A Software Engineering Approach to the Integration of CAD/CAM Systems

by

Sandra Lynn Pennington

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

APPROVED:

Dr. A. Myklebust, Committee Chairman

Dr. M. P. Deisenroth

Dr. J. R. Mahan

Dr. D. J. Nelson

Dr. C. F. Reinholtz

March 5, 1991

Blacksburg, Virginia

A Software Engineering Approach to the Integration of CAD/CAM Systems

by

Sandra Lynn Pennington

Dr. A. Myklebust, Committee Chairman

Mechanical Engineering

(ABSTRACT)

This dissertation presents a new approach to the generation of integration systems for CAD/CAM application programs which enlists the tools and methodologies of computer aided software engineering, also referred to as CASE. A CASE workbench is described which aids in the design, analysis and automatic generation of program code for the integration of CAD/CAM applications based on a variety of integration models.

The requirements and detail design of a CASE integration toolkit, which will be incorporated into the CAD/CAM CASE workbench, is presented. This toolkit assists with the identification of vital information from the CAD/CAM application programs, such as code structure, comment blocks and explicit information about the occurrence and location of variables, arrays and subprograms, that is not normally supplied by compilers or other source code analysis tools. This specialized toolkit helps to quickly acquaint the integration system designer with unfamiliar application programs and greatly assists in the integration process. The information obtained by the CASE integration toolkit is subsequently passed into the CASE design and analysis toolkits and used to assist in the semi-automatic generation of the integration system.

As a basis for many of the ideas in this research, a survey of CAD/CAM integration in industry was undertaken. A wide variety of Fortune 500 companies participated in this

project, providing important information on current integration methods, problems and needs. A synopsis of the survey questions and responses is presented.

Additionally, a new integration model, referred to as the High-level Autonomous Integration Model (HAIM), is also described. The autonomous integration model allows a higher-level of integration than is normally considered through the utilization of the X Window System and database methods. This model includes the transfer of necessary data as well as simultaneous execution of application programs and the ability to share menus and functions between applications.

The CAD/CAM CASE workbench and integration toolkit are designed to assist an integration system designer with the tedious tasks necessary in any CAD/CAM applications integration. This is not meant to infer that the workbench allows complete automation of the integration process. A skilled and knowledgeable designer, however, can save a great deal of time through the utilization of the analysis, design and generation capabilities provided by the CAD/CAM CASE workbench. A description of the necessary skills and tasks performed by the systems integration designer is also included.

Acknowledgements

I would like to express my sincere thanks to Al Bracco of the IBM Automated Design Division in Manassas, Virginia, and to Al Levit of the IBM Engineering Graphics Planning & Strategy organization in Kingston, New York, for their financial and personal support of this project both before and since its inception. A special thanks is also extended to the other IBM organizations who contributed to the funding of this research.

I would additionally like to thank Ken Browning, of the IBM Automated Design Division in Manassas, Virginia, for personally adopting this project and for giving special help wherever necessary, including several trips to Blacksburg and numerous phone conversations.

The many contributions of the local IBM office in Roanoke, Virginia, must also be mentioned here, especially the constant support and assistance of Becki Ramsey who cheerfully provided information, in short order, on a myriad of different topics as well as many professional contacts both within IBM and outside it.

Deepest gratitude and thanks goes to my advisor, Dr. Arvid Myklebust, for his continual encouragement, his knowledgeable guidance and his careful fostering of my professional development far above and beyond what was required. Without the loyal support and respect he has shown me over the past eight years of our association, I would not have attempted or completed my Ph.D requirements.

Thanks are also extended to the members of my advisory committee for their continual support and advice during my long tenure as a student here at Virginia Tech. To Dr. M. Deisenroth for his exceptional teaching skills and enthusiastic dedication to the courses I took with him and for providing me with professional and personal support through his kind words on many occasions too numerous to mention. To Dr. C. Reinholtz for having the patience to serve on both my Master's and my Ph.D. committees. To Dr. D. Nelson for his guidance and direction in the improvement of my knowledge of heat transfer. And finally, to Dr. B. Mahan for his frequent words of encouragement and his willingness to offer assistance whenever required.

I would also like to thank the many students and faculty who offered me help, support, advice and friendship during my seven years as a graduate student, instructor and research associate. Their candid conversations, advice and humor often made the tedious work and long hours bearable.

A very special debt of gratitude goes to my parents for laying a foundation for me to build my life on that has truly given me the strength, confidence and discipline to accomplish things I never thought possible. Their continued love, support and understanding has carried me through many tough times I might not otherwise have navigated.

To my husband Bob, my closest friend, constant companion and strongest supporter, I am forever indebted for changing my life with his love and for helping me to realize the truly important things in life.

Finally, I dedicate this work to the memory of Grace Harriet Dale, my grandmother, a very special person who taught me more than she knew and reminded me of the precious yet fragile nature of life. The memory of her limitless and unconditional love will continue to support and inspire me throughout my life.

Table of Contents

1.0 Introduction 1

1.1 Research Objectives 5

1.2 Summary 6

2.0 Literature Review 7

2.1 Introduction 7

2.2 CAD/CAM Integration 7

2.3 Computer Aided Software Engineering 10

2.3.1 What is CASE? 11

2.3.2 The Software Development Process 13

2.3.2.1 Requirements Analysis and Specification 15

2.3.2.2 Design 15

2.3.2.3 Implementation / Coding 16

2.3.2.4 Test and Release 16

2.3.2.5 Maintenance 16

2.3.3 Structured Methodology Overview 17

2.3.3.1 Structured Analysis 17

2.3.3.2 Structured Design 20
2.3.3.3 Data Modeling 23
2.3.4 Essential Diagrams 24
2.3.5 CASE Toolkits, Workbenches and Methodology Companions 26
2.4 Standards 30
2.5 Summary 31

3.0 Synopsis of Industry Survey on CAD/CAM Integration 32
3.1 Survey Results 34
3.2 Summary 55

4.0 A CASE Approach to the Generation of CAD/CAM Integration Systems 57
4.1 Introduction 57
4.2 Current Methods of Joining CAD/CAM Applications 59
4.2.1 Rigidly-Connected Interfacing 62
4.2.2 Rigidly-Connected Coupling 63
4.2.3 Freely-Connected Interfacing 63
4.2.4 Freely-Connected Coupling 64
4.3 Conclusions Based on Current Methods of Joining Applications 65
4.4 The CAD/CAM CASE Workbench and Integration Toolkit 66
4.5 Application of CASE Methods to the Creation of Integration Systems 68
4.5.1 Rigidly and Freely-Connected Interfacing 68
4.5.2 Rigidly and Freely-Connected Coupling 70
4.6 A Possible Workbench Extension 73
4.7 How to Use the CASE Workbench for the Creation of an Integration System 74
4.8 Summary 76

5.0 Requirements Analysis and System Specification for the CAD/CAM CASE Workbench 77

5.1 Introduction 77
5.2 CAD/CAM CASE Workbench Requirements 79
5.3 CASE Workbench Specification 81
5.4 Summary 88

6.0 Requirements Analysis and Design Specifications for the CAD/CAM CASE Integration Toolkit 89
6.1 Introduction 89
6.2 CASE Integration Toolkit Requirements 90
6.3 Design of the CASE Integration Toolkit 97
6.3.1 Design Considerations 98
6.3.2 Module A - MAIN_PROG 102
6.3.3 Module A.3 - EXTRACT_STMT 102
6.3.4 Module A.5 - PROCESS_STMT 103
6.3.5 Module A.5.2 - PROC_START 103
6.3.5.1 Module A.5.2.3 - P_PGM 104
6.3.5.2 Module A.5.2.5 - P_SUB 104
6.3.5.3 Module A.5.2.7 - P_BLKDAT 104
6.3.5.4 Module A.5.2.9 - P_FUNC 105
6.3.6 Module A.5.6 - PROC_NONEXEC 105
6.3.6.1 Module A.5.6.4 - P_STMT_FUNC 106
6.3.6.2 Module A.5.6.6 - P_REAL 106
6.3.6.3 Module A.5.6.8 - P_INTGR 106
6.3.6.4 Module A.5.6.10 - P_LOGL 107
6.3.6.5 Module A.5.6.12 - P_CHAR 107
6.3.6.6 Module A.5.6.14 - P_DBL_PR 107
6.3.6.7 Module A.5.6.16 - P_COMP 107

6.3.6.8 Module A.5.6.18 - P_COMM 108
6.3.6.9 Module A.5.6.20 - P_IMPL 108
6.3.6.10 Module A.5.6.22 - P_INTR 108
6.3.6.11 Module A.5.6.24 - P_EXT 108
6.3.6.12 Module A.5.6.26 - P_PARAM 109
6.3.6.13 Module A.5.6.28 - P_EQUIV 109
6.3.6.14 Module A.5.6.30 - P_DIMEN 109
6.3.6.15 Module A.5.6.32 - P_DATA 109
6.3.6.16 Module A.5.6.34 - P_SAVE 110
6.3.7 Module A.5.10 - PROC_EXEC 110
6.3.7.1 Module A.5.10.4 - P_ASSN_STMT 111
6.3.7.2 Module A.5.10.6 - P_WRITE 111
6.3.7.3 Module A.5.10.8 - P_PRINT 111
6.3.7.4 Module A.5.10.10 - P_READ 111
6.3.7.5 Module A.5.10.12 - P_FORMAT 112
6.3.7.6 Module A.5.10.14 - P_OPEN 112
6.3.7.7 Module A.5.10.16 - P_CLOSE 112
6.3.7.8 Module A.5.10.18 - P_REWIND 112
6.3.7.9 Module A.5.10.20 - P_BCKSP 113
6.3.7.10 Module A.5.10.22 - P_ENDFIL 113
6.3.7.11 Module A.5.10.24 - P_ENTRY 113
6.3.7.12 Module A.5.10.26 - P_IF 114
6.3.7.13 Module A.5.10.34 - P_DO 114
6.3.7.14 Module A.5.10.38 - P_CALL 114
6.3.7.15 Module A.5.10.40 - P_GOTO 115
6.3.7.16 Module A.5.10.48 - P_RETN 115

6.3.8 Module A.5.14 - P_END 115
6.3.9 Subroutine Utilities 116
6.3.9.1 Module S.1 - CMT_STMT 116
6.3.9.2 Module S.2 - TOKENIZE_LIST_1 116
6.3.9.3 Module S.3 - TOKENIZE_LIST_2 117
6.3.9.4 Module S.4 - GET_PAREN_LIST 117
6.3.9.5 Module S.5 - GET_INT_NUM 118
6.3.9.6 Module S.9 - TOKENIZE_LIST_3 118
6.3.9.7 Module S.11 - GET_STRING_1 119
6.3.9.8 Module S.14 - STMT_TEST 119
6.3.9.9 Module S.18 - TOKENIZE_EXPR 120
6.4 Implementation Considerations 121
6.5 The CAD/CAM Integration Interface 122
6.6 Summary 123

7.0 The High-Level Autonomous Integration Model 124
7.1 Introduction 124
7.2 The Autonomous Model and its Components 127
7.2.1 The System Executive 131
7.2.2 Inter-Client Communications 132
7.3 Application of CASE Methods to the Autonomous Model 134
7.4 Summary 136

8.0 Functions of the Integration System Designer 137
8.1 Necessary Skills and Background Level 137
8.2 Overview of Required Tasks 139
8.3 Summary 141

9.0 Research Summary **142**

9.1 Future Research Plans 144

References **145**

Appendix A. CAD/CAM Integration Survey Form **149**

Appendix B. Flowcharts for the VPI CASE Integration Toolkit **159**

Appendix C. GLOSSARY **254**

Vita **257**

List of Illustrations

Figure 1. Simplified Data-Flow Diagram for Typical Computer Integrated Enterprise 4

Figure 2. Traditional Software Development Life Cycle (SDLC) 14

Figure 3. Structure Chart 21

Figure 4. Basic Toolkits of the Software Development Life Cycle 27

Figure 5. Integration System Model 58

Figure 6. CAD/CAM Integration System Paradigm 60

Figure 7. An Example Integration 75

Figure 8. CASE Workbench Designed for CAD/CAM Integration 82

Figure 9. Flowcharting Symbols 101

Figure 10. The High-level Autonomous Integration Model (HAIM) 128

Figure 11. Multi-Platform Support Enabled by the X Window System 130

Figure 12. Module A - MAIN_PROG 160

Figure 13. Module A.1 - INITIAL_PGM 161

Figure 14. Module A.3 - EXTRACT_STMT 162

Figure 15. Module A.3.12 - CONT_LINE 163

Figure 16. Module A.5 - PROC_STMT 164

Figure 17. Module A.5.2 - PROC_START 165

Figure 18. Module A.5.2.3 - P_PGM 166

Figure 19. Module A.5.2.5 - P_SUB 167

Figure 20. Module A.5.2.7 - P_BLKDAT 168
Figure 21. Module A.5.2.9 - P_FUNC 169
Figure 22. Module A.5.2.9.6 - FUNC_TYPE_INT 170
Figure 23. Module A.5.2.9.8 - FUNC_TYPE_REAL 171
Figure 24. Module A.5.2.9.10 - FUNC_TYPE_DBL 172
Figure 25. Module A.5.2.9.12 - FUNC_TYPE_LOG 173
Figure 26. Module A.5.2.9.14 - FUNC_TYPE_C 174
Figure 27. Module A.5.2.9.15 - PROC_FUNC_ARG 175
Figure 28. Module A.5.6 - PROC_NONEXEC 176
Figure 29. Module A.5.6.T - PROC_NONEXEC_TABLE 177
Figure 30. Module A.5.6.4 - P_STMT_FUNC 178
Figure 31. Module A.5.6.6 - P_REAL 179
Figure 32. Module A.5.6.8 - P_INTGR 180
Figure 33. Module A.5.6.10 - P_LOGL 181
Figure 34. Module A.5.6.12 - P_CHAR 182
Figure 35. Module A.5.6.14 - P_DBL_PR 183
Figure 36. Module A.5.6.16 - P_COMP 184
Figure 37. Module A.5.6.18 - P_COMM 185
Figure 38. Module A.5.6.18.3 - TOKENIZE_COMMON 186
Figure 39. Module A.5.6.18.5 - STORE_COMMON 187
Figure 40. Module A.5.6.20 - P_IMPL 188
Figure 41. Module A.5.6.22 - P_INTR 189
Figure 42. Module A.5.6.24 - P_EXT 190
Figure 43. Module A.5.6.26 - P_PARAM 191
Figure 44. Module A.5.6.26.5 - PROC_PARAM_NAMES 192
Figure 45. Module A.5.6.28 - P_EQUIV 193

Figure 46. Module A.5.6.28.3 - TOKENIZE_EQUIV 194
Figure 47. Module A.5.6.28.6 - STORE_EQUIV 195
Figure 48. Module A.5.6.30 - P_DIMEN 196
Figure 49. Module A.5.6.32 - P_DATA 197
Figure 50. Module A.5.6.32.3 - TOKENIZE_DATA 198
Figure 51. Module A.5.6.32.6 - STORE_DATA 199
Figure 52. Module A.5.6.34 - P_SAVE 200
Figure 53. Module A.5.10 - PROC_EXEC 201
Figure 54. Module A.5.10.T - PROC_EXEC_TABLE 202
Figure 55. Module A.5.10.4 - P_ASSN_STMT 203
Figure 56. Module A.5.10.4.6 - ARRAY_ASSN 204
Figure 57. Module A.5.10.4.7 - VAR_ASSN 205
Figure 58. Module A.5.10.6 - P_WRITE 206
Figure 59. Module A.5.10.8 - P_PRINT 207
Figure 60. Module A.5.10.8.3 - GET_PR_FMT 208
Figure 61. Module A.5.10.10 - P_READ 209
Figure 62. Module A.5.10.12 - P_FORMAT 210
Figure 63. Module A.5.10.14 - P_OPEN 211
Figure 64. Module A.5.10.16 - P_CLOSE 212
Figure 65. Module A.5.10.18 - P_REWIND 213
Figure 66. Module A.5.10.20 - P_BCKSP 214
Figure 67. Module A.5.10.22 - P_ENDFIL 215
Figure 68. Module A.5.10.24 - P_ENTRY 216
Figure 69. Module A.5.10.26 - P_IF 217
Figure 70. Module A.5.10.26.6 - P_IF_VARIANTS 218
Figure 71. Module A.5.10.34 - P_DO 219

Figure 72. Module A.5.10.38 - P_CALL 220
Figure 73. Module A.5.10.38.7 - PROC_CALL_ARG 221
Figure 74. Module A.5.10.40 - P_GOTO 222
Figure 75. Module A.5.10.40.5 - COMPUTED_GOTO 223
Figure 76. Module A.5.10.48 - P_RETN 224
Figure 77. Module A.5.14 - P_END 225
Figure 78. Module S1 - CMT_STMT 226
Figure 79. Module S1.2 - WRT_BUFF 227
Figure 80. Module S2 - TOKENIZE_LIST_1 228
Figure 81. Module S3 - TOKENIZE_LIST_2 229
Figure 82. Module S3.6 - PROC_LAST_TOKEN 230
Figure 83. Module S3.8 - PROC_TOKEN_LIST 231
Figure 84. Module S3.10 - PROC_TOKEN 232
Figure 85. Module S4 - GET_PAREN_LIST 233
Figure 86. Module S5 - GET_INT_NUM 234
Figure 87. Module S6 - PROC_LEN_MOD 235
Figure 88. Module S7 - PROC_NAMES 236
Figure 89. Module S8 - P_DUMMY 237
Figure 90. Module S9 - TOKENIZE_LIST_3 238
Figure 91. Module S9.10 - TOKEN_A_FOUND 239
Figure 92. Module S9.10.7 - TOKEN_B_FOUND 240
Figure 93. Module S10 - NO_TOKEN_A 241
Figure 94. Module S11 - GET_STRING_1 242
Figure 95. Module S12 - INITIAL_SUBPGM 243
Figure 96. Module S13 - GET_STRING_2 244
Figure 97. Module S14 - STMT_TEST 245

Figure 98. Module S14.8 - SFUNC_OR_ASSIGN 246
Figure 99. Module S15 - GET_CHAR_CONST 247
Figure 100. Module S16 - TEST_NAME 248
Figure 101. Module S16.3 - TEST_FOR_NAME 249
Figure 102. Module S16.6 - TEST_FOR_CONST 250
Figure 103. Module S17 - TEST_ARG_ARRAY 251
Figure 104. Module S18 - TOKENIZE_EXPR 252
Figure 105. Module S19 - PROC_SUB_ARG 253

1.0 Introduction

It is virtually impossible, today, to read through an article about CAD/CAM and not encounter some reference to integration. By integration we mean the sharing of necessary data and information between dissimilar CAD/CAM systems and application programs. Although, the components of CAD/CAM systems have rather rapidly evolved since their inception, the problems of system integration have only become more complex and more elusive. The sudden proliferation of CAD/CAM software programs, unfortunately, was accompanied by a lack of standardization and coordination between software developers. These problems resulted in application programs which are unable to communicate with most programs external to their own code. It has been suggested that these integration problems in themselves may have served to diminish the widespread implementation of CAD/CAM systems that was once predicted. It is, perhaps, highly possible that better methods of integration could enable CAD/CAM technology to realize its full potential as predicted by experts nearly two decades ago.

A standard textbook definition of a CAD/CAM system generally states in one form or another that CAD/CAM links the functions of CAD and the functions of CAM through the use of a common database. In reality, however, this is not usually the case. The

major commercial applications which refer to themselves as integrated CAD/CAM systems (e.g. CADAM, CATIA, CAEDS) usually offer true integration only within their own product line and very rarely to products outside it. While this is acceptable if the end user's needs can be met by this one application, it is more common that the majority of CAD/CAM users must combine a number of different CAD and CAM software systems and application programs to adequately perform the tasks they require. These users must then, on an individual basis, try to solve the integration problems that the combined systems present. The problems encountered when integrating two or more programs are formidable. Unfortunately, the current methods of integration are still at a relatively low level. The most common solution enlisted is the data interface which transfers entities from one dissimilar software system to another through the use of a neutral file. This neutral file, which must undergo two separate translations, is frequently written in IGES (Initial Graphics Exchange Specification) format, developed under the leadership of the National Bureau of Standards. Although IGES has been the most popular method of moving data from one system to another, it does have many limitations which have been widely discussed in the literature. Among these problems, perhaps the most restrictive is that IGES can only support simple entities such as lines, circles and dimensions and cannot handle more complex conceptual models or nongeometric information. A future alternative to IGES will be the Product Data Exchange Specification (PDES) which is currently under development by the IGES organization. PDES will allow the representation of more conceptual geometric models as well as the passing of nongeometric information. It is also hoped that PDES will be able to work with more sophisticated CAD systems that utilize solid models. Unfortunately, PDES will probably not be commercially available until the mid 1990's, and while its planned capabilities are quite impressive, it is hard to speculate how successful its actual implementation will be. Its major drawback, as with IGES, is that it is still only a data

transfer type of integration. A large number of proprietary neutral file formats are also frequently utilized but again these systems have inherent problems related to data limitations and multiple translations at this low level.

As an alternative to data transfer, some users are integrating CAD/CAM applications at a higher level. These are mostly custom written interfaces which utilize proprietary programming interfaces in conjunction with commercially available CAD/CAM systems such as CATIA IUA or CADAM Access. While these programming interfaces provide a large number of sophisticated functions, they are quite costly and can be very time consuming to implement. They also lock the user into many pre-defined functions and limitations of the commercial CAD/CAM system such as proprietary database use and management.

As an example of the current methods of CAD/CAM integration, a simplified data-flow diagram for a typical computer-integrated enterprise is shown in Figure 1 on page 4. In the figure, all of the system modules are shown interconnected through data interfaces. This is, in reality, an idealized diagram since most companies are not yet integrated to even this extent. The diagram also shows the use of proprietary programming interfaces such as CATIA IUA or CADAM IUE for "Custom Design and Manufacturing Applications."

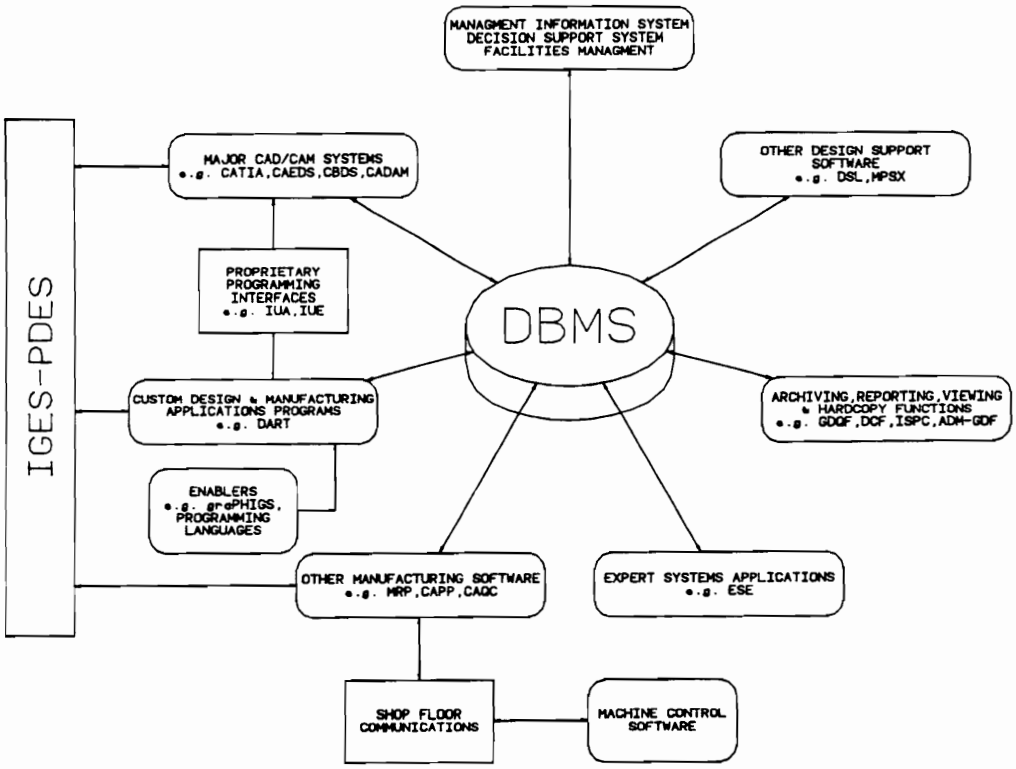


Figure 1. Simplified Data-Flow Diagram for Typical Computer Integrated Enterprise

1.1 Research Objectives

The objectives of this research can be outlined briefly as follows:

I. Establish User Requirements: Survey and report on integration needs in industry including the type of hardware and software systems presently used, the type of integration these systems possess and the type of entities, functions and other types of information that need to be shared for good integration.

II. Detail an integration system generator.

A. Specify an appropriate CASE workbench which is specially designed to provide assistance to an integration system designer for the solution of CAD/CAM integration problems.

B. Design a CASE integration toolkit that is an integrated part of the CAD/CAM CASE workbench and that will focus on the support of the integration system analysis phase. This will primarily include inquiry routines which assist in the identification and extraction of important application program characteristics which are vital to successful integration.

C. Describe an integration interface which will use the detailed information provided by the CASE integration toolkit to provide specialized assistance to the CASE analysis and design tools for the creation of integration systems.

D. Describe a new model for the integration of dissimilar CAD/CAM application programs which will allow the sharing of not only data, such as geometry and analysis results, but also system menus and functions as well.

III. Determine the functions which must be performed by the integration system designer to complete the generation of the integration system.

1.2 Summary

This chapter has presented a brief introduction to the problems of CAD/CAM applications integration as well as the objectives of this research. The next chapter presents a literature review of many current CAD/CAM integration methods and an overview of computer aided software engineering with some applications.

2.0 Literature Review

2.1 Introduction

This chapter cites numerous examples of specific CAD/CAM integrations discussed in the literature along with a brief description of each. A concise overview of computer aided software engineering and some of its applications is also presented. Finally, a brief word on standards concludes the chapter.

2.2 CAD/CAM Integration

CAD/CAM integration has been treated in the literature in a variety of ways but to a large extent the predominant method has always been at the level of data passing between processes. One reason that this method has gained such wide acceptance was the publication of IGES, Version 1.0, as part of an ANSI standard in 1981. Liewald and

Kennicott [Lieu82] were among the first to advocate the use of IGES for the integration of CAD/CAM processes. They were also aware, however, of the great limitations which were presented by data transfer towards the purpose of total integration. This view was stated as follows: "Clearly, no data standard is an ultimate solution to the full scope of CAD/CAM data interface problems... IGES is a baseline from which we can derive immediate and continuing functional benefit while gaining the practical knowledge needed for more advanced efforts." Martin and Downs [Mart85] also emphasize the importance of CAD/CAM integration to the universal acceptance and effectiveness of CAD/CAM implementations. They insist, however, that while data interfaces are an expedient way to transfer data from one process to another, the best integration method would be the use of a centralized CAD/CAM database. While this idea developed a large following, especially among experts, the decline in popularity of turnkey CAD/CAM systems coupled with the rise in segregated CAD and CAM software development has decreased the viability of this solution. Pasemann [Pase85] states that the "ultimate benefit" of CAD applications will not occur until total integration of the various systems is achieved. He overviews the most common methods currently in industrial use which are, again, at the data level. Mayer [Maye87] gives an in-depth technical overview of the functions of IGES. He also discusses the many shortcomings and weaknesses of IGES and predicts that its successor, PDES, will solve the problems which IGES cannot. As the author goes on to mention, however, PDES will not be available until sometime in the 1990's and even though it allows for a larger variety of information to be passed, it still allows integration only at the data level. A more sophisticated example of CAD program integration is described by Rowell, Schwing, and Jones [Rowe88]. The authors provide a set of utility routines which assist in integrating complex analysis codes through the use of a database management system. Some of these utilities include tools for constructing data dictionaries and database schema, tools

for generating subroutines to read from or write to the database, the ability to interactively review and modify values in the database as well as menu building assistance. A managing program allows the user to select the programs that need to be executed from a menu. This method of integration is currently used within several NASA agencies. It provides an effective way to integrate and share data but does not allow any sharing of graphic information or more sophisticated integration functions.

A very impressive application of proprietary programming interfaces for integration of CAD/CAM applications is described in the general information manual for two new IBM software systems: Product Engineering Support (PES) and Computer-Aided Design Integration (CADI) [IBM89]. This integration system is unique in that it not only allows transfer of data and information between supported CAD systems (CADAM and CATIA), an engineering database and a production planning system, but it also allows the user to simultaneously perform functions of both programs interactively while both programs are executing. Another example of integrating CAD applications with proprietary programming interfaces is given by Lu, Myklebust and War [Lu87] in the integration of a helicopter design code with CADAM. This integration did not treat static data but parametric geometric models with all the implied connectivity. Subsequent to this work, Jayaram and Myklebust [Jaya89] devised an automatic generator of geometry interfaces between applications programs and a commercial CAD/CAM system. A prototype was written using CADAM-ISD and IBM's Expert System Environment.

A number of authors have proposed artificial intelligence (AI) techniques for the integration of CAD/CAM systems. Ramanathan [Rama86] presents a model for an "intelligent integration of existing engineering tools" which utilizes database integration and AI techniques. Shin and Maryanski [Shin88] discuss integration problems facing current CAD/CAM and CIM environments and present an approach to solving this prob-

lem using data modeling and knowledge-based dialogue management. Their approach, however, is centered around the integration of information for the CIM environment, such as inventory or cost analysis, and does not directly address the integration of typical CAD models (i.e. geometric descriptions).

2.3 *Computer Aided Software Engineering*

Another important field which is beginning to show promise for CAD/CAM applications is computer aided software engineering or CASE. While CASE was originally applied only to the development of commercial applications, it is beginning to be used in many scientific application environments as well. CASE is making it easier to create software which can accurately meet the requirements originally specified for it. It is also providing methods for automatically generating software and maintaining finished code. The importance of these features along with their possible links to applications such as CAE is discussed by Manuel [Manu87]. Another paper entitled "Integrating CAE, CAD and CASE" by Marshall and Van Dyne [Mars86] discusses the application of CASE methods to the development of microprocessor software to be downloaded to integrated circuits in an emulator environment and the development of integrated CASE tools for the design and manufacture of the integrated circuits themselves. The research does not address the integration of the CAE and CASE systems as the paper title would seem to indicate. Of particular interest in CASE technology is the implementation of automatic programming ideas. Balzar [Balz85] gives a 15 year perspective on work in the automatic programming area. This paper describes methods of acquiring high-level specifications for automatic programming but also some means of determining correct-

ness and methods of translating this into a low-level specification that can be automatically compiled. Luker and Burns [Luke86] describe a program generator as including "the dialogue module which interacts with the user to obtain problem specifications, the intermediate module which is analogous to macro expansion for the specification and finally, the code generation module which produces code and documentation." Though CASE is a relatively new technology, it is beginning to see applications in a wide number of areas. Bird and Schofield [Bird85] describe a software engineering tool with a skeleton code generator which can be used in developing applications for the CAD, CAM and CAE fields. Again, the integration of these systems is not addressed.

A brief overview of CASE is now given to support the ideas presented in this research.

2.3.1 What is CASE?

CASE technology can be simply defined as the automation of software development. Based on the principles and methodologies of software engineering, CASE provides support for all phases of the software development life cycle, including design, implementation and maintenance.

CASE systems are currently in widespread use, especially for commercial applications, and provide many proven benefits such as:

- Improving the quality of systems designed
- Encouraging evolutional and incremental development
- Improving the productivity of the project team

- Enabling project schedules to be met more often

- Providing more automated configuration management

- Allowing applications to be maintained at the design level rather than the code level

The evolution of CASE technology began in the early 1980's with the introduction of documentation and graphical diagramming tools that were based on structured systems methodologies developed in the 1960's and 1970's. These tools not only provided the first software development aids but were also representative of the first attempts to automate software design and analysis tasks. These early tools were primarily PC-based. In the mid-1980's, CASE tools were enhanced to include tools which facilitated design automation such as automated checking of structured diagrams and the addition of data dictionaries or repositories for the storage of these diagrams and related information. The next step in the evolution of CASE technology was the linking of design automation and program automation through the implementation of code generators within CASE toolsets. These tools were primarily mainframe based. During this same time, development began on ***frameworks*** which would help to support and link these various CASE tools together. By providing a data repository as well, these frameworks or ***integrated project support environments (IPSEs)*** are making the integration of design and program automation possible along with the maintenance of completed code.

2.3.2 The Software Development Process

Before we discuss CASE tools in detail, it is important to fully understand the software development process. Several well-defined steps compose this process which leads to well-designed, maintainable code. An important concept in software development is that software development is not a linear process as it is often represented, but more of a cycle. This is easy to understand when we consider that every software program must be continually maintained and frequently expanded to keep up with its user community. The process of development, therefore, is an ongoing one or a cyclic one.

The steps which are generally accepted as those which make up the software development life cycle are shown in Figure 2 on page 14.
This life cycle concept has been in widespread use since the 60's and has been continually developed and refined since that time. As shown in the figure, these life cycle steps include:

- Requirements analysis and specification
- Design
- Implementation / Coding
- Test and Release
- Maintenance

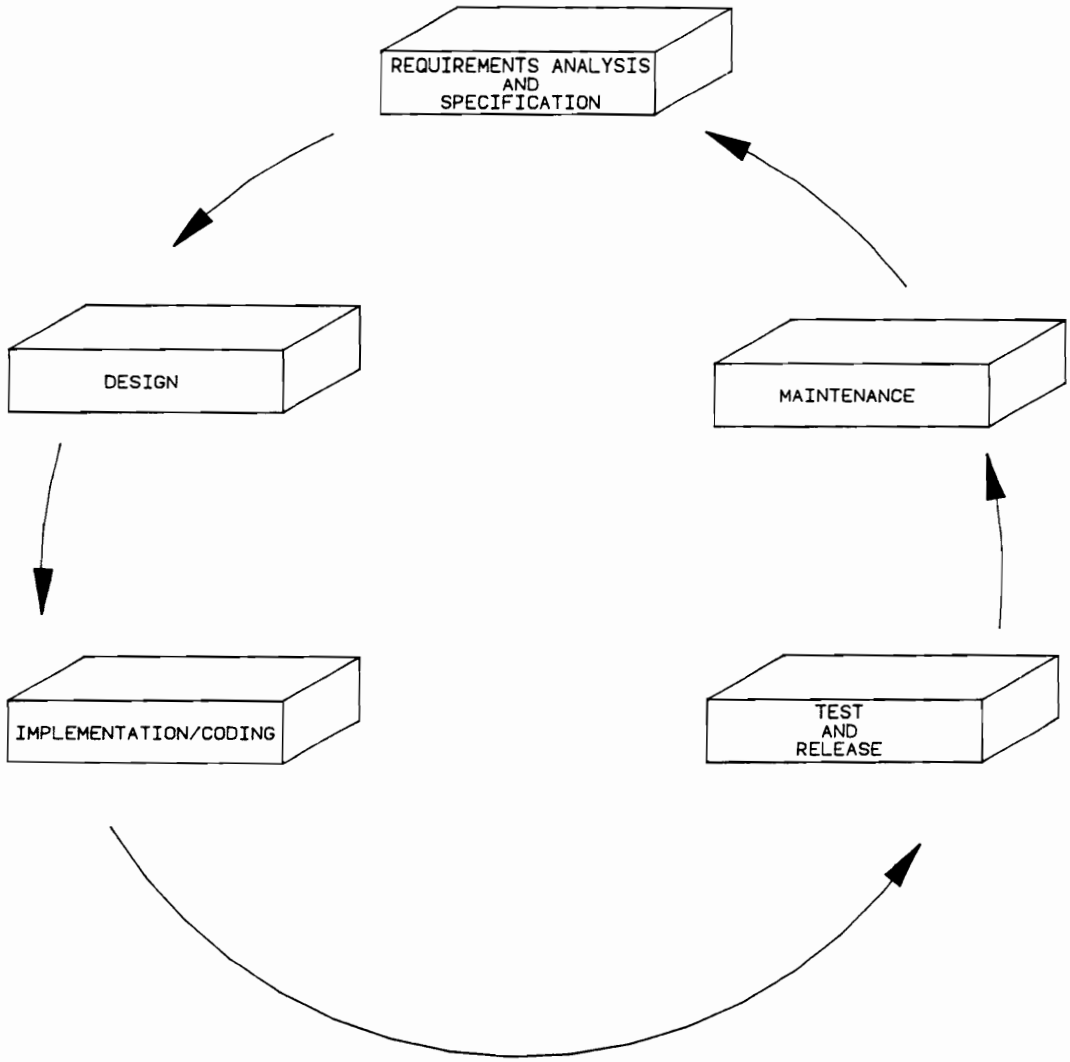


Figure 2. Traditional Software Development Life Cycle (SDLC)

2.3.2.1 Requirements Analysis and Specification

The objective of this phase of the life cycle is to completely define all requirements of the user organization, to carefully analyze these needs and to propose solutions. This phase is usually ***Top-down*** which means that the original analysis begins with the management of the organization and then moves down to those persons who will actually use the system. The scope of the proposed system as well as a general outline of system structure are defined and then taken back to management for further addition and refinement. An acceptable justification of the proposed system as well as a cost analysis must also be provided. A document called a requirements specification is then produced which clearly details the system specification and may also include information such as machine environment constraints, user interface designs and performance goals.

2.3.2.2 Design

In this phase, the previously defined system is very meticulously analyzed to make sure that it meets the original requirements, can be successfully supported by the target hardware and software environment and can allow necessary application integration as well as support future enhancements. Once these items have been carefully checked, a detailed system design or ***design specification*** is developed which includes such things as file format definitions, data structure definitions, program module decompositions and detailed algorithm descriptions. Test methods are also proposed during this phase and a detailed cost analysis for both development and operation is produced.

2.3.2.3 Implementation / Coding

In this phase, the actual coding and debugging of each module is performed exactly as described by the design specification created in the previous phase. If problems are discovered during debugging which indicate logical or structural errors in the design, they must be reported to and solved by the analysts responsible for the design phase. It is not difficult to see why this phase is usually considered the most complex phase of the software development life cycle.

2.3.2.4 Test and Release

Once all coding and debugging is complete, a thorough testing of the software system begins. In this phase, each module built during the coding phase is tested individually with predefined test data and then integrated into a single program structure. Once this is done, the entire program is then carefully tested with predefined test data to ensure the proper program operation. The next step is to send early copies to specific end-users for ***Beta testing***. These users agree to identify any problems or bugs which may still exist in the new system for a certain period of time. Once all reported problems and bugs have been corrected, a final release is created and distributed to the general end-user community.

2.3.2.5 Maintenance

This final phase of the life cycle is an ongoing one in which bugs and problems are continually addressed as they are found, and enhancement suggestions are collected.

Periodically, it may be necessary to release minor upgrade versions of the software to accommodate bug fixes, system enhancements or changes in the software and hardware environment.

2.3.3 Structured Methodology Overview

Perhaps the biggest impact on the use of the software development life cycle concepts was the introduction of ***structured methodologies***. These new ideas helped to impose rigid structure on the requirements analysis, specification and early design stages of the life cycle through the use of charts and diagrams designed to reduce errors and improve the quality of completed code. In the structured approach, input to every step of the life cycle is defined specifically and there is formal checking of consistency, control and quality at the end of each step. Structured methodologies help program designers to visualize code design through the use of diagrams and schematics much the same as CAD helps engineers visualize engineering data. A brief overview of the most popular structured methodologies will provide a better understanding of these concepts.

2.3.3.1 Structured Analysis

Structured analysis utilizes a top down, functional decomposition approach to identify system requirements. The analysis process produces a system specification which serves as a blueprint for the system to be created. The set of tools whose use produces a structured specification are:

- Data Flow Diagrams

- Data Dictionaries

- Process/Control Specifications

Data flow diagrams are all graphical specifications which emphasize the flow of data not the flow of control. They are hierarchical, have many levels and specify requirements not design. When we decompose data flow diagrams into successively lower levels, this is called leveling. The data flow diagram which represents the lowest-level of decomposition of process nodes is called the ***leveled set***.

Data dictionaries are databases for the definitions of all data elements defined in a data flow diagram. These definitions consist of the components which make up the data item and the relationships among them. Structured analysis CASE tools provide a facility which maintains the data dictionary by adding new data flows to the data dictionary automatically when they are created.

Process/control specifications or ***minispecs*** describe the task performed by a single, bottom-level process node (the leveled set) in a data flow diagram. They are called minispecs because they document only one single process in the entire data flow diagram. Minispecs can be written in a variety of forms including the following:

- Pseudocode/Structured English

- Flowcharts

- Program Design Languages

- Formal Computer Languages

- Decision Tables

- Decision Trees

Two similar and popular methodologies for structured analysis are [McCl89, Fish88]:

- DeMarco

- Gane and Sarson

Tom DeMarco, author of the classic book "Structured Analysis and System Specification," developed a structured analysis method which can be described by a seven step process[McCl89]:

1. Create a data flow diagram which represents the way things are done in the current user environment.

2. Using the data flow diagram created in step one, create a logical model of this system through the use of hierarchically leveled data flow diagrams.

3. Develop a logical model of the new system to be built using step two as a base, once again using a set of leveled data flow diagrams.

4. Create the new physical models for the system to be built.

5. Produce cost and schedule estimates for each of the physical models for the systems to be built.

6. Based on step five, select one physical model.

7. Develop a structured specification which includes a set of leveled data flow diagrams, a data dictionary and minispecs.

The Gane and Sarson methodology is similar to the DeMarco methodology described above but has several differences. A data modeling step is included at step two which describes the data stores shown in the data flow diagram. In step three, this information is used to assist with database design. Additionally, the Gane and Sarson version of the data flow diagram has slightly different symbols than those defined by DeMarco.

2.3.3.2 Structured Design

Structured design methodologies provide assistance for the next phase of program development by guiding the transformation of a data flow diagram into a structured design chart. CASE assists at this stage in the development by providing mechanical aids which help to translate detailed analysis requirements into design specifications for implementation. The main product of this phase is the ***structure chart*** which depicts the entire system design in a hierarchical or tree fashion. These charts graphically illustrate the program modules or procedures and the interconnections between them. Data passed between modules are referred to as couples and are notated on the structure chart alongside the connecting arrows which connect these modules. An example of a structure chart is shown in Figure 3 on page 21.

Two of the most common structured design methodologies are [McCl89, Fish88]:

- Yourdon Structured Design
- Jackson Structured Design

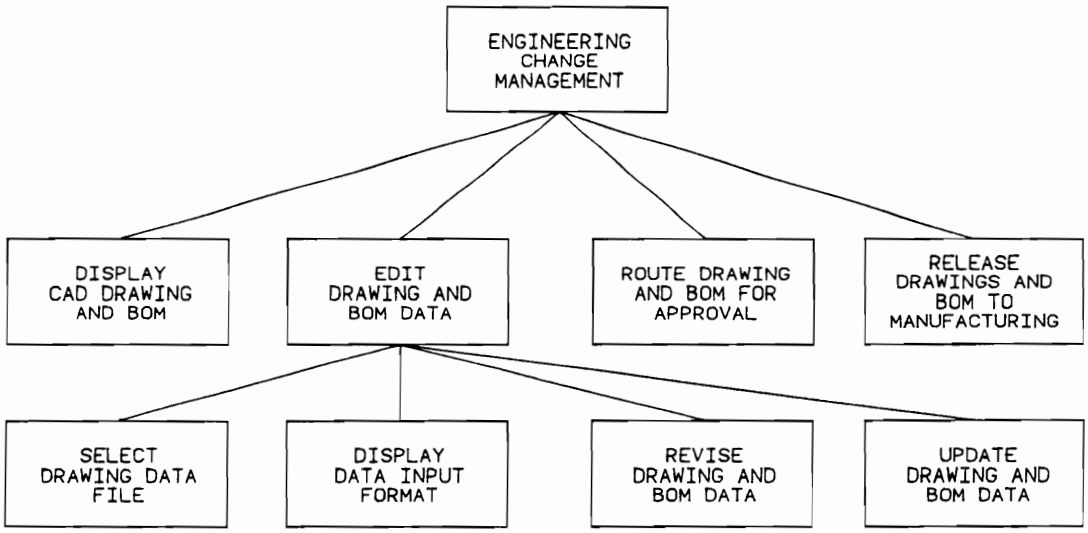


Figure 3. Structure Chart

Yourdon structured design provides a step-by-step approach to system design as well as detailed program design. These steps not only include the development of the design but also its documentation and quality assurance. The four basic steps of the Yourdon methodology are as follows[McCl89]:

1. Draw the data flow diagram.

2. Draw the structure chart. The structure chart is derived from the data flow diagram in one of two ways: transform analysis and transaction analysis. These two techniques provide clear and procedural methods for creating hierarchical structure charts.

3. Evaluate the design. This step evaluates the quality of the design based on coupling, the independence of individual program modules, and cohesion, the measure of relations between elements within a module.

4. Prepare the design for implementation. This is the "divide and conquer" step where program design is broken down into physical implementation units.

Yourdon design methodology can be applied to many different types of problems and is quite widely implemented in CASE tools.

The Jackson Methodology is also hierarchical and provides a step-by-step approach to system design. The major difference, however, is that this methodology is based on an analysis of the data structure and not just the data flow as with the Yourdon method. More simply stated, the Jackson methodology is data-oriented while the Yourdon methodology is procedure-oriented. The Jackson methodology merges conventional

structure charts with a specialized notation to indicate algorithmic execution. Program sequencing, selection and iteration are also indicated on the Jackson designs.

The Jackson program design methodology consists of four steps:

1. Draw the structured diagram for all input and output data in the program.

2. Form all data structures defined in step one into a single program structure.

3. List the operations necessary to convert input to required output. Allocate these operations to a component on the program structure of step 2.

4. Transcribe the program structure into structure text, which is a formal version of pseudocode. Add necessary logic for loops and selection structures.

This method is very popular in data processing organizations. It can be used to design simple programs but cannot be directly applied to complex ones. For this reason, perhaps, there are few implementations of this methodology in CASE tool form.

2.3.3.3 Data Modeling

Another technique which is an important part of the design specification stage and is commonly supported by CASE tools is data modeling. Data modeling assists by allowing the graphical modeling of data to be stored in the database as well as its use by the individual application programs. A data dictionary is used to organize all the data elements in an application's design. Two of the most common data modeling techniques are [McCl89, Fish88]:

- Warnier-Orr Diagrams

- Entity-Relationship Diagrams

The Warnier-Orr diagram uses set theory to describe program design. The diagram is composed of nested set brackets which are used to show the hierarchical structure and process flow of procedures or data. The principle use of the Warnier-Orr diagram is for data modeling such as the designing and refining of file formats and data structures. This diagram does not describe any information related to the data.

The Entity-Relationship diagram looks beyond the simple design of data structures, examining the relationships between the information and data that is used. These diagrams are predominately used to model information that is to be stored in a database. These diagrams are quite straightforward and consist of only two components: entities and relations.

There are a number of CASE tools which incorporate E-R diagrams and many different types of notation are used in these. The most popular notation is probably the one developed by Peter Chen, who is considered to be the original inventor of E-R diagrams.

2.3.4 Essential Diagrams

While several types of methodologies have been discussed here, there is an essential set of three diagrams which can adequately represent any software system. These are:

1. Data Flow Diagram - used during analysis to define program requirements

2. Data Model Diagram - a representation of data entities and the relationships among them

3. Tree Structure Diagram - created during the program design stage to represent the components or modules of a program and their relationships in a single charting structure

A complete CASE workbench should provide some mechanism for drawing and updating at least one type of each of these three diagrams. For real-time systems, other diagram types which show timing, control and sequencing requirements should also be available.

The formal, structured methodologies discussed here form a basis for computer aided software engineering or CASE. It is not hard to see how automation of the many complex and tedious tasks associated with structured methodologies could greatly speed development time and greatly reduce errors. This automation in itself creates structure, which means that whenever we use a CASE tool for a life cycle step that step becomes structured. CASE can best be described as a combination of methodologies and software tools. It encompasses all the phases of the software development life cycle by automating analysis and design tools as well as program implementation, test and maintenance tasks. CASE provides two significant changes to the traditional software development life cycle. The first is that the transformation from one life cycle phase to another is automatic and can easily be repeated. The second is the addition of a database or data repository which maintains the data required for all phases of the life cycle. These ideas force standardization and produce a totally integrated development system which easily facilitates development and maintenance. Current CASE technology is a variety of powerful tools which are predominately graphics-based and are incorporated

into integrated workbenches for ease of use. The next section discusses CASE tools and their place in the software development life cycle.

2.3.5 CASE Toolkits, Workbenches and Methodology Companions

In the previous section, the phases of the software development life cycle were reviewed. At present, there are CASE tools available which address every phase of this cycle. Some specialize in one area only, while others support the entire life cycle. These tools can be grouped into three basic categories which are:

- Toolkits
- Workbenches
- Methodology Companions

A CASE toolkit is a tool which supports one particular phase of the software development life cycle, such as the analysis phase. These toolkits provide graphical assistance for the drawing of necessary diagrams as well as help with tedious record keeping and countless system details. Toolkits can also be categorized by software and hardware as well as by specialized tasks or applications. The most common toolkits which support the software development life cycle are shown in Figure 4 on page 27.

Analysis toolkits automate the creation of the analysis specification, which lists the system requirements. An analysis toolkit usually includes structured diagramming tools for the generation of data flow diagrams and other necessary diagrams, a repository and a specification checker. Analysis toolkits were the first CASE tools to become available

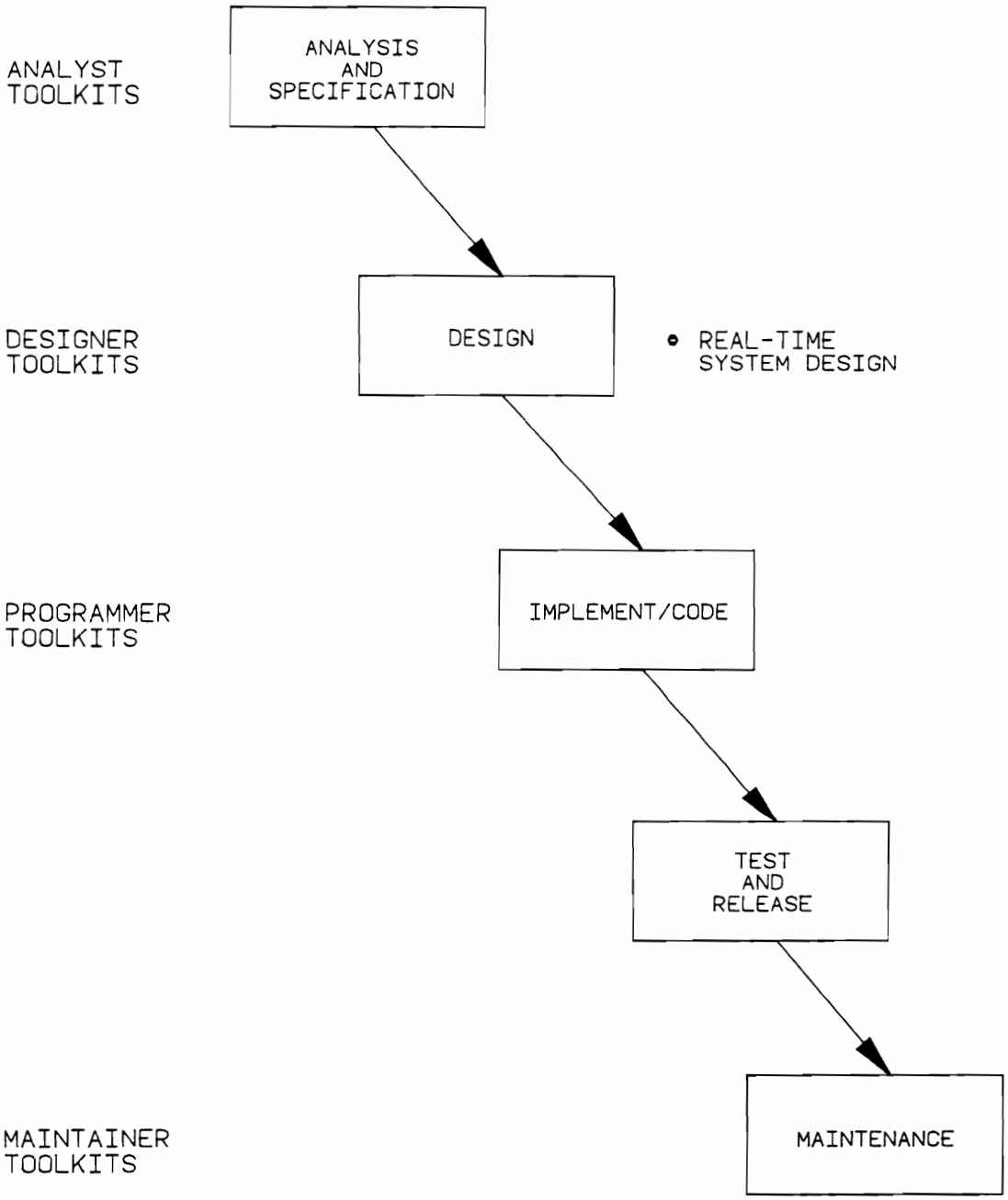


Figure 4. Basic Toolkits of the Software Development Life Cycle

commercially. Many companies currently market CASE tools to support this phase of software development.

