分第四代软件工具是基于主干机的；其次，CASE 是一种通用的软件技术，适用于各类软件系统的开发（如大型软件系统和小型软件系统，商业应用软件和科学计算软件，联机软件和实时应用软件等等），第四代软件工具主要用于开发小型或中型的商业应用软件。

总之，CASE 工具不同于以往的软件工具，主要体现在以下的几个方面：

1. 支持专用的个人计算环境
2. 使用图形功能对软件系统进行说明并建立文档
3. 将软件生存期各阶段的工作连接在一起
4. 收集和连接软件系统中从最初的需求到软件维护各个环节的所有信息
5. 用人工智能技术实现软件开发和维护工作的自动化

## 1.6 CASE 工具实例

CASE 工具能做的事情比第三代和第四代软件工具要多得多。它们实现了许多软件开发和维护工作的自动化，其中包括计划管理，如图 1.3 所示。CASE 工具的例子包括：

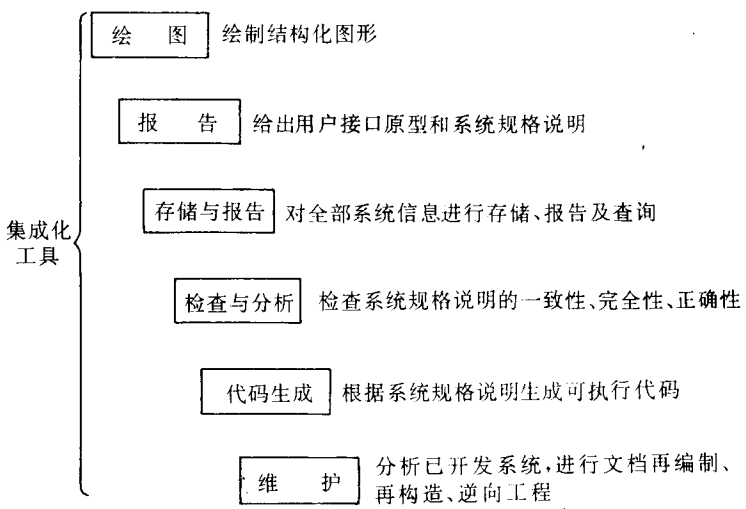


图 1.3　服务于软件开发与维护的各类 CASE 工具

- 画图工具：画出结构化的示图并生成用图形表示的系统规格说明。
- 报告生成工具：建立系统的规格说明和原型。
- 数据词典、数据库管理系统和报告生成工具：存储、报告和查询技术信息、项目管理系统信息。
- 规格说明检查工具：自动检测系统规格说明的完整性、语法正确性和一致性。
- 代码生成工具：根据图形化的系统规格说明，自动生成可执行的代码。
- 文档资料生成工具：产生结构化方法所需的各种技术文档和用户系统文档。

虽然工具是 CASE 的重要组成部分，但 CASE 技术并不仅仅是指软件工具，而是对整个软件环境的重新定义。

CASE 改变了软件开发环境的一个主要原因是因为 CASE 工具一般都在工作站的环境下运行，使软件开发成为一个高度交互的过程，因此，在系统需求定义这样的前期工作中就可以开始进行查错。

## 1.7 CASE 的发展过程

从表 1.2 可看到，CASE 技术的历史始于 80 年代初期，最初推出的是计算机辅助建立文档和画图工具，它们代表了最早的基于 PC 机的软件开发工具以及对软件分析和设计自动化的首次尝试所做的某些工作。这是一些简单的独立画图工具，被用来建立结构化图形，如数据流图、程序结构图、实体关系图。其目的是为了自动产生各类结构化设计方法所要求的结构化文档。不同的 CASE 工具支持不同的设计方法，如 Yourdon 结构化分析和设计方法，Jackson 结构化分析方法和 Martin 的信息工程方法。

**表 1.2 CASE 技术的演进**

| | |
|---|---|
| 八十年代初 | 计算机辅助编制文档。<br>计算机辅助画图。<br>采用分析工具和设计工具。 |
| 八十年代中 | 自动设计分析与自动设计检查。<br>采用自动化系统信息库 |
| 八十年代末 | 根据设计规格说明自动生成代码。<br>将设计自动化与程序自动化联结。 |
| 九十年代初 | 人工智能方法逐步应用于开发。<br>用户接口能适应人的习惯。<br>将可复用技术作为一种开发方法。 |

图 1.4 是一个手工画的结构图，描述了温度监控程序的设计，它使用的是 Yourdon 给出的一套符号。图 1.5 也是一个使用 Yourdon 符号的结构图，但它却是由 CASE 工具自动产生的。图 1.6 是由另一个 CASE 工具使用 Gane & Sarson 符号画出来的数据流图。图 1.7 是用 Martin 的信息工程符号自动产生的实体关系图。

