软件开发 软件工程 CASE 微机 (16)

计算机科学1994Vol.21№.6

74-77

PC/CASE：支持软件开发过程的CASE工具

罗铁根 齐治昌

(国防科技大学计算机系 长沙410073)

TP311.52

摘 要 This paper introduces a CASE tool named PC/CASE that is developed in the enviroment of MS-WINDOWS 3.1 on PC 386/486. The PC/CASE is developed for teaching, and will help users to understand CASE concepts and methodologies. The PC/CASE supports structured programming methodology and the whole procedure of developing software. Therefore, it has given user a friendly computer aided developing.

关键词 Software Engineering, CASE, Structured Programming methodology, Software reuse, DFD.

一、引言

随着计算机产业的迅速发展，诞生了一门新的方法学——"计算机辅助软件工程"(CASE)。CASE主要解决以下问题：

·降低软件开发复杂度：工作台面向图形，形象直观；各种软件资源易被重用；简化软件的管理。

·文档(Documentation)管理、维护方便：文档缺乏和文档描述粗糙是软件维护的主要障碍；CASE工具能自动编写，保留各种文档，而且在很多环节上可以提供自动维护功能，生成的文档齐全，并且格式规范，大大减少了维护时间和开销。

·降低早期故障出现的可能性：早期故障导致生产力下降，软件开销急剧上升；一致性检查、机器自动生成等机制使软件开发早期错误大大减少。

·降低删除故障开销：程序的测试和改错在软件设计工作中往往占到了一个出人意料的百分比，出错开销已成为程序设计总成本的大头；CASE工具能帮助用户找错，并能自动生成新的文档，使用户尽快完成改错。

·降低软件改进开销：用户新的需求和产品功能扩展对维护工作提出了巨大挑战；CASE工具能帮助用户分析软件，自动完成后期软件改进。

·简化维护：维护已成为软件商业中的主要开销因素；CASE工具提供的描述具有通用性，可为不同用户所理解；并且提供软件分析、设计的跟踪能力。

因此，使用CASE工具开发软件将缩短开发时间、降低成本、提高质量且软件产品可靠性好、易于维护。

二、PC/CASE支持的软件开发过程

在CASE系统中，人员、方法、工具是三个最重要的因素，PC/CASE按图1所示支持结构化软件开发的全过程。软件开发人员使用DFD(数据流图)编辑器和DFD一致性检查工具进行结构化分析，生成结构化规范；结构化规范通过结构图生成器自动转

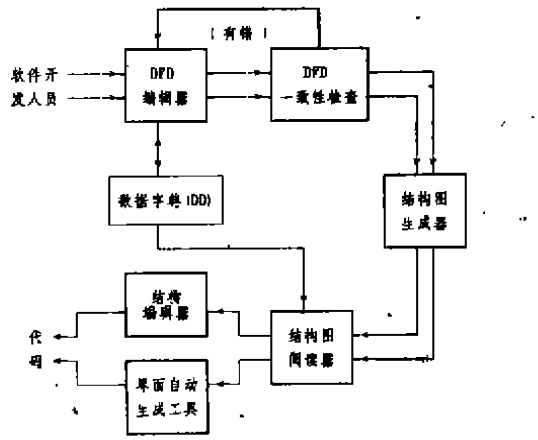

图1 PC/CASE支持的软件开发过程

换为结构图；对结构图的每一个模块结点，使用结构编辑器能编辑结构软件，结构软件即可直接映射成代码；对于界面模块，一种更简便的方法是使用界面自动生成工具，根据用户编辑的界面自动生成代码。

三、DFD设计

DFD工具提供对DFD进行编辑、修改、替换、一致性检查、文件存取等功能，为用户提供一个方便

罗铁根 博士生，研究兴趣：CASE、软件重用、并行编译。 齐治昌 教授，副院长，研究兴趣：数值分析、软件工程、软件开发环境。

的 DFD 设计环境。

DFD 是一个多层结构，每层 DFD 都包括：处理、存储、外部实体、数据流。系统将这四种元素分别用双向链表表示，这样，每层 DFD 都有四个双向链表，必须有一个保留层描述信息的结构，便于 DFD 的层次间切换，每当进入某一层进行编辑时，系统通过层描述结构可获取当前层的信息，从而根据用户操作对四个双向链表进行修改，无需涉及其它层的信息。DFD 的层次结构描述示意图如图2所示。