Design toolkits support the creation of hierarchical tree structured diagrams and procedural logic diagrams. They also provide tools and diagrams for the logical and physical design of databases and files as well as logical data modeling.

The programming phase of the software development life cycle supports many tools that assist in the actual coding of the designed system. The most important tool in this toolkit is a code generator. The code generator provides automatic generation of source code from program design diagrams and information. This automation increases the reliability of the produced code and reduces the amount of development time required. Integrated CASE tools in this area were the last to be commercially available and are still at a relatively low level.

The testing phase of the software development life cycle can be assisted by CASE tools at all phases. Specification checkers at the analysis and design phases help to catch problems before they are generated. Compilers and interpreters with interactive debugging facilities find problems at the coding level. Additionally, CASE tools are usually available which monitor performance, prepare test data and generate audit trails of all system documents, such as specifications, program code and test cases.

Maintenance tools currently dominate the CASE tool marketplace. These toolkits typically provide tools which correct, restructure, replace or enhance existing systems systematically. They make it possible to keep track of large applications through their entire life. These toolkits are in widespread use in almost every type of organization and are required by most government agencies. Another tool which is part of the mainte-

nance toolkit is reverse engineering. These tools allow an existing program to be translated back into its design specification, including logical data models. Changes can now be made at the design level and the code automatically regenerated. Sophisticated tools of this type will revolutionize the CASE tool market but are currently not available except at a relatively low level.

A CASE ***framework*** or ***integrated project support environment (IPSE)*** can be thought of as a special type of toolkit which serves as an infrastructure for integrating individual CASE tools together [Brow88]. The framework provides a method for joining old or new tools as well as those distributed by different vendors. Frameworks record structured data using a database and control the sharing of data between toolkits. They often provide a common user interface as well as tools to assist with monitoring of tool tasks and performance. A number of frameworks are currently available and are beginning to be adopted by CASE tool users. One drawback of a framework is the large amount of specialized customization which must be done for every installation. This type of customization requires a great deal of expertise not only in knowledge of the framework but also of the CASE tools themselves.

A CASE workbench is a collection of CASE toolkits which support every basic phase of the software development life cycle and also includes some software management capabilities. Each toolkit is integrated into the workbench in such a way that all toolkits can easily and automatically share information with the next life cycle phase. This is often accomplished through the use of a framework. CASE workbenches can be general in nature or specifically designed to support one particular type of software development. A CASE workbench also has its own information repository for the storage of all workbench diagrams and information. The future of CASE use and development will be largely centered around integrated CASE workbenches.

A CASE methodology companion is a toolkit or workbench specifically designed to support the steps or rules of a particular structured methodology. This assistance is incorporated into the CASE tool by means of help panels, menu choices and automated specification and code checking. The use of a methodology companion reinforces the use of structured techniques as well as limits the amount of information a developer must enter. Methodology companions are widely available commercially and are often incorporated into CASE workbenches.

2.4 *Standards*

Standards have the potential for indirectly aiding integration, especially across multiple platforms. A number of authors discuss the benefits of the PHIGS standard in creating custom CAD application programs [That88, Wamp88]. Extensions to PHIGS philosophy specifically designed for CAD/CAM are described in [Jaya90]. The importance of standards to CAD/CAM integration of all types cannot be understated. A common remark which was repeatedly made by CAD/CAM integration survey respondents was that adherence to standards of any type, where they were available, was vital to overall success of any application integration.

Another important de facto standard is the X Window System standard for developing graphical user interfaces. The X Window System is a network-based application which is important because it allows portability of applications across multiple hardware and software environments via a network protocol. The X Window System will be further described in Chapter 7.

2.5 Summary

A literature review of integration methods as well as an overview of computer aided software engineering has been presented in this chapter. The next chapter will present a detailed synopsis of the responses to the CAD/CAM integration survey conducted in the winter and spring of 1990. A general description of the companies who participated as well as a brief summary section are included.

3.0 Synopsis of Industry Survey on CAD/CAM Integration

In an effort to ascertain how major companies are solving their own CAD/CAM integration problems and to assist in the identification of data and other entities which need to be shared between CAD/CAM software applications and processes, a survey of industrial CAD/CAM users was conducted. The intent of this survey was to gain a better understanding of the problems which actually exist, so that new solutions could be sought. In February through May of 1990, the survey which appears in Appendix A of this dissertation was mailed to a pre-selected group of company representatives within 41 Fortune 500 companies and government agencies, with more than one division of a company receiving the survey in some cases. These company representatives, who all had direct experience in CAD/CAM applications software, were questioned about:

- the type of CAD and CAM software systems they are currently using including custom codes,

- the level of integration these systems possess,

- the type of integration utilized,

- the type of information which is shared between systems,

- the type of information which needs to be shared to achieve good integration between systems, and

- a variety of other related questions.

While a number of companies declined to participate in the survey for proprietary and other reasons, 26 companies did participate by giving detailed responses to survey questions. To obtain and clarify survey responses, several personal interviews as well as numerous telephone interviews were performed. The cooperation of those persons who did agree to participate was of great help and enabled a good understanding of relevant problems in CAD/CAM integration. While disclosure of the companies names cannot be included, they represent a large number of industries that manufacture a wide range of products including: agricultural equipment, aircraft, aircraft engines, automotive engines, avionics, chemicals and allied products, electronic systems, HVAC systems, major appliances, metal and composite components, microelectronics, power systems, power tools, propulsion, shipbuilding, space systems, steel and related products, tactical missiles, telecommunications and tire and rubber products. The total number of employees for the companies which participated on a nationwide and in some cases worldwide basis can be categorized as follows:

100,000 to 300,000 employees - 32% of the companies surveyed
25,000 to 100,000 employees - 50% of the companies surveyed
15,000 to 25,000 employees - 18% of the companies surveyed

The majority of the companies who were surveyed ranked in the top 30% of the Fortune 500 (where applicable) and breakdown as follows:

Top 10% of Fortune 500 companies - 50%

Top 20% - additional 28%

Top 30% - additional 16%

Top 40% - additional 6%

Several government agencies were also included among the survey respondents.

The survey was concluded in June of 1990 and the responses which were obtained confirmed many preconceived beliefs about existing problems but also presented quite surprising information in many areas. The next section lists each survey question and a synopsis of the responses given. Due to the nature of the responses received, responses to some questions are presented in a much more general manner than others. It is also important to keep in mind that the level of detail reported by survey respondents varied widely and could affect detailed results in some cases. An effort has been made to include as many pertinent comments from the written surveys, personal interviews and phone interviews as possible.

3.1 Survey Results

In this section, responses to survey questions will be reported and discussed question by question. Responses are, for the most part, listed beginning with the most frequent re-

sponse first. All percentages given represent the percentage of the total number of surveys received.

1. **What type of processes or tasks are performed on CAD/CAM systems within your organization? (e.g. design, analysis, manufacturing, etc.)**

   The companies which participated in the survey were predominantly involved with the design of mechanical systems but also included several who deal primarily in the design of electrical systems. Most companies listed between three and five major tasks which are performed by CAD/CAM systems within their organizations. The list below gives the most common responses along with a percentage of the total companies which listed each response. As can be seen from the list, the most common task performed is mechanical design which was listed by 88.5% of the companies surveyed. This was followed closely by mechanical analysis at 84.6%. The rest of the list is ordered with the most frequent responses at the top of the list decreasing down to the least frequent.

   Mechanical design - 88.5%

   Mechanical analysis - 84.6%

      The following is a breakdown of different types of analyses listed. Once again, the percentage of the total companies which included each response is given.

      Structural - 73.1%

      Thermal - 26.9%

      Piping - 7.7%

      Kinematics - 11.5%

      Fluid - 11.5%

      Heat transfer - 7.7%

Others - Hydrodynamics, flutter, HVAC, mathematical, orbit design, interference detection

Drafting - 53.8%

NC manufacturing support - (may include part programming, processing, verification and/or prototyping) - 46.2%

Electrical design - 34.6%

Plant and facilities layout - 26.9%

Manufacturing (No details given) - 19.2%

Conceptual design - 19.2%

Electrical circuit analysis - 15.4%

CMM - 11.5%

Quality assurance - 11.5%

Tolerancing - 11.5%

Tool design - 11.5%

Inspection - 7.7%

Process planning - 7.7%

Reliability and maintainability - 7.7%

Others listed only once included: Documentation of computer installations within the company, EC markup, geometric dimensioning, project control, trade studies

2. What type and make of hardware are you running these systems and applications on? (e.g. mainframe, workstation, etc., and manufacturer)

Survey respondents can be categorized into general hardware groups based on the combinations of equipment they utilize. As shown by the following list, the most common group is that with Mainframes, minicomputers and workstations.

Utilize Mainframes, minicomputers and workstations - 30.8%

Utilize Mainframes and workstations - 15.4%

Utilize Minicomputers, workstations and PC's - 15.4%

Utilize Mainframes, workstations and PC's - 15.4%

Utilize Mainframes, minicomputers, workstations and PC's - 7.7%

Utilize Workstations and PC's - 7.7%

Utilize Mainframes only - 3.8%

Utilize PC's only - 3.8%

Hardware by Type and Manufacturer

The following tabulation gives the Manufacturer's name and hardware type (where specified). The percentage listed represents the percentage of the total number of companies who have at least one piece of equipment of the type listed.

Mainframe Computers

- International Business Machines Corp. - 50%

  3090 - 26.9%

  Unspecified - 23.1%

- Digital Equipment Corp. - 23.1%

  VAX/8800 - 15.4%

  VAX/8650 - 3.8%

  Unspecified - 3.8%

- Amdahl - 11.5%

  5870 - 3.8%

  Unspecified - 3.8%

- Cray - 11.5%

  XMP - 3.8%

  Unspecified - 7.7%

- Control Data Corp. - 7.7%

Unspecified - 7.7%

- Others - 11.5% (includes Convex, Honeywell, Hitachi)

Minicomputers

- Digital Equipment Corp. - 38.5%

  VAX 6000 Series - 7.7%

  VAX 11/780 - 3.8%

  VAX 11/785 - 3.8%

  Unspecified - 23.1%

- Computervision - 11.5%
- Prime Computer - 7.7%
- Intergraph - 7.7%
- International Business Machines Corp. - 3.8%

  4381 - 3.8%

- Wang Computer - 3.8%

Workstations

- Digital Equipment Corp. - 57.7%

  DEC 3100 - 19.2%

  MicroVax - 15.4%

  VaxStation - 7.7%

  Unspecified - 15.4%

- Apollo - 50%

  10000 - 3.8%

  Unspecified- 46.2%

- Silicon Graphics - 34.6%

  Personal Iris - 3.8%

  Unspecified - 30.8%

- SUN - 34.6%
- Hewlett Packard - 19.2%

  HP 900 - 7.7%

  HP 345 - 3.8%

  Unspecified - 7.7%
- IBM - 15.4%

  RT - 11.5%

  RS/6000 - 3.8%

  RS/6000(plan to consider) - 3.8%
- Tektronix - 7.7%

  XD/88 - 3.8%

  Unspecified - 3.8%
- Unspecified - 7.7%
- Others - 3.8% (includes Evans and Sutherland)

PC's

- IBM - 30.8%

  PC's - 15.4%

  PS/2 - 15.4%
- Apple MacIntosh - 19.2%
- Compaq - 7.7%
- Unspec. - 15.4%
- Others - 15.4% (includes Tandy, AST(286), Zenith, AT & T)

3. **List the software systems and/or applications programs you are currently using (i.e. CADAM, IDEAS, in-house programs, etc.). Identify those applications which are commercially available and those which were custom written. If large numbers of applications programs are used, please give general information**

such as the number of programs, all the different types of applications these represent and whether these are commercially available or custom written or a combination of both. (e.g. 5 analysis applications - 2 commercial, 3 custom, etc.)

The majority of the companies surveyed use a combination of commercially available and custom written software (88.5%). 11.5% reported using only commercial software.

The following list enumerates the commercial software in use by survey respondents. The percentages given below indicate the total number of companies surveyed which utilize each software system. Please keep in mind that this is not a measure of the number of seats available. Responses are listed with the most frequent appearing first.

COMMERCIAL SOFTWARE

Mechanical CAD

CADAM - 38.5%

CATIA - 34.6%

Computervision CADDS 4X - 26.9%

AUTOCAD - 15.4%

MCS ANVIL 5000 - 15.4%

UNIGRAPHICS - 7.7%

PGDS - 7.7%

Others listed once each - EUCLID, PRIME MEDUSA, VERSACAD, MICROSTATION CAD, CALMA-DDM, ARIES, P-CAD, INTERGRAPH, INFINITE GRAPHICS IGI-2100, GDQF)

Electrical CAD

MENTOR - 11.5%

Vantage VHDL - 3.8%

VALYSIS - 3.8%

Synopsis Logic Generator - 3.8%

Cadence Standard Cell Layout - 3.8%

Computer Aided Manufacturing

APT360 - 7.7%

AUTOCAM - 3.8%

EXCELLON CAM - 3.8%

Computer Aided Engineering Analysis

PATRAN - 53.8%

I-DEAS (CAEDS) - 50%

NASTRAN - 46.2%

ANSYS - 23.1%

SINDA - 23.1%

ADAMS - 15.4%

TRASYS - 11.5%

ABACUS - 7.7%

MARC - 3.8%

NISA - 3.8%

Others listed only once each - COSMOS-M, RAND-MICAS, MOLDFLOW, MOLDTEMP, FLOW, DAISY, EASY-5W, NEVADA, AUTOKON, TRIFLEX, CAESAR, ISOGEN, ALIAS, NASA Analysis Codes

IN-HOUSE SOFTWARE

The following list enumerates the customized software reported by survey respondents.

Design

surface lofting - 3.8%

geometric definition - 3.8%

animation - 3.8%

preliminary design - 3.8%

markup - 3.8%

tire and mold design - 3.8%

Analysis

structural analysis - 38.5%

thermal, fluid, hydrodynamic analysis - 19.2%

dynamic analysis - 15.4%

composite design - 7.7%

Others listed only once each - piping, rotor analysis, orbit design, sizing codes, power train performance

Manufacturing

quality assurance and control - 23.1%

NC processing and verification - 19.2%

plant layout - 11.5%

reliability and maintainability - 11.5%

tooling Design - 7.7%

CAPP - 7.7%

pipe cutting and bending - 7.7%

time standards - 7.7%

Others listed only once - stamping, formability

4. To what extent are the functions of CAD integrated in your organization? (e.g. design, analysis, etc.)

This question was intended not only to determine what vehicles were being utilized for sharing information between dissimilar applications but also served to ascertain what different companies considered "integration" to mean. To most, integration simply meant getting the data from one application to the next, although these were not always completely successful transfers. To others, integration implies the common storage of all shared information (common database). It is important to note that the majority of companies who felt that they were either completely integrated or well integrated utilized a common database to some extent. In fact, two companies are customizing their present systems to incorporate and utilize a common database solely for integration reasons. The following list provides the type of integration listed and the percentage of total companies which utilized each method:

Data passing through interfaces or translators - 46.2%

Completely integrated through common database - 3.8%

Some codes well integrated while others have little integration - 19.2%

Little integration - 19.2%

NO integration - 3.8%

Currently transforming their codes to facilitate integration through the use of a common database - 7.7%

A suggestion was given by one respondent that the term integration should have been clearly defined for survey participants. The possible definitions they suggested were as follows:

"Several interpretations (of integration) could be:

1. Data shared between systems through manual transcription
2. Data shared ("passed") between systems through software interfaces
3. A single source of data shared by all applications
4. Shared data being configuration managed by a single database manager

Note: We do not feel No. 1 and No. 2 are integration."

While it is appears that most companies surveyed do not necessarily share this opinion, the most successful cases of integration reported in the survey tend to support this belief.

Please answer the following questions (a-e) with reference to CAD integration in your organization:

a. **Do you have an automatic way to pass information between one software system and the next?**

   The majority of the companies surveyed said yes to this question and also listed more than one method of passing information. The following list gives the most common responses with the percentage of total companies who utilize them.

   Proprietary interfaces - 53.8%

   IGES - 50%

   Common database - 19.2%

   EDIF - 7.7%

b. **Are there any significant problems associated with this method?**

   The majority (65.4%) of the companies surveyed said yes, they had experienced significant problems utilizing the methods discussed in the previous question. Some of the reasons given for this were as follows:

   Limitations of the standards make use difficult
   Inability to pass certain types of information does not provide satisfactory results

   Types of information which can be passed are limited

   Speed of transfer is often slow and cumbersome

   Different databases often make transfer impossible

   Accuracy problems are quite common

   Lack of consistency from one method to the next

26.9% of the respondents answered No to this question and said that the current methods of transfer worked quite satisfactorily for their present needs.

Other responses given:

Not applicable - 7.7%

c. What type of information are you sharing?

Most companies listed several answers, these included:

Geometry and drawing information - 88.6%

Attribute information - 30.8%

Textual information - 19.2%

Analysis results - 19.2%

NC data - 15.4%

Documentation - 7.7%

Other responses listed only once included: Parts list, bill of materials, parametric and performance data

d. If no automatic or semi-automatic method of information sharing is used, how do you get the information from one software system to another?

Surprisingly, manual recapture or reentry of data and in some cases recreation of geometry was the most common answer given (65.4%). Many problems were discussed in relation to this response. Reentry of data and models is not only expensive but introduces accuracy problems as well as the problem of creating a different model in the analysis program than was originally created in the design program. Other answers listed included:

Magnetic tape, specialized custom code - 11.5%

Not applicable - 23.1%

e. What type of information are you currently unable to pass between software systems that you would like to be able to share?

Most companies listed more than one type of information that they would like to be able to share but are unable to share at present. These included:

Surface models - 23.1%

Complex 3-D geometry (e.g. splines) - 19.2%

Solid model data - 11.5%

Other responses listed only once or twice each included: parts information, analysis results, reliability and maintainability data, textual data, image data, dimensions and annotations, FEM mesh information

Others answers given included:

None - 19.2%

Not applicable - 7.7%

5. **To what extent are the functions of CAD integrated with those of CAM? (How does CAD final design data get transmitted to the manufacturing floor?)**

The answers to this question were in some cases vague, such as "good integration" or "little integration," and are presented here in the same general manner. The results of the survey indicated that half of the respondents reported some integration of manufacturing operations with CAD functions.

Good integration (often through common database) - 26.9%

Are not integrated - 26.9%

More than one manufacturing operation integrated - 23.1%

Not applicable- 7.7%

Minimal integration - 15.4%

A very successful method reported by at least two companies was an integration within individual product lines rather than a complete integration of all systems.

6. **Have you used the IGES standard? If yes, what has been your experience using IGES? What type of information are you using IGES to pass?**

The majority of companies surveyed were acquainted with IGES as shown below:

YES, we have used IGES - 76.9%

NO, we have not used IGES - 23.1%

Based on survey responses, IGES is being used both between different CAD/CAM applications programs as well as between dissimilar systems often located outside the company or at other company sites.

Many companies shared some of their experiences using IGES which included the following:

Advantages

Does fine with simple 2-D drawings and some wireframe models

Available commercially

Disadvantages

Only successful through the elimination of splines, surfaces, part list data

Lacking in completeness

Only passes a fraction of the necessary information

Has problems passing fonts

Dimensions cause problems

Size of file consumes too much space

Not a viable solution to real problems

The most common types of information being passed by IGES according to survey responses were model data, drawings, 3-D wireframe and FEM grids.

7. Are you counting on future releases of IGES or the release of PDES to solve any of your present software integration problems?

53.8% of the companies surveyed said yes, they were counting on PDES to solve the problems they are currently only working around. A comment made by one of these respondents was that PDES is the "Only game in town."

The remaining 46.2% stated that they were not counting on the future PDES standard and gave a variety of reasons for this including the following:

- Too uncertain
- Too long a time until PDES is released
- Transition between IGES and PDES will be a long and difficult one
- Hope it will provide better capabilities and solutions but are taking a "wait and see" attitude

A number of survey respondents were members of the PDES committee. While these respondents were optimistic in all cases, some of them said that they were not counting on PDES to solve their present integration problems because of its uncertain development time.

8. How many interfaces or translators do you use which are not IGES format?

3 or more interfaces or translators- 42.3%

2 interfaces or translators - 19.2%

1 interfaces or translators- 19.2%

Not applicable - 19.2%

Most of the interfaces and translators which were named were of a commercial or vendor supplied nature. A number of the survey respondents did, however, mention that they had developed their own custom interfaces to enable the sharing of data between some applications programs. As can be seen from these results, the majority of survey respondents are utilizing quite a number of different interfaces and translators to accomplish integration.

9. Have software integration problems affected the productivity of your CAD/CAM software systems? If yes, please give an example.

YES, integration problems have affected productivity - 69.2%

NO, integration problems have not significantly affected productivity - 26.9%

No answer - 3.8%

The majority of companies responded yes to this question and a number of these replied "of course." One respondent did add, however, that while productivity was affected, engineers continually learn to work around integration problems and find new ways to solve them.

One company felt a more serious detriment to productivity was the transition or training of engineers for new or changing software, which requires a massive effort and a major investment. This problem was also mentioned by another company.

10. If you are currently using custom software systems and/or applications programs, did you develop them in-house or have them developed outside your organization?

Developed in-house - 73.1%

Developed both in-house and by outside organizations - 19.2%

No custom development - 11.5%

This was a rather unexpected response which seems to indicate that a large number of companies are in the software development business to some extent.

11. What language are these custom applications written in? Please list the language and the percent of custom applications which apply. (e.g. Fortran - 80%)

FORTRAN

69.2%, most common language

11.5%, second most common language

C

3.8%, most common language

23.1%, second most common language

11.5%, third most common language

23.1%, indicated that most future development will be in C because of their migration to workstations

COBOL

7.7%, second most common language

PL1

3.8%, most common language

7.7%, third most common language

PASCAL

3.8%, most common language

3.8%, third most common language

BASIC

7.7% listed

APL

3.8% listed

12. Do you have access to the source code for these custom applications or do you receive only the executable? Please give (or estimate) the total number of custom applications which you use and approximate the percentage of these for which you have source code.

The response to this question was a rather surprising one in that the majority of companies do have access to most of the source code for their custom applications. The breakdown is as follows:

Have 100% of the source code - 23.1%

Have 95% of the source code - 15.4%

Have 90% of the source code - 11.5%

Have a small % of the source code - 3.8%

Have source code but % not given - 15.4%

Have only the executable code, no source - 15.4%

Other answers given:

No answer - 3.8%

Not applicable - 11.5%

13. Do these custom applications use interactive computer graphics? Please approximate the percentage of the total number of applications given above in question #12 which utilize interactive computer graphics.

As can be seen from the following responses, only a small percentage of the companies surveyed currently include interactive computer graphics in a large number of the custom written programs they use. Many companies, however, stated that adding graphics to these programs would be a priority. It was felt by some respondents that the lack of standards for graphics (such as PHIGS) at the time of development was the primary reason for the lack of graphics in these programs. The following list gives the percentage of programs within each company that currently contain interactive computer graphics (ICG) within the companies surveyed.

95-100% of our custom codes contain ICG -15.4%

70-80% of our custom codes contain ICG - 11.5%

50-60% of our custom codes contain ICG - 11.5%

30% of our custom codes contain ICG - 7.7%

10-15% of our custom codes contain ICG - 11.5%

A very small % of our custom codes contain ICG - 19.2%

None of our custom codes contain ICG - 11.5%

Not applicable - 11.5%

14. Do these custom software systems utilize either the GKS or PHIGS standard? Please approximate the percentage of the total number of applications given above in question #12 which utilize either GKS or PHIGS.

The responses to this question indicate that the GKS and PHIGS standards are not widely used at present. The next question, however, indicates a commitment by many companies to incorporate the PHIGS standard into their future, and in many cases, present codes.

None include these standards - 61.5%

5% or less include them - 15.4%

10% include them - 3.8%

20% include them - 3.8%

No answer - 3.8%

Not applicable - 11.5%

15. If you are currently considering the development of a custom software system of some sort, will you utilize the graphics standards GKS or PHIGS? Why or why not?

46.2% of the companies surveyed said they would be using PHIGS or PHIGS+ in their future software development. Most felt this was important since PHIGS was now a standard and was being readily accepted in major industries as well. Use of a standard is vital to most companies since it provides more flexibility and portability. One company indicated that they would begin to use PHIGS because it had been specified in one of their new government contracts. Another company stated that they would implement PHIGS to insure support on future hardware platforms. In contrast, 30.8% of the companies surveyed said they would not use PHIGS or GKS in future development. Some reasons given for this were as follows:

- The cost of software development is too expensive
- They plan to continue utilizing internal graphics libraries
- Have adopted a new emphasis on acquisition of software rather that development of it

One company said that they would only use PHIGS if the performance was acceptable, especially in the area of animation.

Other responses given were:

Not applicable - 11.5%

No answer - 7.7%

16. If there are any additional comments you wish to make regarding the integration of CAD/CAM applications within your company, please include them here.

57.7% of the survey respondents gave no comments. The remaining 42.3% of respondents had one or several comments, many of which have been included here. Comments about specific applications and problems which may disclose company identity have been omitted. Most of the comments dealt directly with integration but also covered a broad spectrum of other topics. The following is a partial listing of some of the comments which were made.

Integration

- "I think integration will be our thrust in the next 5+ years"
- "Trying to integrate CAD/CAM applications contributes to the high cost of CAD/CAM systems"
- "Integration is the key to competition"
- "The better CAD/CAM integration, the less time is wasted"
- "We have ***islands of integration***, which solves some problems and creates others"
- "Budget restrictions based on contracts make planning for long term integration (especially positions) impossible. This forces the use of vendor supplied products which link together software systems and force strict adherence to standards which cannot solve all problems"

- "Integration of the design, simulation, testing, and manufacturing methods for our products is the key to improving quality, productivity, and employee satisfaction"
- "We want to put as little of our own manpower as possible into creating custom integration software"
- "Redundant entry of data (which cannot be shared between programs) is a recurrent problem which introduces serious errors"
- "A significant need in this area is an integration architecture for all types of engineering data"

Other

- The transfer of product model information electronically between locations is often a requirement which must be met and should be addressed
- CAD system terminology presents a continual problem since the vocabulary understood by engineering is often different from that utilized by manufacturing. More research should be done in this area.
- More involvement of technical people rather than marketing staff in the vendor presentation of systems and applications would be a great advantage. Often vendor systems are not selected because they cannot be benchmarked successfully by marketing personnel.
- While computer accounting is easy to manage on mainframes, it is not as easy to manage on PC's or workstations. This always presents a problem for projects which must be billed for computer usage.
- There is a real need for database managers and repositories for workstation environments
- "Concurrent engineering is vital to competitiveness"

- The cost of training is prohibitive especially when software systems change. One company will spend $7,000,000 over a two year period to train approximately 900 seats in new applications usage.

3.2 Summary

From the results of the integration survey and especially the comments made by survey respondents it is easy to see that CAD/CAM integration is of primary concern to many companies at the present time. One company summarized many of the other respondents feelings when he said that integration was "key to competition." The majority of survey respondents connect their different applications programs through the use of interfaces and translators which assist in data passing. This type of transfer is not always successful, especially when it involves the use of standards such as IGES which do not allow for the transfer of complex surfaces and solids. A small number of respondents are trying to utilize a common database and a database management system for all their applications to achieve a more complete integration. This has been reported to be very successful in at least two cases. The predominant type of information being shared between software systems is geometry and drawing information. Other types of information mentioned include attribute information, analysis results, NC data and textual information. The majority of respondents feel that the lack of good integration in their CAD/CAM systems has cost them a loss of productivity, time and money.

The survey results also show that, of the companies involved, the majority utilize a combination of mainframes, minicomputers and workstations. The software that runs on these systems is, for the most part, a combination of commercial software systems

and custom written codes which address many different areas especially that of engineering analysis. Rather surprisingly, most companies which utilize custom codes develop them in-house and maintain the source code themselves. The majority of these codes are written in Fortran but there seems to be an upcoming increase in the use of the C language brought on by the advent of workstation use. While large numbers of custom codes do not usually include interactive computer graphics at present, it appears that the PHIGS standard will be widely used for the development of future custom software. The major advantages of utilizing the PHIGS standard which were reported by the survey are increased flexibility and portability of developed code. The fact that PHIGS is also supported by most state-of-the-art hardware systems was an important factor as well. Many interesting comments were given by survey respondents which helped to emphasize existing integration problems and bring other types of problems to light.

The next chapter begins with a discussion of the most common methods of CAD/CAM integration currently in use. An application of CASE tools, which assists and partially automates some of the necessary integration tasks, is then presented.

4.0 A CASE Approach to the Generation of CAD/CAM Integration Systems

4.1 Introduction

This chapter presents, in detail, the application of CASE methods to the generation of ***integration systems*** for CAD/CAM application programs. The integration system, or more explicitly the integration software, is the code which actually enables the integration of different CAD/CAM application programs. This may include the storage and retrieval of information to and from a common database or interface file, data conversions of many types and control over the entry points of the original CAD/CAM applications. An example integration system is presented in the last section of this chapter. A model of the integration system is shown in Figure 5 on page 58.

In this research, a semi-automated method of producing this integration system is presented which involves the use of computer aided software engineering (CASE) concepts.

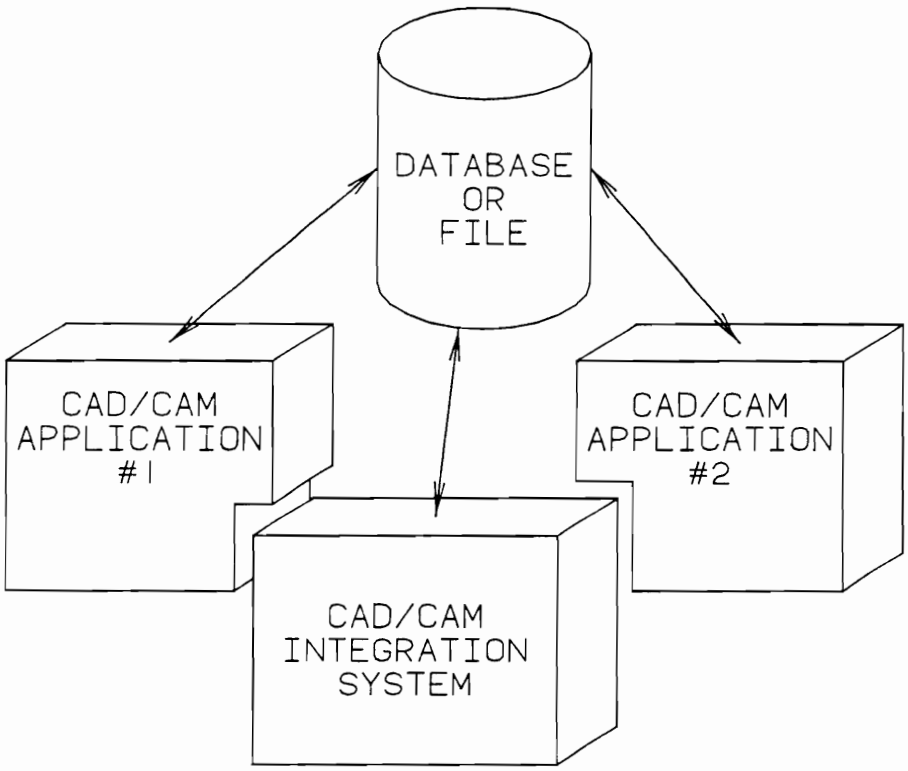


Figure 5. Integration System Model

As discussed in the literature review, CASE provides many ways to more quickly and accurately produce any type of program code.

A CASE workbench is described which is especially equipped to support the creation of these integration systems for CAD/CAM. This CAD/CAM CASE workbench includes toolkits which support the inquiry and extraction of vital information from the programs to be integrated, an interface that prepares and directs this information for use by other tools, CASE design and analysis tools and automatic code generation. A paradigm representing this new method of generating integration systems is shown in Figure 6 on page 60.

To better understand the problems of connecting different CAD/CAM applications, a description of current methods of joining these applications will now be given so that the benefits of CASE application to these problems can be understood.

4.2 Current Methods of Joining CAD/CAM Applications

It is difficult to identify exactly what the term integration refers to when considering the integration of two separate CAD/CAM application programs. As mentioned in the CAD/CAM integration survey results, there does not seem to be common agreement on the exact interpretation of this term and survey responses ranged from extremely low-level methods to very high-level methods. To some CAD/CAM users, integration simply means the sharing of information (data) between one application and the next, even if that means manual transcription of the data from one application to another. While this is, thankfully, not the most common method, some widely used methods re-

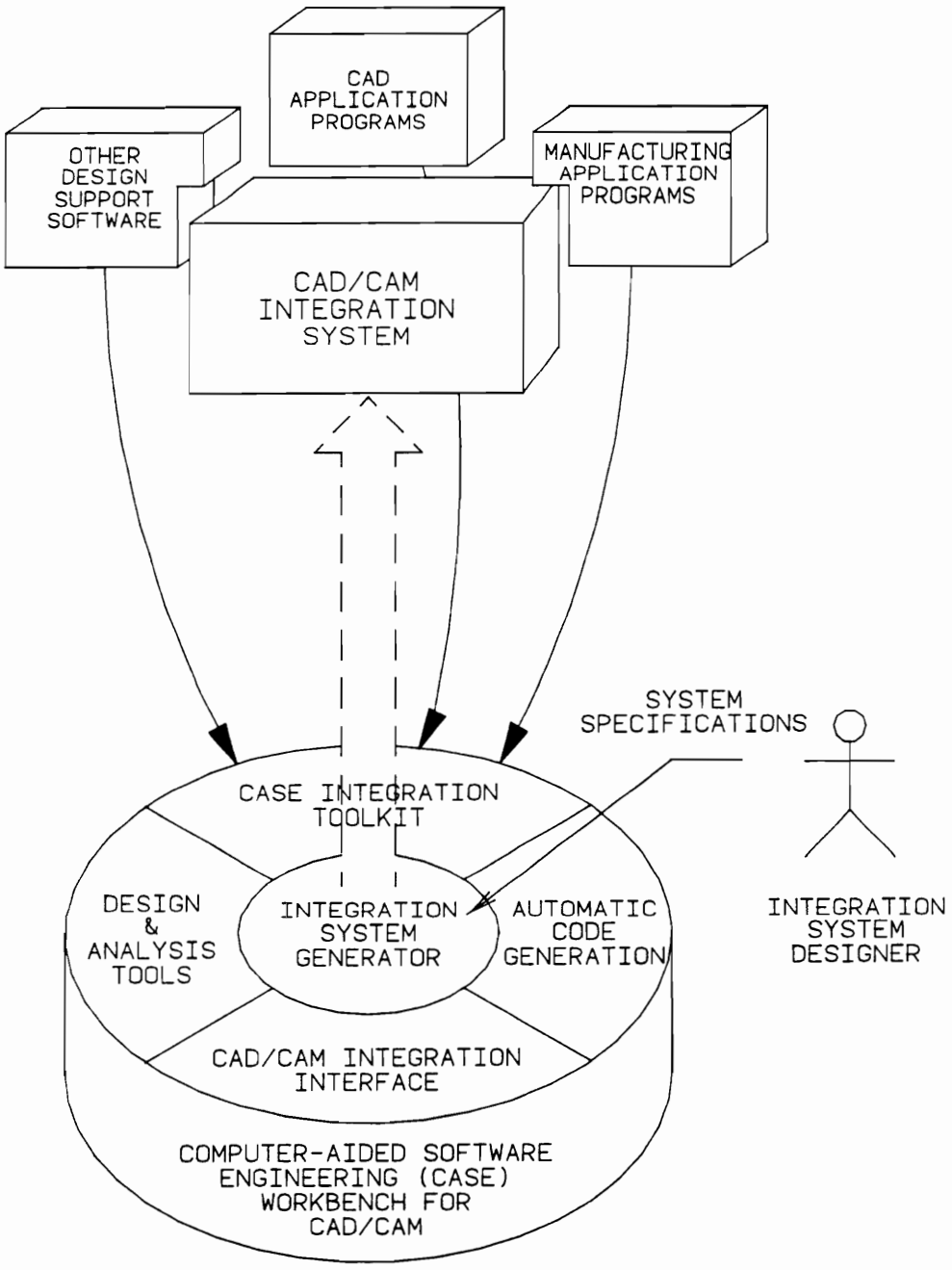


Figure 6. CAD/CAM Integration System Paradigm

quire almost as much effort to implement and use. There are basically two commonly used methods of joining separate CAD/CAM application programs in current use. These are interfacing, or data "passing", and "coupling" or database methods. Probably the most common method currently utilized is interfacing, or "passing" of data. This may be due in some part to lack of knowledge of database methods by CAD/CAM applications programmers. While interfacing requires little or no change in the original applications code, it has great limitations and is difficult to implement. Its greatest advantage is that it requires little knowledge other than that of the programs themselves and some type of programming language. The method of linking two separate CAD/CAM applications which appears to be the most accepted example of true integration is that which involves the use of a central database and a database manager. Among the respondents of the CAD/CAM integration survey, those who reported the most successful and flexible cases of CAD/CAM application joining were using a common database method of some type.

A number of interfacing and coupling (database) techniques can be used to join together two different CAD/CAM application programs. These methods of can be very generally grouped into two categories for ease of discussion: rigidly-connected and freely-connected. Rigidly-connected applications are those which are coupled or interfaced in a method which is specific to the applications being joined and is not easily extended to incorporate subsequent additions or changes. Freely-connected applications are those which are coupled or interfaced in a way which is more generic in application and which has a large number of similarities between the joining of one set of programs and another.

This general categorization helps to identify the four most common ways in which CAD/CAM applications are linked, these are:

- Rigidly-Connected Interfacing
- Rigidly-Connected Coupling
- Freely-Connected Interfacing
- Freely-Connected Coupling

Each of these methods will now be investigated in more detail. It is important to note that most of the examples given here often present a scenario in which two CAD/CAM programs are being considered. While this is an easy way to present and discuss each type of joining, it is not meant to imply that each method will handle only two programs. In most cases, the number of programs to be joined is not a limitation, but it must be mentioned that each additional program considered greatly increases the complexity of the problem.

4.2.1 Rigidly-Connected Interfacing

Rigidly-connected interfacing was the first technique that was widely used to join different CAD/CAM applications. For a typical pair of applications, program one is modified to produce an output file which is identical to the input file expected by program two, this file is then read directly into program two as input. While this allows the results of program one to be shared with program two, it also requires customized code to be written for each program which cannot be reused for the subsequent joining of additional programs. This includes not only data and format specifications, but any necessary data conversions or geometry transformations. It is easy to see that this is a

time consuming and tedious task which does not allow easy addition or modification. It does provide the advantage that, if necessary, this change of the output to an input file specification can be done completely external to the application programs, which is a useful option if source code is not readily available.

4.2.2 Rigidly-Connected Coupling

At the opposite end of the spectrum, is rigidly-connected coupling. This is what the "utopian state" of CAD/CAM probably looks like in some people's minds. In this scenario, two applications utilize the same common data and ideally appear as one large application in which the original applications are accessed as functions. This makes the fact that these programs were once separate entities relatively transparent to the user. A common user interface is usually a desired part of this method. This now presents the user with one program to learn which has fast and efficient operation and just one user interface. The main disadvantage of this method is the obvious complexity of combining the code from multiple applications into one program. Considering the lack of any standards for program construction, database use, data models, etc., it may prove easier to develop a new program than to join different application programs in this manner.

4.2.3 Freely-Connected Interfacing

Because rigidly-coupled interfacing creates the need for so many specialized files to be written between different application programs, the idea of a standard or universal file that could be used for communication with any kind of program became very popular. It is, probably, the most widely used method at present to share information between

dissimilar programs. In this technique, each program requires the capability to translate from a universal file format into its own format, and from its own format into a universal file format. This means that instead of two specialized translations for every program to be interfaced, we have only two translations per program for any number of programs to be interfaced. To facilitate a common file format of this type the ANSI standards committee adopted the IGES (Initial Graphics Exchange Specification) standard in 1981 [Brad83]. IGES is a document standard that is now widely available with most major commercial CAD/CAM systems. It is also used profusely in custom code interfacing as well. While IGES provided a capability not previously available, it has been criticized for its lack of support of many features, especially complex surfaces and solids. To combat some of the problems of IGES, the IGES committee is now working on a new standard called PDES (Product Data Exchange Specification). PDES will address many of the entities that IGES does not. The major problem with PDES, however, is that it will probably not be ready until the mid to late 1990's. An additional problem with the use of a universal file format is that since data translation occurs twice (once upon entry and once upon exit of the IGES file) for every interface, many errors and inaccuracies can be introduced. When accurate data representations are critical, this is a substantial problem.

4.2.4 Freely-Connected Coupling

The last category of joining different CAD/CAM programs that is considered here, can be referred to as freely-connected coupling. This technique also involves the use of a common database for the sharing of data. In this method, however, a database management system is used to communicate the data to all application programs. This is a

good way to join programs that may have been developed independently but now need to share information between them. As can be imagined, the big task here is to modify the applications themselves to communicate with the central database.

4.3 Conclusions Based on Current Methods of Joining Applications

It is evident from the responses to the CAD/CAM integration survey that no simple integration strategy is suitable for all situations. The method of joining different CAD/CAM applications must be selected based on the end use as well as many other factors. For this reason, it is easy to see that several levels of joining applications must be considered by this research for practical use and application, even though some of these methods are tedious and difficult to generate.

It is also of importance to note that survey results and personal investigation point to the fact that database methods are reported to be a more successful way of joining CAD/CAM applications than interfacing. The main reason for this is that data translation (i.e. conversion into some standardized format and subsequent conversion back to the original) can cause substantial problems. Database coupling is more efficient and simplifies the joining of multiple programs. The disadvantage with database communications is that it can be an enormous task to modify the application programs to communicate with a common database unless tools are provided to assist with identifying and generating code for this type of solution.

Finally, remember that CASE methods help to design, generate and maintain efficient code regardless of the end use and are designed to assist in the automation of complex and tedious tasks. They also provide a way to reuse previously designed code thereby reducing development effort. What seems to be a natural justification for the application of CASE methods to the generation of integration systems for CAD/CAM can now be seen. While CASE methods have not been used in this way before, they can provide expert assistance to the problem of joining applications. The next section describes in more detail how CASE technology can be applied to the solution of complex integration problems.

4.4 The CAD/CAM CASE Workbench and Integration Toolkit

In the literature review, a CASE workbench was described as a collection of toolkits which support every basic phase of the software development life cycle as well as software management capabilities. In this research, a new CASE workbench is described which has been specifically designed for the generation of CAD/CAM integration systems. The workbench includes toolkits for the analysis, design, automatic coding and testing of CAD/CAM integration systems as well as their maintenance. A common database for the storage of all workbench diagrams and information is included and a framework is utilized to enable the integration of toolkits within the workbench. The workbench is composed of a combination of commercially available tools (some of which will be modified) as well as tools being designed and created at Virginia Tech. One tool being developed at Virginia Tech, that is addressed in detail in this dissertation,

is a specialized toolkit to assist with the identification of source-code level information which is vital to the successful integration of dissimilar applications both with each other as well as with a common database. This tool, the ***CASE integration toolkit*** will be used by the integration system designer as a way of initially evaluating the applications to be integrated. Review of the information provided by the CASE integration toolkit allows the system designer to quickly understand the structure and specific characteristics of the applications codes. The information provided by this toolkit can then be stored and utilized by subsequent toolkits to expedite the integration system design.

The selected target environment for the CAD/CAM CASE workbench is the IBM RS/6000 workstation running under the AIX operating system. The C programming language has been selected for the development of the CASE integration toolkit. The major function of the CAD/CAM CASE integration toolkit will be to assist with the integration of software whose characteristics are fully known, or whose source code is available. This is justified due to the heavy reliance on in-house codes stated in the results of the CAD/CAM integration survey described in the previous chapter of this report.

A detailed description and specification of the CAD/CAM CASE workbench is given in chapter 5, while a detailed description and design for the CASE integration toolkit is presented in chapter 6. To assist in a more complete understanding of how these tools will be used, a general overview of their application to current methods of joining application programs will now be discussed.

4.5 Application of CASE Methods to the Creation of Integration Systems

When modifying existing programs to allow any type of joining, it is necessary to know as much about the original program source as possible. Determination of the overall program structure as well as identification of the many vital program details often takes weeks and sometimes months to clearly define. While some tools to assist the identification of source details exist, these are usually at a very low-level (e.g. compilers) and do not provide the more high-level information necessary for easy use and understanding. The CASE integration toolkit is designed to analyze source programs for the identification of higher-level type information and to store this information in the CASE workbench and data dictionary for display by any of the workbench toolkits.

The CAD/CAM CASE workbench and integration toolkit are designed to assist the current methods of joining application programs, discussed in the previous sections, each to varying extents. The type of support and assistance provided to the system designer by the CAD/CAM CASE workbench and integration toolkit, for each method, will now be discussed.

4.5.1 Rigidly and Freely-Connected Interfacing

In considering these methods of joining, the first thing that needs to be obtained is the exact manner in which data must be read into each program to be interfaced. Optimally, this would include:

- Occurrences of all READ statements (number and location),
- Variables being READ in,
- Variable type,
- Variable order,
- Format of Variable READ,
- Specifiers used in READ (if any),
- Logical units in use,
- Labels in use, and
- Comments that may precede the READ statement or appear at the beginning of the program and include variable definitions or other important information about the READ statement and other program details.

The next thing which must be addressed is the location of the corresponding variables and information in the applications programs, which must be written to an interface file. This information would include:

- Variables to be written out,
- Variable type verification,
- Variable assignment (for each variable),
- Logical units in use,
- Labels in use, and
- Comments that may precede necessary variables or appear at the beginning of the program and include variable definitions or other important information and descriptions.

To circumvent the tedious method of looking through the program code to find this information and then manually recording it on paper, the CASE integration toolkit can

be called upon to provide this information as well as to produce tier charts showing the overall program structure. This reduces the time necessary for original program evaluation to days instead of weeks or months. Once evaluation is complete, the system designer can use the information obtained, along with the other CASE toolkits, to design necessary code additions or output statements. Code additions may include conversion or manipulation of data to be shared between programs (e.g. units, geometry, etc.) CASE design capabilities will also greatly assist the creation of standard file formats such as IGES. Once these file formats have been initially defined, they could be stored and easily recalled for future use, greatly reducing subsequent development time. After the design is finished, the CAD/CAM CASE workbench may also provide automatic generation of C source code from the graphical designs, for any procedures which must be added to the original code. Maintenance of the completed code, including version control, and documentation of the completed integration system, can also be assisted by the workbench.

4.5.2 Rigidly and Freely-Connected Coupling

Next the case of rigidly and freely-connected coupling is considered. This technique involves the use of a common database to communicate the data between different CAD/CAM applications (and almost always a DBMS). In this method, the application programs themselves must be modified to allow communication with a common database, which makes this type of integration very tedious. The current type of database or file storage used, as well as the program information which must be shared, may also increase the complexity of this integration task. Additionally, to fully accomplish the "marriage" of rigidly-connected applications, a great deal of additional design for such

items as common user interfaces must also be considered by the integration system designer, which is not directly discussed here.

To accomplish this type of joining, the following information centered around program input and output must be known about each program to be joined:

- Occurrences of all READ statements (number and location),
- Variables being read in,
- Variable types,
- Specifiers used in READ (if any),
- Occurrences of all WRITE statements (number and location),
- Variables being written out
- Variable types,
- Specifiers used in WRITE (if any),
- Query language database communications (specific to database used) for all type of operations such as SELECT, INSERT, DELETE, UPDATE, etc.,
- Logical units in use,
- Labels in use, and
- Comments that may precede the READ and WRITE statement or appear at the beginning of the program and include variable definitions or other important information and descriptions.

It may also be necessary to share information between programs which is not currently written to a file or database. This may include data used for intermediate calculations that is not stored for permanent record. For the identification of this information, much more must be known about the variables themselves. This may include:

- Variable or array type and number of occurrences,
- Location of variable or array occurrences,
- Variables equivalenced to,
- Location of assignment(s),
- Variables declared in a COMMON statement, or perhaps even
- Variables passed in a subroutine call.

Once again, the CASE integration toolkit can be called upon to provide this information as well as produce tier charts showing overall program structure.

When all necessary evaluation is complete, the system designer can use the information obtained, along with the other CASE toolkits, to design necessary code additions or output statements in the same way as described previously. CASE design capabilities will also greatly assist in the tedious generation of database query language calls to read from or write to the common database. After the initial creation, database call designs can be stored and easily recalled for future use as macros, greatly reducing subsequent development time. After the design of necessary code modifications or additions is finished, the CAD/CAM CASE workbench may also provide automatic generation of C source code from the graphical designs, for any procedures which must be added to the original code. Maintenance of the completed code, including version control, and documentation of the completed integration system, can also be assisted by the workbench.

4.6 *A Possible Workbench Extension*

The case of rigidly-connected coupling, in which separate applications are ideally "married" together and become one big program, will be more easily and efficiently handled through the use of advanced CASE tools. One area that is just beginning to be widely developed in CASE is ***reverse engineering***. As mentioned in the literature review, this method allows the entry of previously generated code into a CASE workbench which then produces the necessary design schematics automatically. After entering the code from all the programs to be connected, the system designer can more easily redesign (diagrammatically), within the CAD/CAM CASE workbench, the integrated CAD/CAM application and have the workbench regenerate the entire application code to these specifications, producing an entirely new program. A considerable amount of effort would still be required to design this new code, but once the design was complete, a correct and efficient method of generating the new program is already in place to greatly reduce the chances of bugs and improper logic. While reverse engineering will undoubtedly be widely used to solve many complex software re-engineering problems such as this one, currently available reverse engineering tools are still at a rather simplistic level. It will probably be several years before tools with the capabilities discussed here are actually available.

4.7 *How to Use the CASE Workbench for the Creation of an Integration System*

An example of the joining of two simple yet different programs is shown in Figure 7 on page 75.

In this example, program MAIN1 defines two variables, VAR1 and VAR2, in subprogram A. Program MAIN2, a separate application program, requires the reading in of a variable, VAR5, in subprogram C2. Suppose that the variable VAR5 consists of the addition of the variables VAR1 and VAR2 and we wish to share this information between programs. Normally, the user would be required to run application MAIN1, then print out or manually notate the values assigned to the variables VAR1 and VAR2 and Once this is done, these values would then be added together by the user and read, or typed into program MAIN2. To facilitate this process, an integration system can be created to handle the joining of these programs in the following manner. The new integration system or software would act as a new main program which calls program MAIN1 as a subroutine, performs the necessary addition (data conversion) of variables VAR1 and VAR2 and then calls program MAIN2 (as a subroutine), sharing the values of VAR1, VAR2 and VAR5 as common data. By this method, the necessary information can be shared easily between application programs in a more automated way.