早期的 CASE 工具把自动建立文档作为提高软件开发效率的关键。到了 80 年代中期，CASE 工具又增加了两个重要的功能：

1. 自动检查结构化图形。
2. 将结构化图形存入称为词典、中心库(repository)或百科全书的自动化设计信息

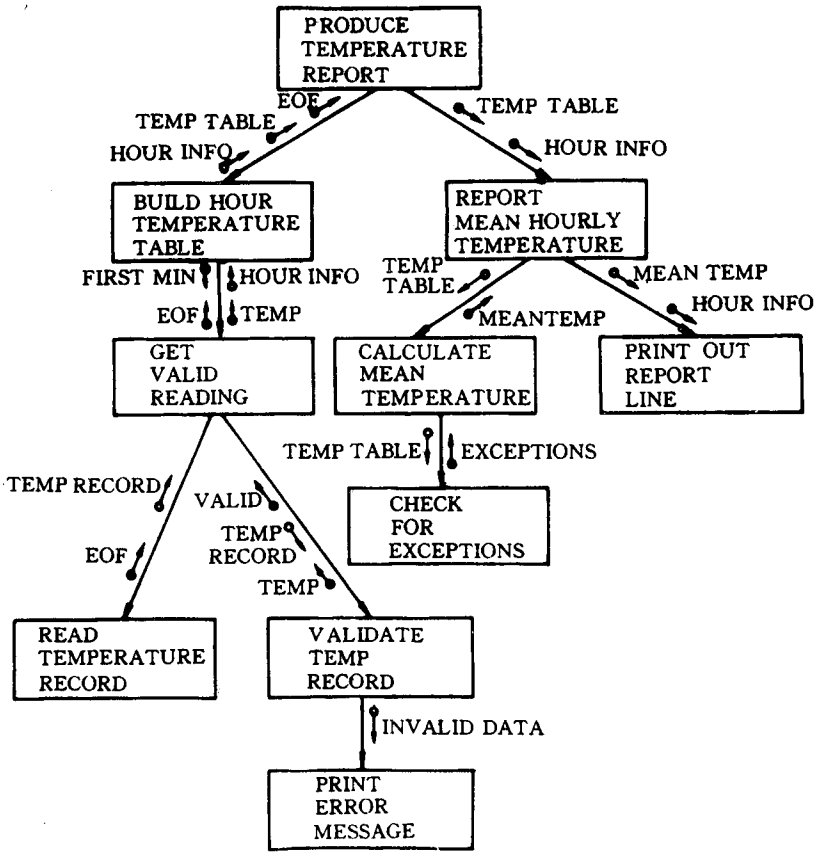


图 1.4　人工绘制的程序结构图

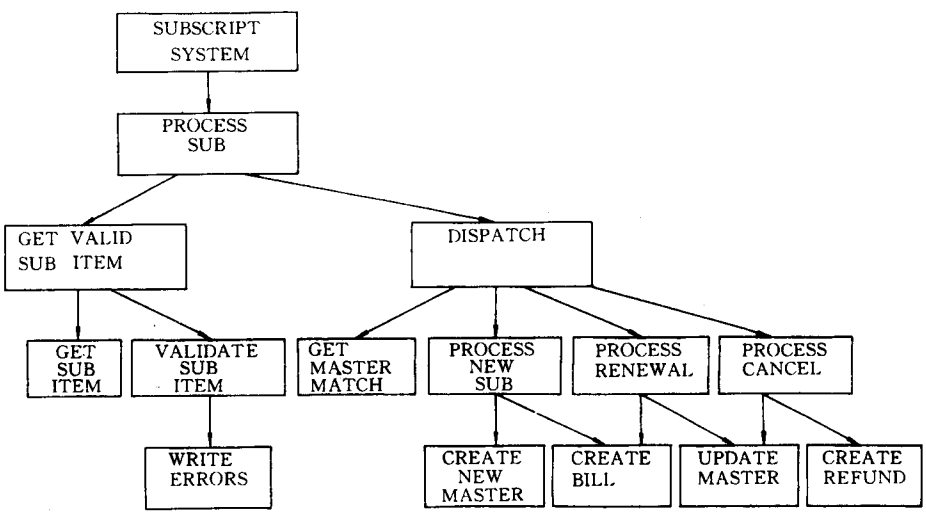


图 1.5　由 INFORMATION ENGINEERING WORKBENCH 生成的结构图

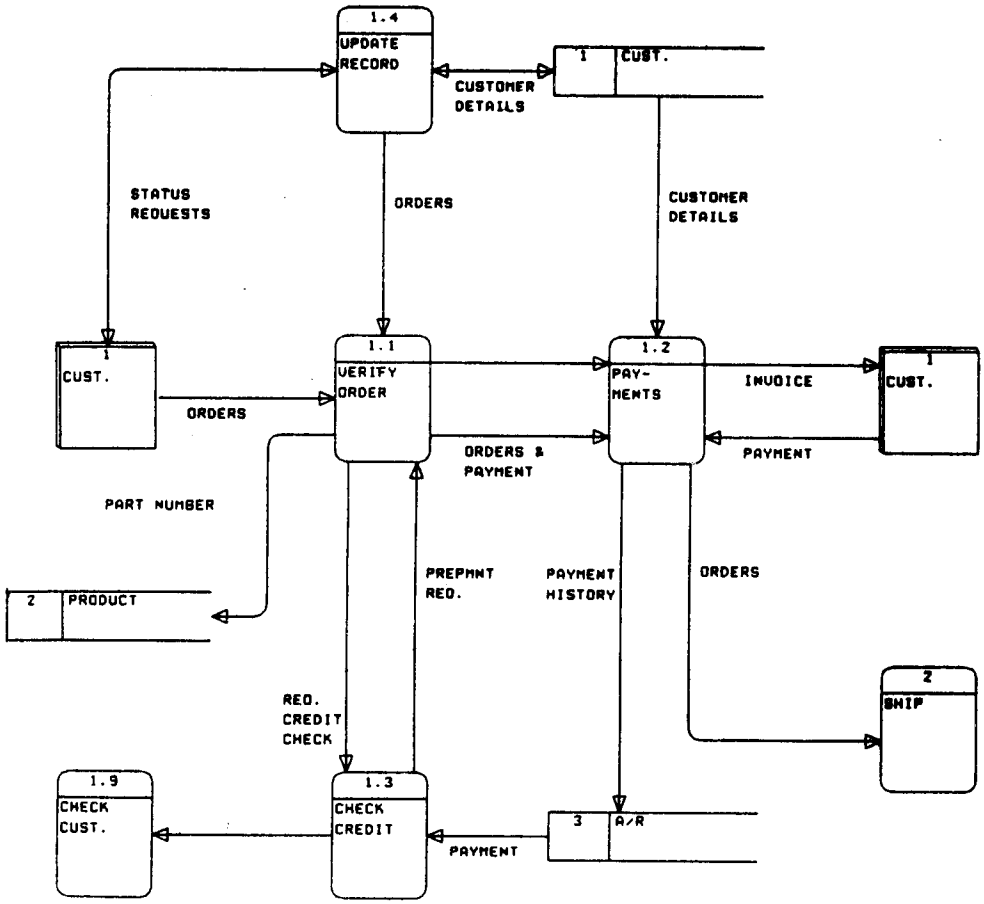


图 1.6　由 EXCELERATOR 绘制的数据流图

库中。

用结构化图形表示的设计规格说明将作为程序设计的依据，但在此之前，为了保证正确性和完整性，必须对它们进行检查。检查是根据它所支持的结构化设计方法的规则进行的。此外，还要把交叉引用信息记录下来，并且消除冗余的信息。

检查以后，这些图被存入称为 CASE 中心库的自动化词典或称为 CASE 库的数据库中。存入以后，就可以很方便地进行更新并实现同一项目内软件人员间的信息共享，以后还可以再用于其它项目的开发。

在这个阶段中，CASE 着重于设计自动化，因为分析和设计被认为是软件生存期中最关键的两个阶段。在生存期的这两个前期阶段把工作做好了，将对软件系统的质量和成本产生很大的影响。使用了 CASE 工具，就能大大地缩短设计的时间并产生高质量的、没有错误的设计规格说明。