·层信息结点　信息包括：父处理结点、父层信息描述结构、父处理结点在整个 DFD 中的编号、四个双向链表的头指针。层描述结构不但可以方便地访问父层和四个双向链表，而且各元素中只需保存当前的编号，通过层信息中的编号加上元素在当前层的编号可以获取元素在整个 DFD 中的编号。

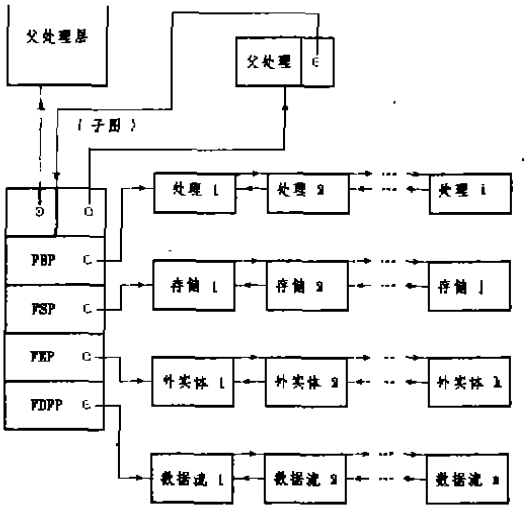

图 2　DFD 的层次结构描述示意图

·处理结点　信息包括：名字、在本层的编号、在窗口中显示的位置、输入数据流、输出数据流、处理规格说明、子图层信息描述结构。通过处理结点描述结构系统可以方便地了解这些信息，并且可以通过数据流指针访问输入、输出数据流结构，通过子图层指针进入子图编辑。

·存储结点　信息包括：名字、在本层的编号、在窗口中显示的位置、输入数据流、输出数据流、存储定义。系统采用存储结点描述结构保留这些信息，并且可以通过存储定义指针访问 DD，获取存储内容和格式的定义。

·外部实体结点　信息包括：名字、在本层的编号、在窗口中显示的位置、输入数据流、输出数据流。外部实体结点描述结构可保存这些信息，供系统查询。

·数据流结点　信息包括：名字、在本层的编号、在窗口中显示的位置（起点和终点座标）、是否为双向数据流、起点和终点结构指针、条目定义。通过数据流结点描述结构系统可以很快了解到这些信息，并且可以通过条目定义指针访问 DD 中的数据条目定义，通过端点指针访问端点信息。

四、数据字典（DD）设计

DD 内容分为两类：数据流和存储。对于数据流，DD 存储其定义，主要用于描述数据流组成成员；对于存储数据项，DD 描述存储内容和格式，DD 在内存中以双向链表保存，先后关系按名字的字典顺序排列，便于查阅、列表。

我们采用一种类似于 BNF（巴科斯-诺尔范式）产生式规则的形式化语言描述数据流、数据存储。该语言使用的基本符号及语义为：

＝：包括；

＋：…和…；

{…}：循环一次或多次；

[…]：选择表中之一元素；

(…)：可选项，即允许零次或一次出现；

'…'：数字，如：'1.5'；

"…"：说明，如："This is a comment"。

五、结构图设计

结构图描述模块之间的调用关系，是一个多叉树结构，其中一个矩形框表示一个模块，矩形框内的字符为模块名字，连线表示模块之间的调用关系。

1　模块规格说明（Mini_Specs）　在 PC/CASE 中，*Mini_Specs* 是用一种伪码语言描述的，其中关键字符含义为：

%：伪码关键字标记；

.：分隔符；

%Subroutine：过程、子程序；

%Inputs：输入表；

%Outputs：输出表；

%function：函数功能描述头；

%External：外部过程；

BEGIN：语句段开始；

END：语句段结束；

LOOP：循环语句段开始；

ENDLOOP：循环语句段结束。

2　由 DFD 直接生成结构图算法　在介绍由 DFD 直接生成结构图算法之前，先定义 DFD 结点

之间的前继关系 BEFORE(x,y)：

1)对任意单向数据流，N_1为起始结点，N_2为终结点，则 BEFORE(N_1,N_2)；

2)对任意双向数据流，N_1为起始结点，N_2为终结点，则 BEFORE (N_1, N_2)，并且 BEFORE (N_2, N_1)；

3)传递性：如果 BEFORE(N_1, N_2)，BEFORE (N_2,N_3)，则 BEFORE(N_1,N_3)，如果 BEFORE(N_1, N_2)，则称 N_1为 N_2的前继，N_2为 N_1的后继。