The CAD/CAM CASE workbench assists this process by locating the assignment of VAR1 and VAR2 in the program code of MAIN1. The integration toolkit also records the fact that VAR1 and VAR2 are already listed in common storage, both in the main program and subroutine A. The toolkit then identifies the location, in MAIN2, where

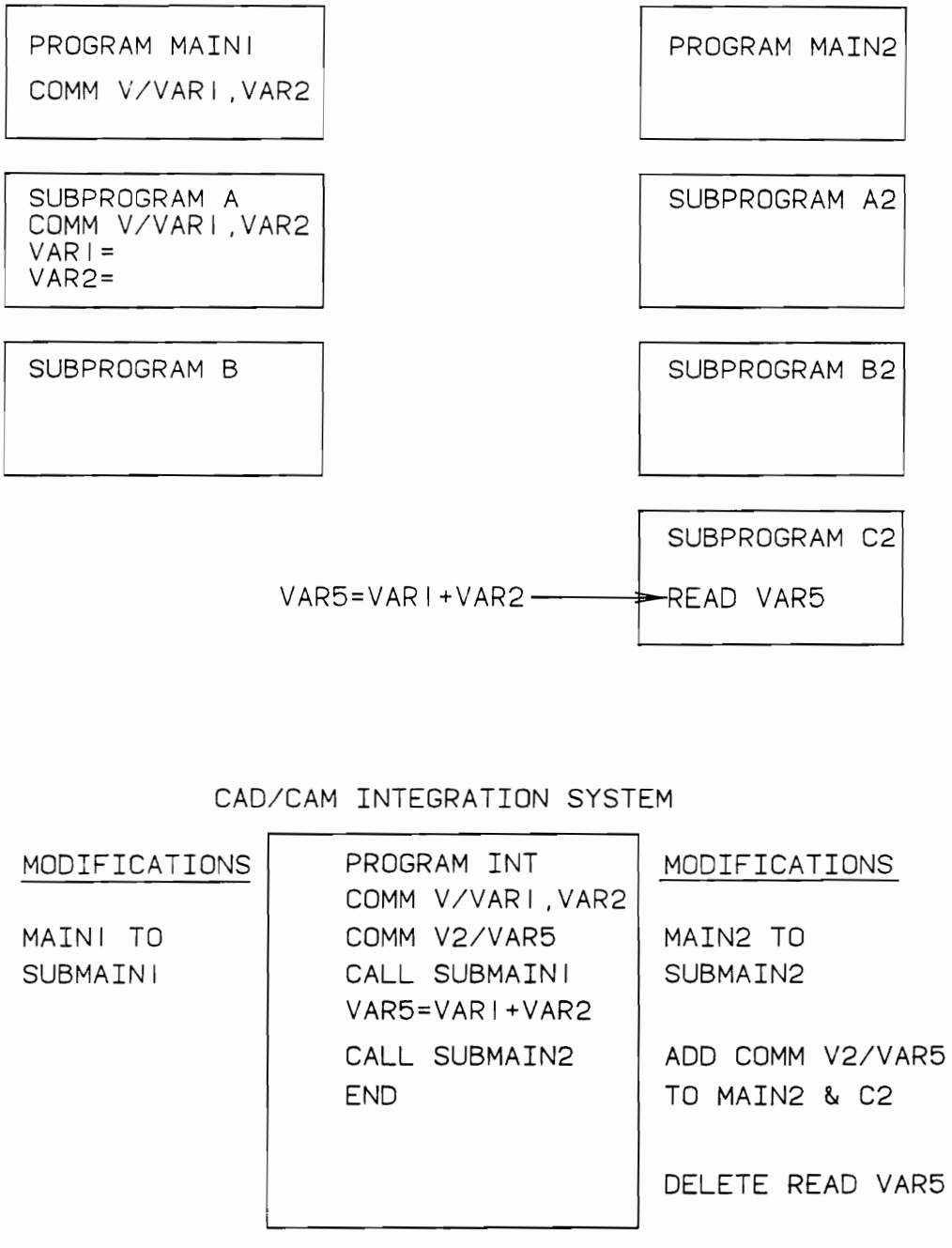


Figure 7. An Example Integration

the variable VAR5 is read in. It also notes that VAR5 is not defined in any type of common storage.

After analysis of the application source codes by the integration toolkit, the necessary information is passed into the CASE design and analysis toolkits for the creation of the integration system described. After this design is complete, code, like that implied in the figure, is generated by the workbench code generator. For the final step, the integration system designer must now change MAIN1 and MAIN2 to subprograms SUBMAIN1 and SUBMAIN2, add VAR5 to common storage in MAIN2 (or SUBMAIN2) and C2 and delete the read statement for VAR5 from C2.

In a typical CAD/CAM integration system, we would additionally find code for database transactions or interface file generation as well as other reading and writing operations. The simple conversion presented here, could easily be a complex geometric transformation between the data of one CAD/CAM system and the next.

4.8 Summary

This chapter has presented a description and categorization of the most common, currently used methods for integrating CAD/CAM application programs. It has also presented a CAD/CAM CASE workbench and integration toolkit which can assist in the design and generation of integration systems which facilitate these methods. The next chapter will discuss in more detail the CAD/CAM CASE workbench which will provide assistance for the generation of integration systems for the different methods of joining applications discussed in this chapter.

5.0 Requirements Analysis and System Specification for the CAD/CAM CASE Workbench

5.1 Introduction

The CAD/CAM CASE workbench assists the integration system designer in the development, maintenance and management of an integration system by providing standard CASE toolkits which support the various phases of the software development life cycle including planning, system analysis, system design, code generation, testing, debugging and maintenance. The workbench also includes custom designed CASE toolkits which assist with integration specific tasks as well as X Window System support to be discussed in chapter 7. By providing these customized tools in a robust CASE environment, we can not only support the integration of CAD/CAM application programs but individual program enhancement and modification as well as new program development. This allows for maximum use of the capabilities of the workbench in addition to encouraging the use of CASE tools for new program creation.

The primary requirements of the integration systems created by the CAD/CAM CASE workbench vary according to the type of integration required and can include any of the following:

- Transfer of data and information including:
  - Generation of specialized input and output files
  - Communication with a common database
  - Communication between applications
- Simultaneous execution of application programs
- Sharing of menus and functions between application programs
- Integration across multiple platforms and sites

Based on these integration system requirements, we can compile a list of requirements for the CAD/CAM CASE workbench which will assist the integration system designer in generating these integration systems. The intent of the next section is to describe all integration tasks which could be substantially aided by the CAD/CAM CASE workbench.

5.2 *CAD/CAM CASE Workbench Requirements*

The CAD/CAM CASE integration workbench should include the following essential components:

- Allow easy access to all the tools of the CASE workbench through an easy-to-use graphical user interface

- Provide the designer with easy access to the specialized functions of the CASE integration toolkit, the CAD/CAM integration interface and the executive generator and X support toolkit to be discussed in the next chapter.

- Allow suitable CASE design and analysis methodology support, including graphical aids for diagramming, to assist in the design of the integration system

- Provide graphical assistance in the creation of information or data models which define entities, objects, relationships and attributes of the application programs to be integrated

- Allow automatic generation of C source code based on information obtained from the design and analysis toolkits

- Provide a library function which can store and retrieve commonly used source code for procedures which may need to be repeated such as unit conversions, coordinate conversions and geometric conversions (e.g. transference of one type of surface, such as B-spline, into a system which commonly uses a different geometric surface de-

scription). In storing these conversions as they are generated, a complete code library of CAD-related functions can be easily created.

- Allow for generated C code to be compiled, tested and then edited

- Facilitate maintenance, including version control, of the final integration code by the CAD/CAM CASE workbench

- Allow for all these phases of the software development life cycle and their corresponding toolkits, including the CASE integration toolkit, to be supported and combined to form an integrated software development environment capable of creating the necessary information and sharing it between tools

- Provide a common database within the workbench for storage of essential diagrams, important information and generated code. Some type of data dictionary, for the storage of all pertinent variables (input data, etc.) should also be included

- Allow the ability to use enablers such as X windows and PHIGS to facilitate the successful integration of application programs

- Provide the ability to create documentation in support of the final integration system and any other necessary documents

- Allow use of the workbench on a variety of hardware platforms with a minimum of modifications

5.3 *CASE Workbench Specification*

This section describes a CASE workbench which will be capable of supporting the integration of CAD/CAM application programs, with some modifications, as previously defined in the requirements section. Once complete, the CAD/CAM CASE workbench will include its own source code generator which will generate code directly from the design and analysis specifications for the required integration system. C has been chosen as the language to be generated by the workbench code generator. Future research may focus on the integration of CAD/CAM applications which utilize X or the PHIGS standard for graphical representation. This is not necessarily a limiting factor since many of the major CAD/CAM systems utilize these standards already and most custom applications will utilize these standards in the future, according to the CAD/CAM integration survey.

The following commercial and custom tools can be assembled and in some cases modified to form the CAD/CAM CASE workbench previously discussed. A schematic of the proposed workbench is shown in Figure 8 on page 82.

The majority of these products comprise IBM's new ***AIX CASE integrated C Solution***, announced in January 1990 [IBM90], which was developed in association with the IBM Business Partners, Cadre Technologies and Interleaf. The selected target environment for the workbench is the IBM RS/6000 workstation running under the AIX operating system.

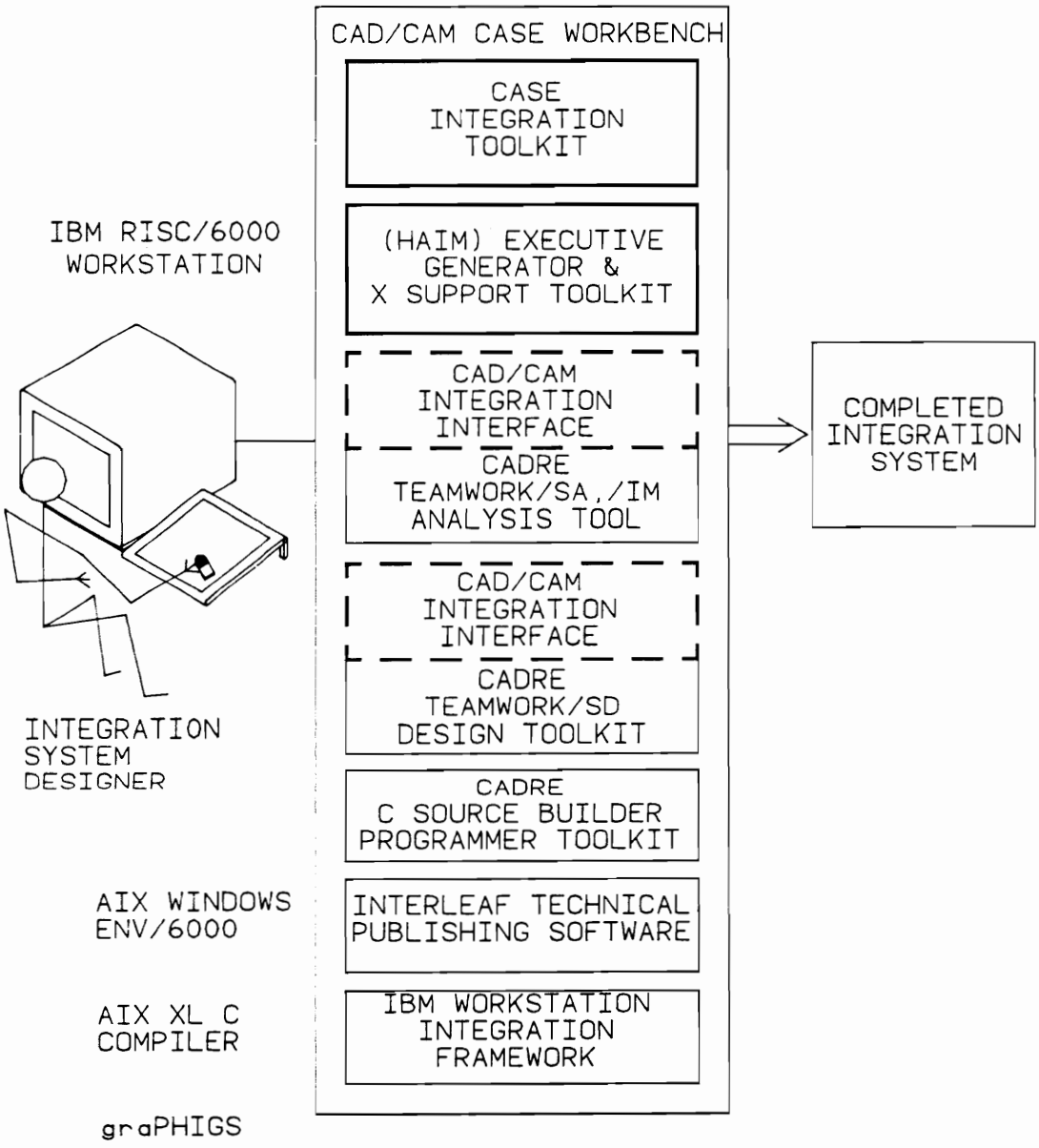


Figure 8. CASE Workbench Designed for CAD/CAM Integration

- **IBM AIX Operating System Version 3**

  This is a new generation of the AIX operating system. It is designed to maximize IBM's second-generation RISC technology and provide significant operating system enhancements.

- **IBM AIX windows Environment/6000**

  This is a graphical user interface environment for the RISC System/6000 family using AIX Version 3. It provides the ability to develop and run advanced graphics applications (such as GL and graPHIGS), IBM enhanced X Windows applications and AIX windows applications.

- **IBM AIX XL C Compiler/6000**

  This is a compiler based on advanced technology that can provide highly optimized code. This extended language (XL) compiler is designed to provide consistency and high performance across multiple languages by sharing the same code optimization technology.

- **CASE Integration Toolkit (VPI)**

  This is a toolkit designed to assist with specific program code integration tasks such as extraction and categorization of information from the CAD/CAM applications to be integrated. This information, which includes such items as program code structure and other specific program characteristics, helps to quickly and easily acquaint the integration system designer with unfamiliar ap-

plication programs and greatly assists in the integration process. A detailed description of the integration toolkit appears in the next chapter.

- **CAD/CAM Integration Interface (VPI)**

  This is a toolkit designed to provide the information extracted by the CASE integration toolkit to be shared and used to assist in the design and analysis of the integration system. A more complete description of this toolkit appears in the next chapter, chapter 6.

- **HAIM Executive Generator & X Support Toolkit (VPI)**

  This is a toolkit designed to assist with the generation of the system executive for the autonomous integration model described in the next chapter. This toolkit controls the invocation of application programs in the autonomous model based on the X Window System. It also provides assistance with the enabling of inter-client communications and network server communications when necessary.

- **Cadre Technologies Teamwork/SA**

  Based on standard Yourdon-DeMarco methods, this structured analysis module provides an editor for creating and editing data flow diagrams graphically and describing the lowest level of decomposition with a simple textual description or mini-specification (these are called ***process specifications*** in Teamwork). Data elements in the data flow diagram are described in data dictionary entries. These data dictionary entries as well as process specifications are "attached" to

the data flow diagram for easy retrieval and updating. Data flow diagrams and their elements as well as process specifications are stored in a central project database. Access to the project database is provided via a C programming language interface. Teamwork/SA will build input/output lists for each process node in the data flow diagram and allow the attaching of program code to process specification. Developers can analyze top-down or bottom-up, resulting in a complete hierarchy of diagrams describing each degree of system complexity. Teamwork/SA also checks to make sure that the hierarchical data flow diagrams are consistent (or more commonly ***balanced***) by ensuring that input and output data flows match between levels. This task is crucial to overall design completeness but is seldom done because of the tedium of doing it with manual techniques [CADR89a, FISH88].

- **Cadre Technologies Teamwork/RT**

  This is a set of real-time extensions for Teamwork/SA which provide extensions to the standard Yourdon/DeMarco data flow diagram for real-time control. Diagrams for real-time control specifications which include state transition diagrams, state event matrices, process activation tables and decision tables are also included in this module.

- **Cadre Technologies Teamwork/IM**

  This is the information modeling module of Teamwork which incorporates Chen notation to assist system analysts in modeling the entities, objects and relationships of application information at a conceptual level through the use of entity-relationship diagrams [CADR89a, FISH88]. Entity-relationship dia-

grams visually describe the data elements of a system by showing the relationships which exist between them. Data definitions and the attributes of their entities and relationships are stored in the data dictionary.

- **Cadre Technologies Teamwork/SD**

An implementation of the Yourdon-Constantine structured design method, this software module features modular decomposition appropriate for many procedural languages such as C, Fortran and Pascal. Structured designs cleary represent a system's ***internal*** architecture, while structured analysis represents a system's user view or ***external*** architecture. Teamwork/SD allows a system designer to graphically derive and document software modules and module interfaces directly from the graphical model created in the analysis stage using structure charts. Structure charts have three components: boxes representing modules, arrows which show the connection between modules and small circles with arrows which represent the data passed from one module to the next. The module boxes of a structure chart contain algorithmic descriptions of a module's functions which can be source code or pseudocode. Teamwork/SD like other Teamwork modules is also supported by a data dictionary [CADR89a, FISH88].

- **Cadre Technologies C Source Builder**

The C Source Builder automatically builds C source code files from Teamwork/SD models. The software builds function definitions and external declarations for each module specification and places these in the appropriate source or include files. It also translates required data dictionary entries to C

declarations and places them in the appropriate source or include file. The system designer can interactively add implementation details to module specifications and data dictionary entries for complete code generation. It is additionally possible to add C language to clarify module specifications which will be embedded right into the generated design. C source code files created by the C Source Builder can be immediately compiled and linked. This module also provides easy editing of module specifications and data dictionary entries. The ability to store generated code for later reuse is also available [CADR89b, FISH88].

- **IBM Framework (currently under development)**

  This is a framework used for building and managing an infrastructure for computer assisted software development. The IBM Framework will link the CASE toolkits, data and other workbench information into a cohesive environment. It will provide version control, configuration control services, correspondence control services, tool integration services and a relational database with data dictionary support.

- **Interleaf Technical Publishing Software**

  This is a family of versatile tools that allows the creation, revision, sharing, managing and printing of professional-looking documents with ease. The Interleaf software can be used to create integration system documents incorporating the graphical analysis and design charts for illustration. Any other documentation required by the system designer can also be created using this tool [Inte90].

5.4 *Summary*

This section has presented an overview of a CASE workbench designed specifically for CAD/CAM integration. A list of workbench requirements have been described followed in the next section by a workbench specification which can meet and surpass the stated requirements. A brief description of each toolkit in the workbench has also been presented. The next chapter discusses the requirements analysis and design of the CASE integration toolkit.

6.0 Requirements Analysis and Design Specifications for the CAD/CAM CASE Integration Toolkit

6.1 Introduction

The CASE integration toolkit is designed to assist with specific program integration tasks such as identification and categorization of information from the CAD/CAM application programs to be integrated. The major function of the CASE integration toolkit is to assist with the integration of software whose characteristics are fully known, that is, whose source code is available. This fact is justified due to the heavy reliance on in-house codes stated in the results of the CAD/CAM integration survey presented in chapter 3. Providing this toolkit in an integrated project support environment, as part of a CASE workbench, will make it possible for the integration system designer to more

quickly and successfully design and generate CAD/CAM integration systems which meet the desired specifications.

Providing the system designer with detailed and comprehensive information about the application programs to be integrated is an important requirement for successful integration. Many of the specific information requirements have been listed in the previous chapter. To most effectively provide assistance for integration, all the previously mentioned information plus additional specifics about the program code must be supplied for complete identification and analysis. A detailed list of requirements is considered in the next section.

6.2 CASE Integration Toolkit Requirements

Since results of the CAD/CAM integration survey show that existing custom CAD/CAM applications are written in FORTRAN, this language has been chosen as the model language to be analyzed for the design of the CASE integration toolkit. While many of the ideas presented here can be applied to the analysis of other programming languages, a large degree of language dependence will also exist in the analysis of each different programming language.

The following list gives detailed requirements for the type of information that would be helpful to the system integration designer in the initial evaluation of applications source code.

Location refers to the location within the main program or individual subprogram unless otherwise noted.

I. Variable Information

- A. Name
- B. Type (declared or implicit)
- C. Location of assignment (including first and last occurrence)
- D. Number of occurrences
- E. Location of occurrences
- F. Number of times passed in a subprogram call statement
- G. Variables equivalenced to
- H. Variables initialization in a data statement

II. Array Information

- A. Name
- B. Type (declared or implicit)
- C. Dimensionality and length (declared or dimensioned)
- D. Location of assignment (including first and last)
- E. Number of occurrences
- F. Location of occurrences
- G. Number of times passed in a subprogram call statement
- H. Variables, arrays or portions of arrays equivalenced to
- I. Arrays initialized in a data statement

III. Parameter Information

- A. Name
- B. Type (declared or implicit)
- C. Location of assignment (including first and last)
- D. Number of occurrences
- E. Location of occurrences
- F. Number of times passed in a subprogram call statement

IV. Subprogram Information

A. Overall Program Architecture

1. Tier chart

B. Function Subprograms

1. Name
2. Type (declared or implicit)
3. Number of places called
4. Location of calls
5. Number of arguments passed
6. Identification of variables passed
7. Number of multiple entries
8. Location of multiple entries
9. Number of multiple returns
10. Location of multiple returns

C. Function Statement (must be embedded in a main program or subprogram)

1. Name
2. Type (declared or implicit)
3. Location of assignment
4. Number of arguments passed
5. Identification of arguments in statement
   a. Variable name
   b. Array name
   c. Expression
   d. Subprogram
   e. Constant Value
   f. Intrinsic function
   g. External subprogram

D. Subroutines

1. Name

2. Type (declared or implicit)
3. Number of places called
4. Location of calls
5. Number of arguments passed
6. Identification of variables passed
7. Number of multiple entries
8. Location of multiple entries
9. Number of multiple returns
10. Location of multiple returns
11. Number of multiple destinations for a return
12. Location of multiple destinations

V. Shared Data Among Program Components

A. Common Data Storage

1. Blank Common Block (only one can exist)

a. Variable or array names in common

b. Number of subprograms in which a blank common occurs

c. Location of occurrences of blank common

2. Named Common Blocks

a. Blockname of each named common

b. Variable or array names in each named common

c. Number of subprograms in which each named common appears

d. Location of occurrences of each named common

B. Block Data

1. Name of block data if named (must be named if more than one exists)

2. Number of block data subprograms

3. Location of block data subprograms

4. Location of occurrences of each block data subprogram

5. Variables which are in common and are initialized in a data statement within a block data subprogram

VI. Input/Output

A. READ Statements

1. Number of occurrences of READ statement

2. Location of each READ statement

3. Identification of specifiers in each READ statement (for list-directed statements)

a. Unit specifier
b. Format specifier

*Optional specifiers

c. END specifier
d. ERR specifier
e. REC specifier
f. IOSTAT specifier

4. Variable(s) or array(s) to be read

B. WRITE Statements

1. Number of occurrences of WRITE statement

2. Location of each WRITE statement

3. Identification of specifiers in each WRITE statement (for list-directed statements)

a. Unit specifier
b. Format specifier

*Optional specifiers

c. ERR specifier
d. REC specifier
e. IOSTAT specifier

4. Variable(s) or array(s) to be written

C. PRINT Statements

1. Number of occurrences of PRINT statement

2. Location of each PRINT statement

3. Identification of specifier in each PRINT statement (for list-directed statements)

   a. Format specifier

4. Variable(s) or array(s) to be printed

D. BACKSPACE Statements

1. Number of occurrences of BACKSPACE statement

2. Location of each BACKSPACE statement

3. Identification of specifiers in each BACK-SPACE statement

   a. Unit specifier

   *Optional specifiers

   b. ERR specifier
   c. IOSTAT specifier

E. ENDFILE Statements

1. Number of occurrences of ENDFILE statement

2. Location of each ENDFILE statement

3. Identification of specifiers in each ENDFILE statement

   a. Unit specifier

   *Optional specifiers

   b. ERR specifier
   c. IOSTAT specifier

F. REWIND Statements

1. Number of occurrences of REWIND statement

2. Location of each REWIND statement

3. Identification of specifiers in each REWIND statement

   a. Unit specifier

   *Optional specifiers

   b. ERR specifier

c. IOSTAT specifier

G. OPEN Statements

1. Number of occurrences of OPEN statement
2. Location of each OPEN statement
3. Identification of specifiers in each OPEN statement
   a. Unit specifier

   *Optional specifiers

   b. ERR specifier
   c. ACCESS specifier
   d. FORM specifier
   e. IOSTAT specifier
   f. RECL specifier
   g. BLANKS specifier
   h. FILE specifier
   i. STATUS specifier

II. CLOSE Statements

1. Number of occurrences of CLOSE statement
2. Location of each CLOSE statement
3. Identification of specifiers in each CLOSE statement
   a. Unit specifier

   *Optional specifiers

   b. IOSTAT specifier
   c. STATUS specifier

VII. Labels

A. Label numbers used in each subprogram

B. Number of labels used

C. Location where each label appears

D. Location(s) where labels are referenced

VIII. DBMS Query Language (e.g. SQL)

A. Type each type of each database access operation

1. SELECT

2. INSERT

3. UPDATE

4. DELETE

B. Number of occurrences of each database access operation

C. Location of occurrence of each database access operation

D. Arguments (e.g. variables, etc.) in each database access statement

E. Associated database locations for each operation

IX. Comments and Other

A. Identification of comment blocks and their size as well as location

B. Identification of single comments, their size and location

C. Number of comment lines within the subprogram

D. Total number of lines within the subprogram

X. Allow the designer to pass extracted information (discussed above) directly into the design and analysis toolkit for the design of the integration system

6.3 Design of the CASE Integration Toolkit

This section includes an overview of the detailed description of the CASE integration toolkit design. Each of the requirements listed in the previous section has been addressed with the exception of the database query language operations (VIII). Since this information is largely dependent on software system selection, it will not be addressed until a specific database has been selected.

The section begins with a discussion of design considerations and notation and is followed by detailed discussion of toolkit program structure. As previously mentioned, the analysis of FORTRAN programs will be considered by this design.

The storage of extracted information is accomplished in one of two ways; through the use of the workbench database and data dictionary or through the use of sequential files. Subprograms are an example of information which could be stored in a sequential file. Program variables and comments will be stored directly into the workbench database and data dictionary where they can be used and displayed by any of the workbench toolkits. Some type of standard file format will be used for the import and export of information to and from the CAD/CAM CASE workbench database. The CDIF (CASE Data Interchange Format) proposed standard developed by CADRE Technologies will be considered. The CAD/CAM integration interface will handle all necessary details associated with these tasks and is discussed in the last section of this chapter.

6.3.1 Design Considerations

It is important to mention here that standard FORTRAN 77 has been used as a guide in the design of the CASE integration toolkit. Additionally, all FORTRAN code to be analyzed has been assumed to be syntactically correct, and thus no checks are performed for correctness. This assumption is valid if the code to be integrated compiles and executes without error before integration begins.

Also of importance in the design of the integration toolkit is the fact that the FORTRAN source file under analysis is only read once by the toolkit for parsing. This single-pass operation allows more efficient execution of the toolkit code.

The design of the CASE integration toolkit is completely detailed on hierarchically decomposed flowcharts. These flowcharts all appear as figures within Appendix B, and are listed by name and module number in the list of illustrations at the beginning of this dissertation. This allows quick identification of page numbers for specific flowchart reference when needed. The flowcharts are drawn according to the methods described by Robert C. Tausworthe in his texts entitled ***Standardized Development of Computer Software, Volumes I & II*** [*TAUS*76, *TAUS*78]. Symbols used on the flowcharts are derived from the document ***American National Standard Flowchart Symbols and Their Usage in Information Processing***, ANSI-X3.5-1970, American National Standards Institute, Inc., New, York, Sept. 1, 1970. These symbols have, in some cases, been refined or extended by Mr. Tausworthe.

The main symbols which appear in the flowcharts for the CASE integration toolkit are as follows:

1. Start and termination symbol - represents the entry or exit point of a flowchart
2. Process box - represents any kind of processing
3. Striped process box - represents a named procedure (subprogram or subroutine) module that has a more detailed representation elsewhere in the same set of flowcharts
4. Decision symbol - represents a specific decision that determines which of alternate paths is to be followed
5. Decision collecting node - represents the merging of alternative flow paths in a program
6. Loop collecting node - represents the iteration point in a looping operation
7. Indexed looping - represent loop initialization, testing and update functions

8. Input/Output symbol - represents and I/O function for processing (input), or the recording of processed information (output).

The notational symbols described above are shown in Figure 9 on page 101.

A general overview is now presented which includes references to many of the toolkit design flowcharts. In the detailed description of FORTRAN statement processing as well as that of other syntactic elements, the following notation will be used to describe standard FORTRAN 77 expressions:

- Upper case letters and words are used to represent FORTRAN keywords and flow chart names.

- Italicized lower case letters and lower case words indicate general entities in a syntactic element or variables from the design flow charts.

- Square brackets ([ ]) are used to indicate optional items.

- An ellipsis (...) indicates that the preceding optional item may appear one or more times in succession.

- Blanks are used in these expressions to improve readability only and have no other significance (just as in FORTRAN).

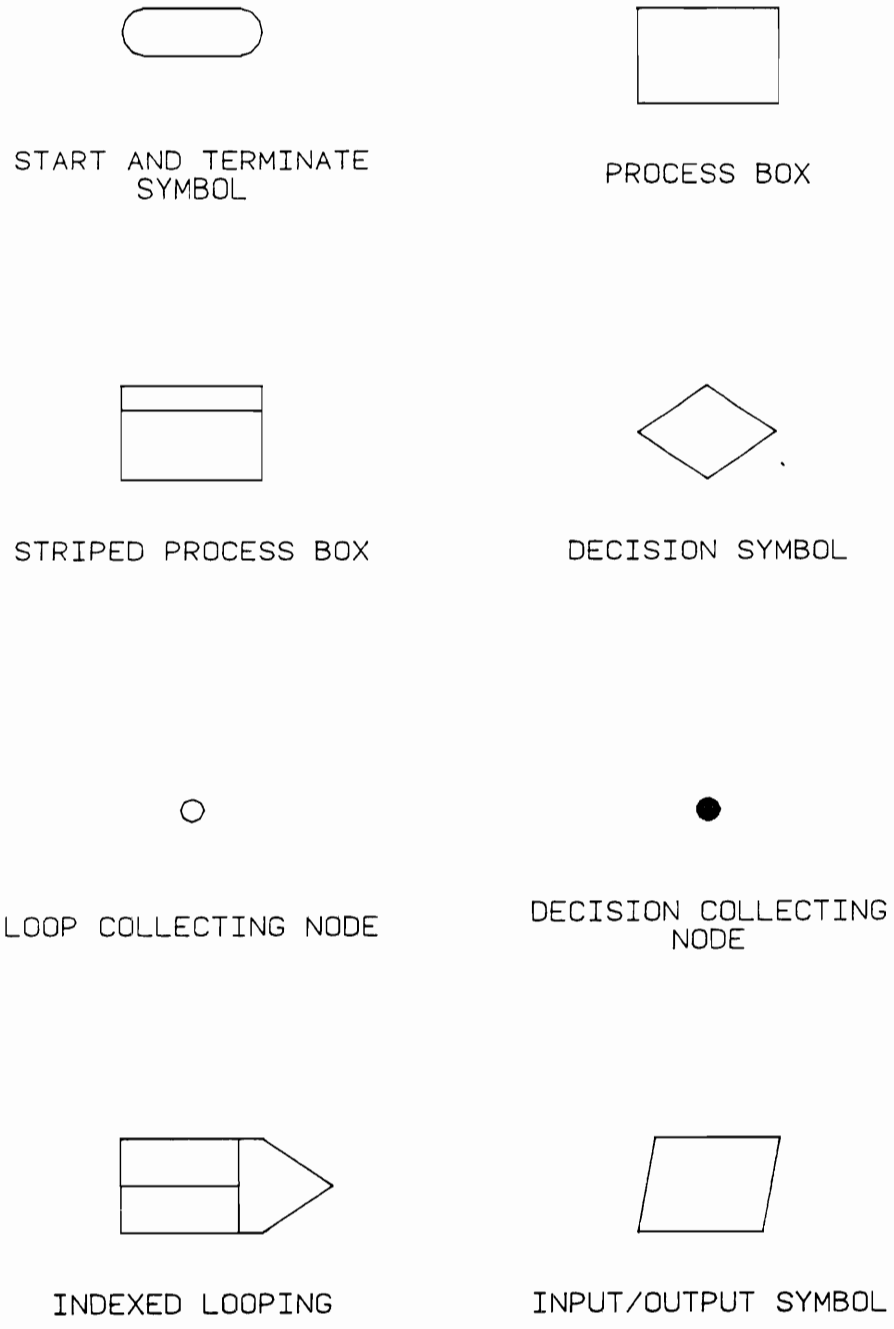


Figure 9. Flowcharting Symbols

6.3.2 Module A - MAIN_PROG

The Module A flowchart, which shows the design of the main program for the integration toolkit, gives the overall program structure and is shown in Figure 12 on page 160. Box number one represents the initialization of all program variables. A complete listing of the variables to be initialized is shown in the expanded chart INITIAL_PGM. Box number two finds and opens the source file to be analyzed. Box number three represents the reading of one source line and details its processing. Comment lines are processed at this time as well as several other items and only FORTRAN statements are retained for further analysis. This process is expanded in chart EXTRACT_STMT. The next box is a decision box which checks for the end of the file being processed. If this is not the end of the file, the prepared FORTRAN statement is now processed as shown in chart PROCESS_STMT. The final process box on this chart closes the file under consideration once the analysis is complete.

6.3.3 Module A.3 - EXTRACT_STMT

Module A.3 shows the detailing of the EXTRACT_STMT process and is shown in Figure 14 on page 162. Two major tasks are handled at this level, the processing and storage of all comment statements for later use (expanded in chart CMT_STMT) and the preparation of FORTRAN statements for further analysis. This analysis includes the removal of all blank lines from the source code, the appending of all continuation lines so that each FORTRAN statement can be considered in its entirety and the identification of all statement labels. Once these operations are complete, the remaining

statement is stored from column 7 to column 72, all blanks are removed and lower case letters are converted to upper case.

6.3.4 Module A.5 - PROCESS_STMT

Module A.5 shows the detailing of the PROCESS_STMT process and is shown in Figure 16 on page 164. This chart begins the decomposition of the major portion of toolkit code. The prepared FORTRAN statement is taken as input and is identified as a program or subprogram start statement, a nonexecutable statement, an executable statement or an END statement. Based on this information, the statement processing is expanded in chart PROC_START, chart PROC_NONEXEC, chart PROC_EXEC or END_STMT. A variable called *PGM_SEC* keeps track of which section of the program code the fortran statement has been processed in. When program execution begins, this variable is set to "start", if the statement is not processed as a start statement, the variable *PGM_SEC* is reset to be a nonexecutable statement. This process continues through the program code until all nonexecutable and executable statements are processed in this same manner and a FORTRAN END statement is encountered. Control is then returned to the main program for preparation and processing of the next FORTRAN source line.

6.3.5 Module A.5.2 - PROC_START

Module A.5.2 shows the detailing of the PROC_START process and is shown in Figure 17 on page 165. This chart shows the processing of statements identified as start statements by the variable *PGM_SEC* in the PROCESS_STMT subroutine. These are

the PROGRAM statement (detailed in chart P_PGM), the SUBROUTINE statement (detailed in chart P_SUB), the BLOCK DATA statement (detailed in chart P_BLKDATA) and the FUNCTION statement (detailed in chart P_FUNC). A processing flag, *P_STATE*, is returned indicating whether or not the FORTRAN statement was processed or not. The statements processed by the PROCESS_STMT subroutine are now addressed. Complete representation of their major functions is clearly illustrated in the flowcharts which appear in Appendix B.

6.3.5.1 Module A.5.2.3 - P_PGM

P_PGM, shown in Figure 18 on page 166, processes the PROGRAM statement. The general form of this statement is

PROGRAM *name*

6.3.5.2 Module A.5.2.5 - P_SUB

P_SUB, shown in Figure 19 on page 167, processes the SUBROUTINE statement. The general form of this statement is

SUBROUTINE *name* [([arg_1 [, arg_2] ...])]

6.3.5.3 Module A.5.2.7 - P_BLKDAT

P_BLKDAT, shown in Figure 20 on page 168, processes the BLOCK DATA statement. The general form of this statement is

BLOCK DATA [*name*]

6.3.5.4 Module A.5.2.9 - P_FUNC

P_FUNC, shown in Figure 21 on page 169, processes the FUNCTION statement. The general form of this statement is

[*type*[**n*]] FUNCTION *name* ([arg_1 [, arg_2] ...])

6.3.6 Module A.5.6 - PROC_NONEXEC

Module A.5.6 shows the detailing of the PROC_NONEXEC process, and is shown in Figure 28 on page 176. All standard nonexecutable FORTRAN 77 statements are considered in the design of this module. The process begins with a test for statement functions or assignment statements. Identified statement functions are processed by the module P_STMT_FUNC while assignment statements cause program control to be returned to the next higher-level of decomposition (PROCESS_STMT) for evaluation as an executable statement. If the statement is not a statement function or assignment is processed individually based on which FORTRAN statement type it is identified as. Once processing is complete, the variable *P_STATE* is returned to indicate whether or not the statement was processed or not. Because of the complexity of this chart, inclusion of all process expansion names was not possible. These have been detailed in a table which follows the PROC_NONEXEC chart and has the module number A.5.6.T.

The statements processed by the PROC_NONEXEC subroutine are now addressed. Complete representation of their major functions is clearly illustrated in the flowcharts which appear in Appendix B.

6.3.6.1 Module A.5.6.4 - P_STMT_FUNC

P_STMT_FUNC, shown in Figure 30 on page 178, processes a statement function declaration. The general form of this is

$$name(\ arg_1[, arg_2]...) = expression$$

6.3.6.2 Module A.5.6.6 - P_REAL

P_REAL, shown in Figure 31 on page 179, processes the REAL statement. The general form of this statement is

$$\text{REAL}\ [*n]\ name_1[(dimension_1)]\ [,\ name_2[(dimension_2)]]\ ...$$

6.3.6.3 Module A.5.6.8 - P_INTGR

P_INTGR, shown in Figure 32 on page 180, processes the INTEGER statement. The general form of this statement is

$$\text{INTEGER}\ [*n]\ name_1[(dimension_1)]\ [,\ name_2[(dimension_2)]]\ ...$$

6.3.6.4 Module A.5.6.10 - P_LOGL

P_LOGL, shown in Figure 33 on page 181, processes the LOGICAL statement. The general form of this statement is

COMPLEX [*n] $name_1$[($dimension_1$)] [, $name_2$[($dimension_2$)]] ...

6.3.6.5 Module A.5.6.12 - P_CHAR

P_CHAR, shown in Figure 34 on page 182, processes the CHARACTER statement. The general form of this statement is

CHARACTER [*n] $name_1$[($dimension_1$)] [, $name_2$[($dimension_2$)]] ...

6.3.6.6 Module A.5.6.14 - P_DBL_PR

P_DBL_PR, shown in Figure 35 on page 183, processes the DOUBLE PRECISION statement. The general form of this statement is

DOUBLE PRECISION $name_1$[($dimension_1$)] [, $name_2$[($dimension_2$)]] ...

6.3.6.7 Module A.5.6.16 - P_COMP

P_COMP, shown in Figure 36 on page 184, processes the COMPLEX statement. The general form of this statement is

COMPLEX [*n] $name_1$[($dimension_1$)] [, $name_2$[($dimension_2$)]] ...

6.3.6.8 Module A.5.6.18 - P_COMM

P_COMM, shown in Figure 37 on page 185, processes the COMMON statement. The general form of this statement is

$$\text{COMMON } [/[name_1]/] \; list_1 [[,]/[name_2]/ \; list_2] \ldots$$

6.3.6.9 Module A.5.6.20 - P_IMPL

P_IMPL, shown in Figure 40 on page 188, processes the IMPLICIT statement. The general form of this statement is

$$\text{IMPLICIT } type_1 \; (list_1) \; [, \; type_2 \; (list_2) \;] \ldots$$

6.3.6.10 Module A.5.6.22 - P_INTR

P_INTR, shown in Figure 41 on page 189, processes the INTRINSIC statement. The general form of this statement is

$$\text{INTRINSIC } name_1 \; [, name_2 \;] \ldots$$

6.3.6.11 Module A.5.6.24 - P_EXT

P_EXT, shown in Figure 42 on page 190, processes the EXTERNAL statement. The general form of this statement is

$$\text{EXTERNAL } name_1 \; [, name_2 \;] \ldots$$

6.3.6.12 Module A.5.6.26 - P_PARAM

P_PARAM, shown in Figure 43 on page 191, processes the PARAMETER statement. The general form of this statement is

PARAMETER ($name_1$ = $const_1$ [, $name_2$ = $const_2$] ...)

6.3.6.13 Module A.5.6.28 - P_EQUIV

P_EQUIV, shown in Figure 45 on page 193, processes the EQUIVALENCE statement. The general form of this statement is

EQUIVALENCE ($name\_a_1$[($list\_a_1$)] , $name\_b_1$[($list\_b_1$)])
[, ($name\_a_2$[($list\_a_2$)] , $name\_b_2$[($list\_b_2$)])] ...

6.3.6.14 Module A.5.6.30 - P_DIMEN

P_DIMEN, shown in Figure 48 on page 196, processes the DIMENSION statement. The general form of this statement is

DIMENSION $name_1$($dimension_1$) [, $name_2$($dimension_2$)] ...

6.3.6.15 Module A.5.6.32 - P_DATA

P_DATA, shown in Figure 49 on page 197, processes the DATA statement. The general form of this statement is

DATA $list_1$ /$clist_1$/ [[,] $list_2$ /$clist_2$/] ...

6.3.6.16 Module A.5.6.34 - P_SAVE

P_SAVE, shown in Figure 52 on page 200, processes the SAVE statement. The general form of this statement is

SAVE $name_1$ [, $name_2$] ...

6.3.7 Module A.5.10 - PROC_EXEC

Module A.5.10 details the design of the PROC_EXEC process, and is shown in Figure 53 on page 201. All standard, executable FORTRAN 77 statements are considered by the design of this module. The process begins with a test for assignment statements. Identified assignment statements are then processed by the module P_ASSN_STMT. If the statement is not an assignment statement, it is checked against all possible executable statements and then processed. Once processing is complete, the variable *P_STATE* is returned to indicate whether or not the statement was processed or not. Because of the complexity of this chart, inclusion of all process expansion names was not possible. These have been detailed in a table which follows the PROC_EXEC chart and has the module number A.5.10.T.

The statements processed by the PROC_EXEC subroutine are now addressed. Complete representation of their major functions is clearly illustrated in the flowcharts which appear in Appendix B.

6.3.7.1 Module A.5.10.4 - P_ASSN_STMT

P_ASSN_STMT, shown in Figure 55 on page 203, processes an assignment statement. The general form of this statement is

$$name\ (list)\ =\ expression$$

6.3.7.2 Module A.5.10.6 - P_WRITE

P_WRITE, shown in Figure 58 on page 206, processes a WRITE statement. The general form of this statement is

$$\text{WRITE}\ (\left[token\_a_1 = \right]\ token\_b_1\ \left[,\ \left[\ token\_a_2 = \right]\ token\_b_2\ \right] \ldots)\ list$$

6.3.7.3 Module A.5.10.8 - P_PRINT

P_PRINT, shown in Figure 59 on page 207, processes a PRINT statement. The general form of this statement is

$$\text{PRINT}\ format\ \left[,\ list\right]$$

6.3.7.4 Module A.5.10.10 - P_READ

P_READ, shown in Figure 61 on page 209, processes a READ statement. The general form of this statement is

$$\text{READ}\ (\left[token\_a_1 = \right]\ token\_b_1\ \left[,\ \left[\ token\_a_2 = \right]\ token\_b_2\ \right] \ldots)\ list$$

6.3.7.5 Module A.5.10.12 - P_FORMAT

P_FORMAT, shown in Figure 62 on page 210, processes a FORMAT statement. The general form of this statement is

FORMAT (*list*)

6.3.7.6 Module A.5.10.14 - P_OPEN

P_OPEN, shown in Figure 63 on page 211, processes an OPEN statement. The general form of this statement is

OPEN ([$token\_a_1 =$] $token\_b_1$ [, [$token\_a_2 =$] $token\_b_2$] ...)

6.3.7.7 Module A.5.10.16 - P_CLOSE

P_CLOSE, shown in Figure 64 on page 212, processes an CLOSE statement. The general form of this statement is

CLOSE ([$token\_a_1 =$] $token\_b_1$ [, [$token\_a_2 =$] $token\_b_2$] ...)

6.3.7.8 Module A.5.10.18 - P_REWIND

P_REWIND, shown in Figure 65 on page 213, processes an REWIND statement. The general form of this statement is

REWIND *constant*

REWIND ([*token_a*$_1$ =] *token_b*$_1$ [, [*token_a*$_2$ =] *token_b*$_2$] ...)

6.3.7.9 Module A.5.10.20 - P_BCKSP

P_BCKSP, shown in Figure 66 on page 214, processes an BCKSP statement. The general form of this statement is

BCKSP *constant*

BCKSP ([*token_a*$_1$ =] *token_b*$_1$ [, [*token_a*$_2$ =] *token_b*$_2$] ...)

6.3.7.10 Module A.5.10.22 - P_ENDFIL

P_ENDFIL, shown in Figure 67 on page 215, processes an ENDFIL statement. The general form of this statement is

ENDFIL *constant*

ENDFIL ([*token_a*$_1$ =] *token_b*$_1$ [, [*token_a*$_2$ =] *token_b*$_2$] ...)

6.3.7.11 Module A.5.10.24 - P_ENTRY

P_ENTRY, shown in Figure 68 on page 216, processes the ENTRY statement. The general form of this statement is

ENTRY *name* [([arg$_1$ [, arg$_2$] ...])]

6.3.7.12 Module A.5.10.26 - P_IF

P_IF, shown in Figure 69 on page 217, processes the IF statement. The general form of this statement is

IF (*logical_expr*) $label_1, label_2, label_3$

IF (*logical_expr*) *THEN*

IF (*logical_expr*) *expression*

ELSE IF (*logical_expr*) *THEN*

6.3.7.13 Module A.5.10.34 - P_DO

P_DO, shown in Figure 71 on page 219, processes the DO statement. The general form of this statement is

DO *label* [,] *name* = $expr_1$, $expr_2$ [$expr_3$]

6.3.7.14 Module A.5.10.38 - P_CALL

P_CALL, shown in Figure 72 on page 220, processes the CALL statement. The general form of this statement is

CALL *name* [([arg_1 [, arg_2] ...])]

6.3.7.15 Module A.5.10.40 - P_GOTO

P_GOTO, shown in Figure 74 on page 222, processes an GOTO statement. The general form of this statement is

GOTO *label*

GOTO (*label*$_1$[, *label*$_2$] ...) [,] *expression*

6.3.7.16 Module A.5.10.48 - P_RETN

P_RETN, shown in Figure 76 on page 224, processes an RETN statement. The general form of this statement is

RETURN [*label*]

6.3.8 Module A.5.14 - P_END

P_END, shown in Figure 77 on page 225, processes an END statement. The general form of this statement is

END

6.3.9 Subroutine Utilities

6.3.9.1 Module S.1 - CMT_STMT

CMT_STMT, shown in Figure 78 on page 226, processes the comment statements found in the FORTRAN source file. The location of comment statements in the source file are checked to determine if they are part of a contiguous block or if they appear singly. Comment statements which are determined to be contiguous are stored into a buffer so that the number of contiguous comment statements can be determined. Once a non-contiguous comment statement is detected, the buffer is written to a file for permanent storage along with the comment location and block size. The process of searching for contiguous comment statements is then repeated.

6.3.9.2 Module S.2 - TOKENIZE_LIST_1

TOKENIZE_LIST_1, shown in Figure 80 on page 228, processes a character string containing a list of names or tokens in the following form:

$$token_1 \ [, \ token_2] \ ...$$

The string under consideration is scanned for text delineated by commas. As each token is identified, it is stored in an array of type character, which is returned to the user along with the number of tokens found in the string (*ntoken*). One use of this module is to extract the variable names found in the argument list of a SUBROUTINE statement.

6.3.9.3 Module S.3 - TOKENIZE_LIST_2

TOKENIZE_LIST_2 , shown in Figure 81 on page 229, processes a character string containing a list of names or tokens in the following form:

$$token_1[(list_1)] \; [, \; token_2[(list_2)]] \ldots$$

The string under consideration is scanned for tokens delineated by commas, which may optionally be followed by a parentheses delineated list. As the string is processed, the tokens and the list elements are stored in the character arrays *token* and *list*. If a token is not followed by a list element then the *list* element is set to "NULL" to indicate this. The number of tokens found is noted in variable *ntoken*. The most common use for this module is to tokenize the variable lists found in declarations statements like REAL and INTEGER.

6.3.9.4 Module S.4 - GET_PAREN_LIST

GET_PAREN_LIST , shown in Figure 85 on page 233, extracts a character string which is delineated by matching parentheses as follows:

$$(p\_list)$$

The string to be processed must begin with a parentheses and the string *p_list* may contain any number of matching parentheses. The algorithm used, determines matching parentheses by incrementing or decrementing a parentheses counter. When a left parentheses is encountered, the counter is incremented. When a right parentheses is encountered the counter is decremented. The process is terminated when the counter value

becomes zero. A common use of this module is to extract the argument list from a SUBROUTINE statement.

6.3.9.5 Module S.5 - GET_INT_NUM

GET_INT_NUM , shown in Figure 86 on page 234, extracts an integer number from a character string. The module requires a pointer to the starting digit of the number. The string is scanned until a non-digit character is encountered. The module returns both the character and integer representation of the number.

6.3.9.6 Module S.9 - TOKENIZE_LIST_3

TOKENIZE_LIST_3 , shown in Figure 90 on page 238, extracts tokens from a character string of the following form:

$$[\,token\_a_1 =]\ token\_b_1\ [,\ [\ token\_a_2 =]\ token\_b_2\ ]\ \ldots$$

The character to be processed string contains pairs of tokens (*token_a* and *token_b*), the first of which is optional, that are separated by an equal sign. The module determines the number of token pairs *ntoken* and these are stored in the character arrays *token_a* and *token_b*. To indicate that *token_a* is not present in a particular token pair, the corresponding element of the array *token_a* is set to "NULL". This module assists in the processing of the READ and WRITE statements by tokenizing the device/format descriptors.

6.3.9.7 Module S.11 - GET_STRING_1

GET_STRING_1 , shown in Figure 94 on page 242, extracts a substring from a character string. The module requires a pointer to the start of the first substring. The string is then scanned until a delimiter is encountered. The module considers any of the following characters as a delimiter: "*", "=", "(" or "end-of-string". The module returns the substring and the number of characters in the substring. One use of this module is extraction of the subroutine name in a SUBROUTINE statement.

6.3.9.8 Module S.14 - STMT_TEST

STMT_TEST , shown in Figure 94 on page 242, tests a statement to determine if it is an assignment statement, statement function declaration or neither. The general form for an assignment statement is

$$name[(list)] = expression$$

where *name* is the variable or array name, *list* contains the array subscript descriptions and *expression* is a valid arithmetic, character or logical expression. For a simple variable assignment, the subscript *list* would not be present. For an array assignment, the subscript *list* may contain arithmetic expressions, functions references and references to other arrays.

The general form for a statement function declaration is

$$name(list) = expression$$

where *name* is the function name, *list* is an argument list and *expression* is a valid arithmetic, character or logical expression. The statement function must appear in the non-executable section of the subprogram and the argument list must be composed of nonrepeated variable names.

The module begins the identification process by testing to see if the statement contains an equal sign and begins with a valid variable or array name. If the variable name is followed immediately by an equal sign, then the statement is an assignment statement. If the name is followed by a parentheses delineated list, followed immediately by an equal sign, then the statement can be either be an array assignment or a statement function. The variable *name* is next checked to see if it has been declared as an array. If *name* is declared as an array, then the statement is an assignment statement. Otherwise, the statement is a function statement. This module returns two flags, *stmt_func* and *assign_stmt*. These flags are set to either "YES" or "NO" to indicate whether or not the statement was a statement function or an assignment statement.

6.3.9.9 Module S.18 - TOKENIZE_EXPR

TOKENIZE_EXPR, is shown in Figure 104 on page 252. This module operates on a character string which contains either arithmetic, logical or character expressions and extracts a list of variable or array names found in the expression. The algorithm used has four basic steps. In the first step, the string is scanned for any operators which are then replaced with a comma. For example, the arithmetic operators are: "+", "-", "*", "/", "**". Other possible operators are noted in Figure 104. In the second step, all grouping parentheses are removed. Grouping parentheses are matching pairs of parentheses which are preceded by an operator. This step is performed by scanning the string

until a left parentheses is found which is preceded by a comma. Then module GET_PAREN_LIST is used to locate the matching right parentheses. This process is repeated until all grouping parentheses are removed. The third step is to scan the string for any constants (real, integer or character) which are preceded and followed by commas and to delete them. In the last step, the string is "cleaned-up" by replacing all multiple commas by single commas and by deleting all leading and trailing commas. The string is now in a form which can be processed by module TOKENIZE_LIST_3. This module returns the number of tokens found (ntoken), along with the variable names and subscript descriptors stored in character arrays *token* and *list*, respectively.

6.4 *Implementation Considerations*

In compiler development, the process of tokenization is typically referred to as lexical analysis [*AHO*88].