改进 CASE 技术的下一步工作是将设计自动化与程序自动化结合起来。如果设计自动化是指用计算机辅助实现软件的分析与设计，那么程序自动化就意味着代码的自动生

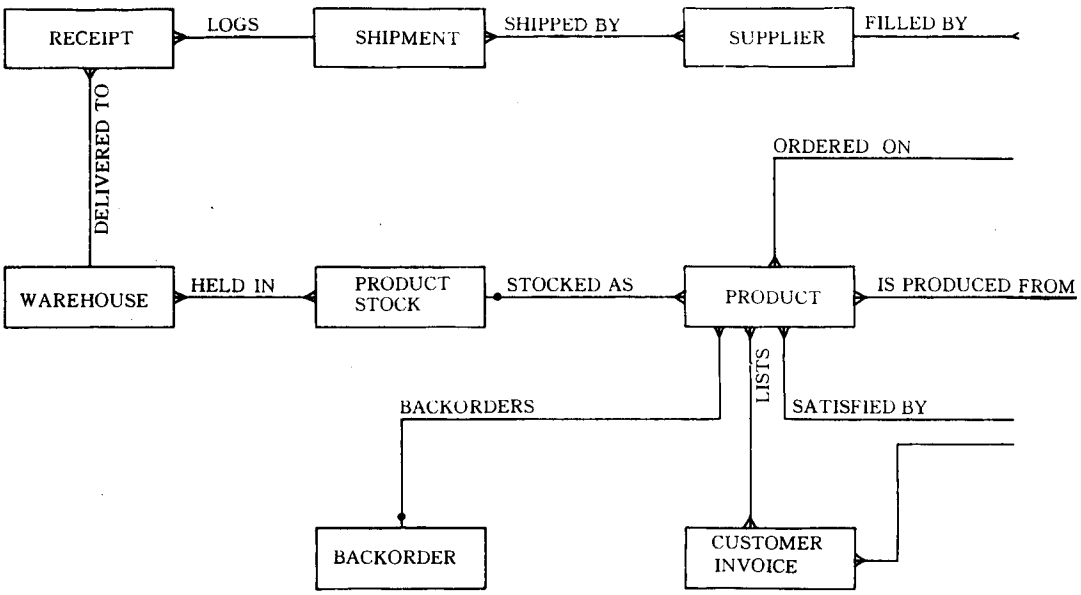


图 1.7　由 INFORMATION ENGINEERING FACILITY 生成的实体关系图

成。将这两者联系在一起则意味着一个完整的软件的百分之八十到百分之九十可以由结构化设计图实现(见图 1.8)。

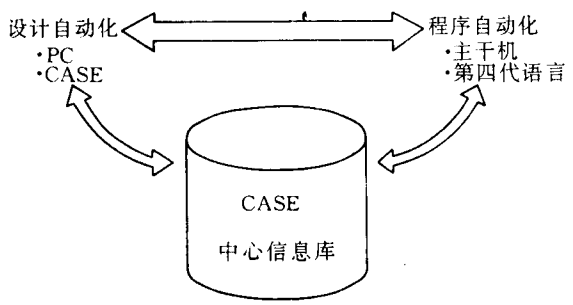


图 1.8　设计自动化与程序自动化结构

设计自动化是由 80 年代初期和中期开发的基于 PC 机的 CASE 工具支持的，程序自动化则由第四代应用软件生成器和代码生成器来实现，大多数的代码生成器都是基于主干机的。因此为了把设计自动化和程序自动化结构合在一起就需要在这两者之间架起一座桥梁，把两种硬件环境——PC 机和大型主干机，以及两种软件技术—CASE 和第四代软件工具联系起来。这座桥梁的一个重要组成部分就是在 CASE 环境中增加一个中心库。CASE 中心库是存储并组织与某个软件有关的所有信息的机构。在这些信息中，有的用于项目管理；有的则用于代码的自动生成。信息的内容涉及到所要解决的问题、该问题

的域、软件的开发过程、数据模型、过程模型、原型、项目资源和项目历史以及开发单位的有关信息等。

除了将设计自动化和程序自动化结合起来，CASE 中心库还第一次使软件的可复用性成为可行的概念。软件部件的可复用性是提高软件开发效率的关键 。在以后的软件开发过程中，开发人员不需要建立每一个新的软件或对原有的软件进行临时的修改，而是可以重复使用已经存入 CASE 中心库中的各种软件模块，不但能重复调用源程序模块，还能重复使用项目计划、原型模型、数据模型和设计规格说明等模块。

## 1.8 CASE 的目标

归纳起来，CASE 有三大作用，这三大作用从根本上改变了软件系统的开发方式：

1. 一个具有快速响应、专用资源和早期查错功能的交互式开发环境。
2. 对软件的开发和维护过程中的许多环节实现了自动化。
3. 通过 一个强有力的图形接口，实现了直观的程序设计。

CASE 技术的最终目标是通过一组集成的软件工具，实现整个软件生存期的自动化。但是，目前还没有达到这一目标，我们必须努力使之尽快实现。

## 1.9 人们认识的变化

也许目前描述 CASE 的最好方式是把它看成为一种认识上变化。CASE 代表了软件开发技术发展过程中在认识上的一种根本变化，这种认识上的变化为实现软件技术根本性的进步铺平了道路。CASE 并不是软件开发技术发展过程中在认识上的第一次重要的变化，60 年代后期，结构化技术的引入就代表了这种认识上的变化。由于出现了软件危机，当时的观点是：软件开发是一个复杂的、冗长乏味和容易出错的过程。不应该把软件开发看成是每个程序员个人的技巧，而应该把它看成类似工程的、规格化的过程。结构化技术强调了程序设计标准步骤和管理的控制 ，其观点是：软件的质量和开发效率最好是通过规格化、形式化和标准化来控制。