算法

步骤1　置正向跟踪经过的泡点集 InTraceSet、反向跟踪经过的泡点集 OutTraceSet 为空；

步骤2　选择外部数据源 E、E'；

步骤3　在该层 DFD 中跟踪 E 的输出数据流，将所有流经的处理 B_i 加入 InTraceSet 中；

InTraceSet=InTraceSet+{B_i}；

步骤4　在该层 DFD 中反向跟踪外部数据源 E'的输入数据流，将 E'的所有前继处理 B_j 加入 OutTraceSet 中；OutTraceSet=OutTraceSet+{B_j}；

步骤5　建立一个"多箱"图，顶点为该层的父结点，儿子为交集：InTraceSet∩OutTraceSet；

步骤6　对于 B_i∈InTraceSet-OutTraceSet，寻找其父结点 F，F 定义为在该层 DFD 的 InTraceSet 中最后一个对 B_i 有输出数据流的处理结点。即不存在 F'满足：BEFORE(F,F')，且 F'对 B_i 有输出数据流。如：InTraceSet = {B_1, B_2, B_3}，OutTraceSet = {B_1, B_2, B_4}，并且 BEFORE(B_1, B_2)，BEFORE(B_1, B_3)，BEFORE (B_2, B_3)，则：B_3 = InTraceSet-OutTraceSet。对 B_3有输出数据流的处理结点集 Δ = {B_1,B_2}。在 Δ 中 B_1为 B_2的前继，而 B_2在 Δ 中没有后继，故 B_2为最后一个对 B_3有输出数据流的处理结点。因而，B_2为 B_3的父结点。

步骤7　对 B_i∈OutTraceSet-InTracsSet，寻找其父结点 F，F 定义为在该层 DFD 的 OutTraceSet 中最后一个接收从 B_i 来的输出数据流的处理结点。即不存在 F'满足：BEFORE(F,F')，且 F'接收从 B_i 来的输出数据流。如：InTraceSet={B_1,B_2,B_3}，OutTraceSet={B_1,B_2,B_4}，并且 BEFORE(B_1,B_2)，BEFORE (B_1, B_3)，BEFORE (B_1, B_2)，则：B_4 = OutTraceSet-InTraceSet。有从 B_3输出数据流的处理结点集 Δ={B_1,B_2}。在 Δ 中 B_1为 B_2的前继，而 B_2在 Δ 中没有后继，故 B_2为最后一个有从 B_3输出数据流的处理结点。因而，B_2为 B_3的父结点。

步骤8　对于有子图的处理结点 B_k，递归执行步骤1～7，生成结构图子图，其父结点即为 B_k。

3　结构图显示　结构图实际上是一个用二叉树结构描述的多叉树，因而在显示之前必须先计算各结点的显示位置，使得结构图不但能在用户区全部显示出来，而且各结点之间疏密程度适当，树看起来比较平衡。PC/CASE 采用了一个比较简单的算法解决这个问题：

步骤1　计算各层结点个数 Widths[i]；

步骤2　求出宽度最大值 MaxWidth，即 Widths[i]中的最大值；

步骤3　根据 MaxWidth 定义用户区域宽度 UserWidth；

步骤4　计算各层结点位置，对于第 i 层，第 NO 个结点：

x=(int)(UserWidth/Widths[i]) * NO；

y=i * LAYERHEIGHT；

其中，(x,y)为显示位置，LAYERHEIGHT 为系统定义的两层之间的距离。

六、软件一致性

在 PC/CASE 中将自动进行 DFD、DD、Mini_Specs 的一致性检查。检查内容包括：

1)DFD　·DFD 中没有命名或孤立的流、处理、存储或外部实体；·DFD 中无输入或输出的处理结点；·仅在上下文图中出现的外部实体；·上下文图中无存储；·DFD 中同名实体，含多个层次上出现的存储；·DFD 中上下层间 I/O 不平衡的结点。

2)DD　·DD 中没有定义的条目；·语法上不合法的定义，如：引入非法字符；·DD 中自定义的项；·DD 中循环定义的项；·多个条目使用同一定义。

3)DFD+DD　·在 DD 中没有定义的 DFD 实体名；·DD 中定义了，但在 DFD 中不存在的命名；·DD 中命名和 DFD 中实体名的类型不一致者，如：在 DFD 中，StoreMsg 是数据流，而在 DD 中 StoreMsg 却被定义为存储实体。