One tool which assists with the development of lexical analyzers and is commonly available with the UNIX operating system is called LEX. LEX is a program generator that produces a C program for lexical analysis. The input to LEX provides the specification for the lexical analyzer and consists of a set of ***regular expressions*** which provide suitable notation for describing the desired tokens. Tokens are simply groups of characters of the source language that logically belong together, like variable names. Associated with each token is an ***action.*** The action of a token consists of a C program fragment. This fragment is executed whenever the regular expression for a given token is matched. In the case where the token is a identifier (or name), the corresponding

action might be to call a function which would place the identifier into a table, if it did not already exist there, and return the table address.

The implementation of several of the toolkit tokenization modules could be assisted through the use of a tool such as LEX.

6.5 *The CAD/CAM Integration Interface*

To accomplish the successful sharing of information extracted by the CASE integration toolkit with the workbench design and analysis tools, an integration interface will be designed. This interface will characterize the integration system to be created and pass extracted information into the CASE design and analysis toolkits. The structure of this integration system will include four major types of information:

- Variable declaration and assignment
- Translation and/or conversion of the data to be shared
- Reading and writing functions
- Subprogram entry points

The interface will also select pertinent information (variables, subprograms, input/output, etc.) from the extracted data and generate entries in the workbench data dictionary or database for later access by the design and analysis toolkits. It effectively acts as a filter determining which information is needed by the toolkits and passing only

this information from that stored by the integration toolkit. Statistical (e.g. number of occurrences, etc.) and structural information (tier chart) about the application codes is also passed to the design and analysis toolkits. The import and export of this information, to and from the workbench database, will be accomplished through some type of CASE standard format. Since this format is usually proprietary, selection of this format will be based on the actual software enlisted.

6.6 Summary

This chapter has presented the requirements and design specifications for the CASE integration toolkit. This toolkit is a vital part of the CAD/CAM CASE workbench and is designed to assist with specific program integration tasks such as identification and categorization of critical integration related information from the CAD/CAM application programs to be integrated. An integration interface, for the sharing of information between the integration toolkit and design and analysis toolkits has also been described. The next chapter discusses the autonomous integration model, a new integration model that provides a higher-level alternative to the integration of CAD/CAM applications. The use of the CAD/CAM CASE workbench and integration toolkit for the design and generation of this model is also presented.

7.0 The High-Level Autonomous Integration Model

7.1 Introduction

It has been thought for many years that the ideal method of CAD/CAM integration would be along the lines of what has been called here, ***rigidly-connected*** coupling. With some in-depth consideration of this idea, however, we may start to realize that this is not entirely true. In rigidly-connected coupling as you will recall, two or more separate programs are "married" together to form one program in which there is now only a single user interface and a single method of execution. The original programs actually loose their individual identities to become a part of this new program. While this creates the most efficient merging of these programs possible (if it can be accomplished), it actually creates more long-term problems than it solves. Anyone familiar with CAD/CAM software systems will know that they remain in a constant state of change. This not only refers to simple bug fixes, enhancements and updates for compatibility with ever

changing hardware and software environments but also to more radical changes like new methods of geometric modeling and analysis, new companies with new software and old companies which go out of business. Very few companies use all the same software systems for more than five to ten years at a time and most much less than this. With this in mind, consider again the rigidly-connected coupling of two or more programs which may take years to complete. What happens if one of the codes needs to be changed, removed or replaced? The software is no longer in its original form, so who is responsible for making these changes now; is it the developing organization or the integration organization? It is not hard to see that this could be just short of a nightmare and may require the integration to be restarted from scratch depending on the complexity of the change. For these reasons, it seems much more feasible to consider some form of integration which allows individual programs to be integrated and yet maintain their own identity and autonomy. This would allow easy change or modification of these programs with minimum impact on the application programs.

In chapter 4, existing methods of joining CAD/CAM application programs as well as ways to assist in the automation of their creation were discussed. It is, however, quite obvious that most of these methods are rather limited in their capabilities, provide little flexibility and the majority represent rather low-level ways of joining different application programs. They also do not provide high-level options such as the capability to view menus of one program while executing the other or starting functions of one program without terminating the execution of the other program, and few address the problems of integrating across multiple platforms or multiple locations. In many cases, these higher-level considerations are as important as the lower-level ones involving data transfer.

With the previously stated criteria in mind, consider a method of integrating application programs which would approach the problem from a higher-level and allow some type of simultaneous execution of multiple programs which can be displayed for an individual user. If a facility is then added to allow these two programs to share data and information through menu selections, perhaps, an acceptable, higher-level method of integration could be achieved. This research asserts that the best and most practically applicable solution to this problem would be through the use of a window system. Window systems allow the display of multiple input and output areas, or windows, which can act as virtual terminals running independent client application programs with a high degree of device independence. These systems additionally provide features for simultaneous program execution as well as application inter-communication. While window systems, including X, do not provide integration of the type previously described, they can be used to enable the creation of a system which would support the requirements that have been given.

The X Window System, developed jointly by MIT and Digital Equipment Corporation, has been chosen for use in this investigation. The reasons for this selection are primarily because X has become a de facto standard over the past several years and it does not require the user to adapt some previously defined user interface but provides the tools to create one.

7.2 *The Autonomous Model and its Components*

In considering a new integration model, it is important to first identify the necessary requirements which need to be considered. This list of the primary end-user requirements of the new integration model should include the following:

- Complete transfer of necessary data and information
- Autonomous integration of application programs allowing easy change and modification of individual programs with low impact on the application programs
- The capability to view menus of one program while executing the other
- The capability to start functions of one program without terminating the execution of the other program
- Integration across multiple platforms and multiple sites

The new autonomous model described can meet and in several ways exceed the requirements listed above through the use of an executive program, based on X, which invokes simultaneous execution of two or more CAD/CAM application programs that can be displayed in different windows at the network server. The autonomous model additionally allows these programs to communicate with each other through inter-client communications. It is also assumed that these programs will be integrated with a central database for storage and retrieval of data and information. A diagrammatic representation of the autonomous model is shown in Figure 10 on page 128.

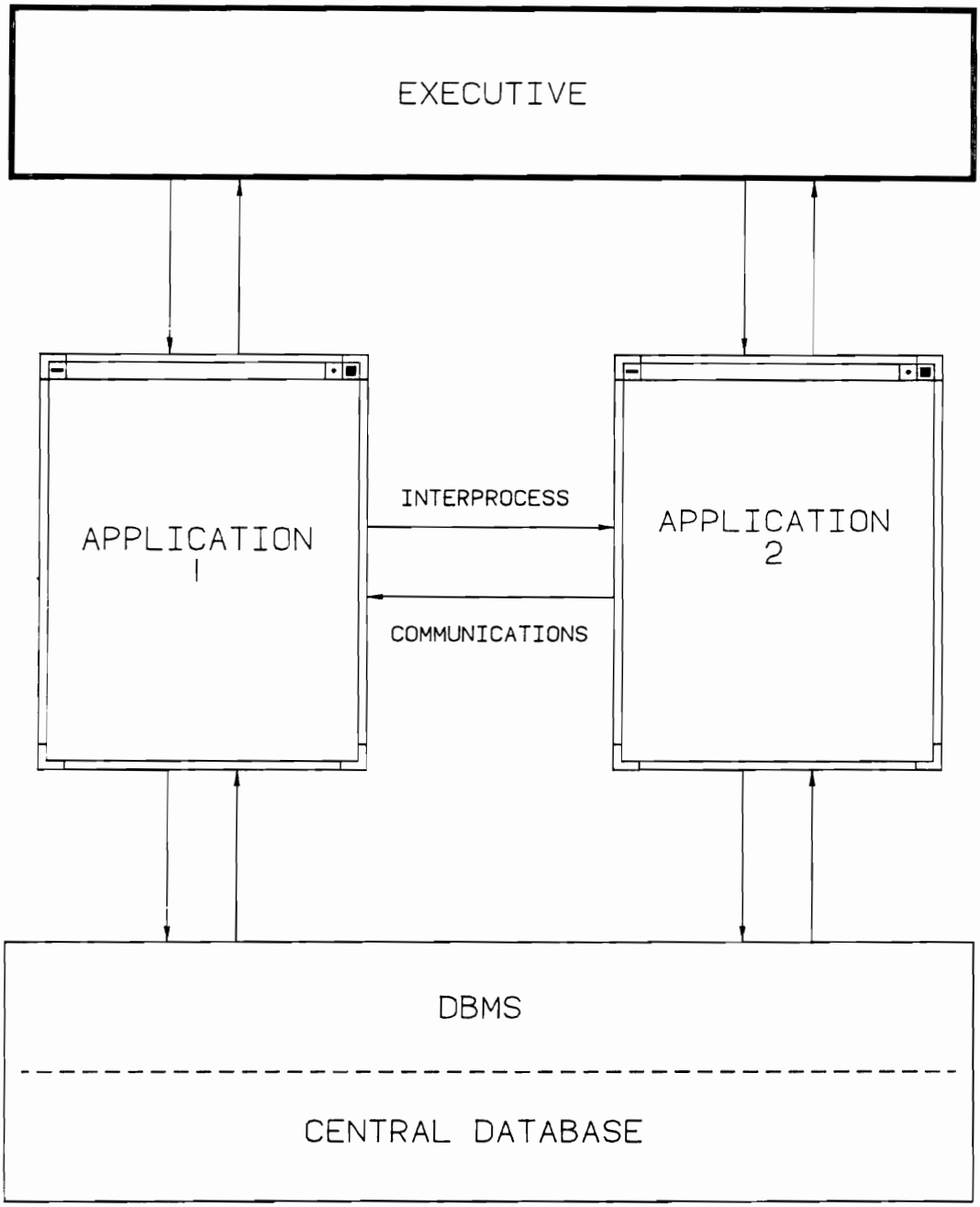


Figure 10. The High-level Autonomous Integration Model (HAIM)

In this model, the input and output functions of the individual application programs are the main program operations which must be modified through the help of the CAD/CAM CASE workbench and integration toolkit. This fact allows for a minimum of source code modification as well as a large degree of individual application program autonomy. Through X we also have the ability to suspend or continue the action within application windows, thereby allowing us full control of program execution. Finally, the integration of application programs which do not reside on the same processor is possible as long as each client system supports the X network protocol. This is a very important advantage since in reality actual CAD/CAM users are frequently sharing data and information between different computer systems (PC, mainframe, workstation) that are often manufactured by different companies and may reside at different sites as well as in different states or even countries. The autonomous model will provide the CAD/CAM application user with not only a device independent integration method but also a site independent integration method. A schematic of the multi-platform support which will be provided for the autonomous integration model by the X network protocol is shown in Figure 11 on page 130.

The autonomous integration model consists of two separate components, these are:

- The System Executive

- Inter-Client Communications

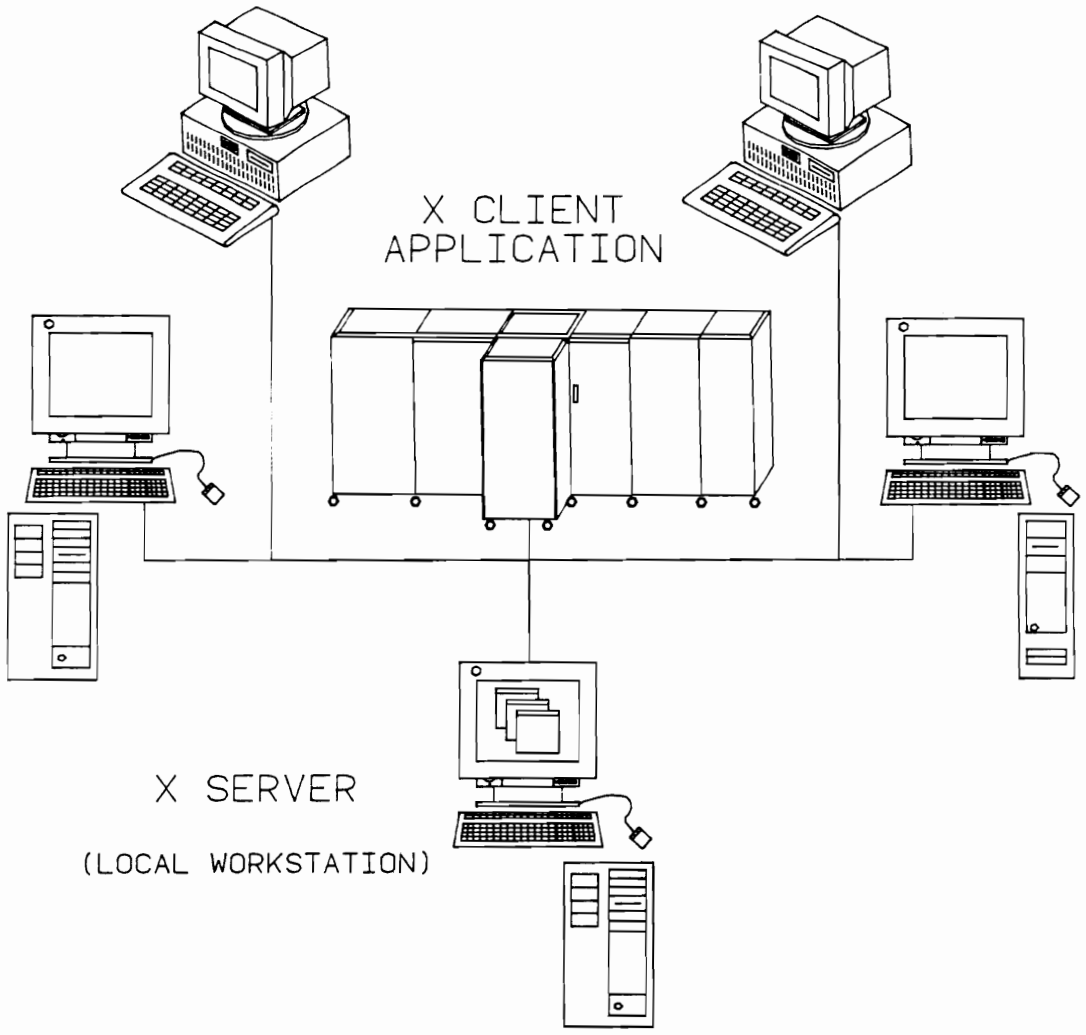


Figure 11. Multi-Platform Support Enabled by the X Window System

7.2.1 The System Executive

The system executive will provide a window-based multitasking environment for the concurrent execution of the target applications. It will be designed using the capabilities of X as well as knowledge of the general operation of CAD/CAM application programs. The system executive will undertake the task of starting the CAD/CAM applications and displaying them in an acceptable manner at the network server regardless of where they reside. The tasks of the system executive will rely not only on the capabilities of the X Window System but also on the hardware and software characteristics considered. The main tasks that will be performed by the system executive are scheduling, window management and system interfacing.

The executive scheduler will provide the multitasking support for the concurrent execution of application programs. It will control their invocation as well as termination and also maintain the interactive nature of each program. The scheduler will be capable of suspending or resuming the execution of any application program as it deems necessary.

The executive window manager will provide graphics support through the use PHIGS or Xlib for the complete control of windows (e.g. resizing, moving, hierarchy, etc.). It will also maintain rules for window characteristics which it can enforce upon CAD/CAM applications that attempt to open their own windows on the network server. A large task that will be performed by the window manager is the interpretation and handling of input from the keyboard, mouse or other input devices. The window manager will additionally allow for iconifying and deiconifying of windows. Since the IBM RISC/6000 will be utilized in this research, some of the window management features

that will be required may already be available on this hardware. This may reduce the amount of window manager design required.

The system interface will provide all communications between the system executive and the underlying AIX operating system. X will perform many of these operations inherently.

7.2.2 Inter-Client Communications

Inter-client communications will enable application programs to communicate directly with each other as well as exchange and share data through the use of the X Window System. A primary goal for the autonomous integration of CAD/CAM applications is to facilitate this sharing of information while these applications are concurrently running. Inter-client communications will provide for this sharing at two different levels:

- Direct Communications - where data is passed directly from process to process (a technique suitable for relatively small quantities of information).

- Indirect Communications - where a common database address is passed between processes instead of the actual information (this technique is suitable for transferring larger quantities of information).

In every CAD/CAM application, permanent, high integrity archiving of important drawings and other information is a vital necessity. While data files and interfaces are frequently used, they do not provide the versatility of the DBMS approach. Most large CAD/CAM applications, always enlist a relational database and some type of database

management system which can often be used to store other related project information (e.g. CADAM, CATIA). The results of the CAD/CAM integration survey also show that when dissimilar applications share a common database, the problem of integration is minimal and overall integration can be accomplished quite successfully. By modifying dissimilar applications to share a common database, the largest part of the data integration problem can be solved. A good example of this process is discussed by [Rowe88]. In this paper (also mentioned in the literature review) a program is discussed which manages the modification of complex analysis codes to utilize a common relational database for the purposes of integration. The project has been so successful, that it is now being used by several NASA agencies as well as a few private aerospace companies. For these reasons, it will be assumed that the CAD/CAM applications programs to be utilized in the autonomous integration model will first be modified to share a common database through the use of the CAD/CAM CASE workbench as discussed in chapter 4. This will, it is hoped, allow the most satisfactory integration possible.

Although a common database could reside anywhere on the communications network, the most reliable location would likely be on a network mainframe processor. Workstations may not possess the hardware capabilities to handle such large amounts of data and do not usually possess the same level of reliability for this volume of data, at present.

7.3 *Application of CASE Methods to the Autonomous Model*

As shown in previous chapters, CASE methods can assist with the integration of CAD/CAM application programs in many ways. These methods can also be applied to the creation of the high-level autonomous integration.

To accomplish this type of integration, the following information which is centered around program input and output, must be known about each program to be integrated:

- Occurrences of all READ statements (number and location),
- Variables being read in,
- Variable types,
- Specifiers used in READ (if any),
- Occurrences of all WRITE statements (number and location),
- Variables being written out
- Variable types,
- Specifiers used in WRITE (if any),
- Query language database communications (specific to database used) for all types of operations such as SELECT, INSERT, DELETE, UPDATE, etc.,
- Logical units in use,
- Labels in use, and
- Comments that may precede the READ and WRITE statement or appear at the beginning of the program and include variable definitions or other important information and descriptions.

As with the case of freely connected coupling, it may also be necessary to share information between programs which is not currently written to a file or database. For the identification of this information, much more must be known about the variables themselves. This may include:

- Variable or array type and number of occurrences,
- Location of variable or array occurrences,
- Variables equivalenced to,
- Location of assignment(s),
- Variables declared in a COMMON statement, or perhaps even
- Variables passed in a subroutine call.

Once again, the CASE integration toolkit can be called upon to provide this information as well as produce tier charts showing overall program structure.

When all necessary evaluation is complete, the system designer can use the information obtained, along with the other CASE toolkits, to design necessary code additions or output statements in the same way as described previously. CASE design capabilities will also greatly assist in the tedious generation of database query language calls to read from or write to the common database. After the design of necessary code modifications or additions is finished, the CAD/CAM CASE workbench also provides automatic generation of C source code from the graphical designs, for any procedures which must be added to the original code.

Another important task of the workbench is the generation of the system executive for the autonomous integration model. This system executive will allow the invocation and simultaneous execution of the CAD/CAM application programs to be integrated. Ru-

dimentary X interface code must also be added to the client application programs for inter-client communications and possibly for client communication with the server. Since these problems will be highly specific to the X Window System, a specialized tookit referred to as the HAIM System Executive and X Support toolkit will manage the design and creation of the necessary code. The developed code for these tasks may be very similar from one integration to the next and may, in large part, be reusable thus saving much development time and designer effort.

Maintenance of all completed code, including version control, and documentation of the new integration system, can also be assisted by the workbench.

7.4 Summary

This chapter has presented a new, higher-level model for the integration of CAD/CAM application programs referred to as the high-level autonomous integration model (HAIM). The autonomous model provides integration of CAD/CAM application programs through the use of a system executive which facilitates simultaneous execution of two or more programs while also supporting communication between these programs as well as with a common database. The application of CASE methods to assist in the generation of this integration model is also discussed. The next chapter discusses the required skills and major tasks of the integration system designer.

8.0 Functions of the Integration System Designer

In this section, the necessary skills and tasks required of the integration system designer will be discussed. The importance of this person in the integration process cannot be understated. Considering the complex tasks which must be performed, the involvement of a skilled systems integrator will most likely always be a required part of any integration process. The actual tasks of a systems integrator, who performs any type of application integration have always been hard to define. There seems to be no general agreement on what this person actually does. To avoid this confusion, these issues are addressed here.

8.1 Necessary Skills and Background Level

The job of any systems integrator is a complex one but it is also a primary factor in the success of any software system. To be fully prepared to integrate dissimilar CAD/CAM systems with confidence, the integration system designer mentioned in this research,

must have or obtain knowledge about and experience with a large variety of technologies and tools related to CAD/CAM and CASE. These include the following:

- CASE Technology
  - Experience with integrated CASE workbench environments
  - Pertinent structured methodology knowledge
  - Complete understanding of essential diagramming rules
  - Experience with automatic code generation from CASE designs
- Computer Languages and programming
  - The language generated by the workbench
  - The language which the application programs are written in
- Database technology
  - Will require familiarization with specific query languages and databases depending on the applications to be integrated
- General knowledge of standard or common neutral file formats
  - IGES/PDES
  - Proprietary or other common CAD/CAM interfaces

It is also important for the systems integration designer to have good communication skills so that the end-user requirements are completely understood before the integration process begins. The system designer will also have to spend time becoming familiar enough with the application programs themselves to best understand and plan for the new integration system. Finally, a working knowledge of the X Window System and X protocol will be necessary for generating the high-level integration model if it is desired.

8.2 *Overview of Required Tasks*

The tasks required of the system designer are many and become quite involved and varied as the task of generating an integration system progresses. The many duties of the integration system designer have been mentioned throughout this dissertation. To provide the synopsis of these tasks, an example integration process will be considered here to allow the description of the integration system designer's duties in context.

Consider the integration of two different CAD/CAM application programs, Program A and Program B. The modification of these programs for communication with a common database is the primary objective.

The duties of the system designer in the generation of a suitable integration system for this problem are as follows:

- The system designer first meets with the application programs representatives or committees to obtain the source codes for the application programs and to ask necessary questions. Selection of specific common database (or file format) must also be decided at this time. Identification of fundamental program differences, such as units coordinate systems or geometric definitions must also be determined by the system designer during this initial program investigation.

- The system designer next uses the CASE Integration toolkit to parse the Program A and Program B application programs for identification of all necessary data. Determination of variables, functions and other necessary information is stored in

sequential files and/or a workbench database for later use and review by the integration system designer.

- Using the analysis, information modeling and design tools, within the CASE workbench, as well as the assistance provided by the CAD/CAM integration interface, the necessary integration system design is graphically created by the designer.

- The Graphical designs are now used by the code generator to create C source code. Since this is a very interactive process like the analysis and design phases, the system designer must constantly interact with the code generator to accomplish the final source code creation.

- If data or unit conversion is required, it must also be designed by the designer at this time and incorporated into the integration system. As previously mentioned, the C source builder will be capable of storing this type of code in a library for later reuse by other application integrations.

- Now that the software for the integration system has been completed, the designer must next make the necessary modifications to the original source programs to incorporate it.

- The designer next compiles, debugs and tests the new integration system. Subsequent changes and modifications to the integration system can be performed on the workbench at any time.

- The system integration designer is also responsible for documenting the new integration system, both from a technical level and a user standpoint. The CAD/CAM CASE workbench will also assist with this task.

- Once released, the new code may still be cataloged and maintained by the workbench which also provides version control.

8.3 Summary

This chapter has described the necessary skills and tasks of the integration system designer. A brief example enumerating many of these tasks has also been given. The next and final chapter of this dissertation summarizes this research and provides a brief description of future work.

9.0 Research Summary

This research has presented an easier more efficient way to generate CAD/CAM integration systems through the use of computer aided software engineering (CASE). A specialized CASE workbench, has been described which assists an integration system designer in the creation of an integration system for dissimilar CAD/CAM application programs. The CASE workbench provides coverage for all phases of the software development life cycle including the planning, system analysis, system design, programming and maintenance of the developed system.

An important tool of the CAD/CAM CASE workbench, the CASE integration toolkit, has also been designed in this research. This toolkit provides phase-level tools that focus on the quick analysis and identification of application program characteristics that are important for integration. These include detailed program information which is not normally available to the designer such as comment blocks and their locations, explicit information about the occurrence and location of variables, arrays and subprograms as well as detailed information about input and output operations. The complete design of this toolkit has been included, and appears in Appendix B. Toolkit information will be later accessed by an integration interface which will generate a skeleton of the inte-

gration system structure and will provide it along with pertinent integration toolkit information to the CASE design and analysis tools. This interface will serve as a driver for integration program design and will also generate necessary data dictionary entries for identified integration system variables.

To gain a better understanding of existing integration methods as well as the entities, functions and other types of information which need to be shared between different CAD/CAM applications, an industry-wide survey was conducted which involved the participation of a number of major CAD/CAM users. The participants from these Fortune 500 companies were asked questions about their current CAD/CAM systems, their methods of integration, the type of information they are sharing between applications and their future plans and needs in this area. The results of this survey provided a good basis for many of the ideas in this research and assisted with the specification of tools required by the CAD/CAM CASE workbench and integration toolkit. It also provided valuable information which in being used by other research efforts.

A new model for the integration of dissimilar CAD/CAM application programs has also been described. This new integration model will be capable of sharing not only data, such as geometry and analysis results, but also system menus and functions as well. The generation of this new integration model will be accomplished through the use of the CAD/CAM CASE workbench, the integration toolkit and other specialized workbench tools.

Also discussed were the integral responsibilities of the person who utilizes the CAD/CAM CASE workbench, the integration system designer. A skilled and knowledgeable system designer is critical to the successful and efficient application of CASE tools to any integration problem.

9.1 Future Research Plans

This work is part of a three year project which will be implemented and refined through December of 1992.

Future work will be divided into three major areas:

- The design and implementation of the autonomous integration model,
- The design and implementation of the CAD/CAM integration interface, and
- The implementation of the CASE integration Toolkit Design

References

[*Aho*88] Aho, A. V., Sethi, R., and Ullman, J.D., ***"Compilers - Principles, Techniques, and Tools,"*** Addison-Wesley, 1988.

[*Balz*85] Balzar, R.M., "A 15 Year Perspective on Automatic Programming," ***IEEE Transactions on Software Engineering***, Vol. 11, No. 11, Nov. 1985.

[*Bird*85] Bird, M., and Schofield, N., "A Practical Approach to Software Engineering by Using an Interaction Handler and Skeleton Code Generator," ***Computer Aided Design (GB)***, Vol. 17, No. 8, pp. 374-8, Oct. 1985.

[*Brad*83] Bradford, M. S., and Wellington, J., "IGES, A Key Interface Specification for CAD/CAM Systems Integration," ***National Bureau of Standards***, January 1983.

[*Brow*88] Brown, A. W., "Integrated Project Support Environments," ***Information and Management***, Vol. 15, pp. 125-134, 1988.

[*CADR*89*a*] CADRE, "TEAMWORK Product Overview", CADRE Technologies Inc., Teamwork Div. Providence, RI, 1989.

[*CADR*89*b*] CADRE, "Unified CASE UC1100 C Source Builder," ***CADRE Microcase Div. Ref. #900613,*** CADRE Technologies Inc., Beaverton, OR, May, 1989.

[*Date*90] Date, C. J., ***"An Introduction to Database Systems, Volume I ,"*** Addison-Wesley, 1990.

[*Fish*88] Fisher, A. S., ***"CASE Using Software Development Tools,"*** Wiley and Sons, 1988.

[*Gill*85] Gillenson, M. L., ***"Database Step-by-Step"***, Wiley and Sons, 1985.

[*IBM*89] International Business Machines Corp., "Engineering Management for CIM, Product Engineering Support and Computer-Aided Design Integration - General Information," ***IBM Technical Document #GH12-5619-0***, Stuttgart, Germany, 1989.

[*IBM*90] International Business Machines Corp., "Advanced Interactive Executive/CASE", ***IBM Document #G320-9939-00***, White Plains, NY, 1990.

[*Inte*90] Interleaf, Inc., "Interleaf Technical Publishing Software 4," ***Interleaf Document #PN 72018-001***, Interleaf, Inc., Cambridge, MA, 1990.

[*Jaya*89] Jayaram, S., and Myklebust,A. "Automatic Generation of Geometry Interfaces Between Applications Programs and CAD/CAM Systems," ***Computer-Aided Design***, December 1989.

[*Jaya*90] Jayaram, S., and Myklebust,A. "CADMADE-Towards a Standard Environment for the Creation of Design and Manufacturing Software," ***Proceedings of ICED***, Dubrovnik, Yugoslavia, August 1990.

[*Liew*82] Liewald, M.H., and Kennicott, P.R., "Intersystem Data Transfer via IGES," ***IEEE Computer Graphics & Applications***, May 1982.

[*Lu*87] Lu, L.J., Myklebust, A., and War, S., "Integration of a Helicopter Sizing Code with a Computer-Aided Design System," ***Journal of the American Helicopter Society***, October 1987.

[*Luke*86] Luker, P.A., and Burns, A., "Program Generators and Generation Software," ***The Computer Journal***, Vol. 29, No. 4, 1986.

[*Manu*87] Manuel, T., "Integration is Crucial to CASE's Future," ***Electronics***, Vol. 60, No. 19, pp. 77-80, Sept. 17, 1987.

[*Mars*86] Marshall, J., and Van Dyne, D., "Integrating CAE, CAD and CASE," ***Digital Design*** Vol. 16, No. 7, pp. 40-2, 44, 46, June 1986.

[*Mart*85] Martin, R., and Downs, J.E., "The Role of Data in CADCAM Integration," ***Computer-Aided Engineering Journal (GB)***, Vol. 2, No. 3, pp. 97-101, June 1985.

[*Maye*87] Mayer, R.J., "IGES, One Answer to the Problems of CAD Database exchange," ***BYTE Magazine***, Vol. 12, No. 6, June 1987.

[*McCl*89] McClure, C. ***"CASE is Software Automation,"*** Prentice Hall, 1989.

[*Need*90] Needleman, R., ***"Info World: Understanding Networks"***, Symon and Schuster, 1990.

[*Nye*88] Nye, Adrian, ***"Definitive Guides to the X Window System, Volume One - Xlib Programming Manual,"*** O'Reilly & Assoc., Inc., 1988.

[*Nye*90] Nye, Adrian and O'Reilly, Tim, ***"Definitive Guides to the X Window System, Volume Four - X Toolkit Intrinsics Programming Manual,"*** O'Reilly & Assoc., Inc., 1990.

[*Pase*85] Pasemann, K., "Interfaces for CAD/CAM-Integration," ***CAMP'85: Computer Graphics, Applications for Management and Productivity - Proceedings***, Berlin, Germany, Sept. 24-27, 1985.

[*Purd*90] Purdum, J., ***"C Programming Guide"***, Que Corp., 1988.

[*Rama*86] Ramanathan, J., "Intelligent Integration of Existing Engineering Tools," ***Proceedings of the Fifth Annual Control Engineering Conference***, Rosemont, IL, May 6-8, 1986.

[*Rowe*88] Rowell, L. F., Schwing, J. L., and Jones, K. H., "Software Tools for the Integration and Execution of Multidisciplinary Analysis Programs," ***AIAA/AHS/ASEE Aircraft Design, Systems and Operations Meeting*** , Atlanta, Georgia, Sept. 7-9, 1988.

[*Shin*88] Shin, D.G., and Maryanski, F., "Knowledge-Aided Engineering Environment for Design and Manufacturing," ***COMPSAC 88: The Twelfth Annual International Computer Software & Applications Conference***, Chicago, Illinois, Oct. 5-7, 1988.

[*Taus*76] Tausworthe, Robert C., ***Standardized Development of Computer Software, Part I - Methods***, Jet Propulsion Laboratory, SP 43-29, July 1976.

[*Taus*78] Tausworthe, Robert C., ***Standardized Development of Computer Software, Part II - Standards***, Jet Propulsion Laboratory, SP 43-29, Part II, August 1978.

[*That*88] B. R. Thatch and A. Myklebust, "A PHIGS-Based Graphics Input Interface for Spatial Mechanism Design," ***IEEE Computer Graphics and Applications***, Vol. 8, No. 2, March 1988.

[*Wamp*88] Wampler, S.G., Myklebust, A., Jayaram, S., and Gelhausen, P., "Improving Aircraft Conceptual Design - A PHIGS Interactive Graphics Interface for ACSYNT," AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, Atlanta, GA, Sept. 7-9, 1988.

[*Youn*90] Young, Douglas A., ***"The X Window System, Programming and Applications with Xt,"*** Prentice Hall,Inc., 1990.

Uncited References

Buckley, F. J., ***Implementing Software Engineering Practices,*** Wiley and Sons, 1989.

Department of Defense, "Defense System Software Development," DOD-STD-2167A, 1988.

Date, C. J., ***"Relational Database: Selected Writings,"*** Addison-Wesley, 1986.

Dart, S. A., Ellison, R. J., Feiler, P. H. and Habermann, A. N., "Software Development Environments," ***IEEE Computer***, pp.18-22, Nov., 1987.

Gibson, M. L., "The CASE Philosophy," ***BYTE***, Vol. 14, No. 4, pp. 209-218, April, 1989.

Jenkins, A.M., "Surveying the software generator market," ***Datamation***, Vol. 31, No. 17, pp. 105-7, 110-11, 114-15, 118-20, Sept. 1, 1985.

McClure, C., "The CASE Experience," ***BYTE***, Vol. 14, No. 4, pp. 235-244, April 1989.

QED Information Sciences, Inc., ***"CASE - The Potential and the Pitfalls,"*** QED Information Sciences, Inc., 1989.

Voelcker, J., "Automating Software: Proceed with Caution," ***IEEE Spectrum***, Vol. 25, No. 7, pp. 25-27, July 1988.

Appendix A. CAD/CAM Integration Survey Form

Industry Survey of CAD/CAM Integration

Virginia Polytechnic Institute
CAD/CAM Laboratory
114E Randolph Hall
Blacksburg, VA 24061

Dr. Arvid Myklebust
Professor and Director, VPI CAD/CAM Laboratory
(703)231-7195

Sandra Pennington
Research Associate
(703)231-7457

Your assistance in the efforts of this research are greatly appreciated.

1. **What type of processes or tasks are performed on CAD/CAM systems within your organization? (e.g. design, analysis, manufacturing, etc.)**

2. **What type and make of hardware are you running these systems and applications on? (e.g. mainframe, workstation, etc., and manufacturer)**

3. List the software systems and/or applications programs you are currently using (i.e. CADAM, IDEAS, in-house programs, etc.). Identify those applications which are commercially available and those which were custom written. If large numbers of applications programs are used, please give general information such as the number of programs, all the different types of applications these represent and whether these are commercially available or custom written or a combination of both. (e.g. 5 analysis applications - 2 commercial, 3 custom, etc.)

4. To what extent are the functions of CAD integrated in your organization? (e.g. design, analysis, etc.)

Please answer the following questions (a-e) with reference to CAD integration in your organization:

a. Do you have an automatic way to pass information between one software system and the next?

b. Are there any significant problems associated with this method?

c. What type of information are you sharing?

d. If no automatic or semi-automatic method of information sharing is used, how do you get the information from one software system to another?

e. What type of information are you currently unable to pass between software systems that you would like to be able to share?

5. To what extent are the functions of CAD integrated with those of CAM? (How does CAD final design data get transmitted to the manufacturing floor?)

6. **Have you used the IGES standard? If yes, what has been your experience using IGES? What type of information are you using IGES to pass?**

7. **Are you counting on future releases of IGES or the release of PDES to solve any of your present software integration problems?**

8. **How many interfaces or translators do you use which are not IGES format?**

9. **Have software integration problems affected the productivity of your CAD/CAM software systems? If yes, please give an example.**

10. **If you are currently using custom software systems and/or applications programs, did you develop them in-house or have them developed outside your organization?**

11. **What language are these custom applications written in? Please list the language and the percent of custom applications which apply. (e.g. Fortran - 80%)**

12. **Do you have access to the source code for these custom applications or do you receive only the executable? Please give (or estimate) the total number of custom applications which you use and approximate the percentage of these for which you have source code.**

13. **Do these custom applications use interactive computer graphics? Please approximate the percentage of the total number of applications given above in question #12 which utilize interactive computer graphics.**

14. **Do these custom software systems utilize either the GKS or PHIGS standard? Please approximate the percentage of the total number of applications given above in question #12 which utilize either GKS or PHIGS.**

15. **If you are currently considering the development of a custom software system of some sort, will you utilize the graphics standards GKS or PHIGS? Why or why not?**

16. **If there are any additional comments you wish to make regarding the integration of CAD/CAM applications within your company, please include them here.**

Please fill in the following information completely or attach your business card. This information will not be included in the tabulation of survey results but will allow us to contact you should we need additional information. Thank you again for your help.

Company: _____

Name: _____

Title: _____

Address: _____

_____

_____

Phone: _____

Appendix B. Flowcharts for the VPI CASE Integration Toolkit

This section contains the program specification flowcharts for the CASE Integration Toolkit. All flowcharts arranged by module number. Primary charts have numbering which begins with the letter "A" followed by their dewey decimal expansion. Subroutines which are used in multiple locations have numbering which begins with the letter "S" and follow the primary charts in this appendix. The flowcharts are also listed by module number and name in the list of illustrations which follows the table of contents.

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: MAIN_PROG	MODULE: A
DESIGNED BY: SANDRA PENNINGTON	NOTE: DESCRIBES THE OVERALL PROGRAM STRUCTURE
DATE: FEBRUARY 1991	

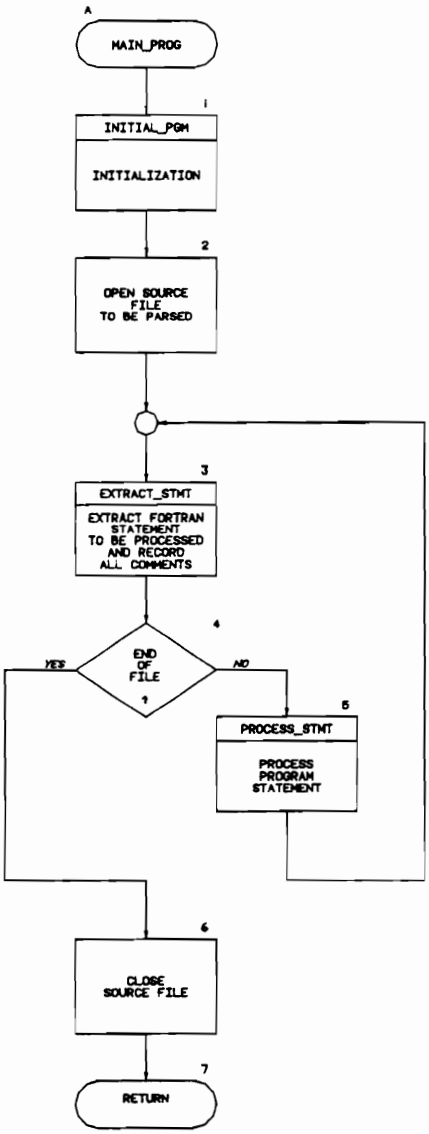


Figure 12. Module A - MAIN_PROG

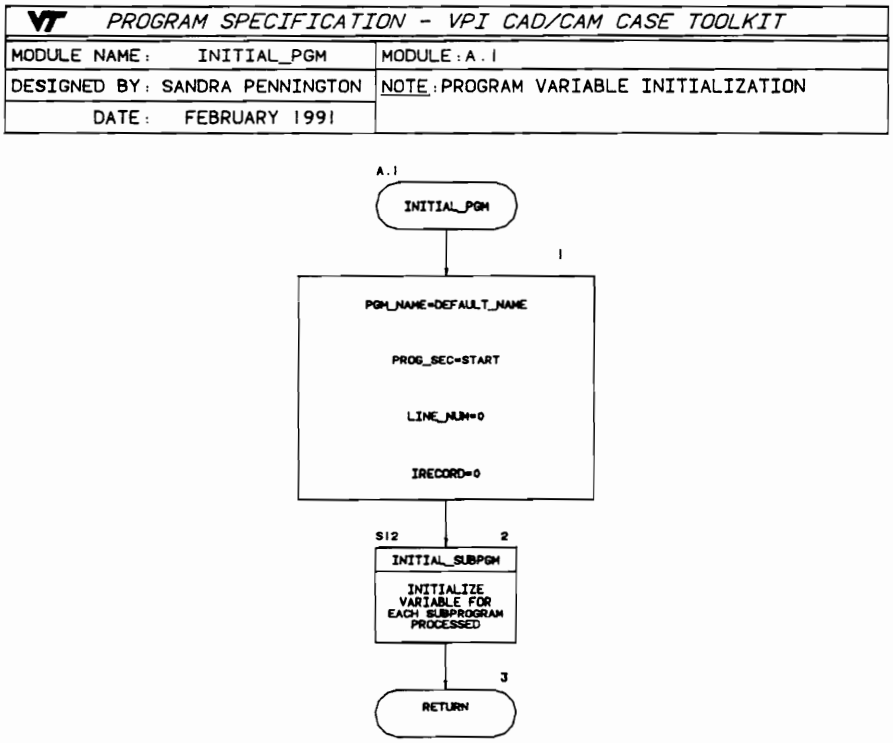


Figure 13. Module A.1 - INITIAL_PGM

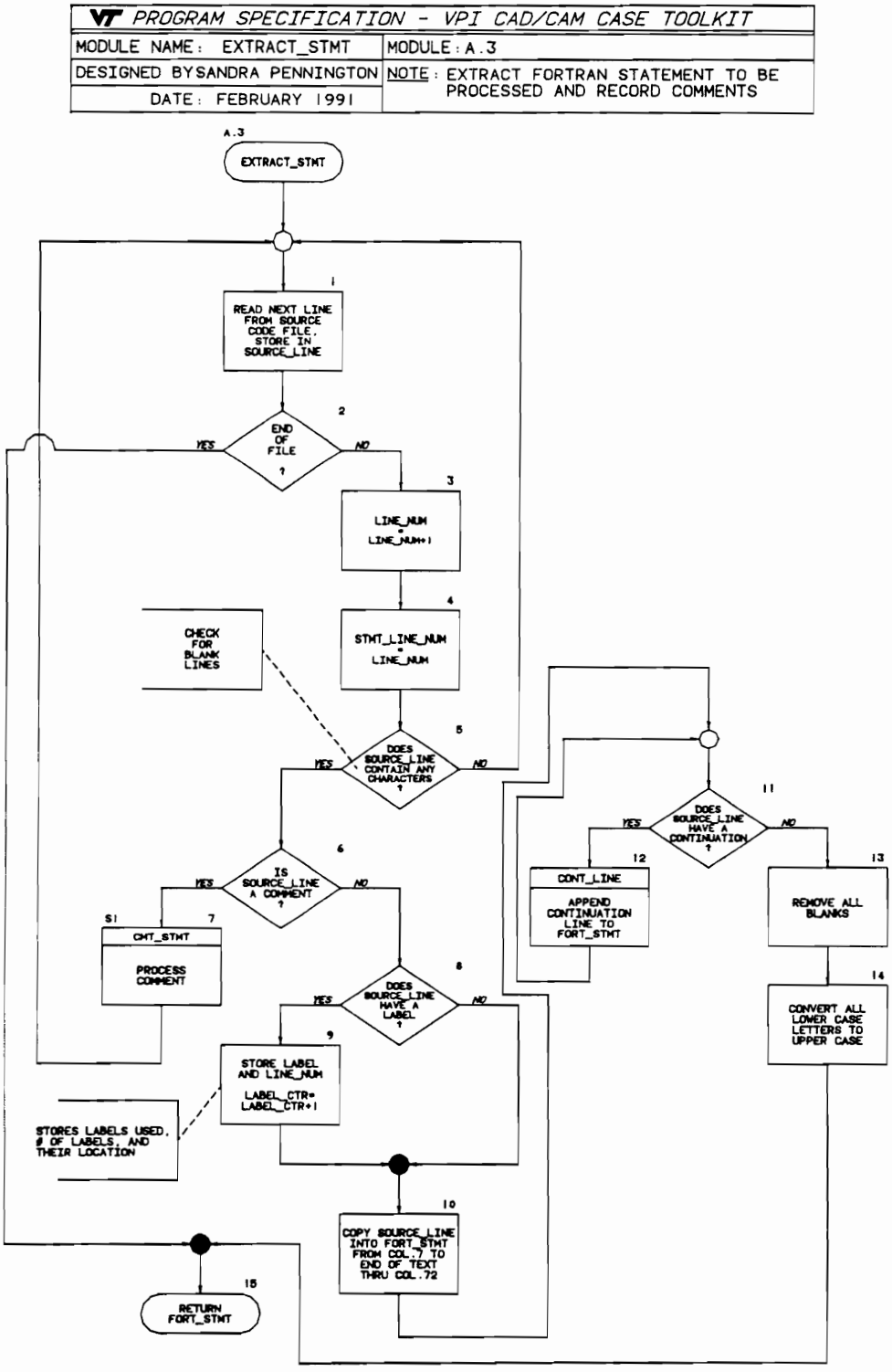


Figure 14. Module A.3 - EXTRACT_STMT

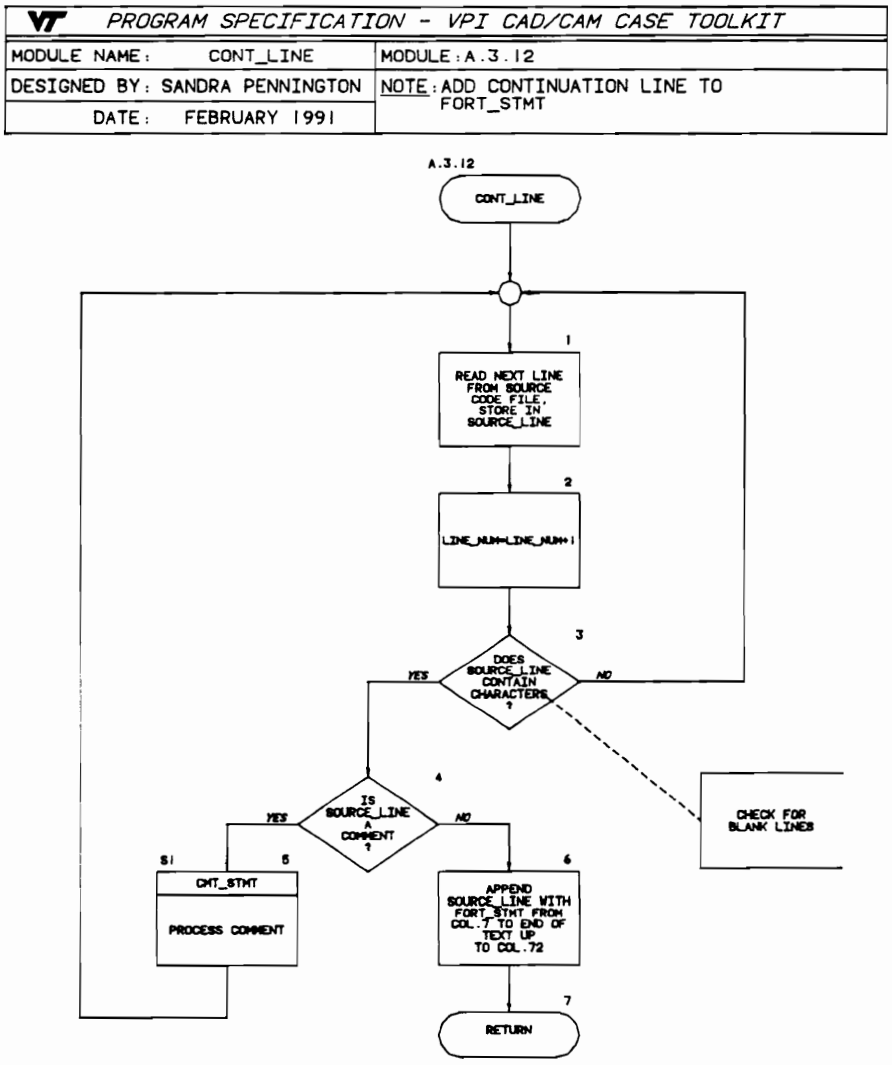


Figure 15. Module A.3.12 - CONT_LINE

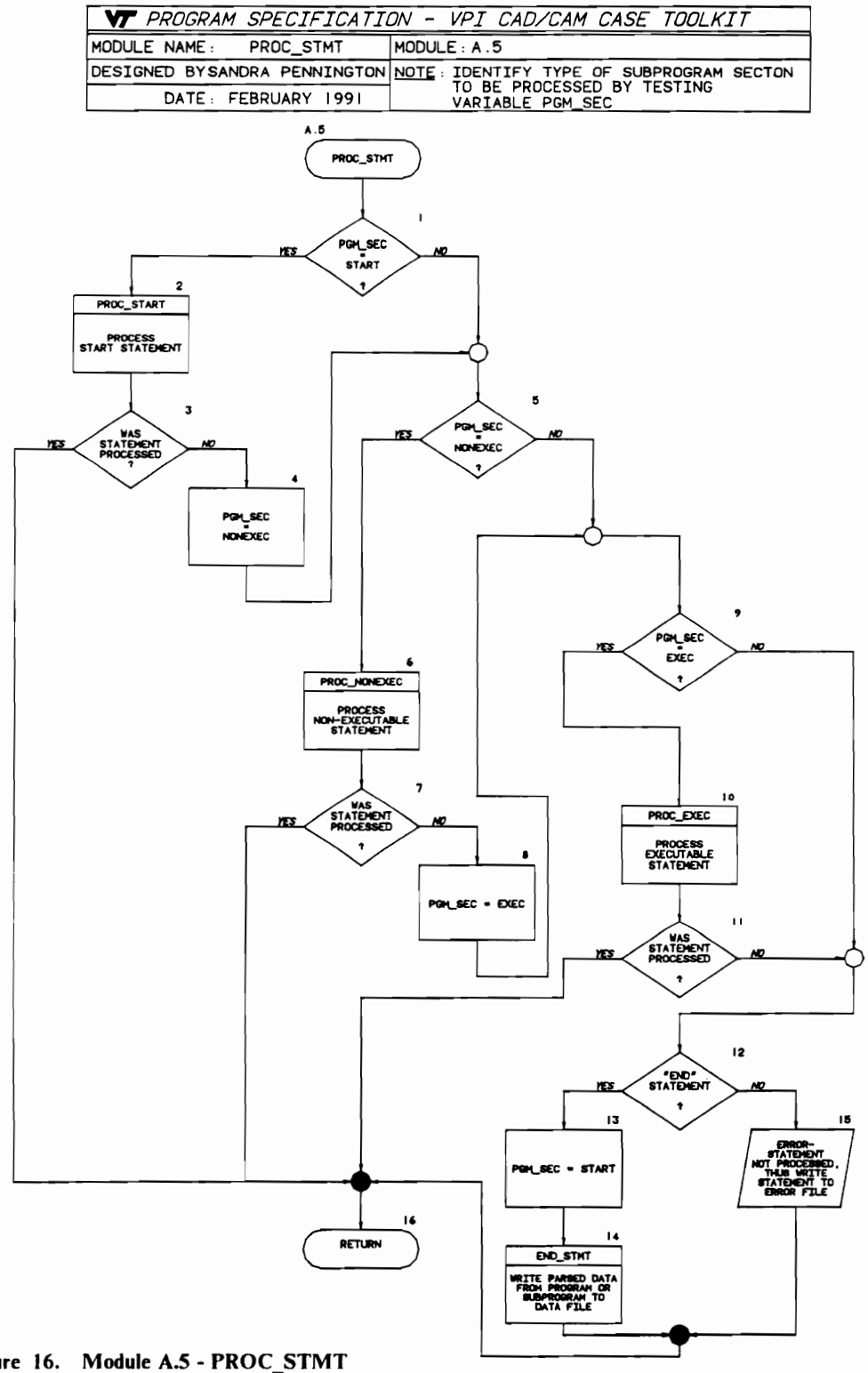


Figure 16. Module A.5 - PROC_STMT

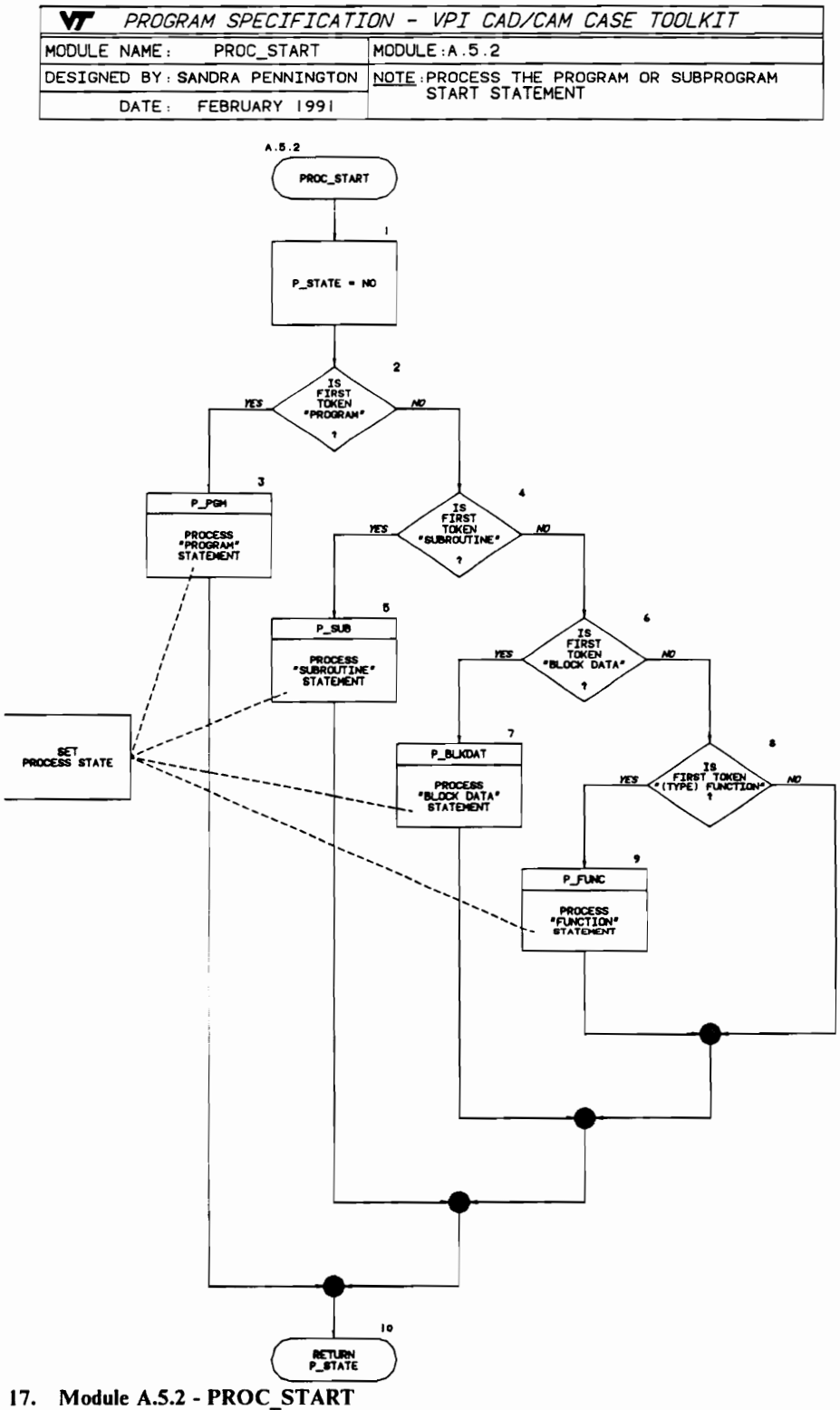


Figure 17. Module A.5.2 - PROC_START

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_PGM	MODULE: A.5.2.3
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS PROGRAM STATEMENT
DATE: FEBRUARY 1991	