70 年代后期，出现了另一次有关软件开发过程认识的变更，新的观点认为：可以通过使用某些软件工具来简化软件的开发过程，从而提高软件的生产率。他们认为如果能有更多的人(包括那些具有较少专业技术的人)开发出一些软件应用系统，软件危机就可能得到缓和。第四代软件工具和最终用户工作环境的引入说明了这种认识的变化。

到了 80 年代后期，我们又看到了对软件开发过程的另一次认识上的变化。从某种意义上说，这种变化代表了早期较为保守的软件开发观点。其看法是：软件开发是一个很困难的任务，需要使用各种类似工程技术的规格化方法，而且开发复杂的软件系统，应该由那些经过高级训练的专业软件开发人员来完成。虽然最终用户工作环境和第四代软件工具已经做出了很有价值的贡献，但经验表明，它们在软件开发中的作用是有局限性的。它们主要用于建立中小规模的商业应用软件，而不是作为通用的软件开发工具。

最近的这次认识上的变化是对二十多年来已经被定义的软件工程基本原理的重新发

现。结构化技术是一个重新提出来的主题，但这一次强调的是软件开发过程的自动化。在这个阶段中，CASE 是结构化技术的复兴。

CASE 的观点是：软件开发和维护应该被看成是一个形式化和规格化的过程，应该实现更为全面的正确性检查和处理过程的自动化。显著提高软件的质量和开发效率的唯一可行的方法是通过自动化来实现。

## 1.10 自动化的方向

本书的主题是软件的自动化，讨论了作为实现软件开发自动化工具的 CASE 技术。我们将说明什么是 CASE，什么不是 CASE 以及 CASE 的发展前景。表 1.3 给出了 CASE 技术的各种基本定义。

**表 1.3 CASE 的有关定义**

| |
|---|
| CASE：计算机辅助软件工程。 |
| CASE 技术：一种软件技术。为软件的开发、维护和项目管理提供一种自动化工程原理，包括自动化结构化方法和自动化工具。 |
| CASE 工具：一种软件工具。对某个具体的软件生存期任务实现自动化（至少是某一部分的自动化）。 |
| CASE 系统：一种集成的 CASE 工具。使用一个公共的用户接口，并在一个公共的计算机环境下运行。 |
| CASE 工具箱：一组集成的 CASE 工具。被设计用来协同工作以实现某个软件生存期阶段或某类具体的软件作业的自动化（或部分地实现自动化）。 |
| CASE 工作台：一组集成的 CASE 工具，被设计用来协同工作以实现整个软件生存期的自动化（或提供自动化的辅助手段），包括分析、设计、编码和测试。 |
| CASE 方法：一种“可自动化”的结构化方法。为软件的开发和维护的整个过程或某个方面定义了一个规程化、类似工程的方法。 |
| CASE 方法的同伴：一组 CASE 工具。实现某种具体的 CASE 设计方法自动化的任务、并（或）自动产生由该设计方法所要求的文档和其它资料。 |
| CASE 工作站：一个装有各种 CASE 工具的 32 位的工作站或个人计算机。实现各种软件生存期功能的自动化。 |
| CASE 硬件平台：一级、二级或三级的硬件系统体系结构。为 CASE 工具提供一个操作平台。 |

本章已经介绍了软件自动化和计算机辅助软件工程技术的主题。随后将在第二章和第三章中定义 CASE 结构。第五章对 CASE 工具进行了分类。第四章讨论了 CASE 系统的硬件平台。第六至第九章将讨论 CASE 在软件开发过程中的作用，第六章介绍 CASE 工具的使用情况。第七章是引进 CASE 工具的指南。第八章讨论了 CASE 技术如何改变传统的软件生存期。第九章讨论了 CASE 技术同第三代、第四代和第五代软件技术的

关系。

在本书第十至第十三章中，讨论了到本世纪末软件开发的前景。90年代的软件开发环境将在第十章中介绍。第十一章描述了“能适应用户习惯的环境”，这是90年代软件开发环境的一个智能用户接口。方法论驱动程序——为90年代软件生存期提供方法论指南的专家系统将在第十二章中讨论。第十三章说明软件的可复用性如何和为什么将成为90年代最主要的软件开发策略。最后一章给出一个简短的结束语以结束本书。