4)DFD+Mini_Specs　·DFD 中处理结点和 Mini_Specs I/O 不一致者，要求 DFD 中底层处理结点的输入数据流(输出流)和模块规格说明中的 Inputs(Outputs)一致。

5)Mini_Specs 和生成的代码　·代码中没有说明的变元及没有定义的数据类型；·代码中重复定义的数据类型、常元及重复说明的变元；·Mini_Specs 的 Inputs 表中元素在代码中没有被引用；·Mini_Specs 的 Outputs 表中元素在代码中没有被赋值。

七、总结

PC/CASE 系统使用了较好的 DFD 层次描述方法、由 DFD 自动生成结构图算法、完备的软件一致性检查，是一个支持软件开发全过程的 CASE 工具。该系统界面友好、工作台面向图形，能自动生成或保留中间软件文档，能自动生成 C 语言代码，为软件开发人员进行结构化程序设计提供了一个友好的开发环境。

参考文献

[1] T. G. Lewis, CASE: Computer-Aided Software Engineering, 1991, Van Nostrand Reinhold, New York

[2] Alan S. Fisher, CASE Using Software Development Tools, 1988, John Wiley & Sons, Inc.

[3] Poh-Tin Lee et al., Modelling of Visualised Data Flow Diagrams Using Petri Net Model, Software Engineering Journal, January 1992, p4-12

[4] Anthony Earl, Using a Raference Model for CASE Frameworks to Describe PCTE^{++}, SEE V.3, 1991, p123-142

[5] Richard Snpdgress, The Interface Description Language: Defition and Use, 1989, Computer Science Press

[6] R. J. Norman, M. Chen, Working together to Integrate CASE, 1992, IEEE Software

[7] C. Mcclure, CASE Is Software Automation, 1989, Prentice Hall. Englrwood Cliffs, New Jersey 07632

（上接第73页）

（1994年3月 MIT 的 R. Davis 在浙江大学讲学，提到知识表达的问题：较强的问题求解方法及合适的知识表达方法均不能彻底解决智能系统开发问题—主要由于现实世界和人们对问题的理解始终有一段距离（失真）。然而，我们认为软件系统的定义不尽然，因为人们对软件系统的功能需求是清楚的，只要能提供这样一种完善的领域模型定义语言（跟经验知识的描述不同），将为软件定义的实现提供自动化的可能。）

参考文献

[1] R. T. Yeh, 新型的软件演进风范，计算机科学，1991年第6期

[2] N. E. Fuchs, Specification are (preferably) exectable, Software Engineering Journal, Sep. 1992

[3] I. J. Hayes, Specifications are not (necessarily) executable, Software Engineering Journal, Nov. 1989

[4] P. Zave, Salient Features of an Executable Specification Language and Its Environment, IEEE SE-12, 2(1986)

[5] C. Rich, et al., The Programmer's Apprentice: A Research Overview, IEEE Computer 1988, 21(11): 10-25

[6] V. Berzins, et al., Using Transformations in Specification-Based Prototyping, IEEE SE-19, 5(1993)

[7] Michael R. Lowry, Software Engineering in Twenty-First, AI Magazine, 1992, Vol 14, №.3（亦见本刊94№2）

[8] Michael R. Lowry, R. D. McCartney, Automating Software Design, AAAI Press, 1991

[9] D. R. Smith, et al., Research on Knowledge-based Software Environments at Kestrel Institute, IEEE Tran. on Software Engineering, 1985, 11(11): 1278-1295

[10] D. R. Barstow, A Perspective on Automatic Programming, Readings in Artificial Intelligence and Software Engineering, 1986, Eds. C. Rich and R. C. Waters, 537-559, San Mateo, Calif: Morgan Kaufmann

[11] R. Balzer, A 15 Year Perspective on Automatic Programming, IEEE SE-11, 11(1985)

[12] V. Berzins, Luqi, An Introduction to the Specification Language Spec, IEEE Software, Mar. 1990

[13] P. Zave, An Operational Approach to Requirements Specification for Embedded Systems, IEEE SE-8, 3(1982)

[14] Y. Tung, et al., Multiple Views of an Executable Software Specification Language, J. Systems Software, 21(1993)