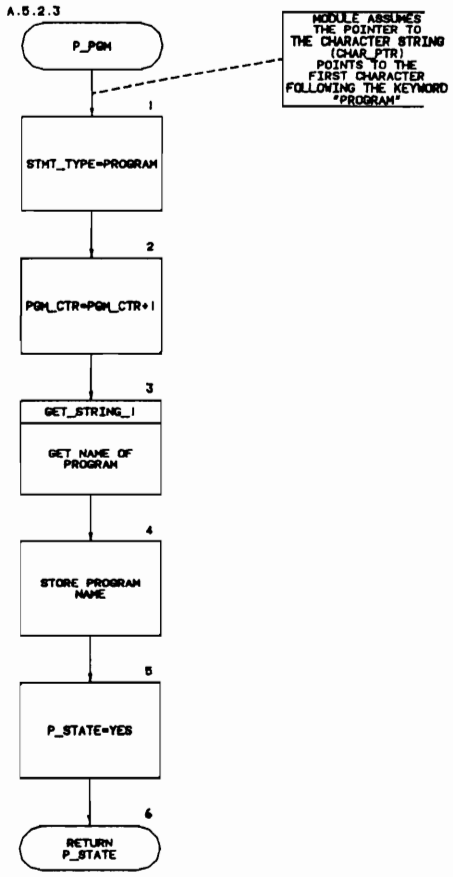


Figure 18. Module A.5.2.3 - P_PGM

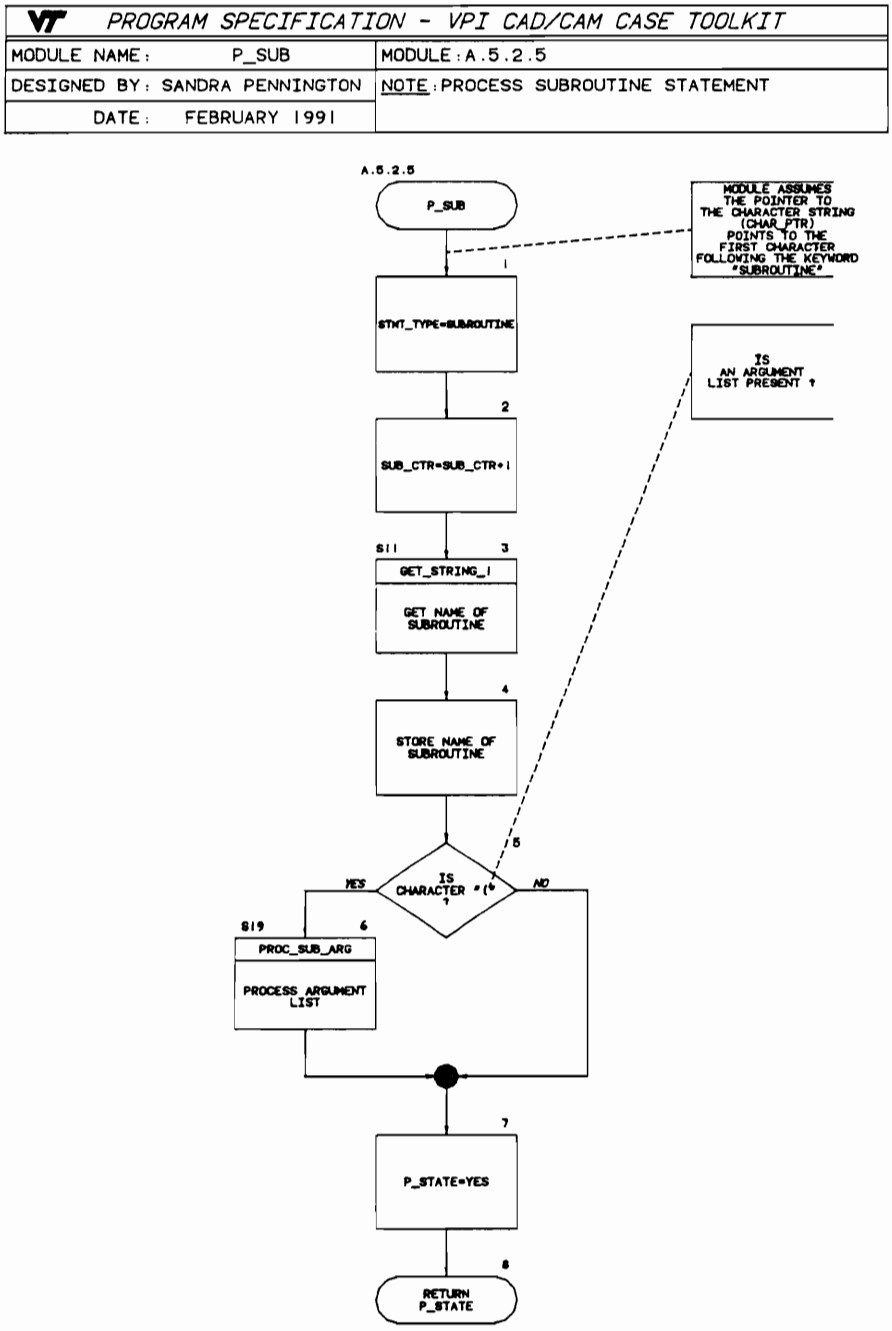


Figure 19. Module A.5.2.5 - P_SUB

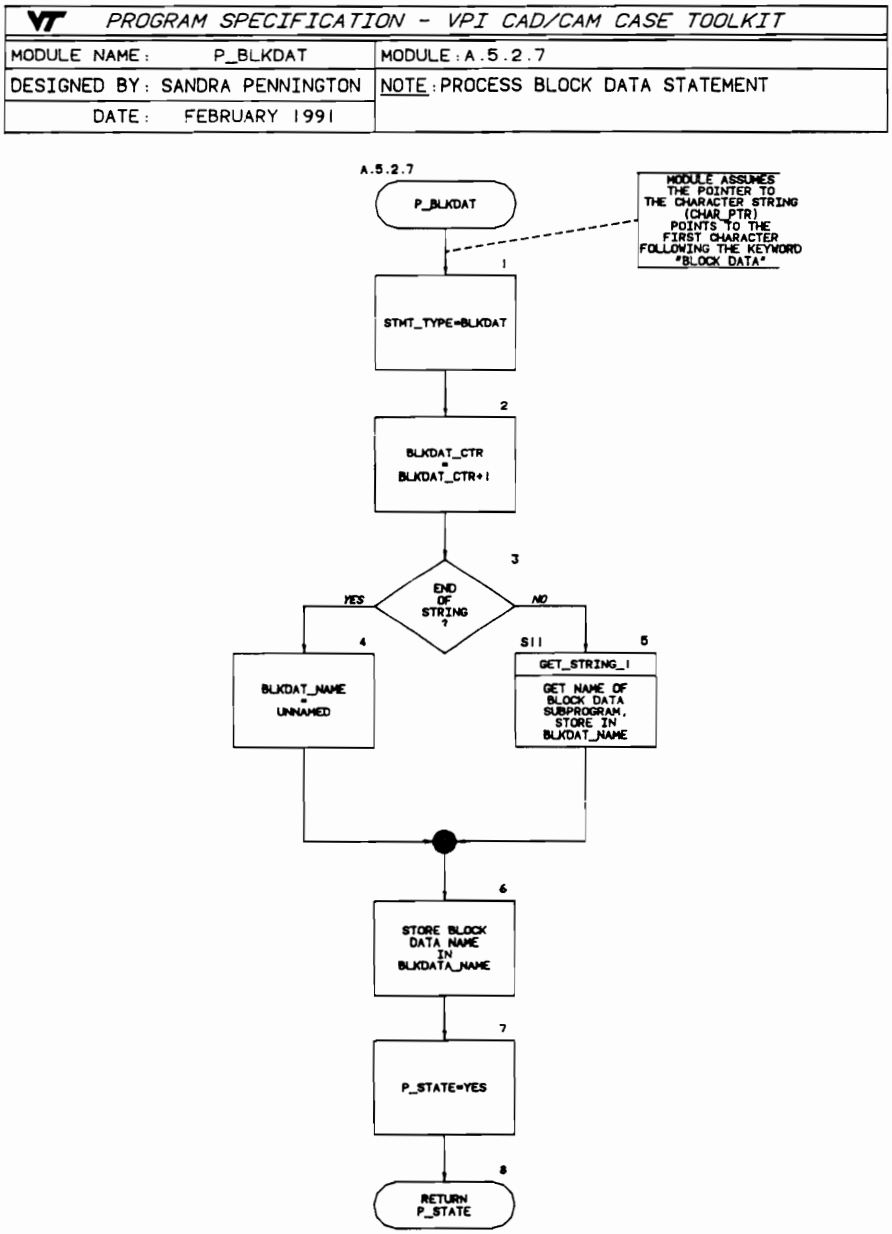


Figure 20. Module A.5.2.7 - P_BLKDAT

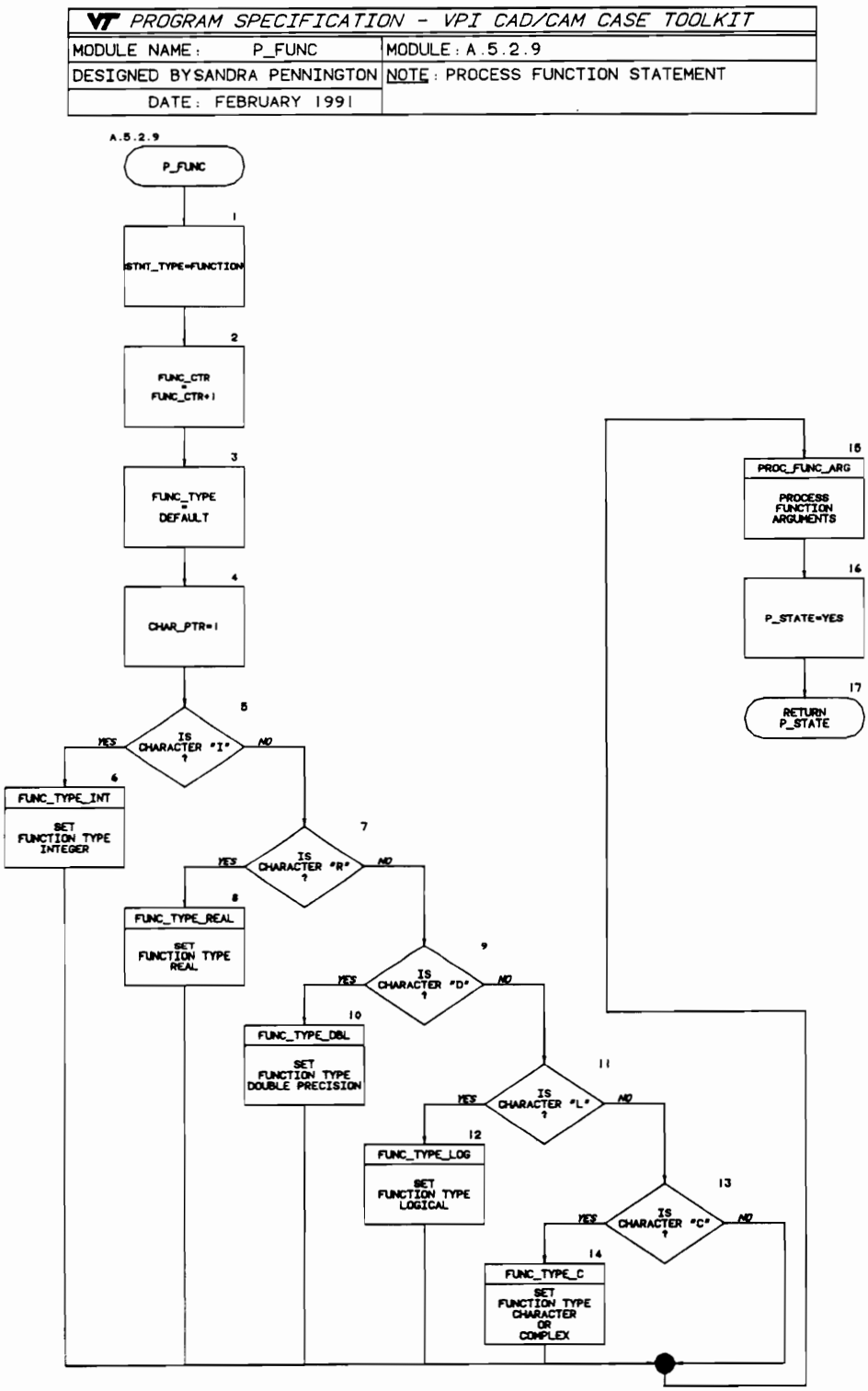


Figure 21. Module A.5.2.9 - P_FUNC

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: FUNC_TYPE_INT	MODULE: A.5.2.9.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: SET FUNCTION TYPE TO INTEGER
DATE: FEBRUARY 1991	

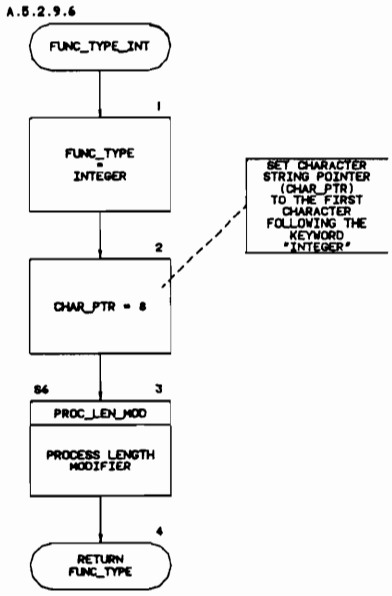


Figure 22. Module A.5.2.9.6 - FUNC_TYPE_INT

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: FUNC_TYPE_REAL	MODULE: A.5.2.9.8
DESIGNED BY: SANDRA PENNINGTON	NOTE: SET FUNCTION TYPE REAL
DATE: FEBRUARY 1991	

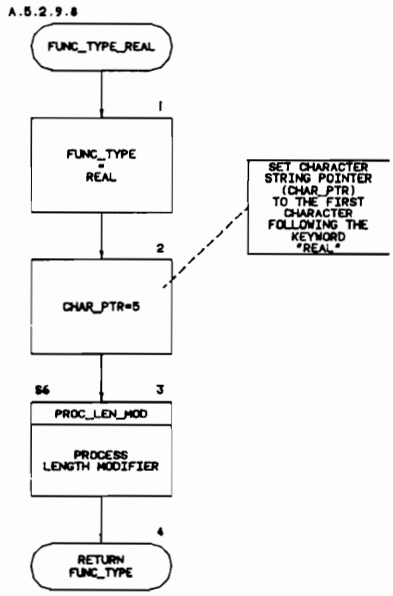


Figure 23. Module A.5.2.9.8 - FUNC_TYPE_REAL

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: FUNC_TYPE_DBL	MODULE: A.5.2.9.10
DESIGNED BY: SANDRA PENNINGTON	NOTE: SET FUNCTION TYPE TO DOUBLE PRECISION
DATE: FEBRUARY 1991	

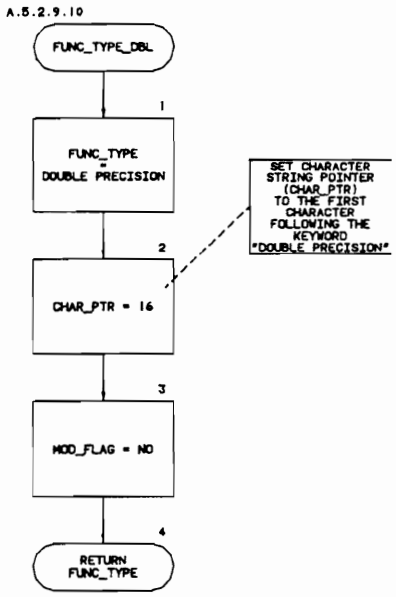


Figure 24. Module A.5.2.9.10 - FUNC_TYPE_DBL

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: FUNC_TYPE_LOG	MODULE: A.5.2.9.12
DESIGNED BY: SANDRA PENNINGTON	NOTE: SET FUNCTION TYPE TO LOGICAL
DATE: FEBRUARY 1991	

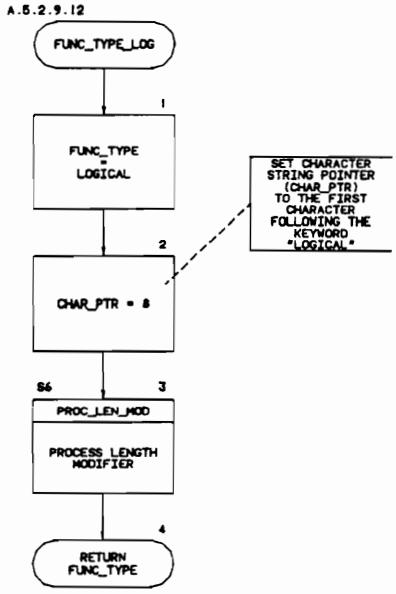


Figure 25. Module A.5.2.9.12 - FUNC_TYPE_LOG

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: FUNC_TYPE_C	MODULE: A.5.2.9.14
DESIGNED BY: SANDRA PENNINGTON	NOTE: SET FUNCTION TYPE TO CHARACTER OR COMPLEX
DATE: FEBRUARY 1991	

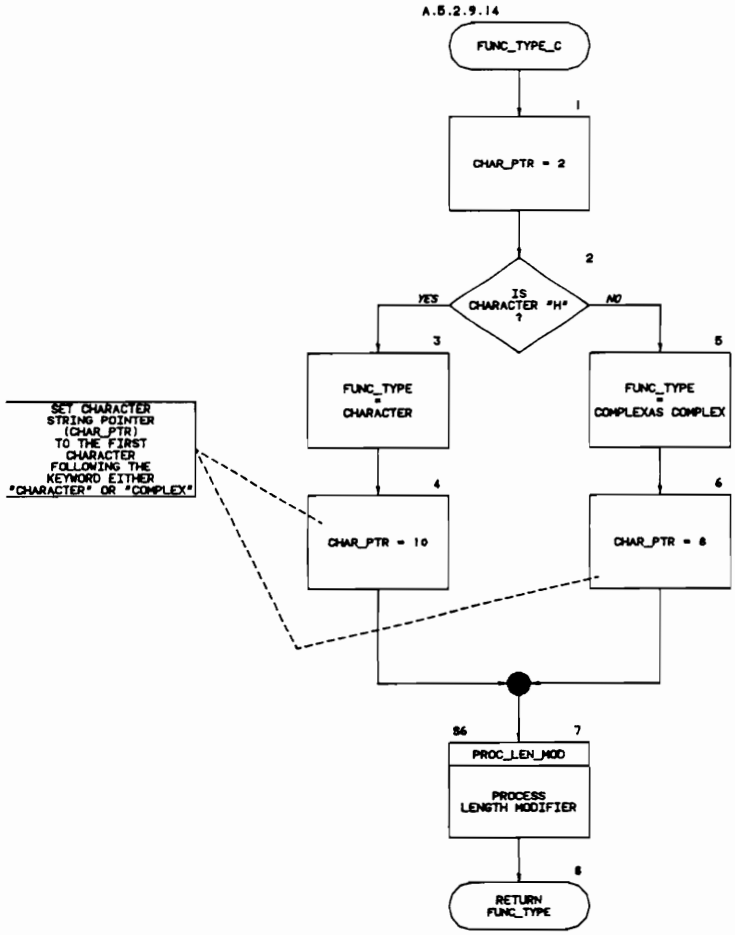


Figure 26. Module A.5.2.9.14 - FUNC_TYPE_C

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_FUNC_ARG	MODULE: A.5.2.9.15
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS FUNCTION ARGUMENTS
DATE: FEBRUARY 1991	

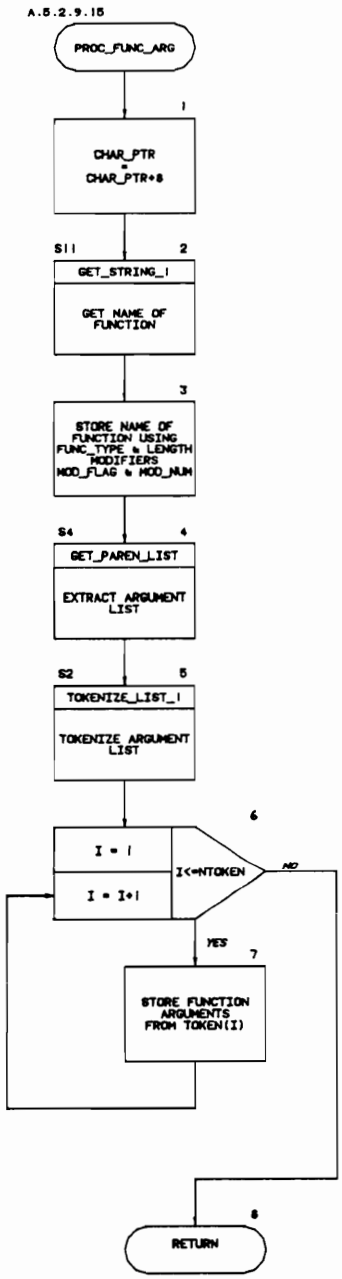


Figure 27. Module A.5.2.9.15 - PROC_FUNC_ARG

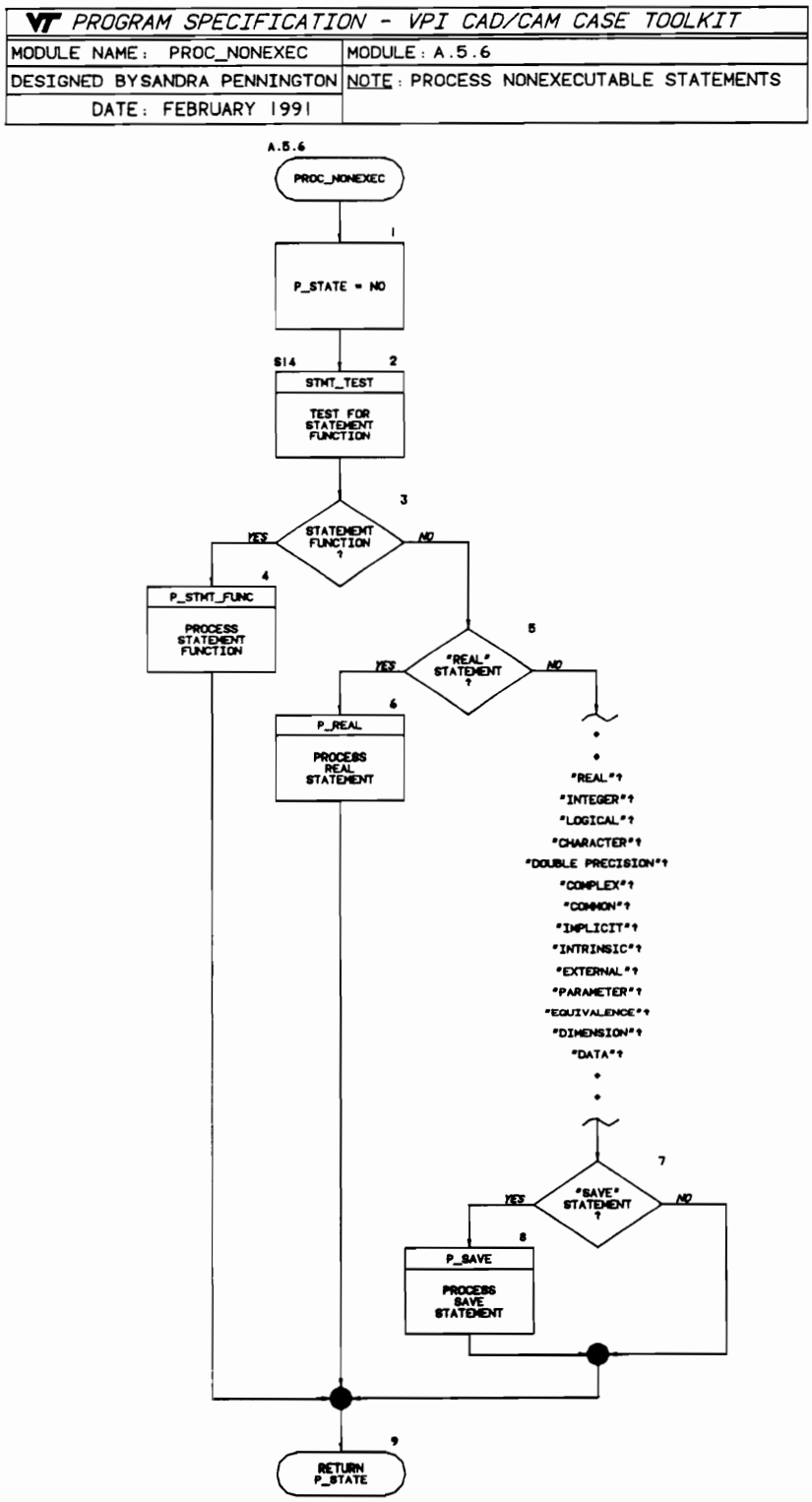


Figure 28. Module A.5.6 - PROC_NONEXEC

A.5.6.T

STATEMENT	MODULE NAME	MODULE #
STATEMENT FUNCTION	P_STMT_FUNC	A.5.6.4
REAL	P_REAL	A.5.6.6
INTEGER	P_INTGR	A.5.6.8
LOGICAL	P_LOGL	A.5.6.10
CHARACTER	P_CHAR	A.5.6.12
DOUBLE PRECISION	P_DBL_PR	A.5.6.14
COMPLEX	P_COMP	A.5.6.16
COMMON	P_COMM	A.5.6.18
IMPLICIT	P_IMPL	A.5.6.20
INTRINSIC	P_INTR	A.5.6.22
EXTERNAL	P_EXT	A.5.6.24
PARAMETER	P_PARAM	A.5.6.26
EQUIVALENCE	P_EQUIV	A.5.6.28
DIMENSION	P_DIMEN	A.5.6.30
DATA	P_DATA	A.5.6.32
SAVE	P_SAVE	A.5.6.34

Figure 29. Module A.5.6.T - PROC_NONEXEC_TABLE

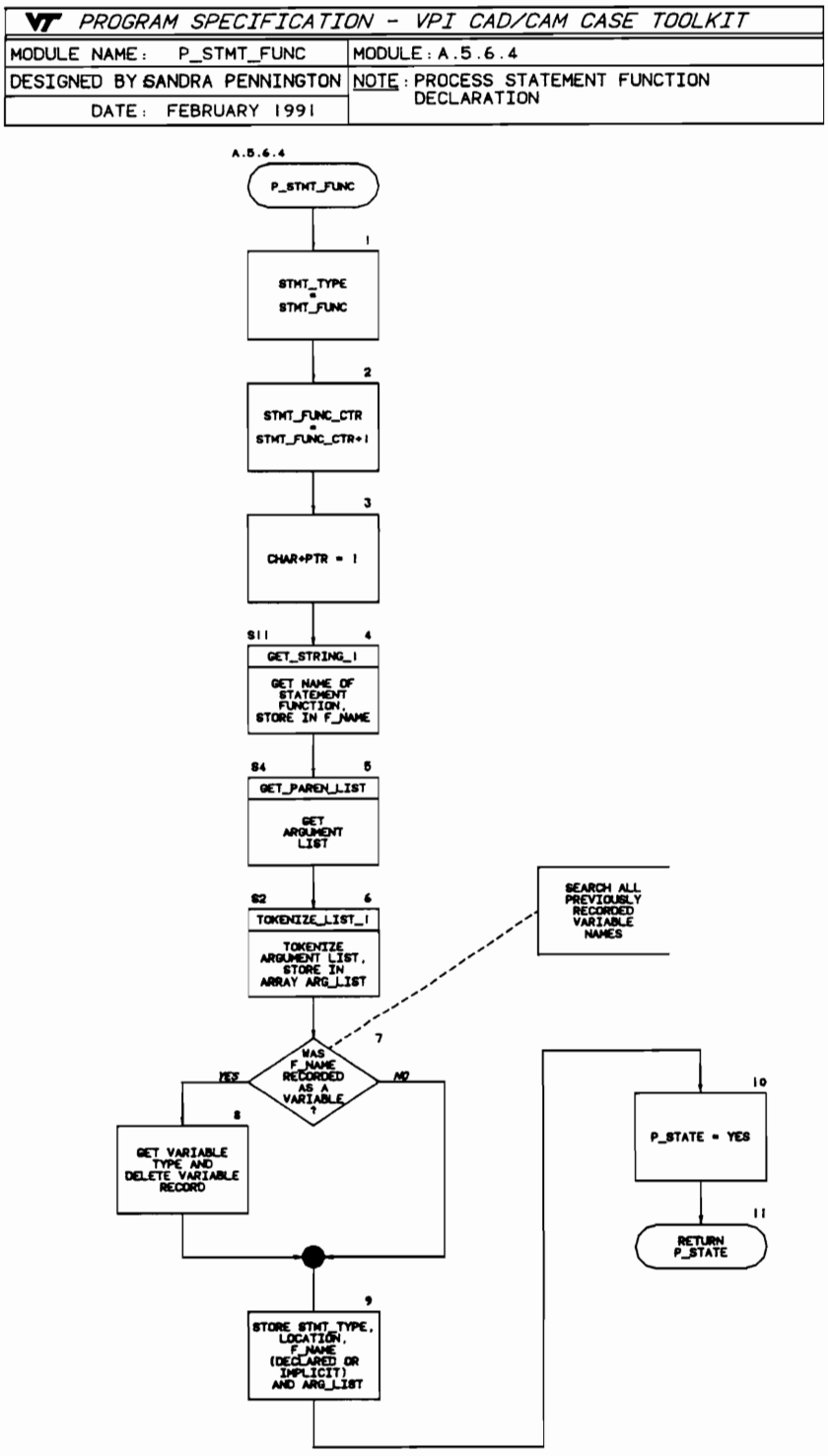


Figure 30. Module A.5.6.4 - P_STMT_FUNC

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_REAL	MODULE: A.5.6.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS REAL STATEMENT
DATE: FEBRUARY 1991	

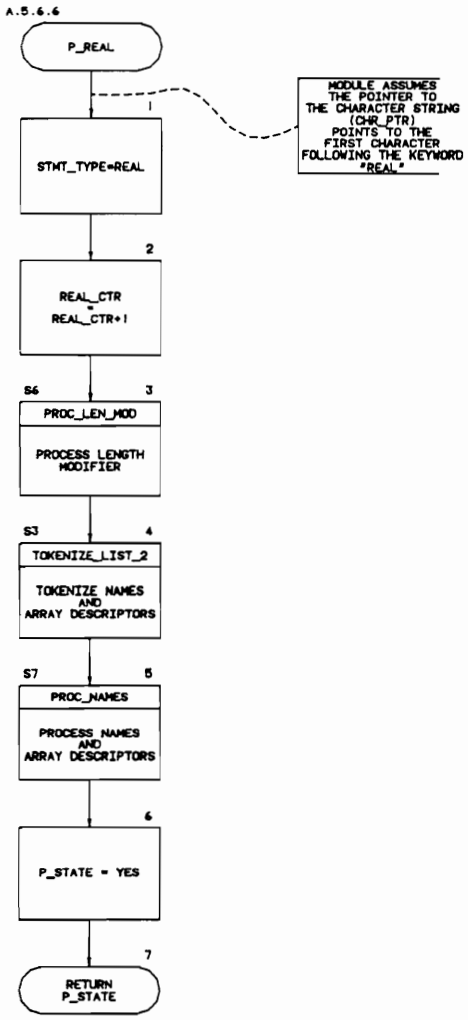


Figure 31. Module A.5.6.6 - P_REAL

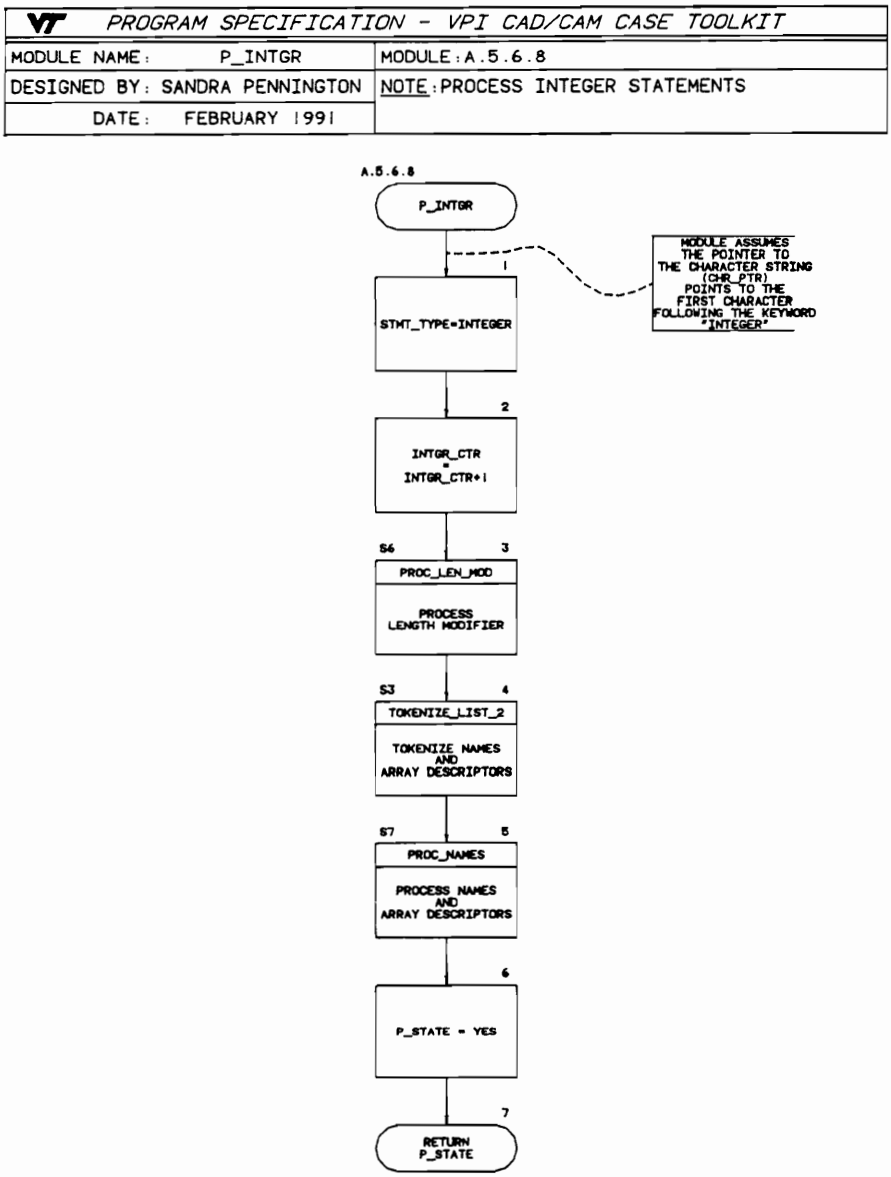


Figure 32. Module A.5.6.8 - P_INTGR

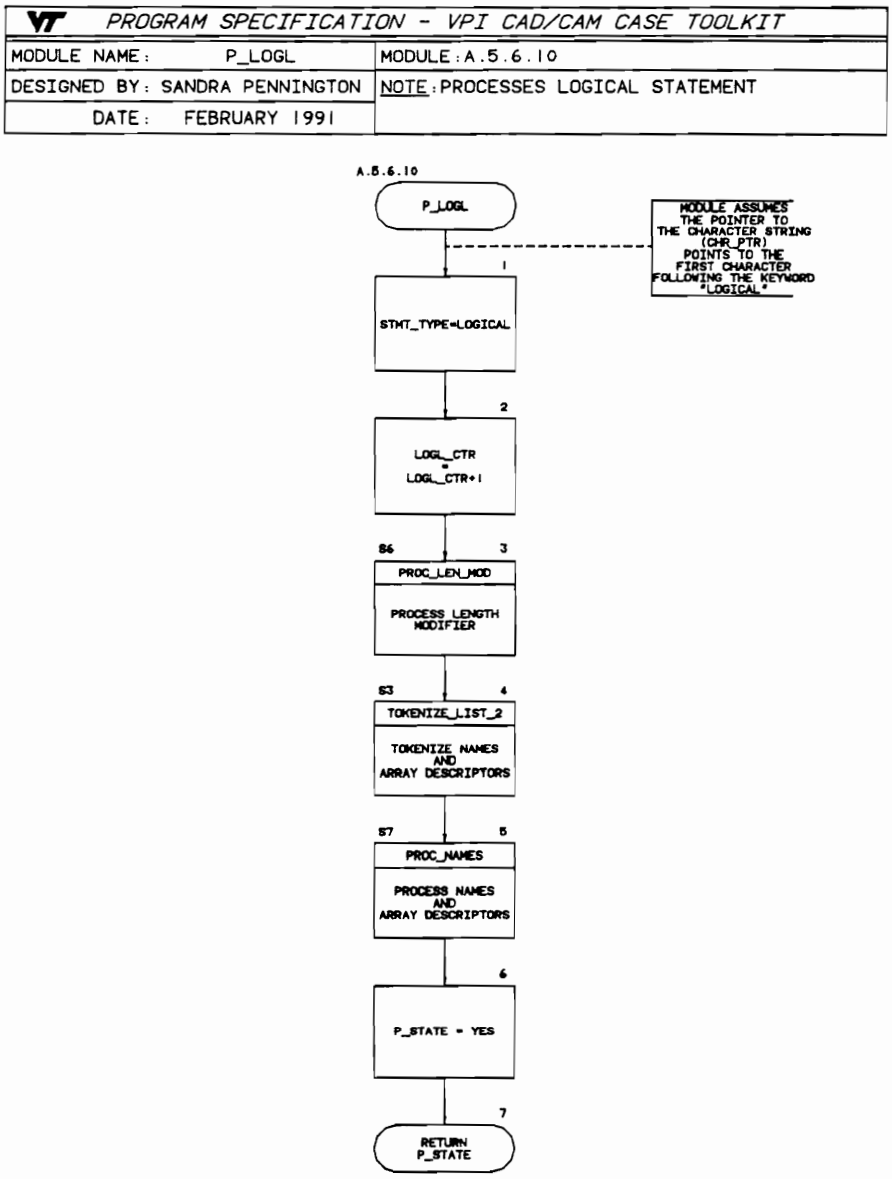


Figure 33. Module A.5.6.10 - P_LOGL

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_CHAR	MODULE: A.5.6.12
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESSES CHARACTER STATEMENT
DATE: FEBRUARY 1991	

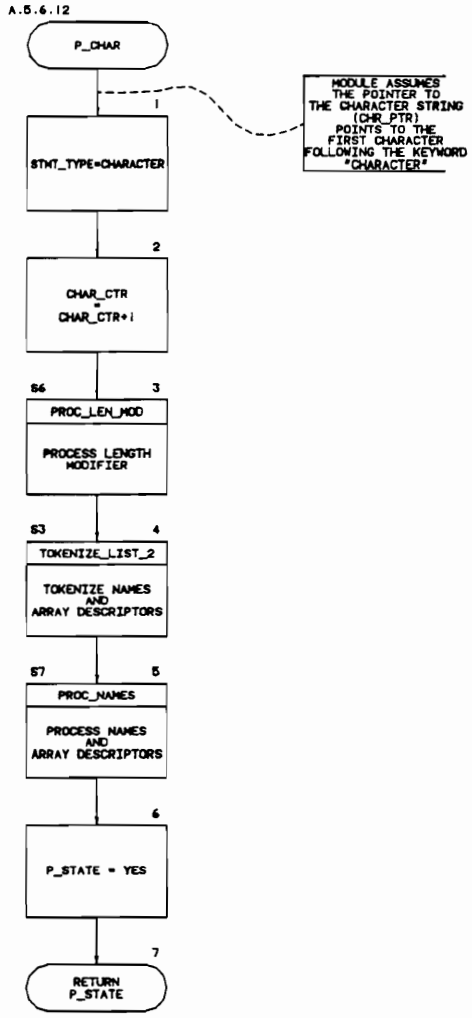


Figure 34. Module A.5.6.12 - P_CHAR

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_DBL_PR	MODULE: A.5.6.14
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS DOUBLE PRECISION STATEMENT
DATE: FEBRUARY 1991	

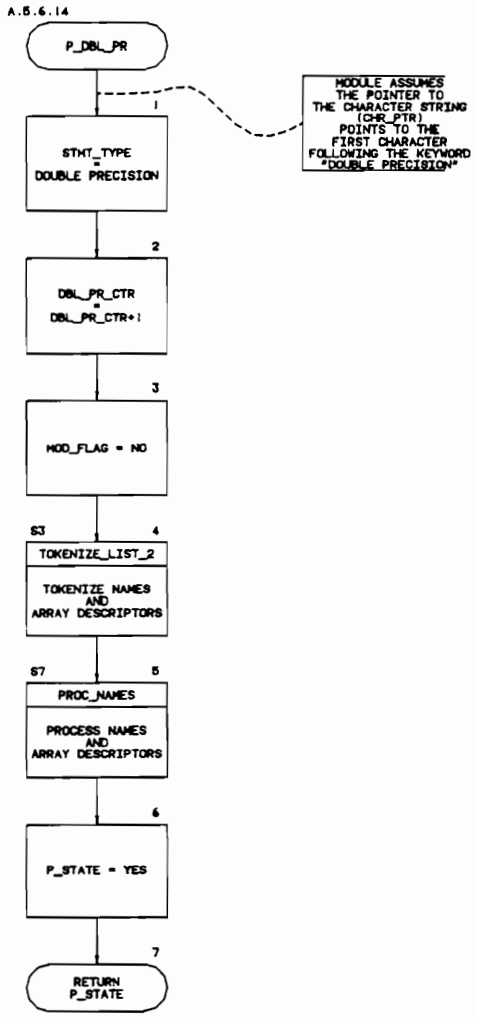


Figure 35. Module A.5.6.14 - P_DBL_PR

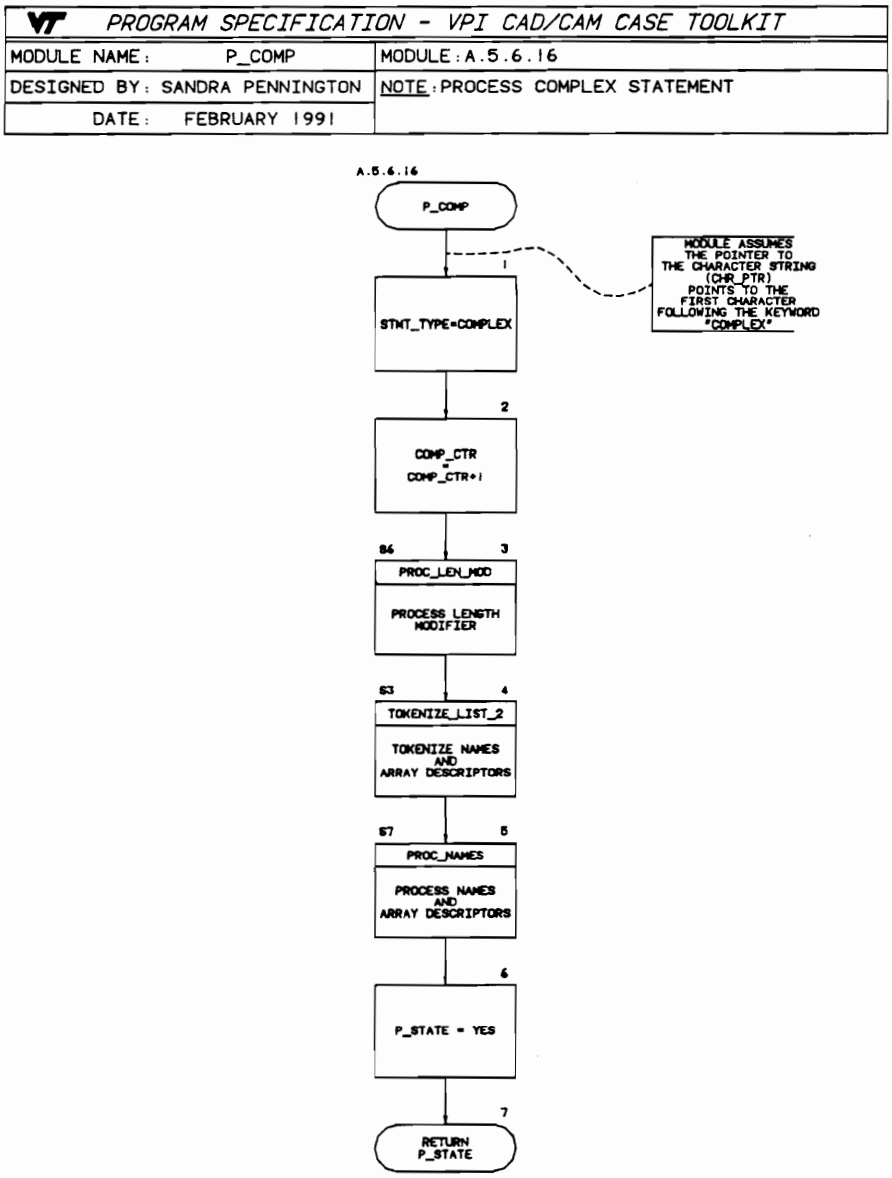


Figure 36. Module A.5.6.16 - P_COMP

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_COMM	MODULE: A.5.6.18
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS COMMON STATEMENT
DATE: FEBRUARY 1991	