# 第二章　CASE 软件开发环境

## 2.1　新的软件开发环境

随着硬件和软件技术的进步，软件开发环境从 50 年代和 60 年代初期的独立方式变成 60 年代后期和 70 年代初期的批处理方式，然后又是 70 年代后期到 80 年代初期的分时处理方式。80 年代后期，一种基于个人工作站的新方式正在成为更加理想的软件开发环境。工作站对于改进软件开发环境具有深远的意义，强有力的工作站工具和随之而来的类似工程技术的结构化方法论的引入，以自动化的方式改变了软件的开发过程。

CASE 工作站是一个完整的环境，包括硬件和软件两部分，其目的是为软件系统的开发、维护和项目管理提供计算机化的辅助手段。

此外，工作站还提供了字处理、信息存储与检索、电子邮件和日历等功能，它是一种专门为每位软件开发人员提供最大限度支持的个人计算机。

使用 CASE 工作站的主要目的是提高软件的开发效率。提高软件开发质量自然也应是其追求的目标之一，表 2.1 列出了其它一些经常为人们所提到的观点。

**表 2.1　软件工作站的目标**

- 最大限度地提高软件开发人员的生产率。
- 提高软件质量并减少错误。
- 简化软件的开发过程。
- 加速软件开发过程。
- 更有效地利用计算机资源。
- 降低软件的成本。
- 提高软件开发环境的可靠性。
- 提供具有更好的人机界面的开发环境。
- 从根本上改进软件的开发过程。
- 自动生成软件的文档。
- 自动生成可执行代码。
- 支持快速建模。
- 在各个开发过程中实现查错自动化。
- 使项目管理自动化。
- 实现软件开发过程的形式化和标准化。
- 实现软件开发步骤和工具的集成化。
- 实现软件文档编制的标准化。
- 实现软件的可复用性。
- 对软件开发和维护实现更为全面的控制。
- 改进软件在不同环境之间的可移植性。

## 2.2 CASE 工作台

CASE 工作台是一组范围广泛的智能集成化软件工具，构成了工作站的“软环境”。软件开发人员可根据自己的需要对 CASE 工作台进行剪裁，以便于执行项目管理、设计和维护这样一些专门的任务，代替 60 年代和 70 年代那些支持批处理工作方式和第三代语言这样一类传统工具。

CASE 工作台在覆盖软件生存期的宽度上不同于 UNIX 和 Interlisp 这样早期的程序设计环境。UNIX 是一个分时操作系统，统一的文件格式是集成一组程序设计工具的主要手段，作为一个适用的程序设计环境，它不支持具体的程序设计语言或软件开发设计方法。而 Interlisp 则是一个只能支持 Lisp 语言的程序设计环境，这就使得 Interlisp 语言具有剪裁和优化其工具的优点。所有的 Interlisp 工具都是用 Lisp 语言写的，并提供了高度集成化的操作系统实用函数、编辑程序和查错程序。

70 年代和 80 年代的程序设计环境主要集中在编码和实现阶段，而 CASE 工作台则为软件系统制定规格说明 、设计、实现、测试和文档生成提供了各种强有力的工具。CASE 是一个通用的软件支持环境，它能支持所有的软件开发过程的全部技术工作及其管理工作。

为了达到提高软件开发效率和简化软件开发过程的目标，CASE 工作台不能只是由一组工具组成，即使 70 年代和 80 年代最好的工具也是如此。因为这些工具并不是用在专用的个人计算机环境中，允许不同的工作站上的软件开发人员对其进行调整；它不能使用强有力的图形功能来增强人机接口；不能同其他的工具互相配合、将软件开发过程的各个方面联系在一起；它也不能收集正在进行的某个开发过程和软件产品进展有关的信息；最后，它们并不是智能工具，不能自动执行开发任务。

反之，CASE 工作台的集成软件工具能够为软件开发过程提供全面的支持：

- 生成用图形表示的系统需求和设计规格说明
- 检查、分析和交叉引用系统信息
- 存储、管理并报告系统信息和项目管理信息
- 建立系统的原型并模拟系统的工作原理
- 生成系统的代码及有关的文档
- 实施标准化和规格化
- 对程序进行测试、验证和分析
- 连接外部词典和数据库

为了提供全面的软件开发支持，一个完整的 CASE 工作台必须具有以下的功能：

- 图形功能
- 查错功能
- 中心信息库
- 高度集成化的工具包
- 对软件生存期的全面覆盖

- 支持建立系统的原型
- 代码的自动生成
- 支持结构化的方法论

在本章中，我们将讨论前四个功能，后面的四个功能在下一章接着讨论。

## 2.3 图形功能

使用CASE工作台时，给人们印象最深的是图形编辑程序。用户可以很方便地生成或消除各种目标，也可以用鼠标器在屏幕上任意地移动目标，其操作的方便程度实在令人感到惊讶。图形是一个非常重要的功能，图形接口的功能越强，用户的软件开发效率就越高。