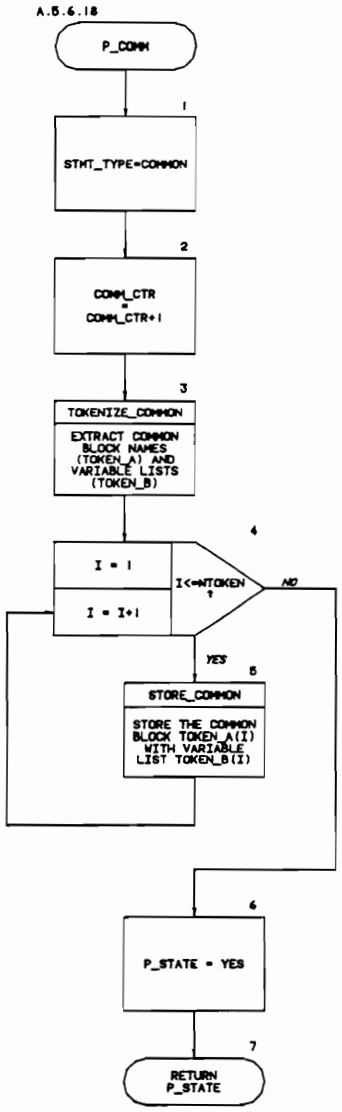


Figure 37. Module A.5.6.18 - P_COMM

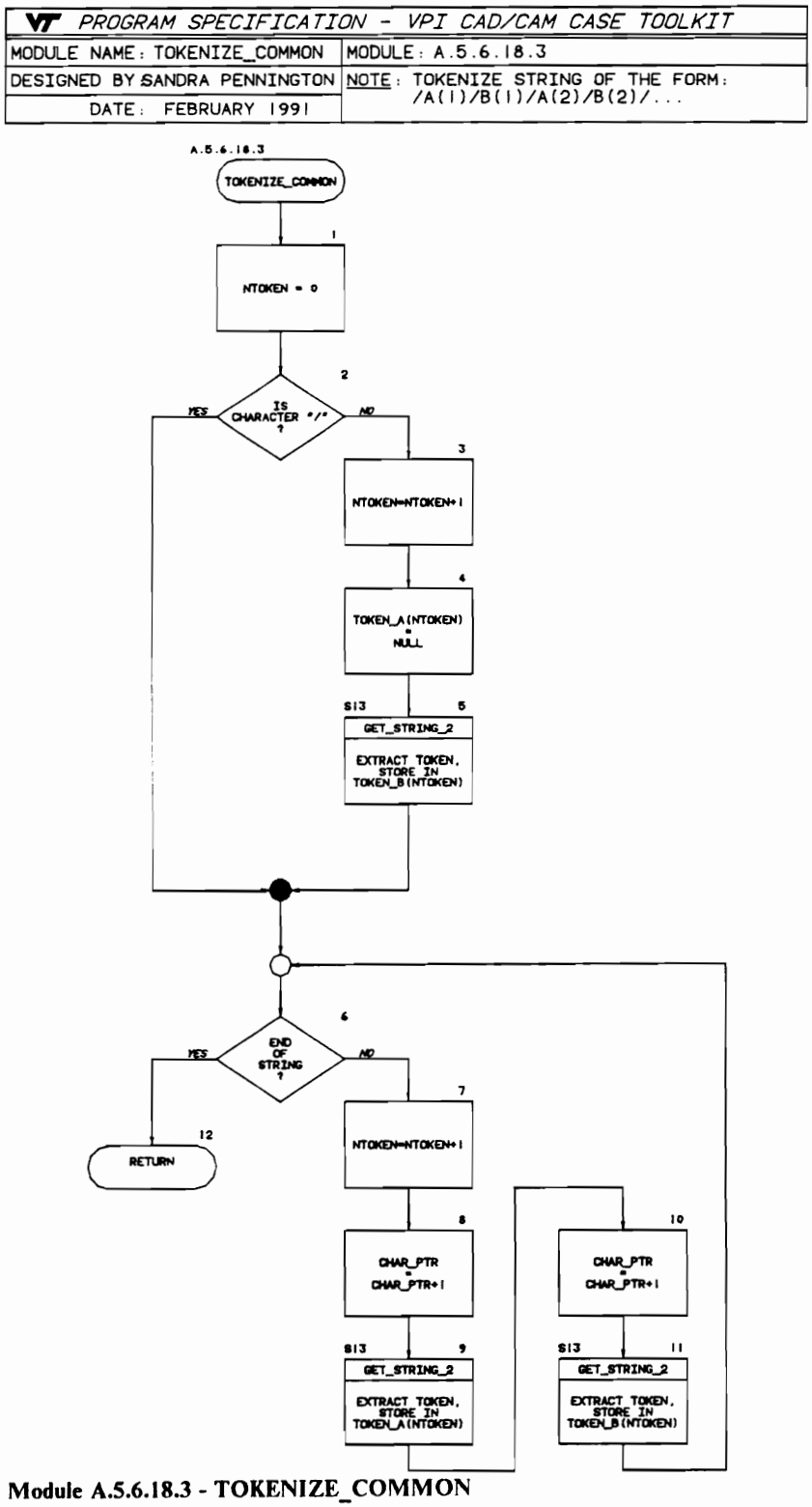


Figure 38. Module A.5.6.18.3 - TOKENIZE_COMMON

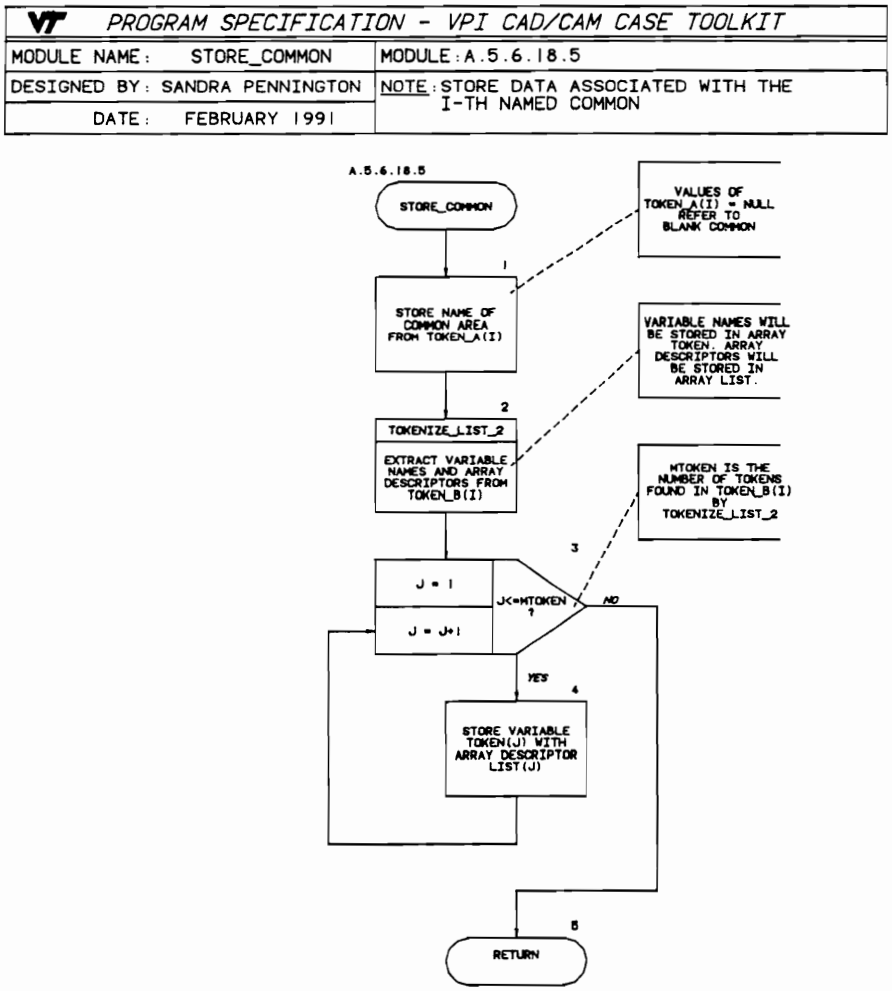


Figure 39. Module A.5.6.18.5 - STORE_COMMON

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_IMPL	MODULE: A.5.6.20
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS THE IMPLICIT STATEMENT
DATE: FEBRUARY 1991	

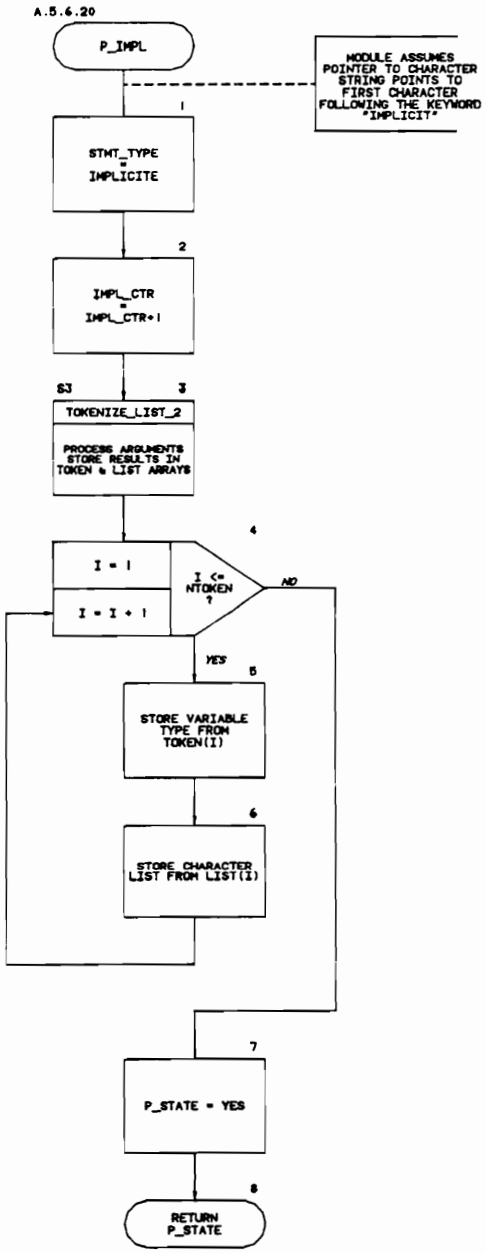


Figure 40. Module A.5.6.20 - P_IMPL

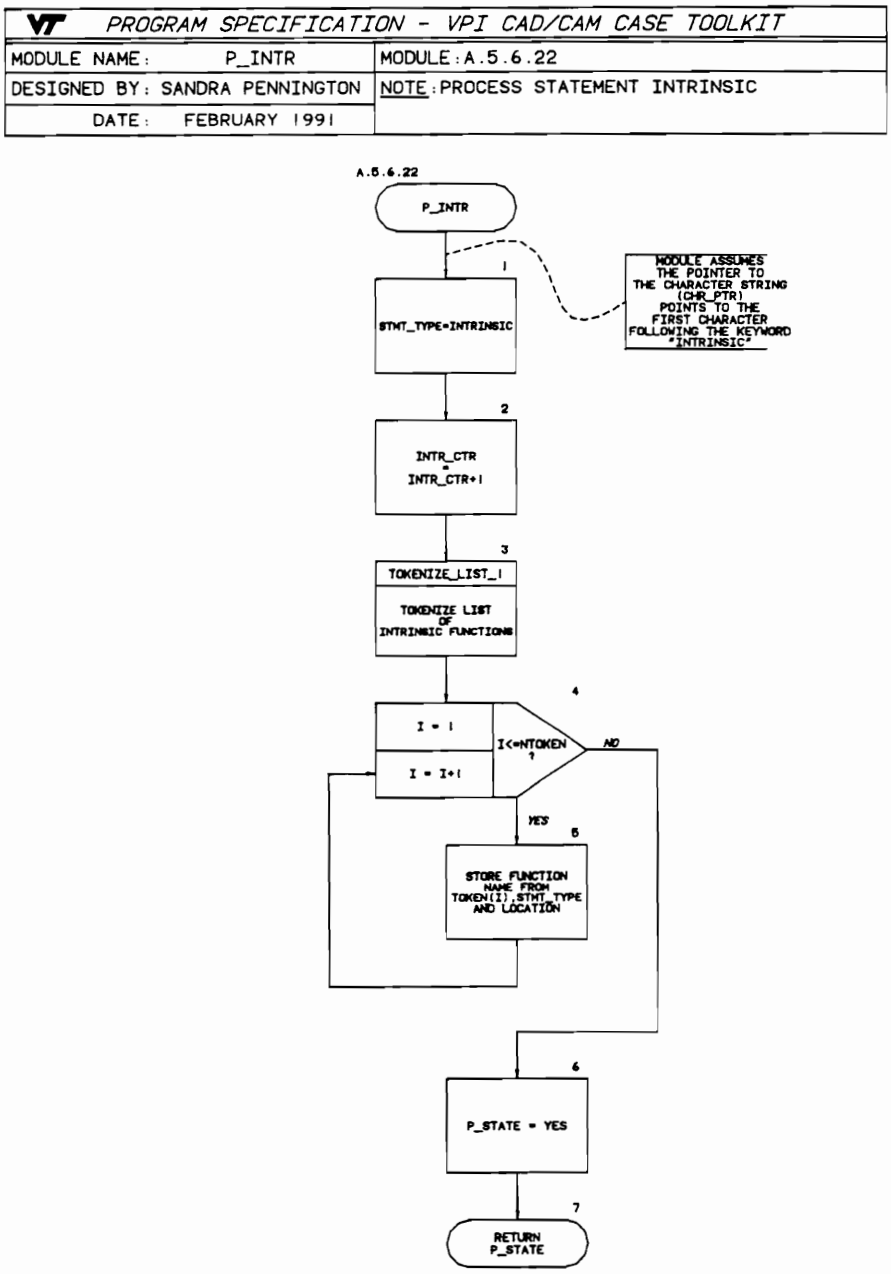


Figure 41. Module A.5.6.22 - P_INTR

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_EXT	MODULE: A.5.6.24
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS STATEMENT EXTERNAL
DATE: FEBRUARY 1991	

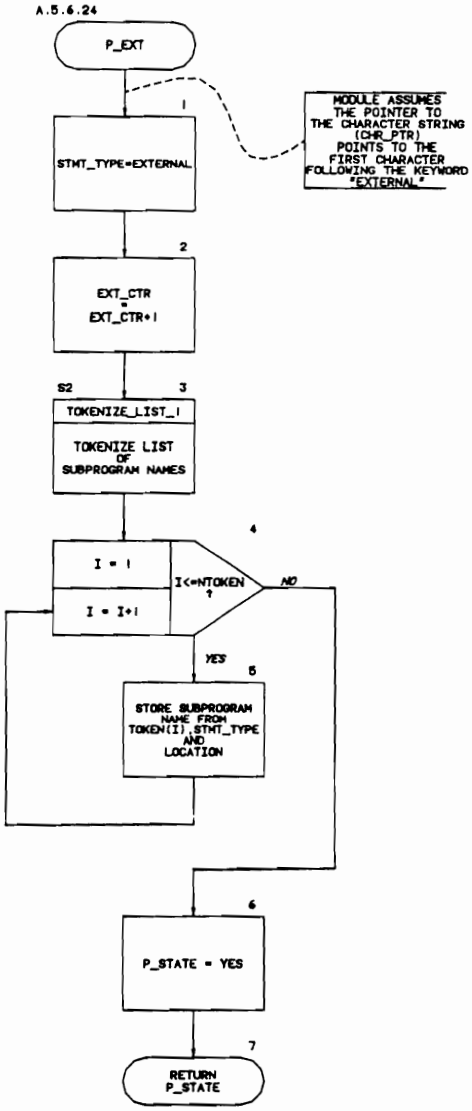


Figure 42. Module A.5.6.24 - P_EXT

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_PARAM	MODULE: A.5.6.26
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS PARAMETER STATEMENT
DATE: FEBRUARY 1991	

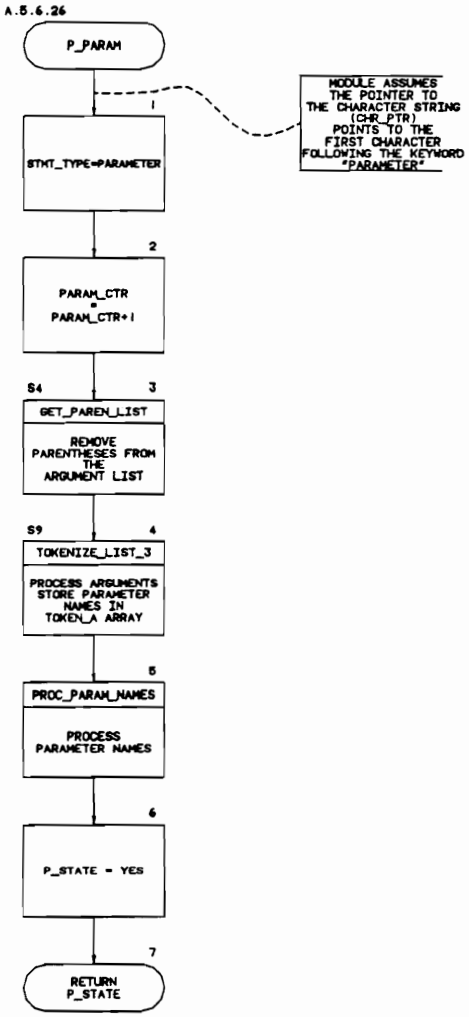


Figure 43. Module A.5.6.26 - P_PARAM

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_PARAM_NAMES	MODULE: A.5.6.26.5
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS PARAMETER NAMES
DATE: FEBRUARY 1991	

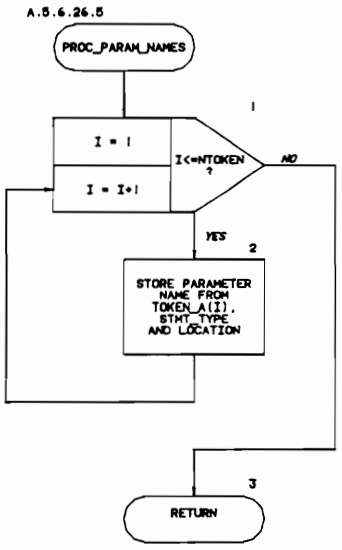


Figure 44. Module A.5.6.26.5 - PROC_PARAM_NAMES

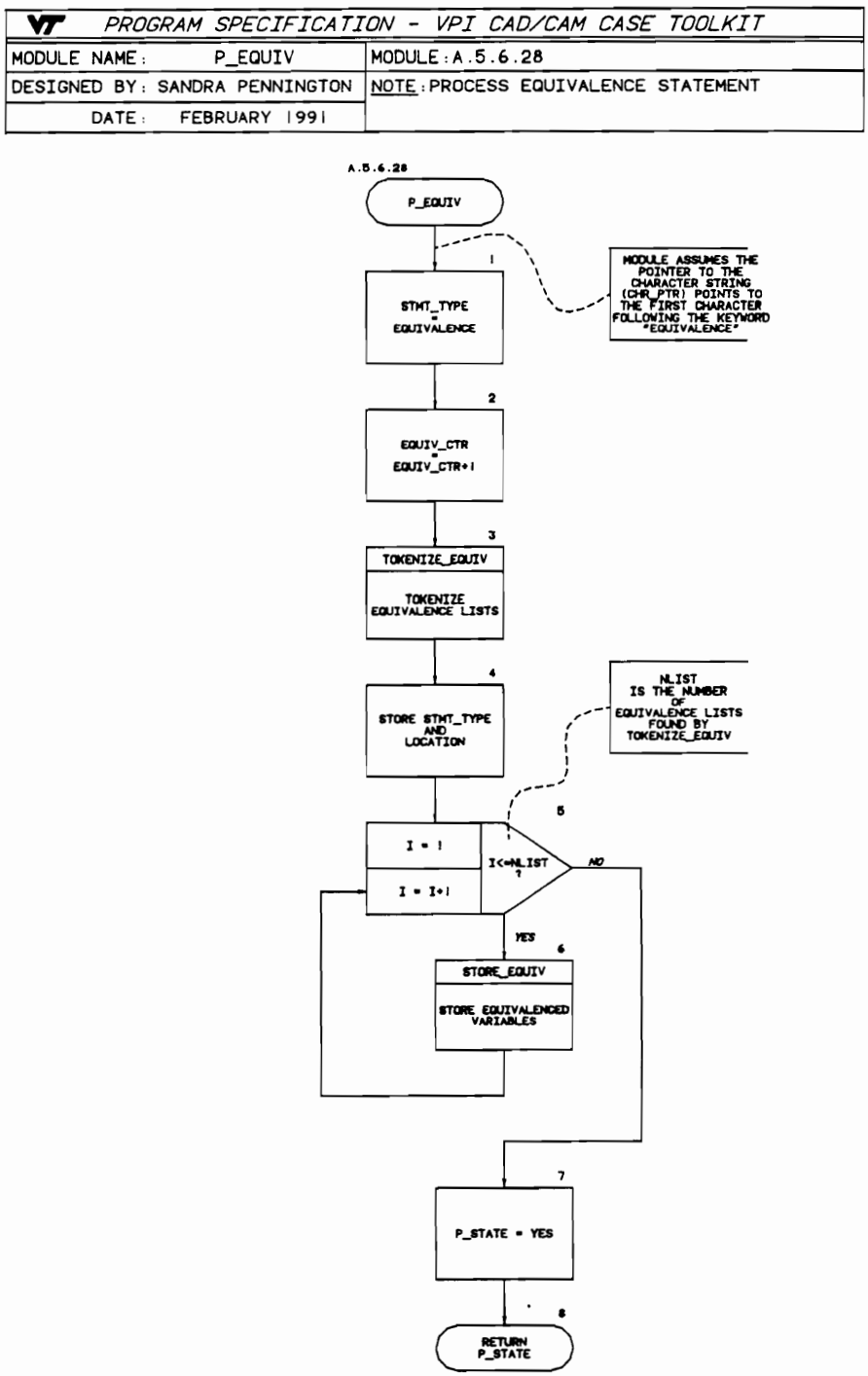


Figure 45. Module A.5.6.28 - P_EQUIV

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TOKENIZE_EQUIV	MODULE: A.5.6.28.3
DESIGNED BY: SANDRA PENNINGTON	NOTE: TOKENIZE A LIST OF THE FORM: (TOKEN1),(TOKEN2),(TOKEN3),...
DATE: FEBRUARY 1991	

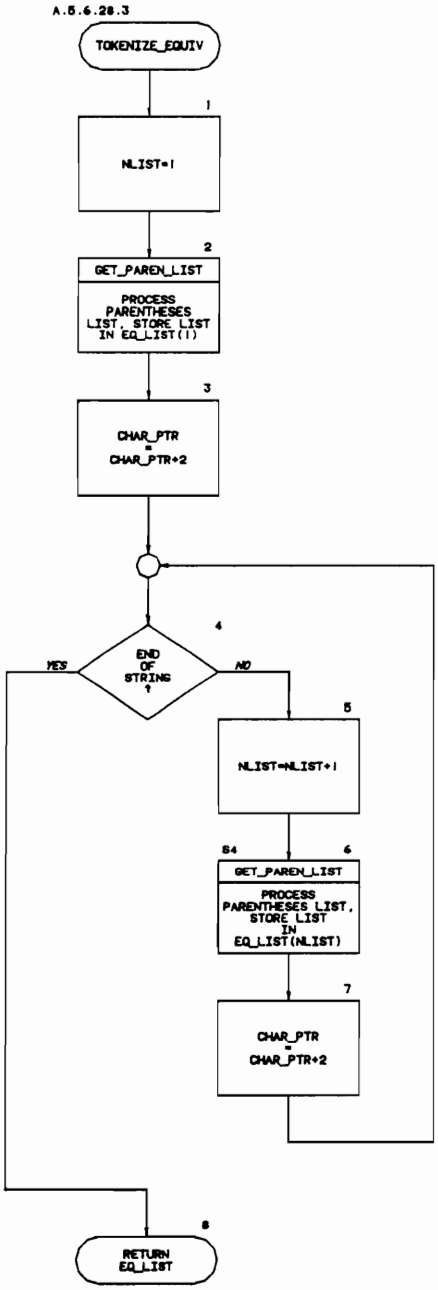


Figure 46. Module A.5.6.28.3 - TOKENIZE_EQUIV

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: STORE_EQUIV	MODULE: A.5.6.28.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: STORE EQUIVALENCED VARIABLES
DATE: FEBRUARY 1991	

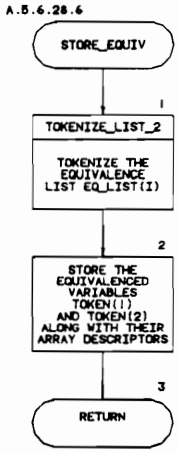


Figure 47. Module A.5.6.28.6 - STORE_EQUIV

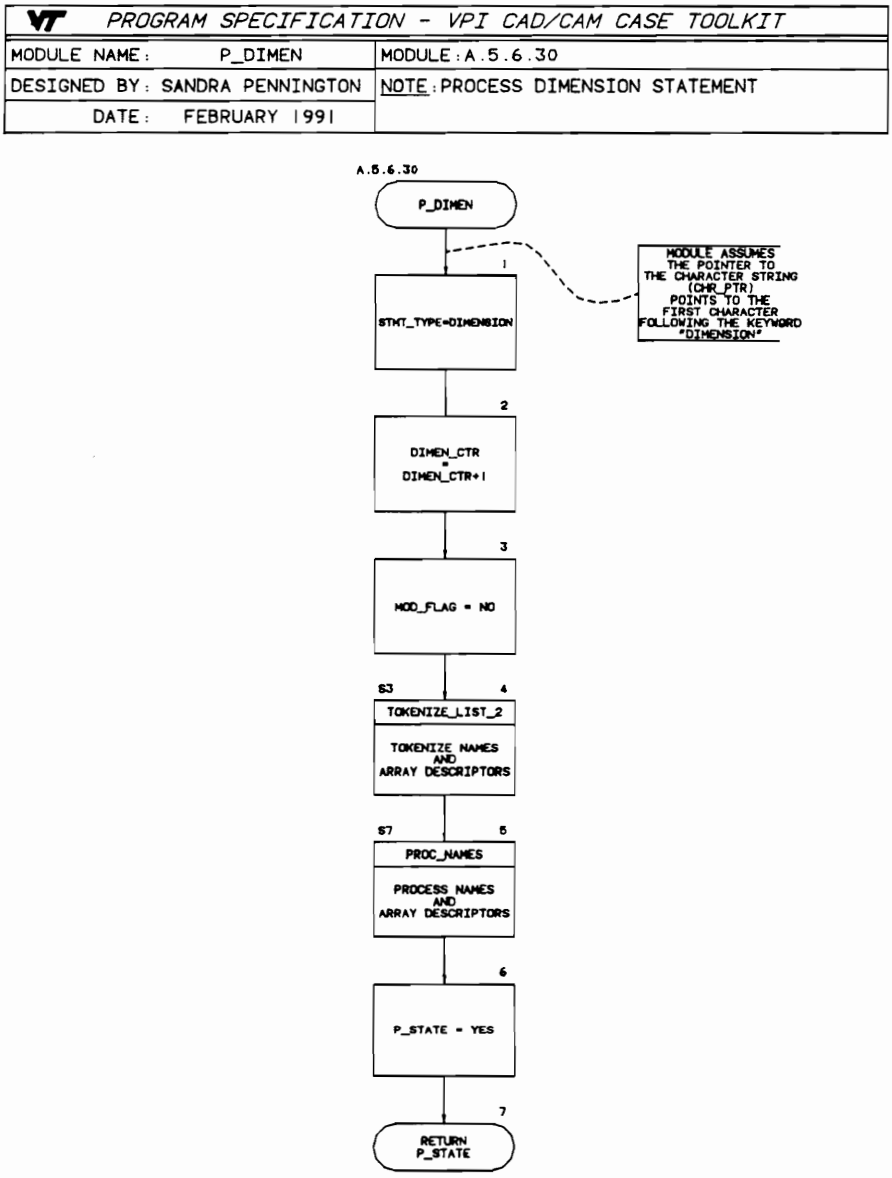


Figure 48. Module A.5.6.30 - P_DIMEN

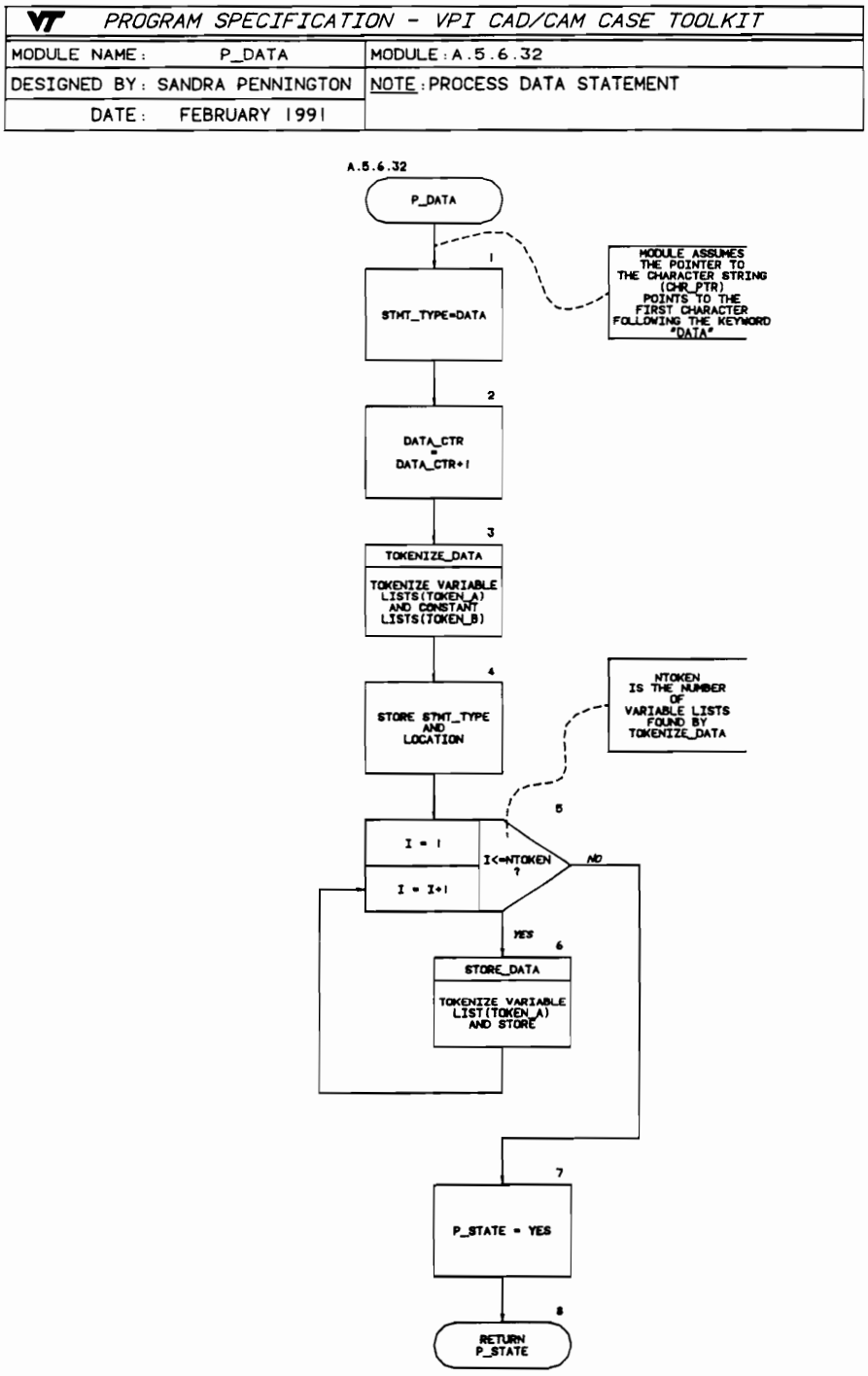


Figure 49. Module A.5.6.32 - P_DATA

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TOKENIZE_DATA	MODULE: A.5.6.32.3
DESIGNED BY: SANDRA PENNINGTON	NOTE: TOKENIZE A STRING OF THE FORM: A(1)/B(1)/A(2)/B(2)/...
DATE: FEBRUARY 1991	

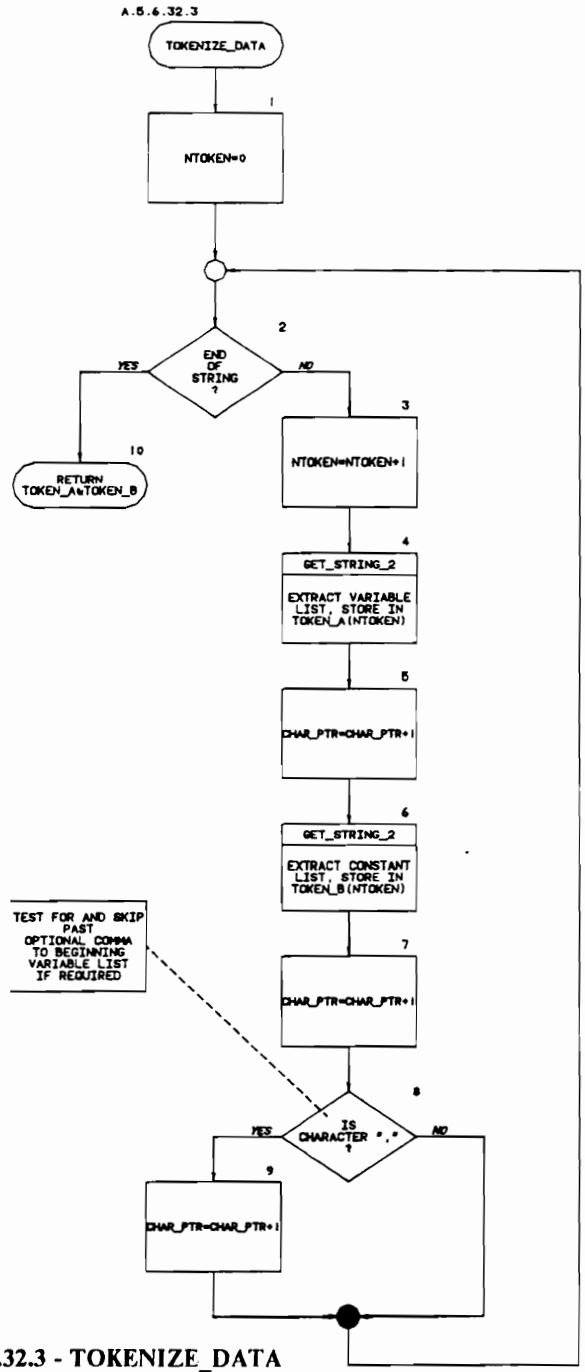


Figure 50. Module A.5.6.32.3 - TOKENIZE_DATA

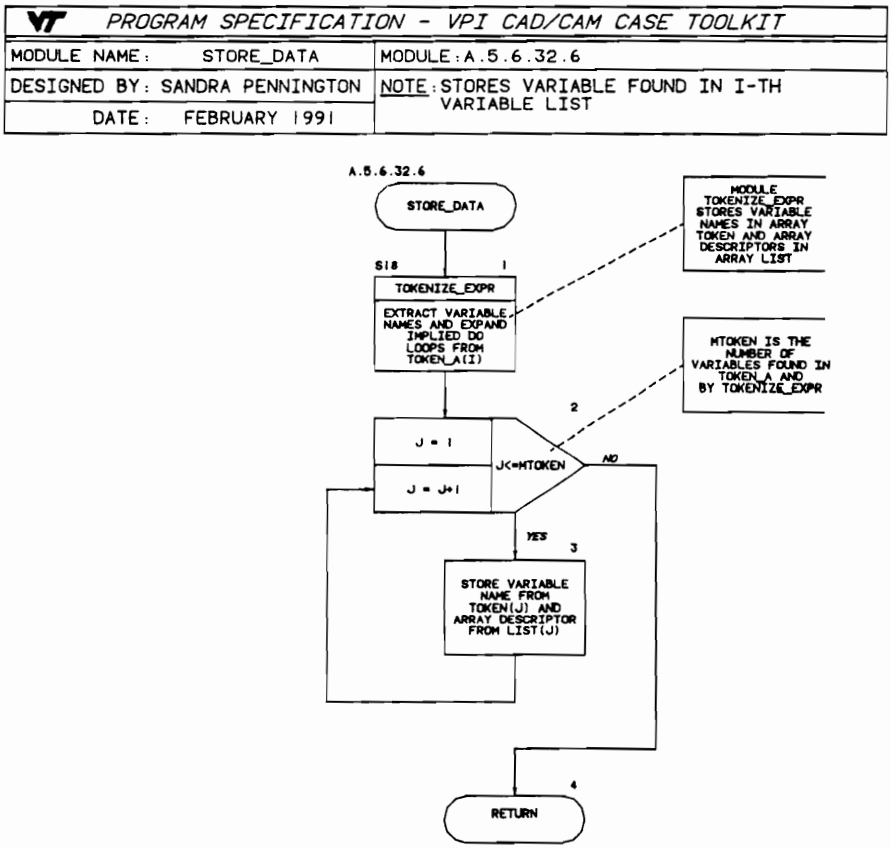


Figure 51. Module A.5.6.32.6 - STORE_DATA

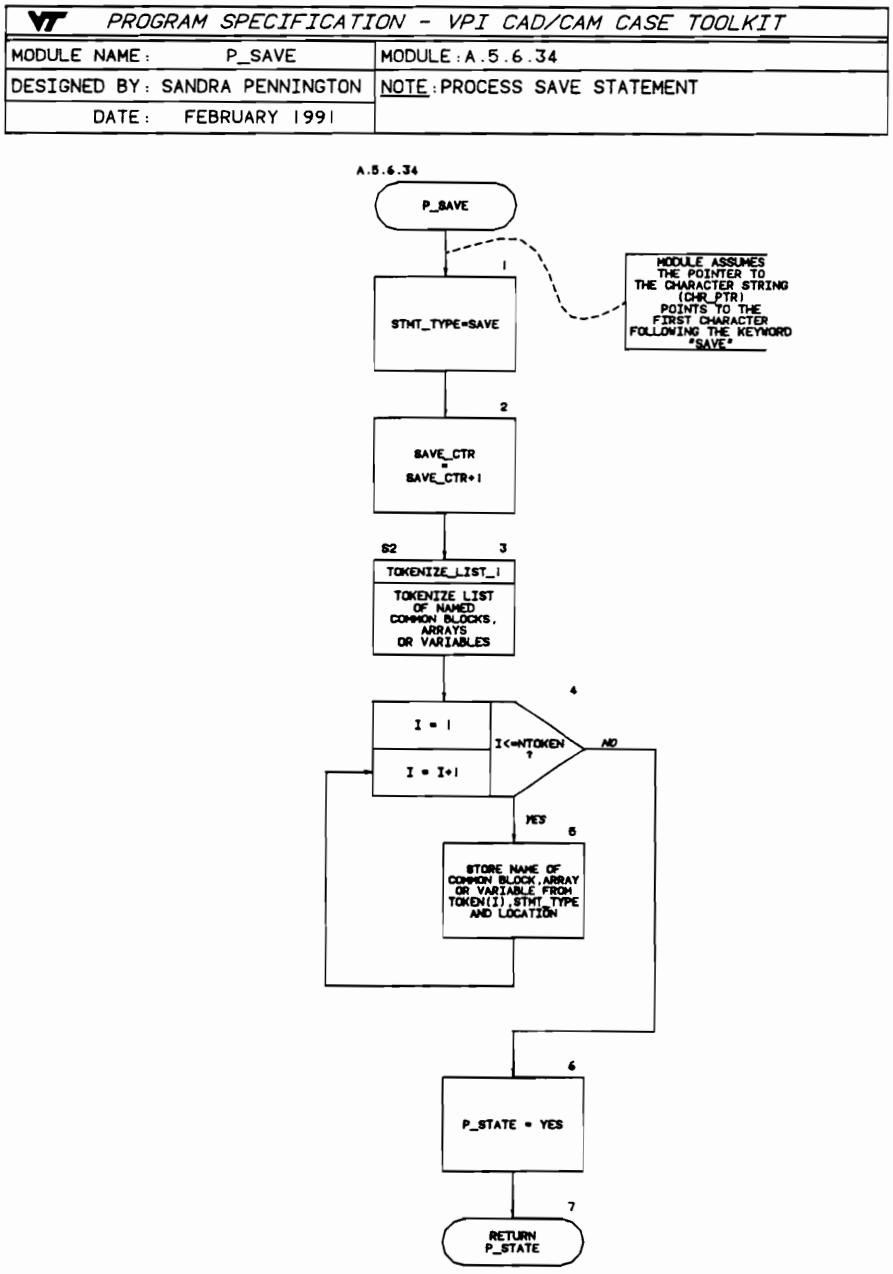


Figure 52. Module A.5.6.34 - P_SAVE

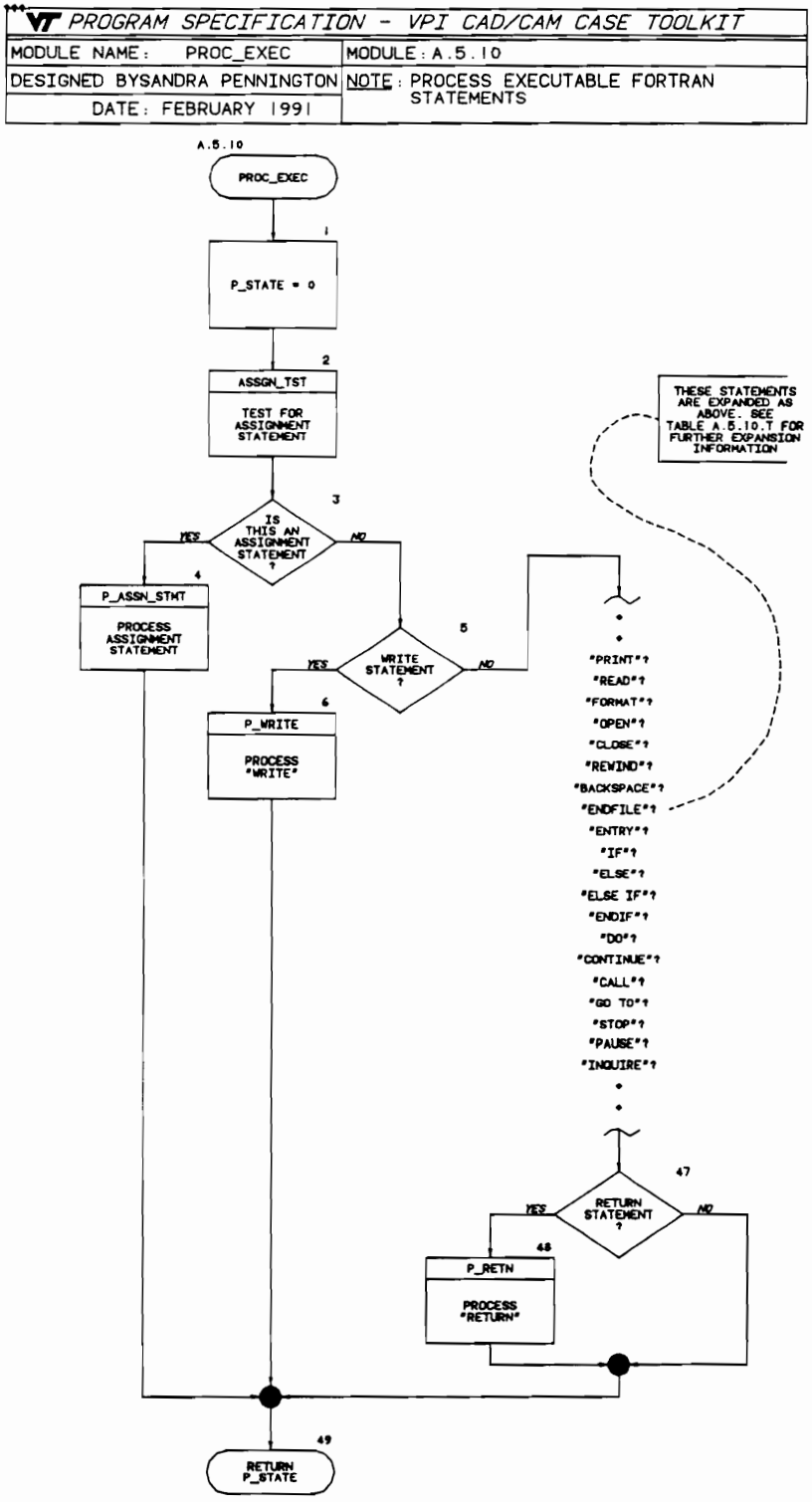


Figure 53. Module A.5.10 - PROC_EXEC

A.5.10.T

STATEMENT	MODULE NAME	MODULE #
ASSIGNMENT STMT	P_ASSN_STMT	A.5.10.4
WRITE	P_WRITE	A.5.10.6
PRINT	P_PRINT	A.5.10.8
READ	P_READ	A.5.10.10
FORMAT	P_FORMAT	A.5.10.12
OPEN	P_OPEN	A.5.10.14
CLOSE	P_CLOSE	A.5.10.16
REWIND	P_REWIND	A.5.10.18
BACKSPACE	P_BCKSP	A.5.10.20
ENDFILE	P_ENDFIL	A.5.10.22
ENTRY	P_ENTRY	A.5.10.24
IF	P_IF	A.5.10.26
ELSE	P_DUMMY	S8
ELSE IF	P_IF	A.5.10.30
ENDIF	P_DUMMY	S8
DO	P_DO	A.5.10.34
CONTINUE	P_DUMMY	S8
CALL	P_CALL	A.5.10.38
GO TO	P_GOTO	A.5.10.40
STOP	P_DUMMY	S8
PAUSE	P_DUMMY	S8
INQUIRE	P_DUMMY	S8
RETURN	P_RETN	A.5.10.48

Figure 54. Module A.5.10.T - PROC_EXEC_TABLE

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_ASSN_STMT	MODULE: A.5.10.4
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS ASSIGNMENT STATEMENT
DATE: FEBRUARY 1991	

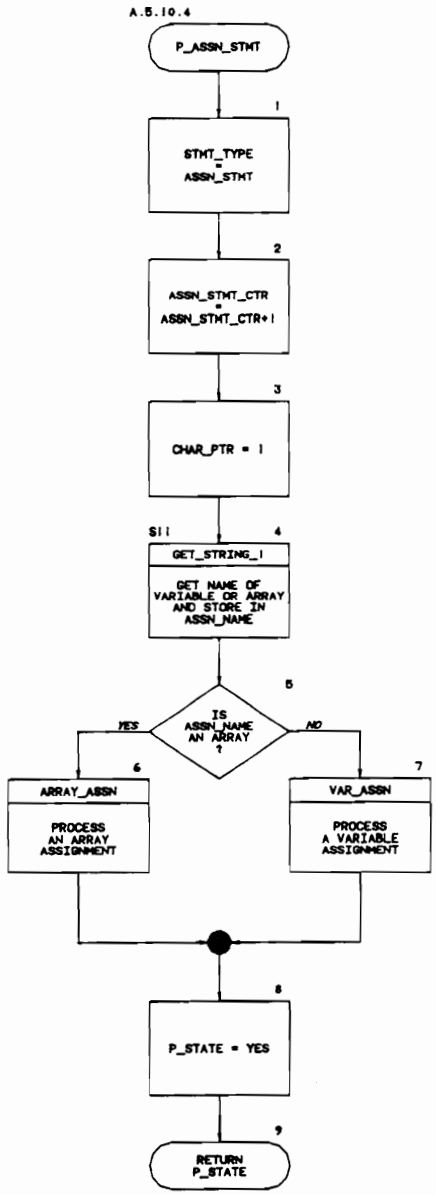


Figure 55. Module A.5.10.4 - P_ASSN_STMT

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: ARRAY_ASSN	MODULE: A.5.10.4.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS ARRAY ASSIGNMENT
DATE: FEBRUARY 1991	

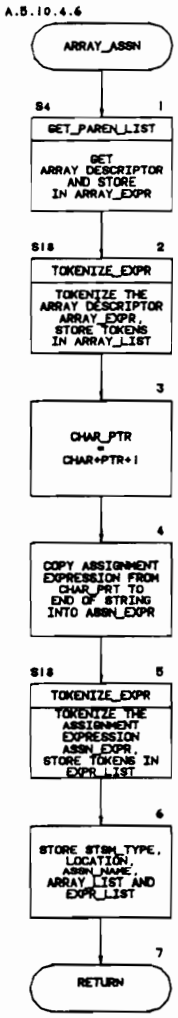


Figure 56. Module A.5.10.4.6 - ARRAY_ASSN

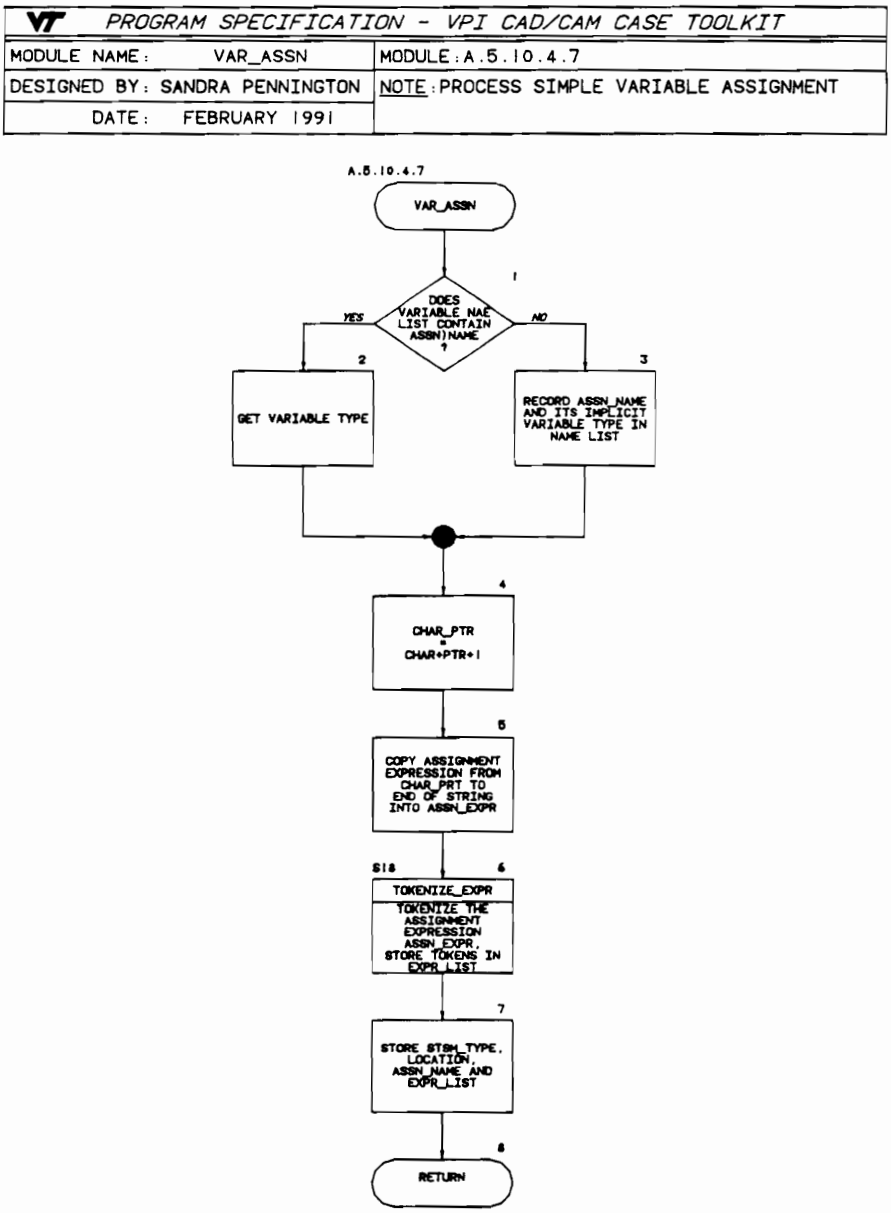


Figure 57. Module A.5.10.4.7 - VAR_ASSN

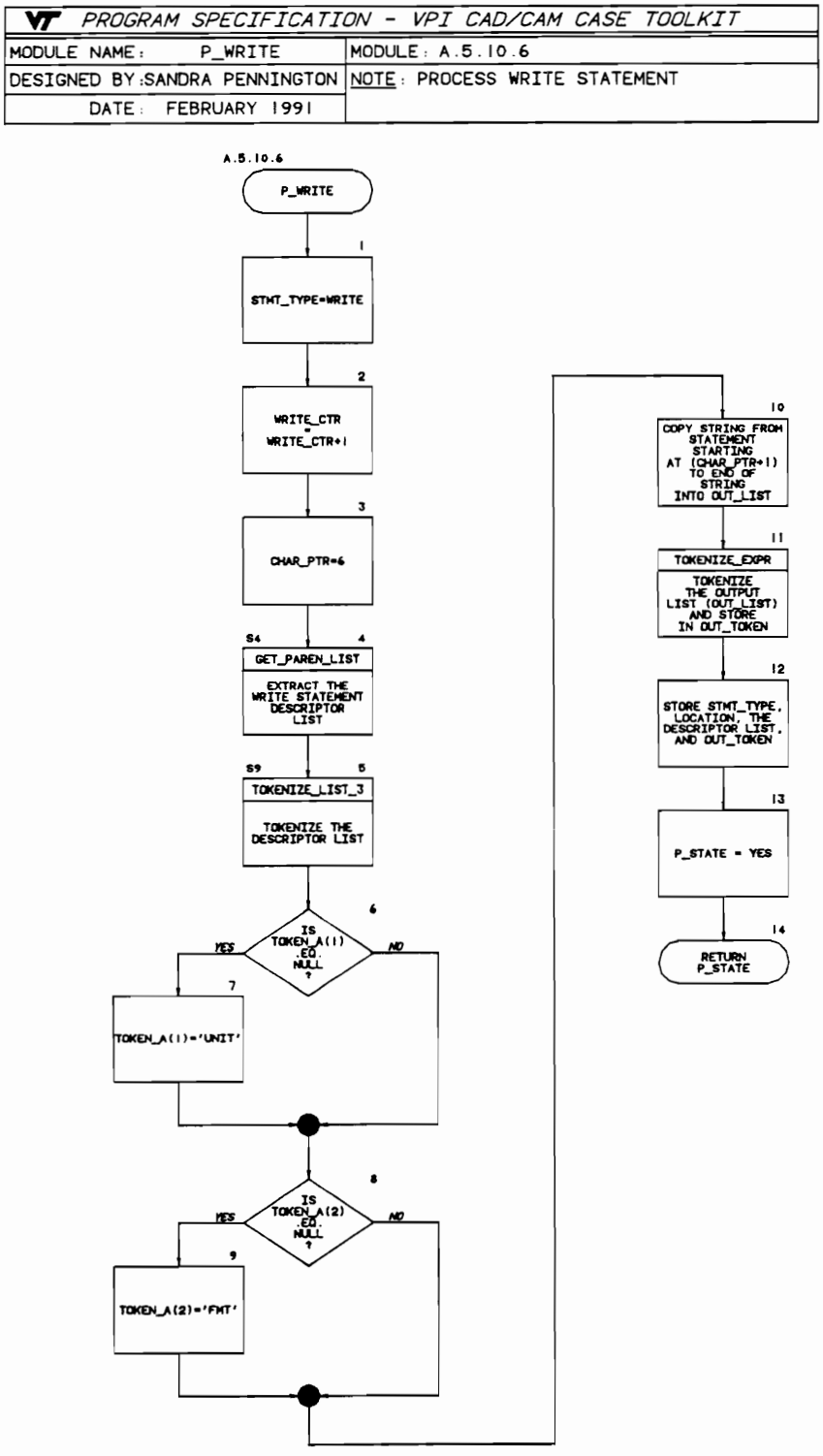


Figure 58. Module A.5.10.6 - P_WRITE

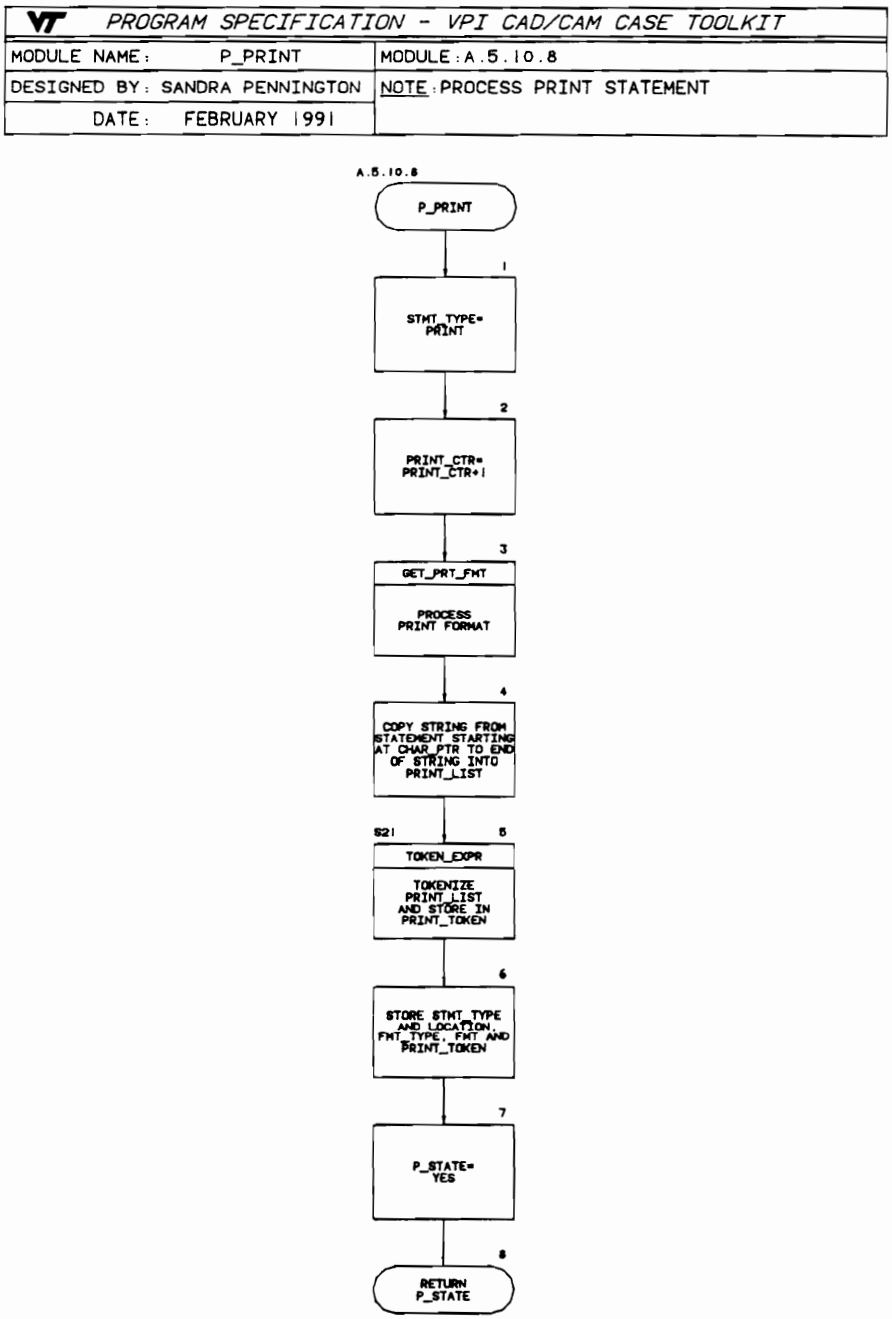


Figure 59. Module A.5.10.8 - P_PRINT

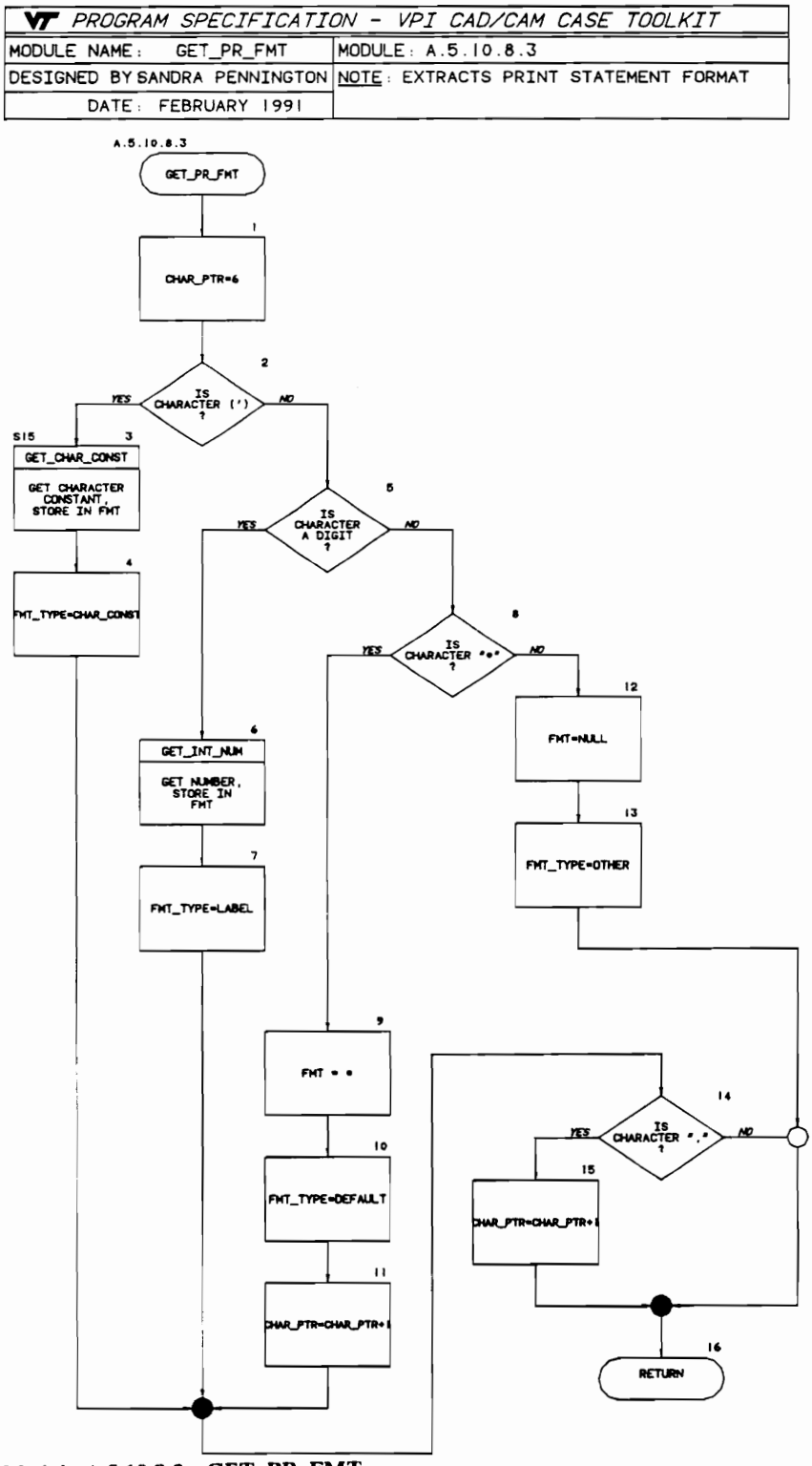


Figure 60. Module A.5.10.8.3 - GET_PR_FMT

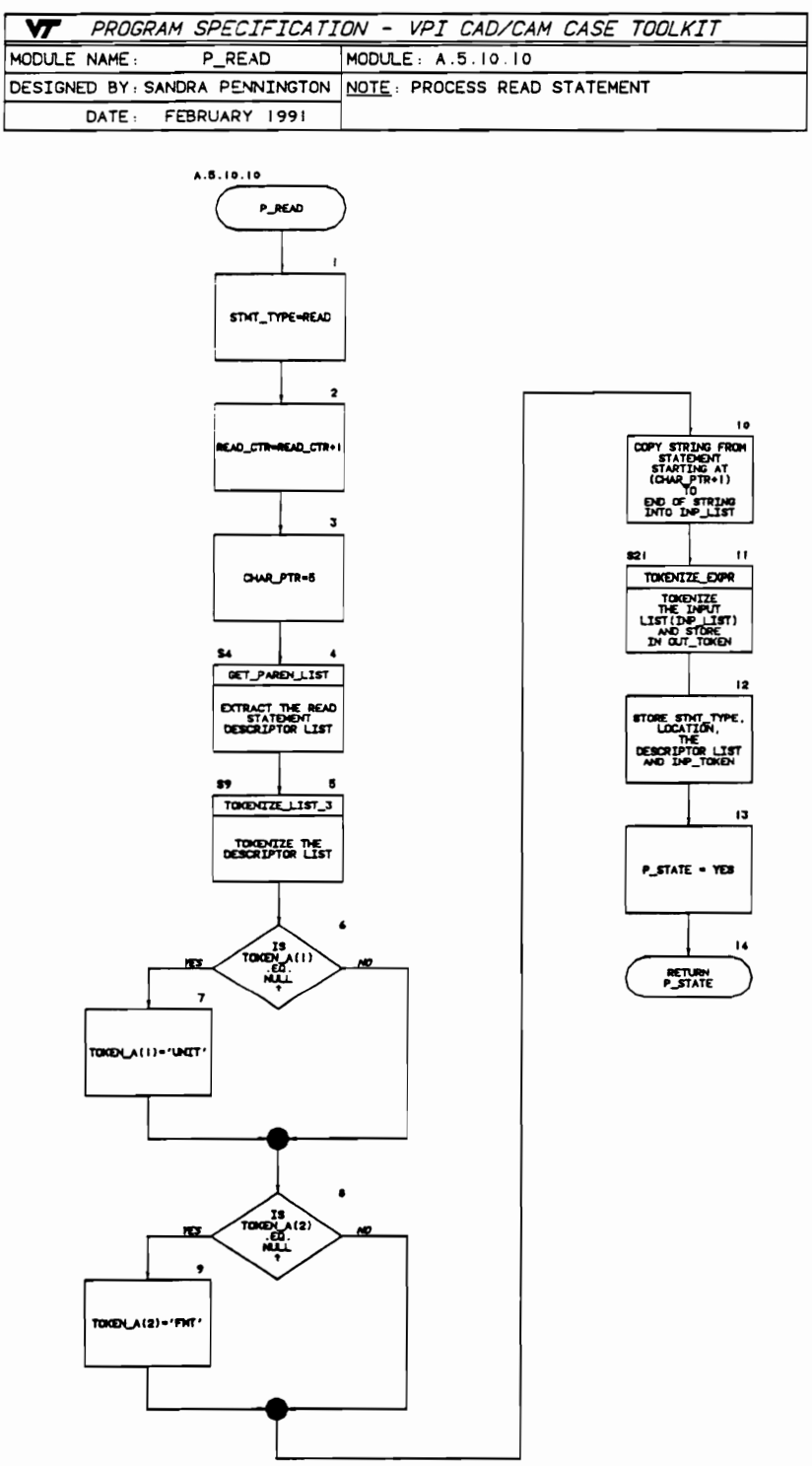


Figure 61. Module A.5.10.10 - P_READ

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_FORMAT	MODULE: A.5.10.12
DESIGNED BY: SANDRA PENNINGTON	NOTE: STORE LOCATION OF FORMAT STATEMENT
DATE: FEBRUARY 1991	

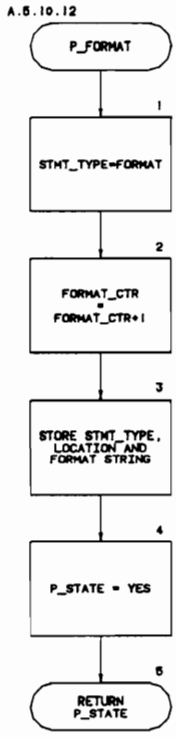


Figure 62. Module A.5.10.12 - P_FORMAT

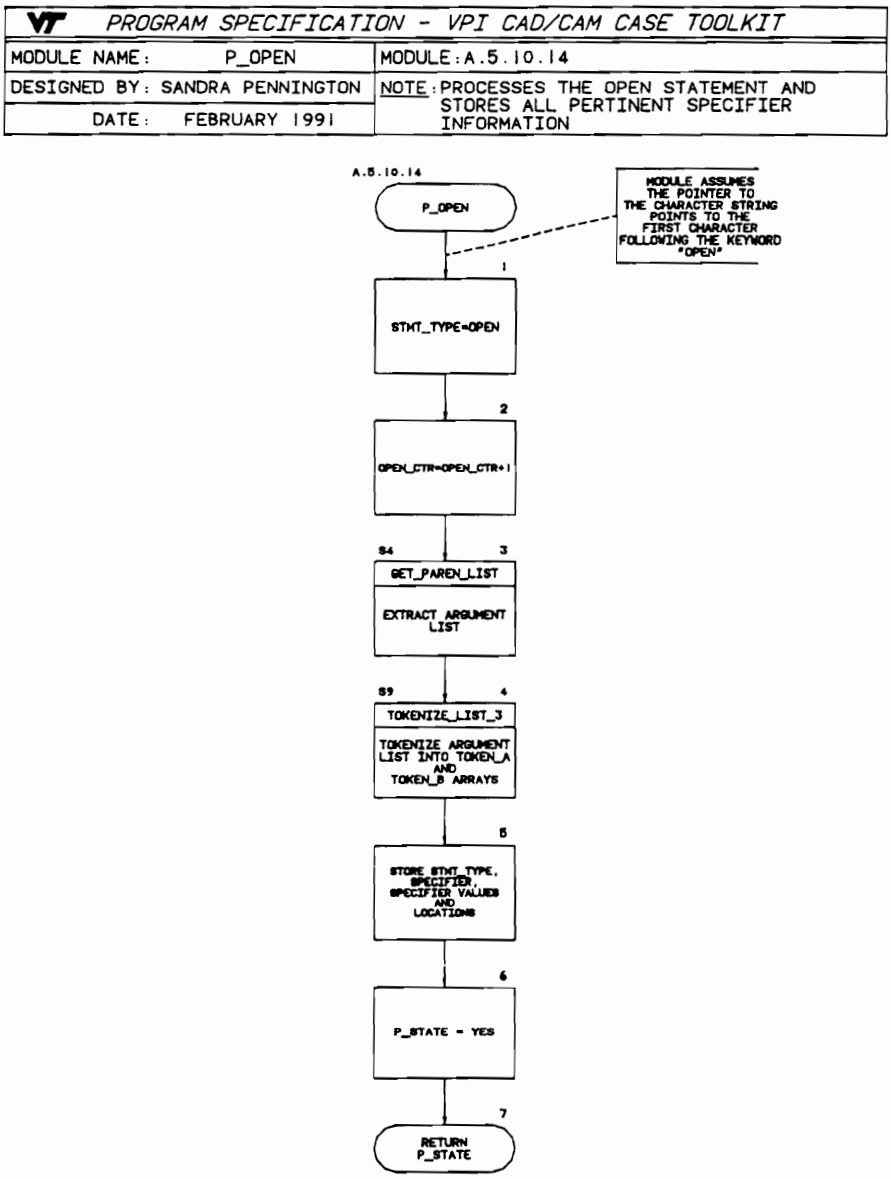


Figure 63. Module A.5.10.14 - P_OPEN

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_CLOSE	MODULE: A.5.10.16
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESSES THE CLOSE STATEMENT AND STORES ALL PERTINENT SPECIFIER INFORMATION
DATE: FEBRUARY 1991	

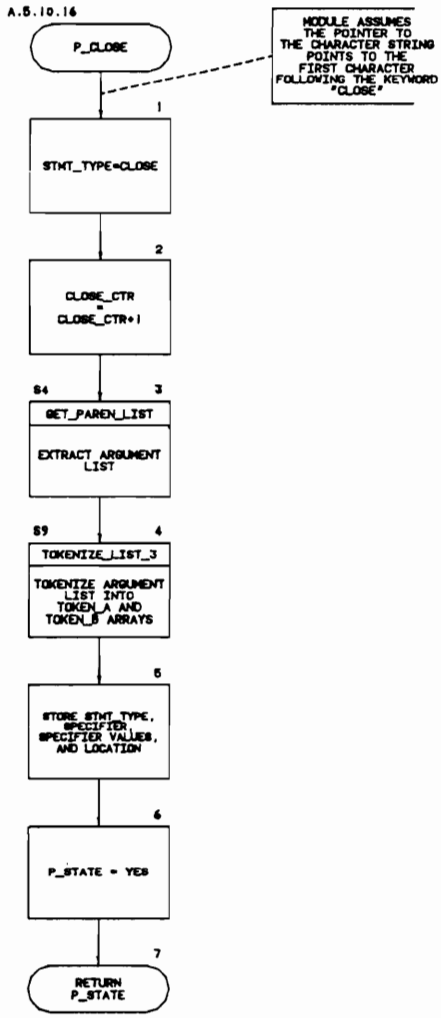


Figure 64. Module A.5.10.16 - P_CLOSE

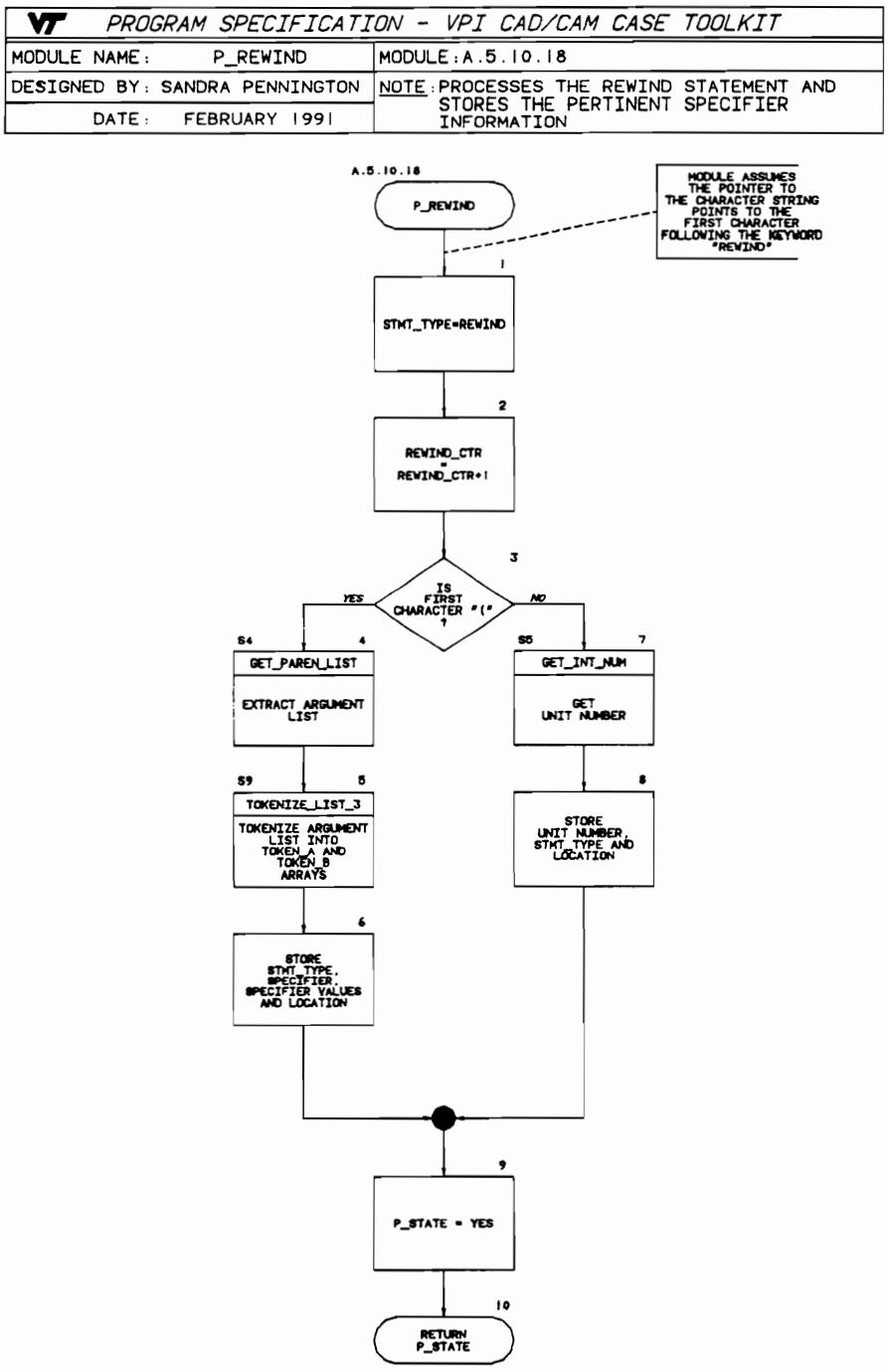


Figure 65. Module A.5.10.18 - P_REWIND

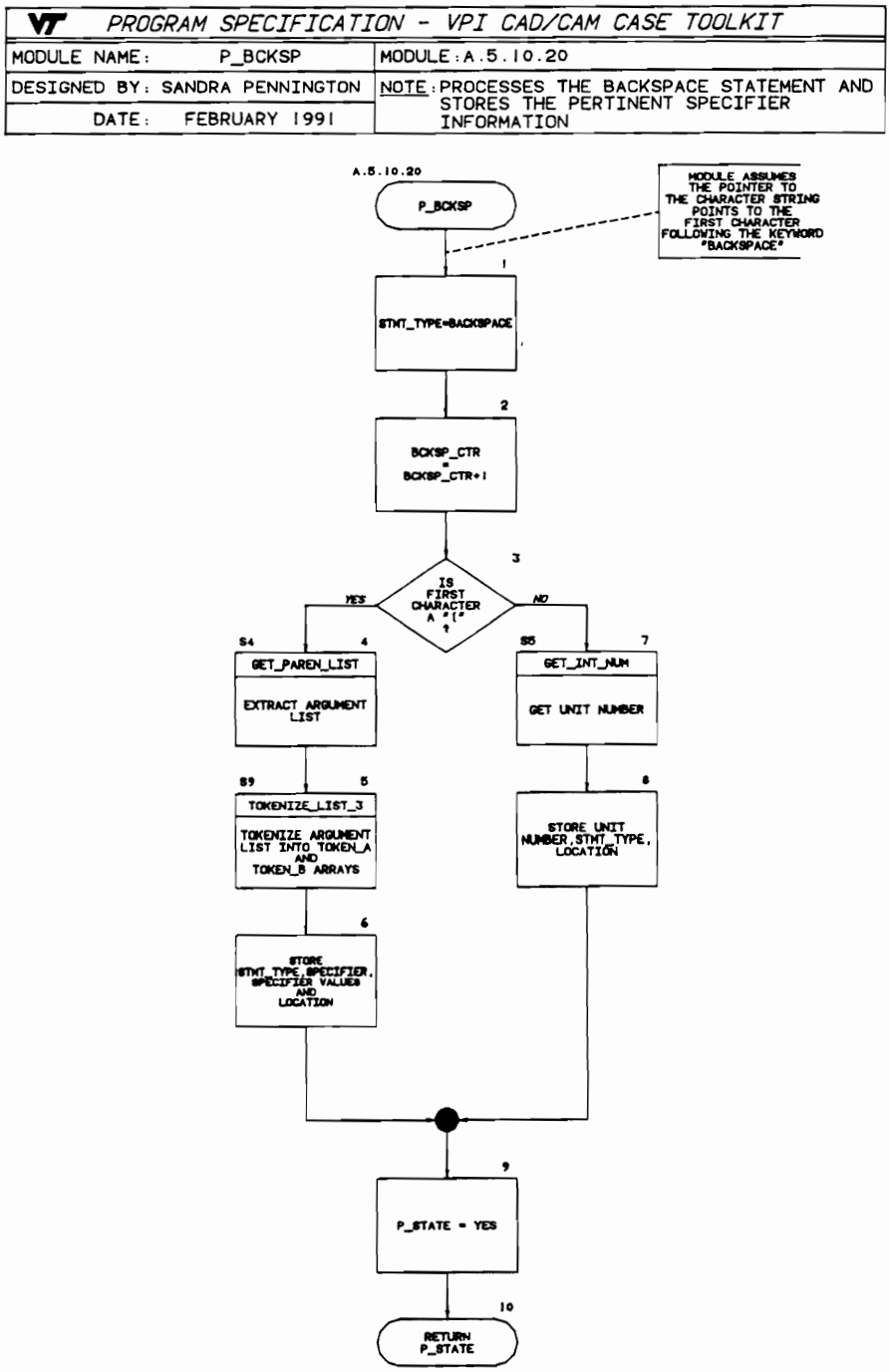


Figure 66. Module A.5.10.20 - P_BCKSP

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_ENDFIL	MODULE: A.5.10.22
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESSES THE ENDFILE STATEMENT AND STORES THE PERTINENT SPECIFIER INFORMATION
DATE: FEBRUARY 1991	

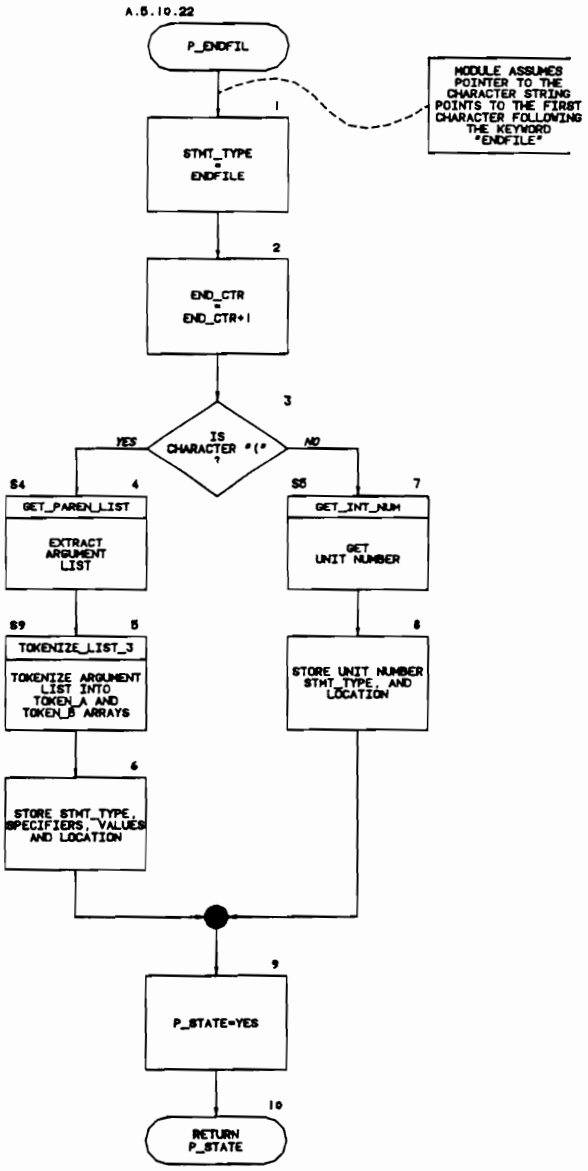


Figure 67. Module A.5.10.22 - P_ENDFIL

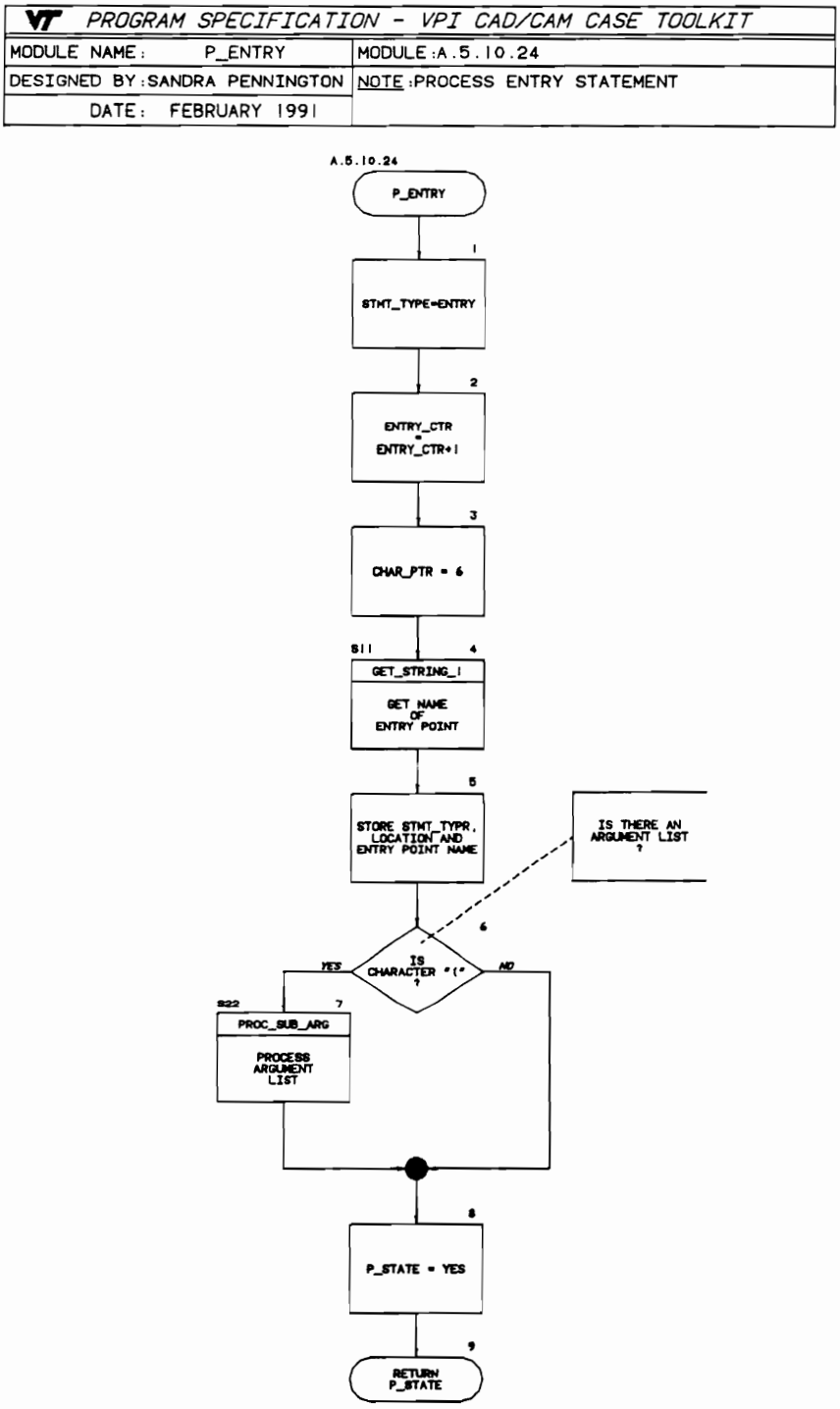


Figure 68. Module A.5.10.24 - P_ENTRY

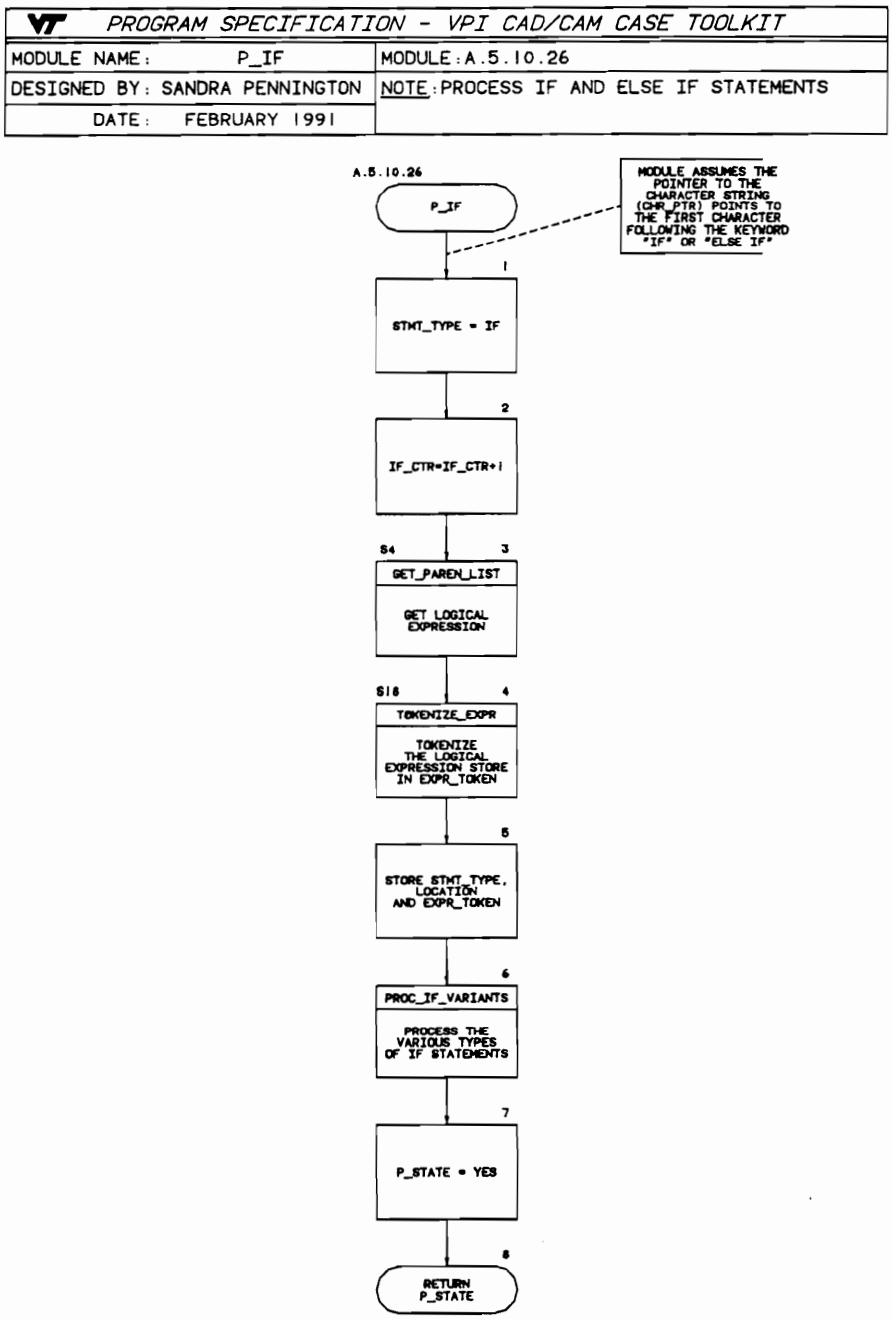


Figure 69. Module A.5.10.26 - P_IF

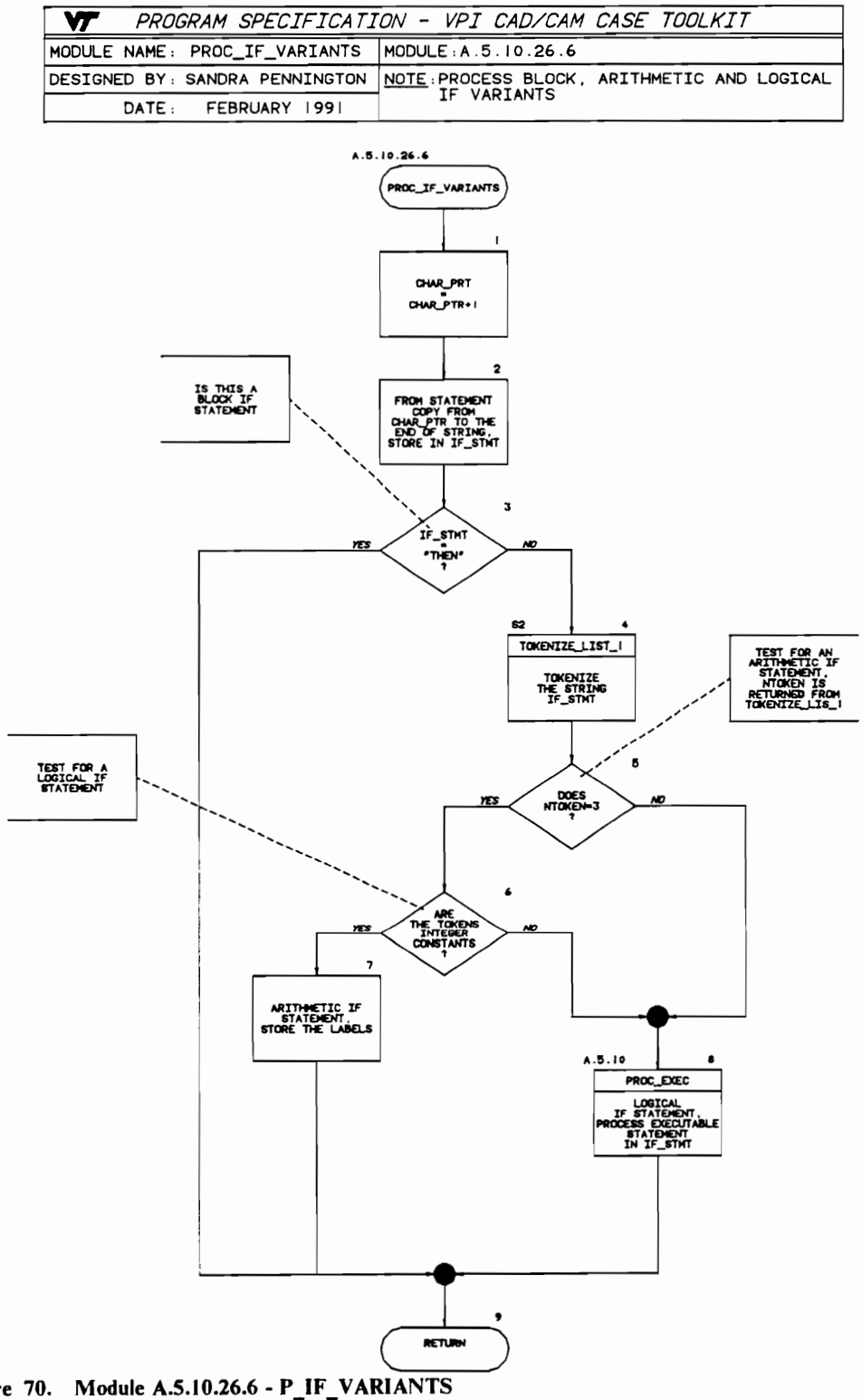


Figure 70. Module A.5.10.26.6 - P_IF_VARIANTS

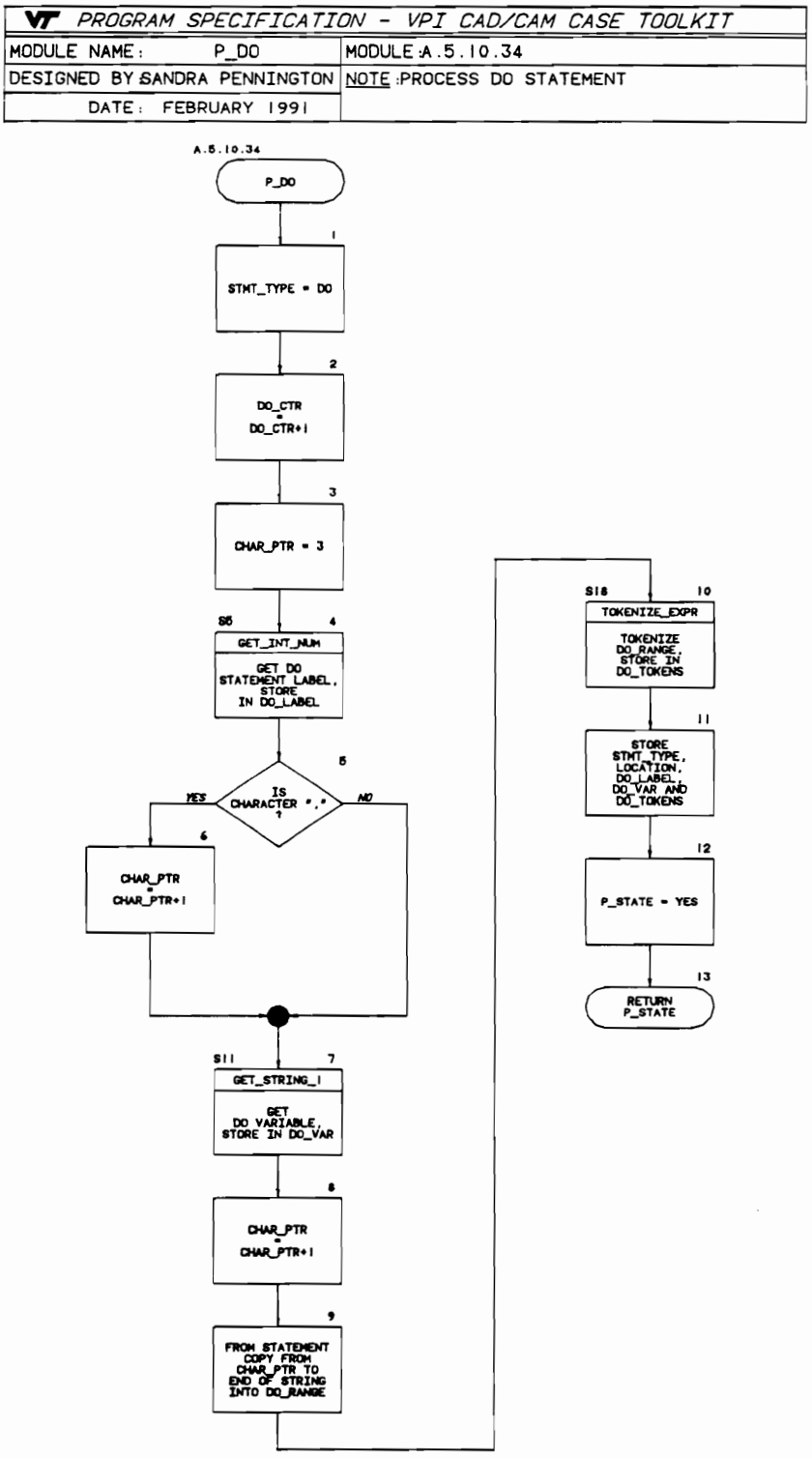


Figure 71. Module A.5.10.34 - P_DO

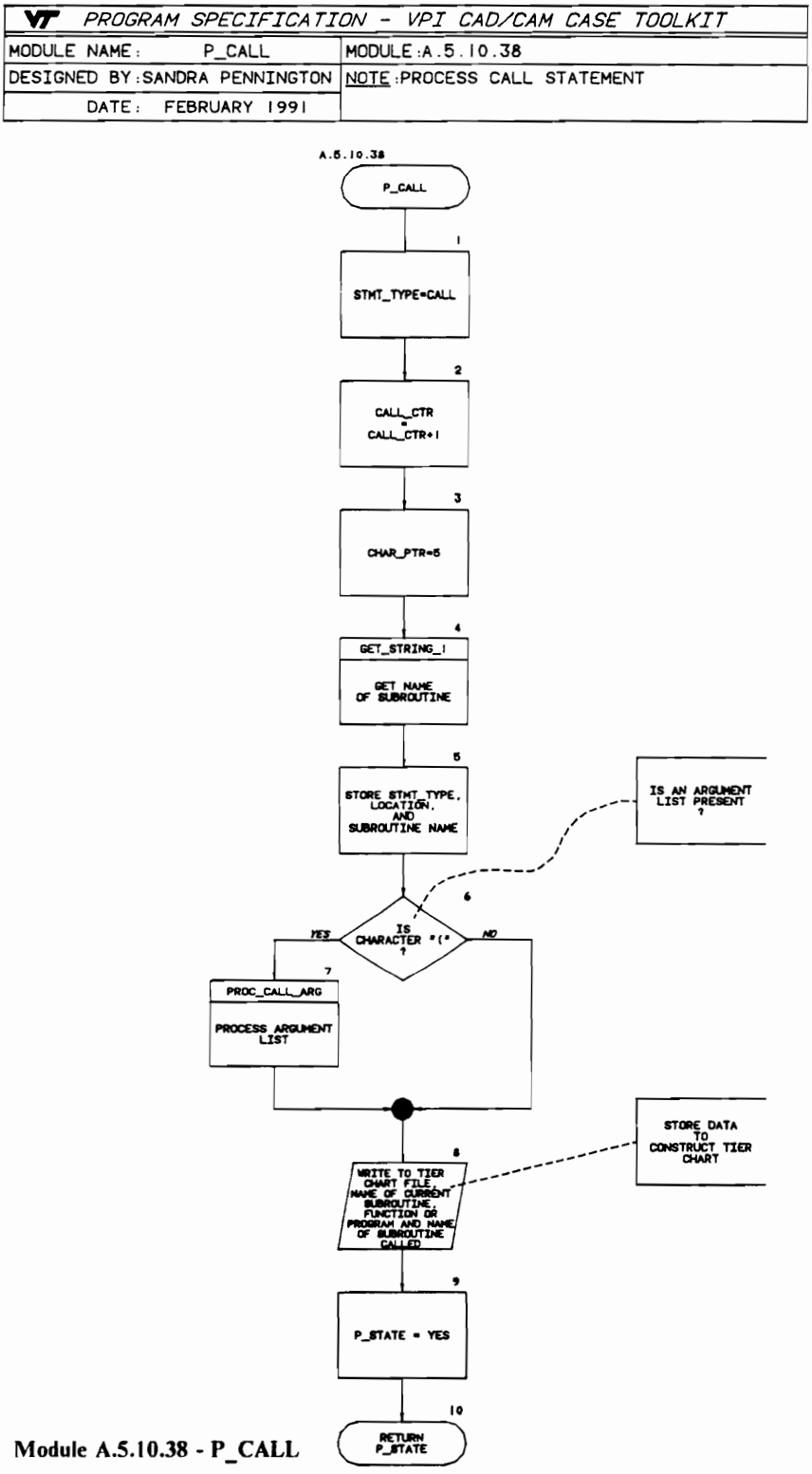


Figure 72. Module A.5.10.38 - P_CALL

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_CALL_ARG	MODULE: A.5.10.38.7
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS ARGUMENT LIST FROM CALL STATEMENT
DATE: FEBRUARY 1991	

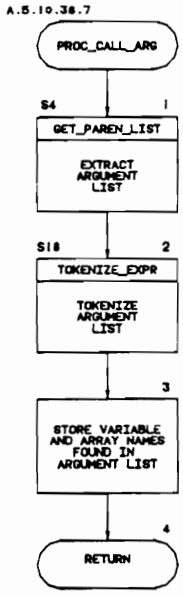


Figure 73. Module A.5.10.38.7 - PROC_CALL_ARG

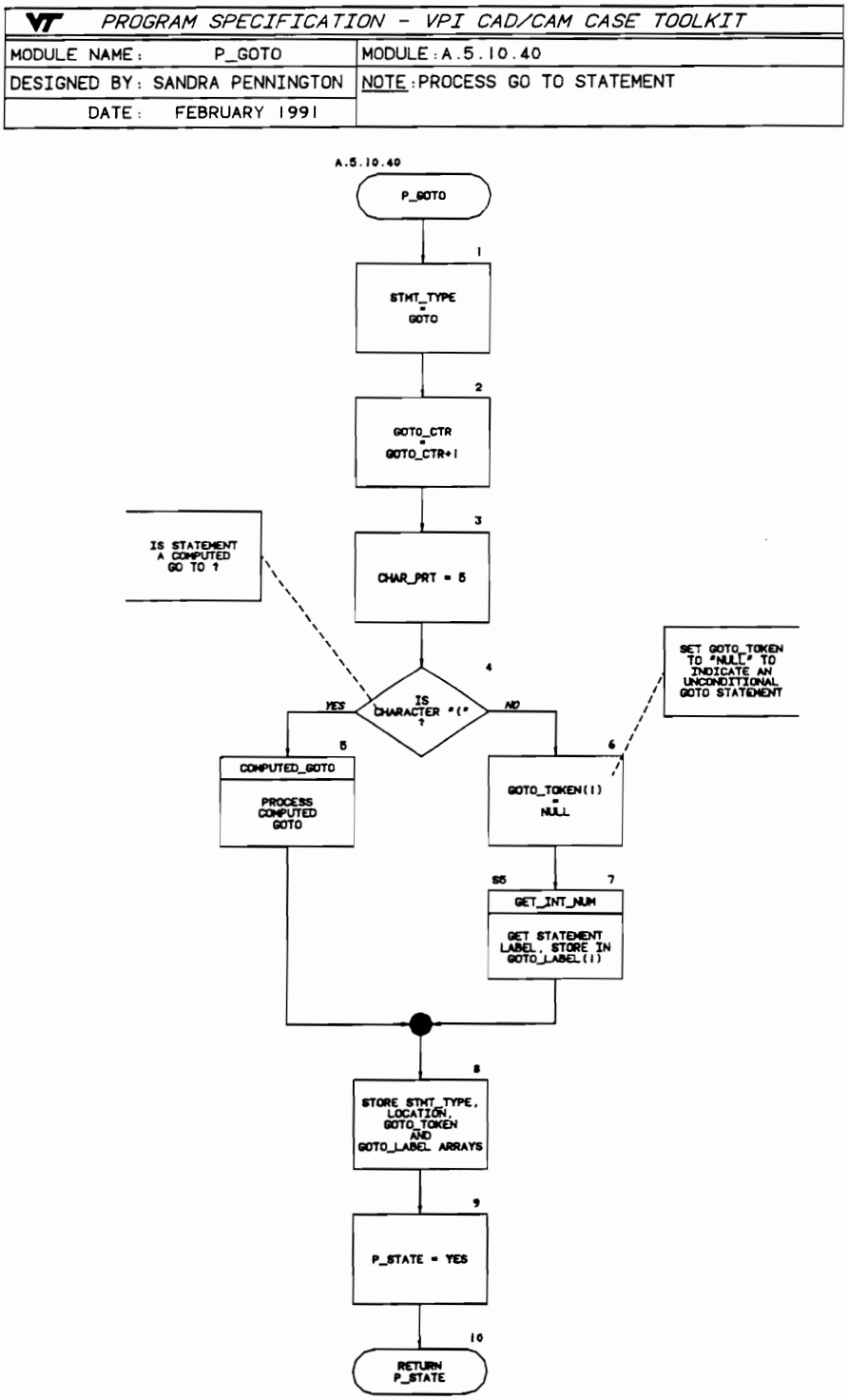


Figure 74. Module A.5.10.40 - P_GOTO

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: COMPUTED_GOTO	MODULE: A.5.10.40.5
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS COMPUTED GO TO STATEMENT
DATE: FEBRUARY 1991	