表2.2是一个订货系统的规格说明，图2.1则是系统规格说明的图形表示(即使用了数据流图)。一般人都喜欢图形方式，因为人脑适于接收形象信息。用文字记叙的文本是一维的，而图形则是多维的，因为它借助了大小、形状、颜色这些实实在在的特性。由于图象语言比文本语言更为丰富，因此，图形也比文本表示更多的信息，而且读者还能更快地理解其中的意义。

**表2.2 订货系统规格说明的文字描述**

**订货系统程序规格说明**

该系统是根据订货文件来处理各种订货事务的。有三种类型的事务：新的订货，重新确认订货和取消订货。每种事务首先必须被确认，然后再根据主文件进行处理。对于新的订货，系统将建立起该顾客的记录，并根据结算的结果产生帐单；对于重新确认的订货，系统将修改截止日期并产生结算帐单；对于取消订货，相应的记录被标识为删除并退还款项。

## 2.4 为什么要使用图形

图形表示法在软件开发中起着重要的作用。图形可用来定义程序的规格说明和表示程序的设计，图形还为把设计变换成程序代码提供了实现的蓝图。图形也是软件文档的一种重要形式。

清晰的图形在复杂系统的设计和程序开发的过程中发挥了关键性的作用。即使是在开发人员一个人承担的系统设计和程序开发过程中，图形也能为开发人员提供清晰的思路，如果图形选择不好，就会妨碍思路，但如果选好了，就能加快工作速度并提高产品的质量。

在多人协同工作的情况下，图形是一种重要的沟通工具。在开发过程中，需要一种规范化的图形技术，使开发人员能够更好地交换思想，才能把系统的各个组成部分精确地集成起来。当查错时，图形也是一个很有价值的工具，它能帮助你更好地理解别的程序员的思路，以便更好地对程序进行跟踪，查出错误的根源或了解经过某些修改后对程序所产生

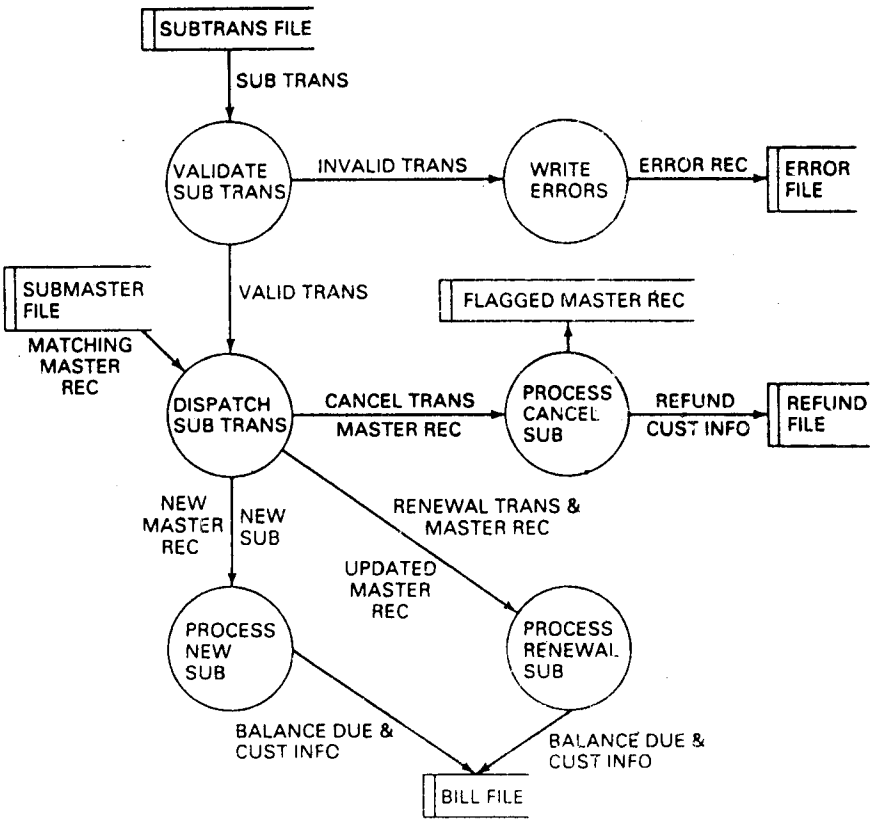


图 2.1　订货系统的数据流图

的影响。

在一起工作的人越多，对图形的准确性要求也就越高。对于较大项目开发组中的成员来说，要详细地了解其他人的工作是很困难的，甚至是不可能的。为了解决这一问题，每个成员对系统都应该有大概的了解，并知道自己所做的工作属于系统的哪个部分；每个人都必须有清楚的、准确定义的接口，并能在各种不同的程序上对系统或程序进行观察。最重要的是，对系统每个部分的描述方法都必须是一致的、标准化的，只有这样，系统各部分之

**表 2.3　对文档的要求**

好的系统和程序文档是结构化原理的基本组成部分，因此，好的文档必须能满足下列的要求：

- 有助于提高软件系统的可理解性。
- 使软件系统的生成和更新变得更加容易、成本更低、并能自动实现。
- 能给出系统的高层视图，解释各类数据和过程模块的作用及其相互之间的关系。
- 能详细地描述每个数据结构和过程模块。
- 能为系统设计中的问题需求提供蓝图，并将该设计蓝图转换为程序代码。

间的通讯才容易进行，并避免由于误解而导致程序上的错误。

图形实际上是软件模型化的语言，因为它为软件的描述提供了一种简明的、没有歧义性的方法，是产生好的系统和程序文档的基础(见表 2.3)。图形也是软件分析和设计的基础，因为在软件的分析和设计过程中，用图形技术为软件系统建立模型，就能体现出不同的结构化方法论的特性。

## 2.5 结构化图形方法的优越性

数据流图和实体关系图这一类的结构化图形技术体现了程序文档的基本结构原理。结构化图形技术有许多优点。首先，图形和文字表示法的结合增加了软件系统的可理解性。图形在这方面是特别有用的，因为它比文字描述更加明确。另外，由于图形更加简单明了，对于含有相当信息量的问题，画出图形比写出文字的描述所花的时间要少得多。

其次，结构化图形技术是自顶向下分层的，这意味着它支持自顶向下的结构化开发方法，在分解的过程中的每一步上，能够在不同程度上描述系统、程序或数据结构。这是一种描述程序逻辑和数据结构的标准方法，用以说明分解过程的步骤及其相应的结果。

最后一点，结构化图形技术能帮助开发人员处理程序开发过程中所产生的大量细节，因为它是一种逻辑表示法，而不是面向程序设计的表示法。逻辑表示法对于用户和非专业的管理人员来说更有意义。

## 2.6 结构化图形的使用

图形技术是同程序设计方法论一道发展起来的。50 年代和 60 年代，用程序流程图来描述详细和复杂的程序逻辑，但由于它未能给出程序的高层视图或结构化的视图，因此不再受到人们的欢迎。70 年代，结构化技术开始得到了广泛的应用，结构化图形技术，如数据流图、程序结构图也随之出现，它们中的一些被用作分析工具，另一些被用作程序设计工具，还有其他的一些被用作数据库和(或)文件的设计工具。表 2.4 根据这种分类法，列出了结构化图形技术的一些例子。

数据流图、分解图和控制流图是分析工具的例子(见图 2.2 至图 2.4)。例如，数据流图用于描述数据在通过一个系统时所发生的变换；控制流图增加了描述实时系统时所需的控制信息；分解图用于表达组织机构及其基本职能。

程序结构图(图 2.5)用作程序设计工具，它定义了程序的层次组织，包括各程序模块以及模块的数据接口。动作顺序图和 Warnier-Orr 图(见图 2.6 和图 2.7)用来表示程序的层次结构以及详细的程序逻辑。

实体关系图和数据模型图用来描述数据结构(图 2.8 和图 2.9)。每个数据实体及其属性，还有它同其他数据实体的关系都可以在这些图中加以定义。此外，获取事务规则时，可以用基数来区分可选规则与必选规则之间的关系。

表 2.4 结构化图形方法

| 结构化图形 | 过程型 | 数据型 |
|---|---|---|
| 分析工具 | | |
| 数据流图(data flow diagram) | × | |
| 控制流图(control flow diagram) | × | |
| 判定表及判定树(decision table, tree) | × | |
| 矩阵(matrix) | × | × |
| 分解图(decomposition diagram) | × | |
| 依赖关系图(Dependency diagram) | × | |
| 设计工具 | | |
| 结构图(structure chart) | × | |
| 活动图(action diagram) | × | |
| Warnier-Orr 图 | × | × |
| 状态转换图( state transition diagram) | × | |
| 判定表及判定树 | × | |
| 伪码(pseudocode) | × | |
| 流程图 (flowcharts) | × | |
| 屏幕布局(screen layout) | | × |
| 对话流(dialoge flow) | | × |
| 数据库及文件设计 | | |
| 数据模型(data model) | | × |
| 实体关系图(entity relationship diagram) | | × |
| 数据结构(data structure) | | × |
| 逻辑记录(logical records) | | × |
| 物理数据库及文件(physical database and file) | | × |