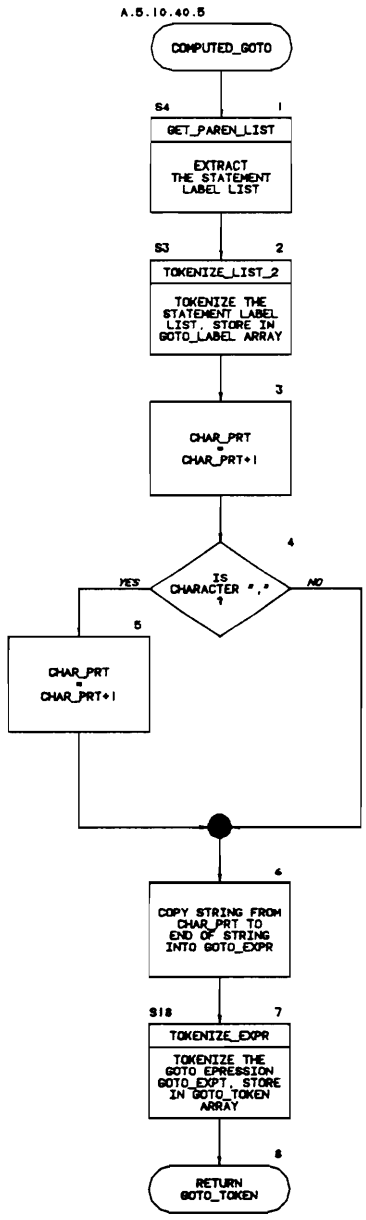


Figure 75. Module A.5.10.40.5 - COMPUTED_GOTO

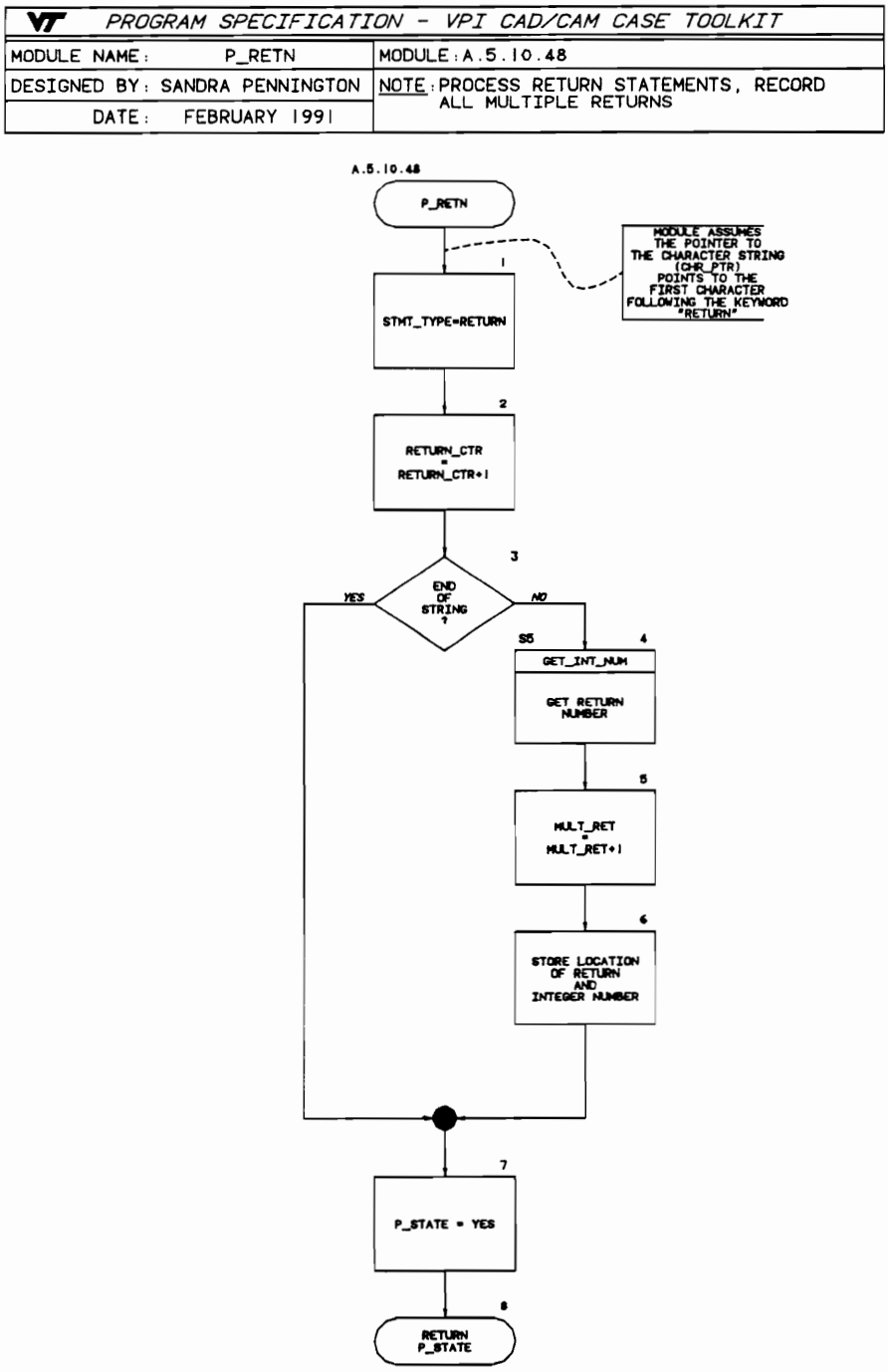


Figure 76. Module A.5.10.48 - P_RETN

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_END	MODULE: A.5.14
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS END STATEMENT
DATE: FEBRUARY 1991	

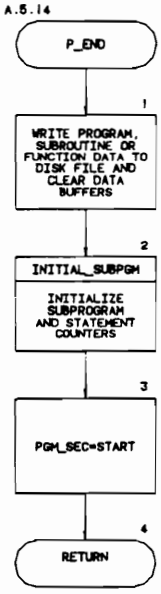


Figure 77. Module A.5.14 - P_END

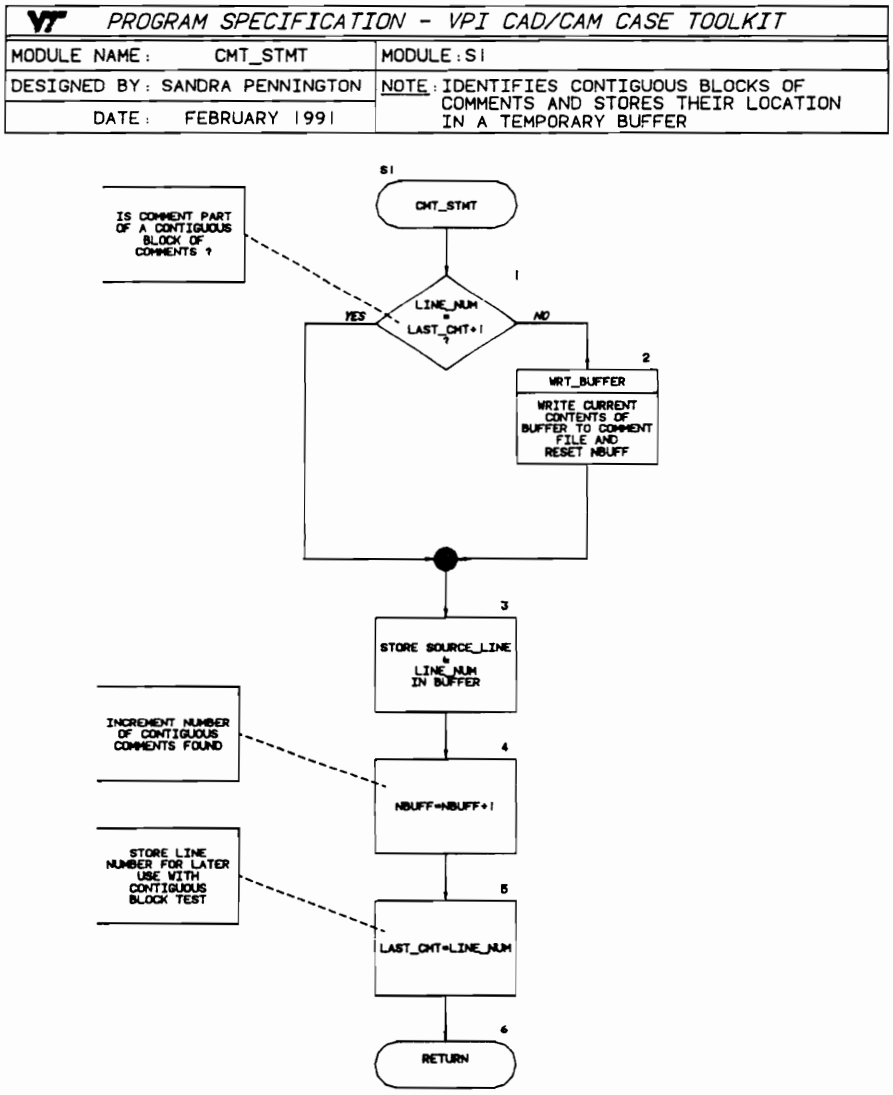


Figure 78. Module S1 - CMT_STMT

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: WRT_BUFF	MODULE: S1.2
DESIGNED BY: SANDRA PENNINGTON	NOTE: WRITES COMMENT BUFFER TO FILE AND RESETS BUFFER COUNTER
DATE: FEBRUARY 1991	

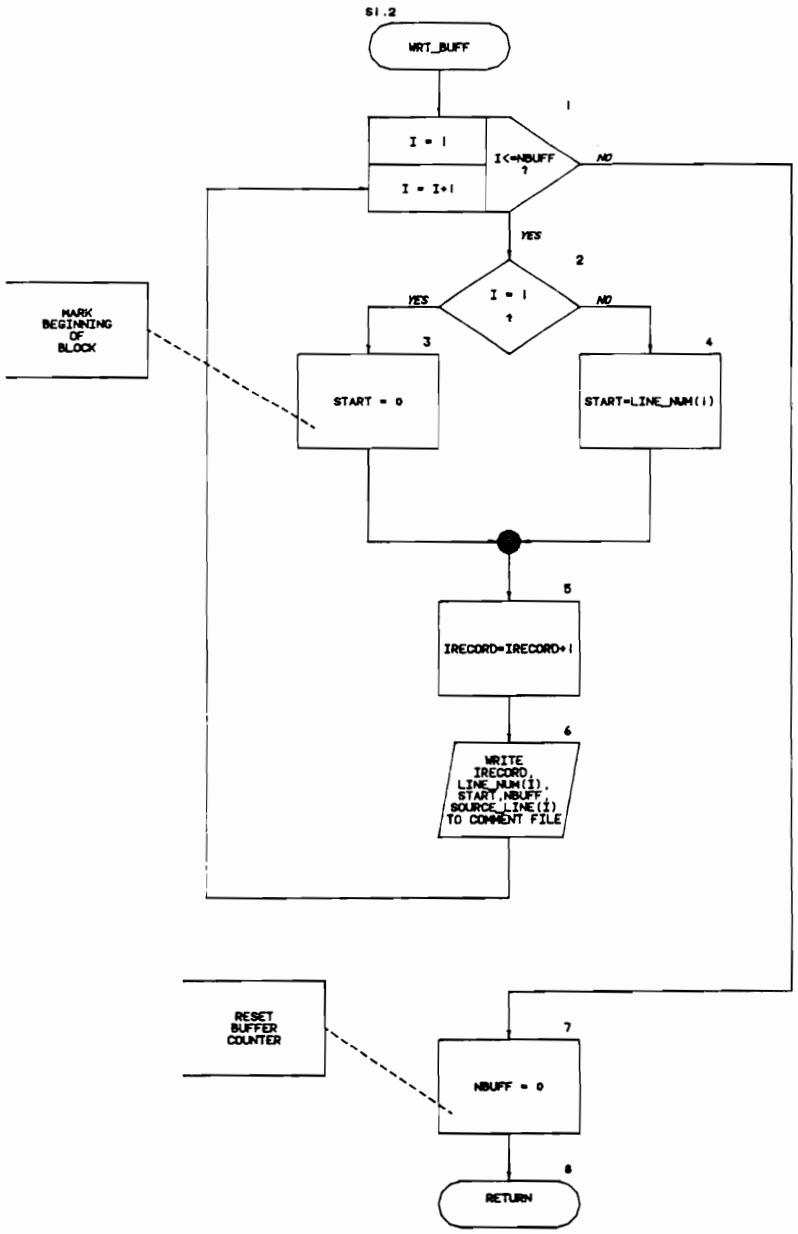


Figure 79. Module S1.2 - WRT_BUFF

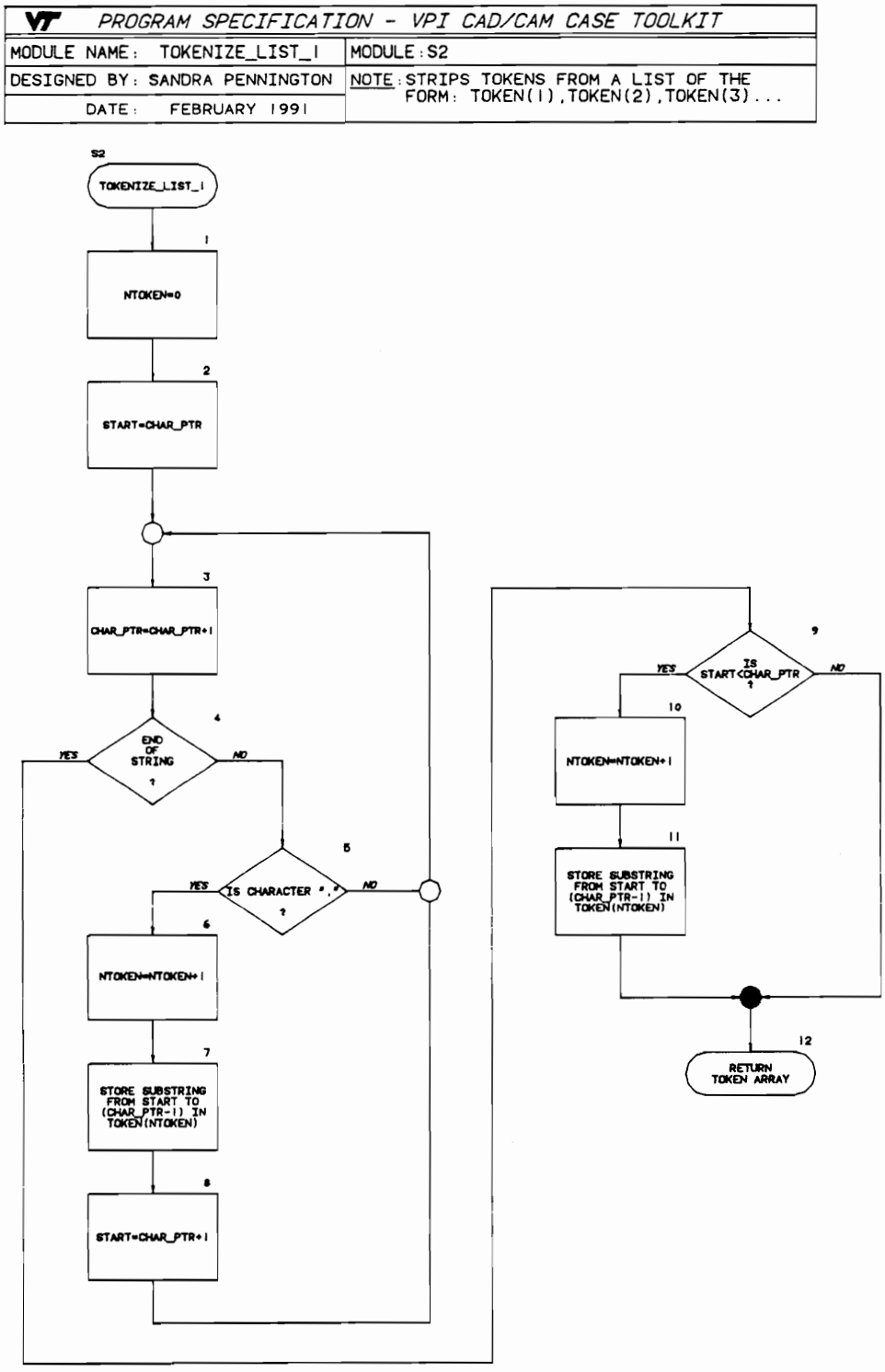


Figure 80. Module S2 - TOKENIZE_LIST_1

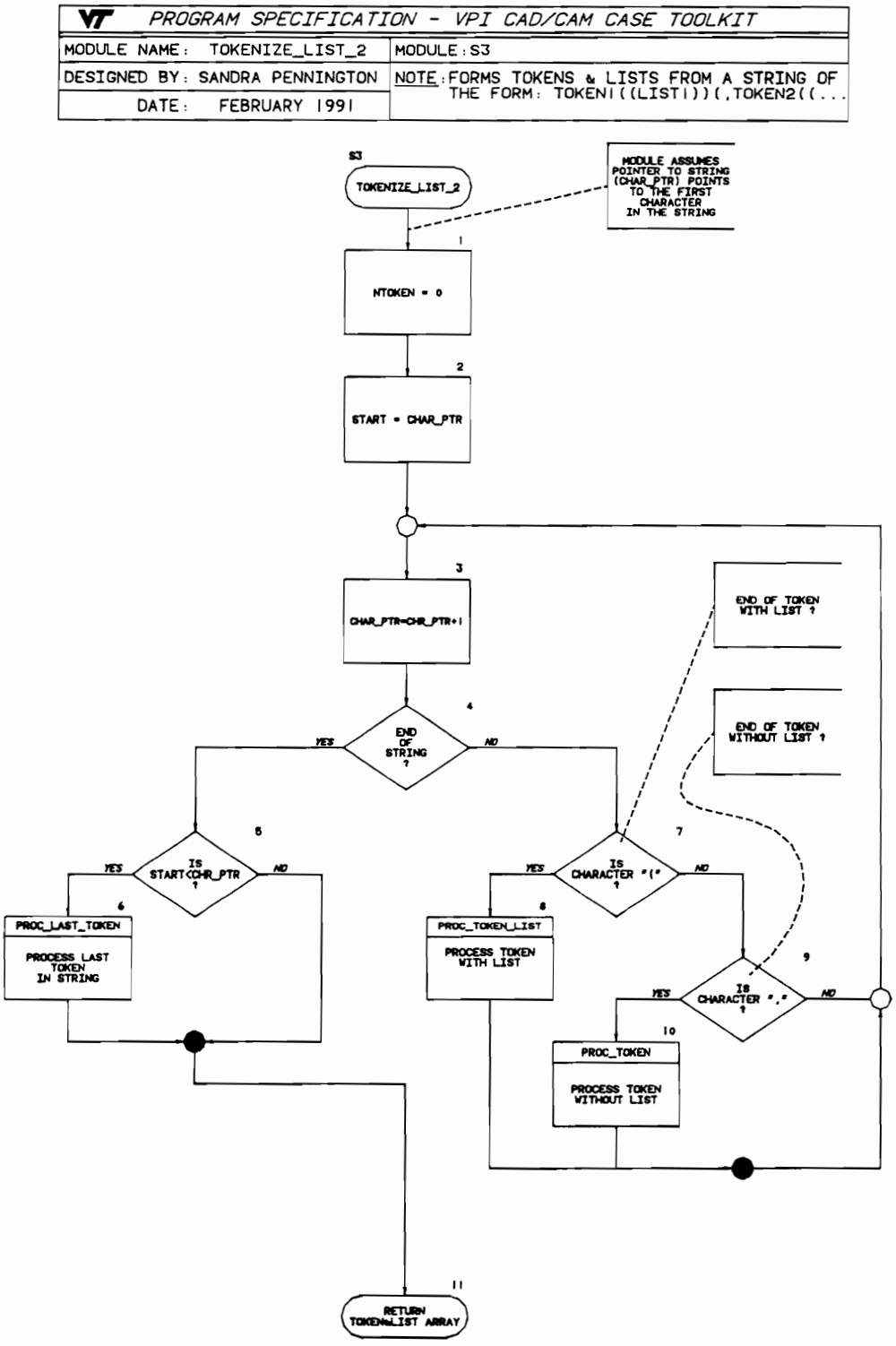


Figure 81. Module S3 - TOKENIZE_LIST_2

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_LAST_TOKEN	MODULE: S3.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS LAST TOKEN IN STRING
DATE: FEBRUARY 1991	

Figure 82. Module S3.6 - PROC_LAST_TOKEN

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_TOKEN_LIST	MODULE: S3.8
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS TOKEN WITH LIST
DATE: FEBRUARY 1991	

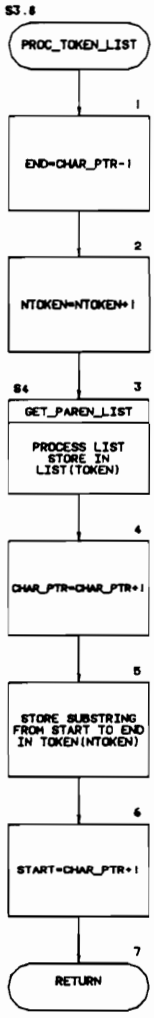


Figure 83. Module S3.8 - PROC_TOKEN_LIST

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_TOKEN	MODULE: S3.10
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS TOKEN WITHOUT LIST
DATE: FEBRUARY 1991	

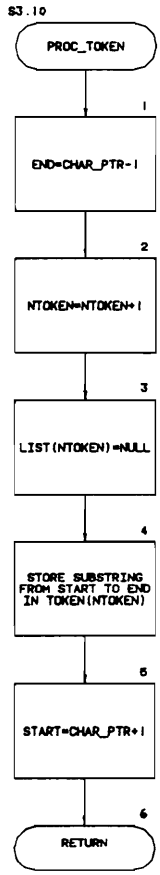


Figure 84. Module S3.10 - PROC_TOKEN

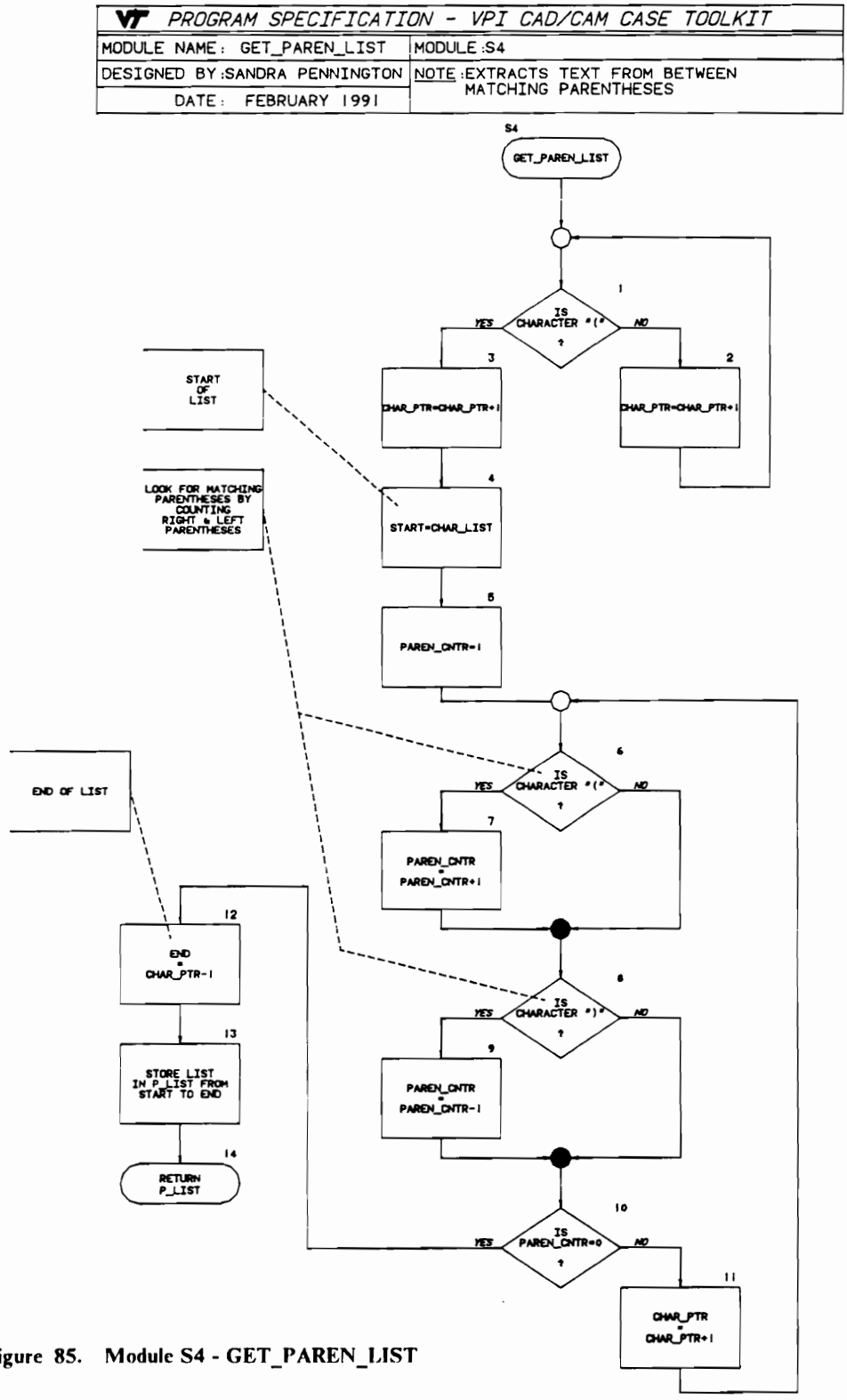


Figure 85. Module S4 - GET_PAREN_LIST

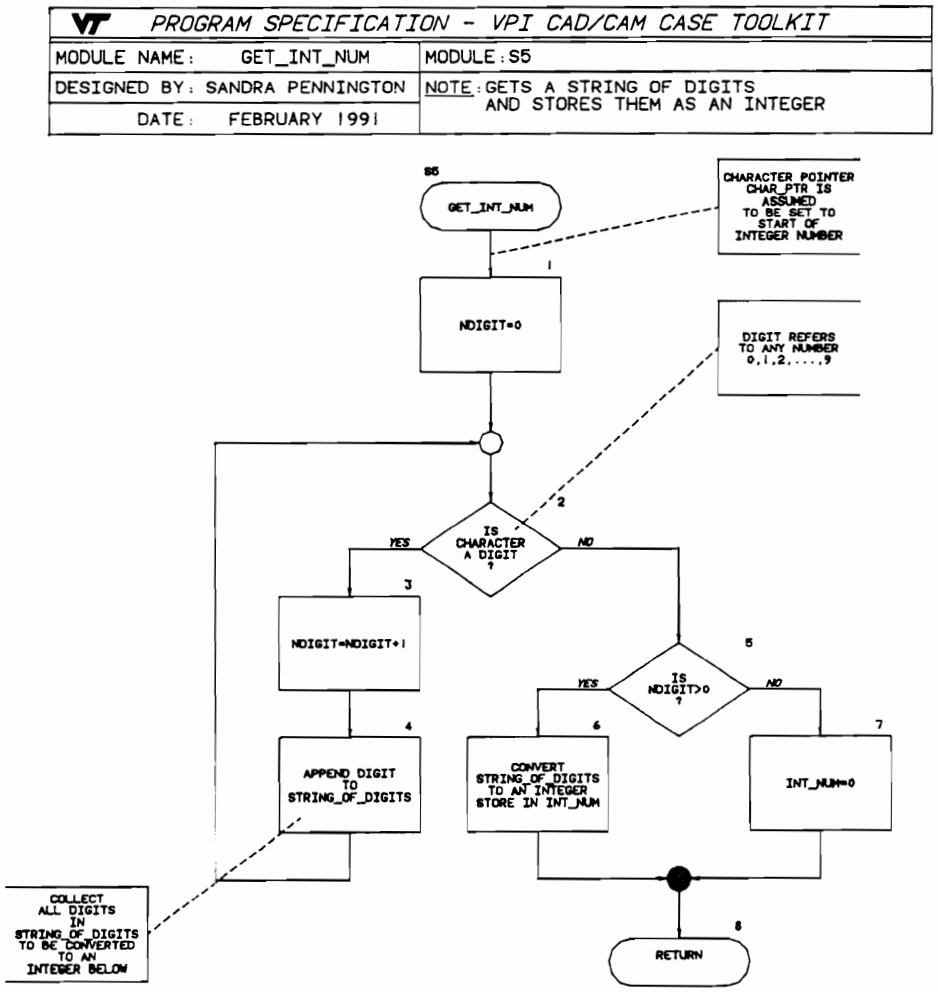


Figure 86. Module S5 - GET_INT_NUM

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_LEN_MOD	MODULE: S6
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS LENGTH MODIFIER
DATE: FEBRUARY 1991	

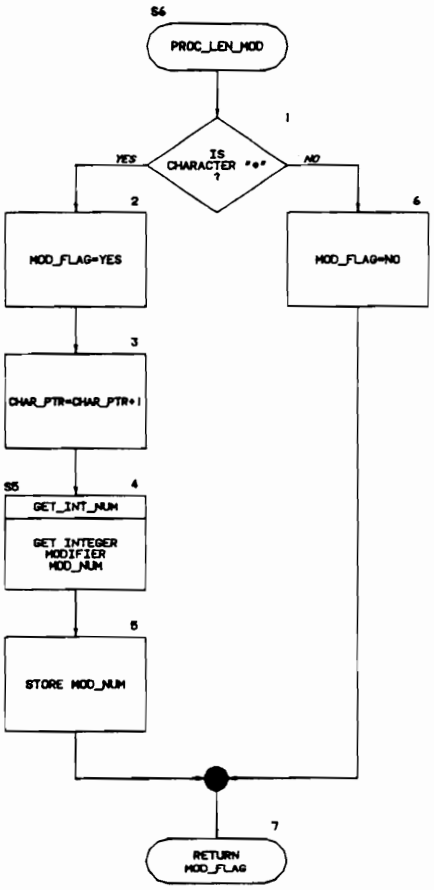


Figure 87. Module S6 - PROC_LEN_MOD

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_NAMES	MODULE : S7
DESIGNED BY: SANDRA PENNINGTON	NOTE : PROCESS VARIABLE AND ARRAY NAMES
DATE: FEBRUARY 1991	

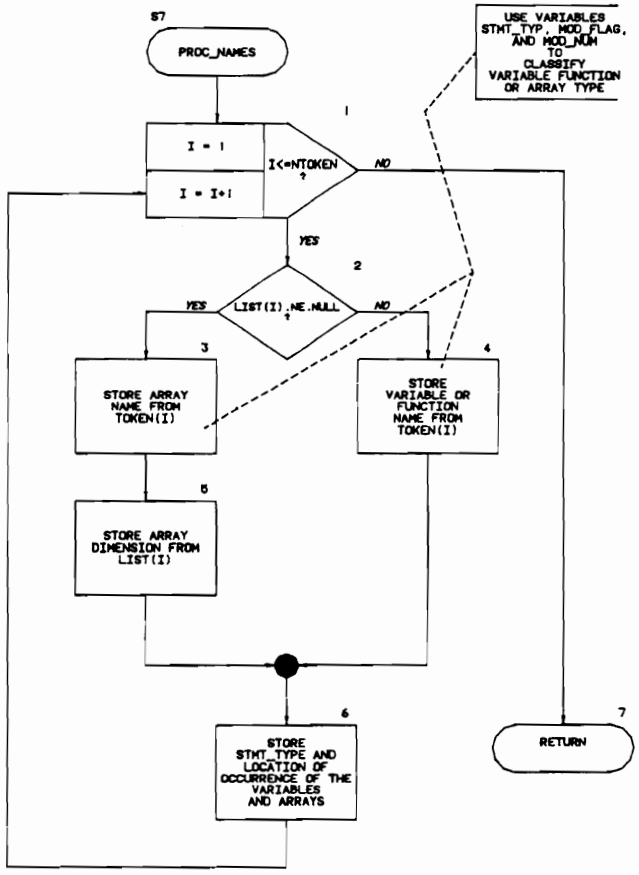


Figure 88. Module S7 - PROC_NAMES

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: P_DUMMY	MODULE : S8
DESIGNED BY: SANDRA PENNINGTON	NOTE : IDENTIFIES STATEMENT BUT DOES NOTHING
DATE: FEBRUARY 1991	

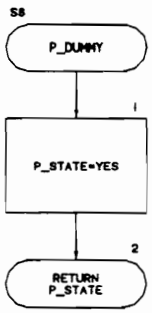


Figure 89. Module S8 - P_DUMMY

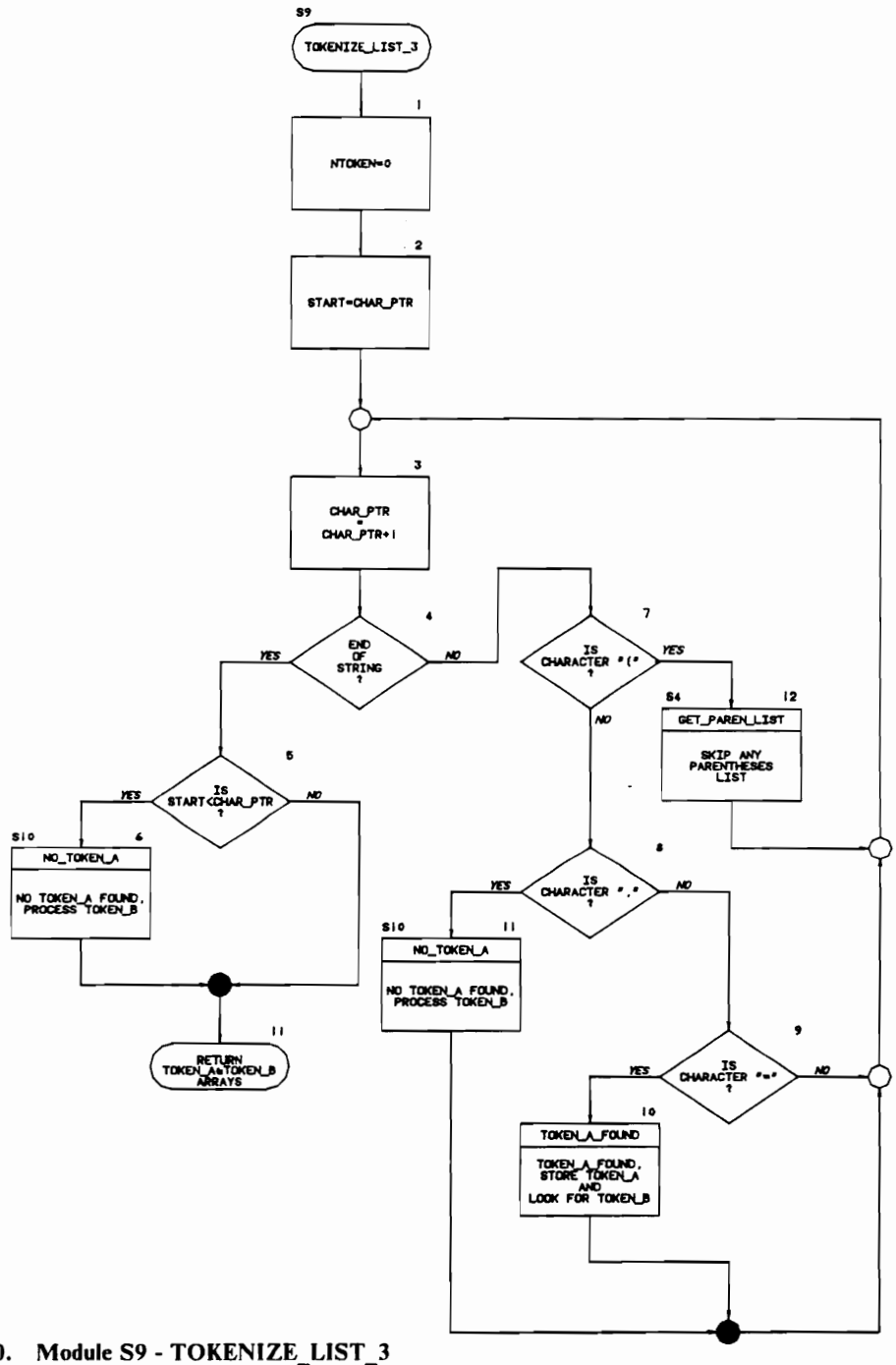


Figure 90. Module S9 - TOKENIZE_LIST_3

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TOKEN_A_FOUND	MODULE: S9.10
DESIGNED BY: SANDRA PENNINGTON	NOTE: TOKEN_A FOUND, STORE TOKEN AND LOOK FOR TOKEN_B
DATE: FEBRUARY 1991	

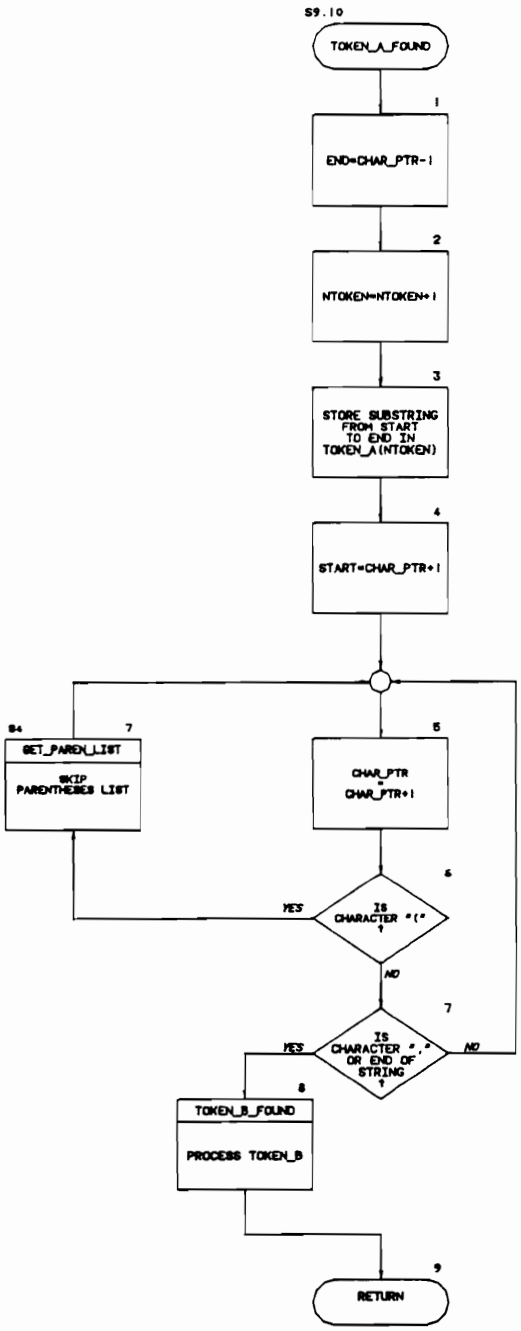


Figure 91. Module S9.10 - TOKEN_A_FOUND

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TOKEN_B_FOUND	MODULE: S9.10.7
DESIGNED BY: SANDRA PENNINGTON	NOTE: TOKEN_B FOUND, STORE TOKEN_B
DATE: FEBRUARY 1991	

Figure 92. Module S9.10.7 - TOKEN_B_FOUND

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: NO_TOKEN_A	MODULE: S10
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS TOKEN_B, NO TOKEN_A FOUND
DATE: FEBRUARY 1991	

Figure 93. Module S10 - NO_TOKEN_A

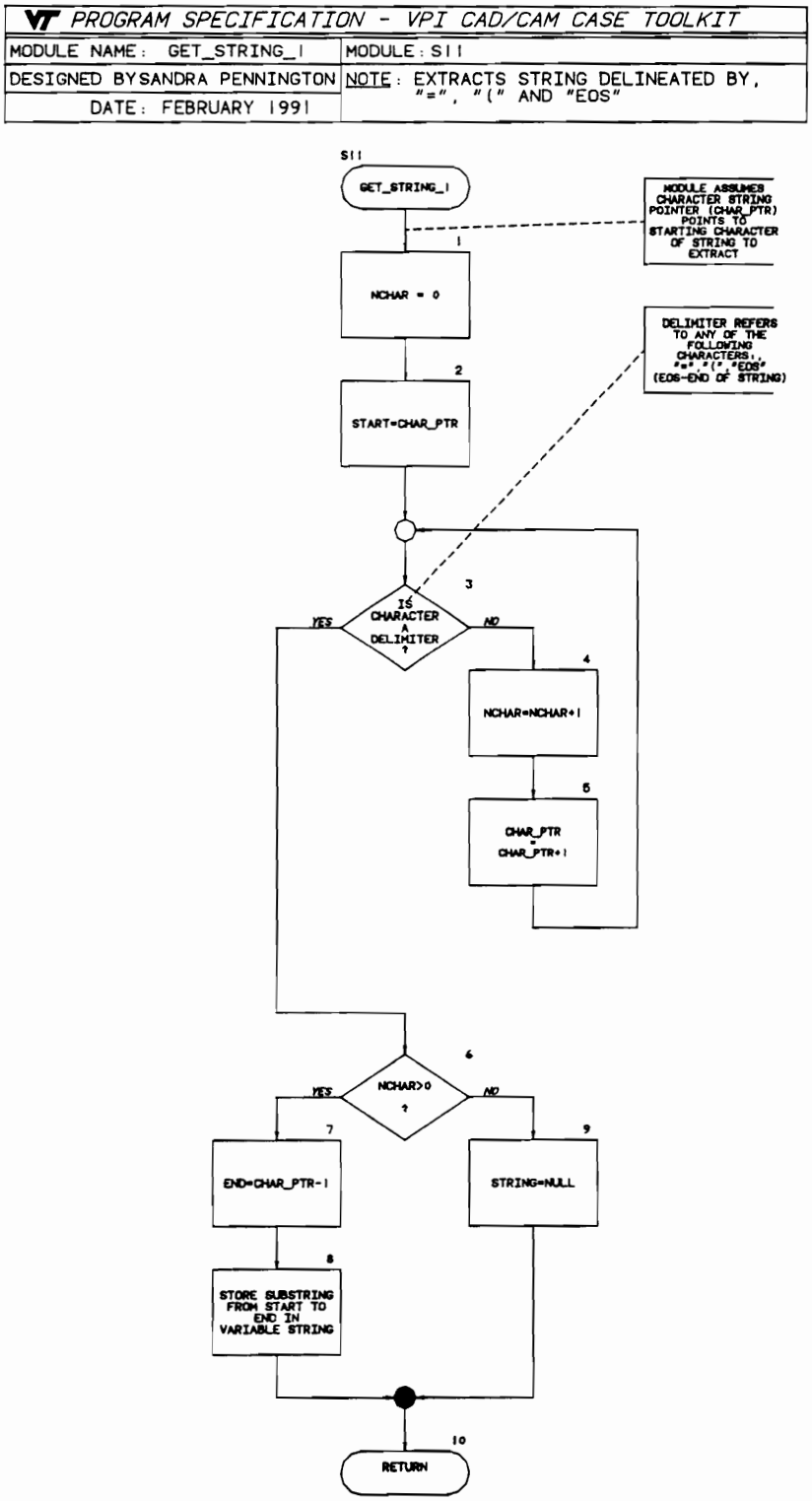


Figure 94. Module S11 - GET_STRING_1

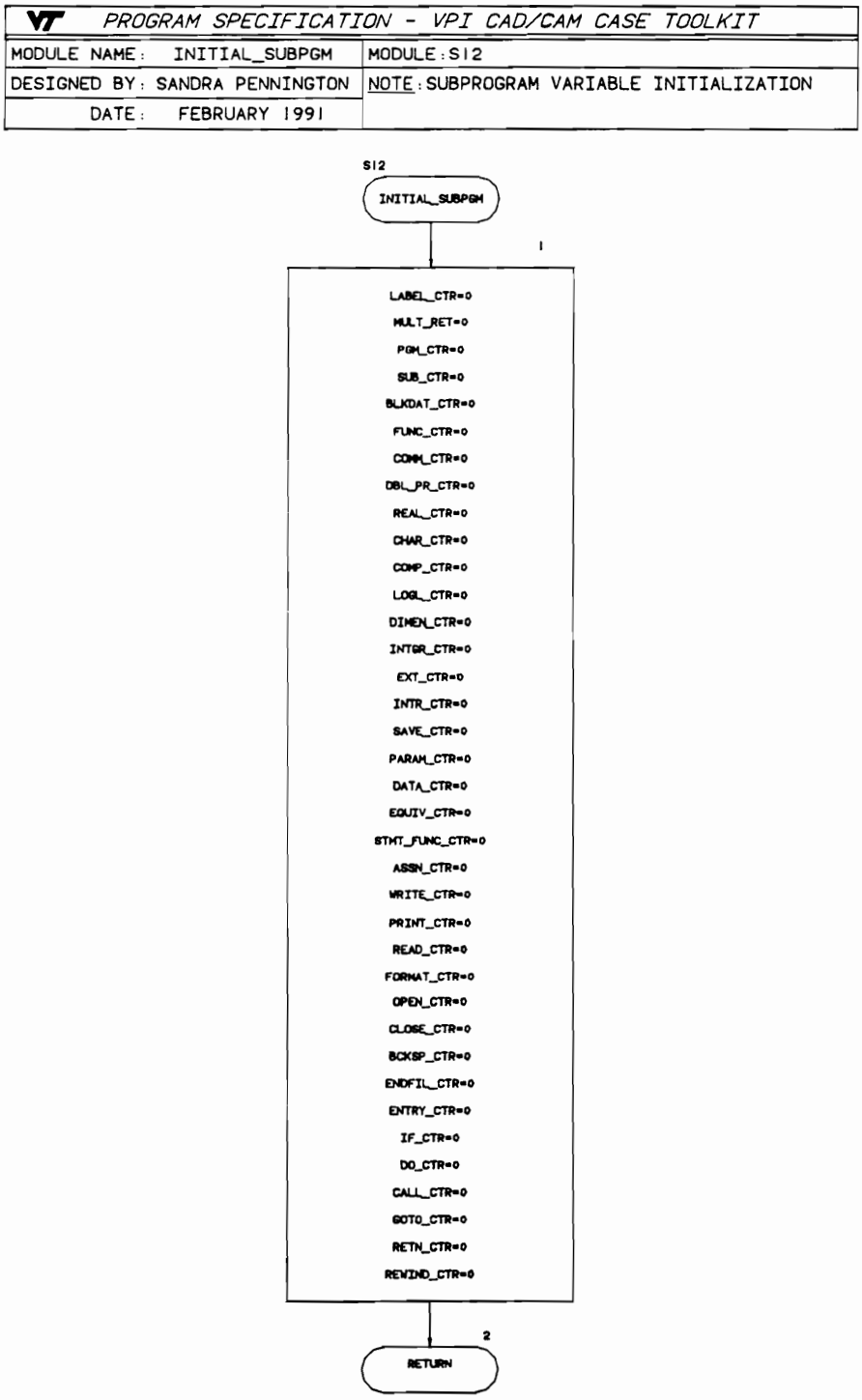


Figure 95. Module S12 - INITIAL_SUBPGM

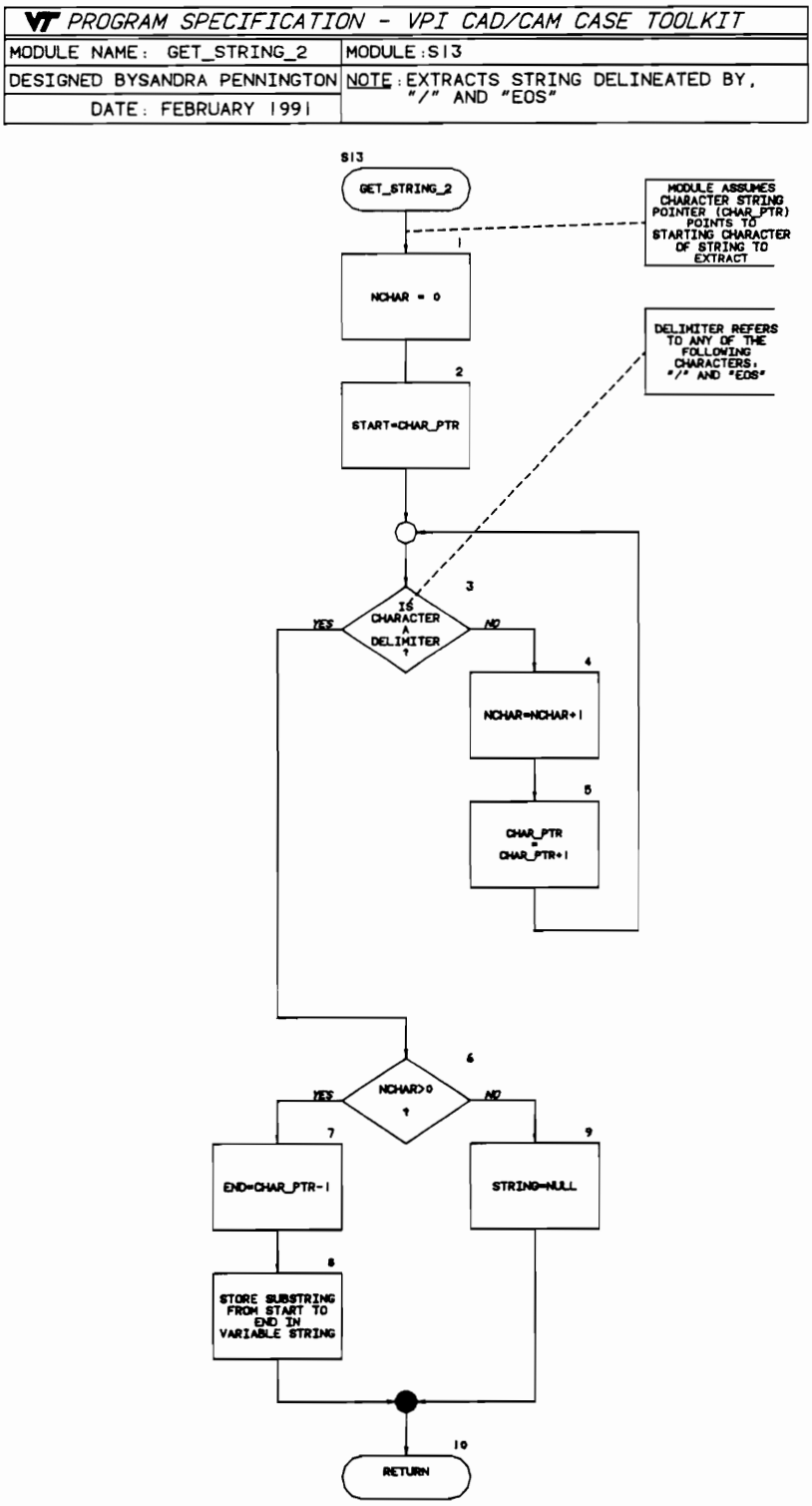


Figure 96. Module S13 - GET_STRING_2

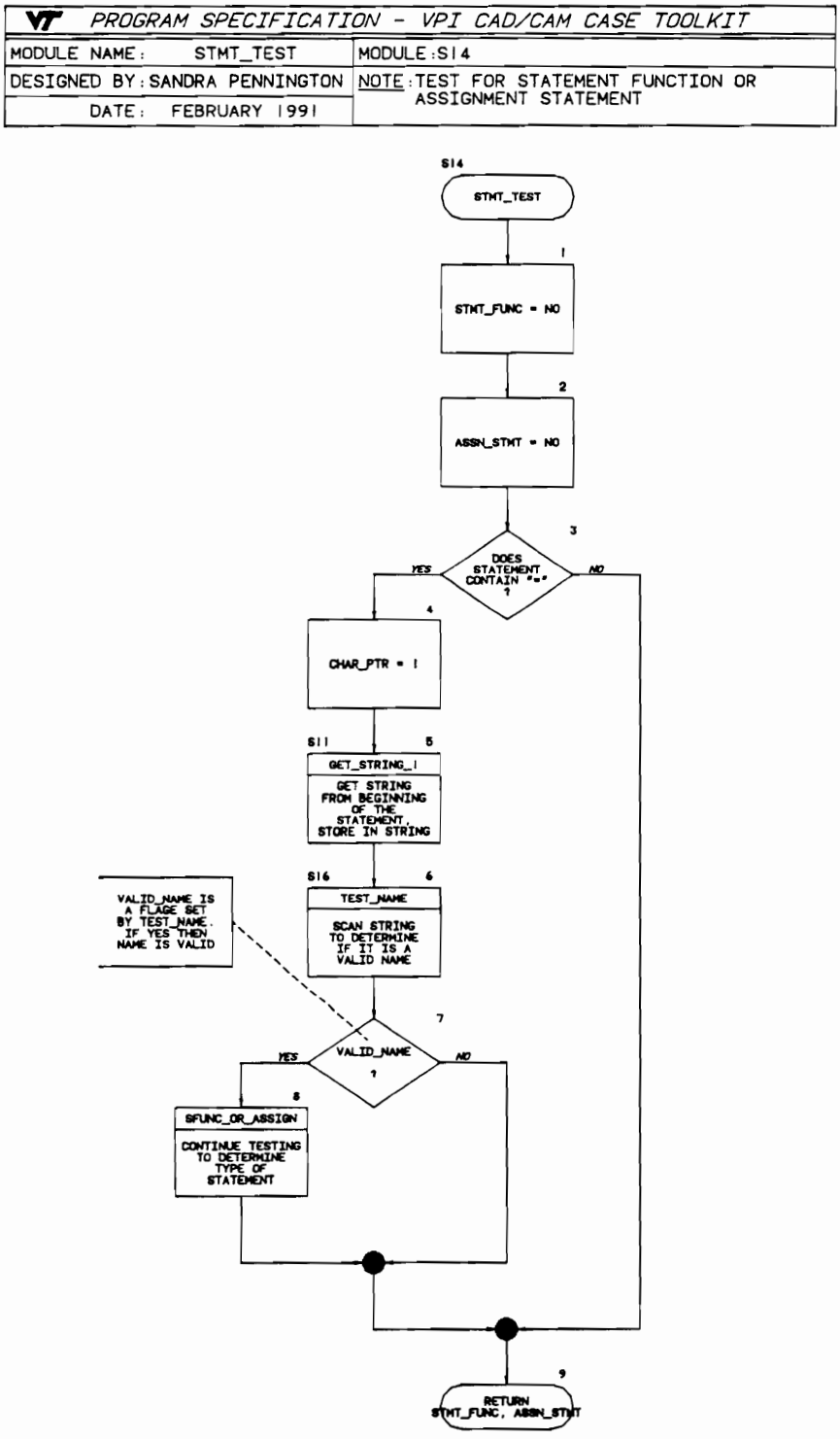


Figure 97. Module S14 - STMT_TEST

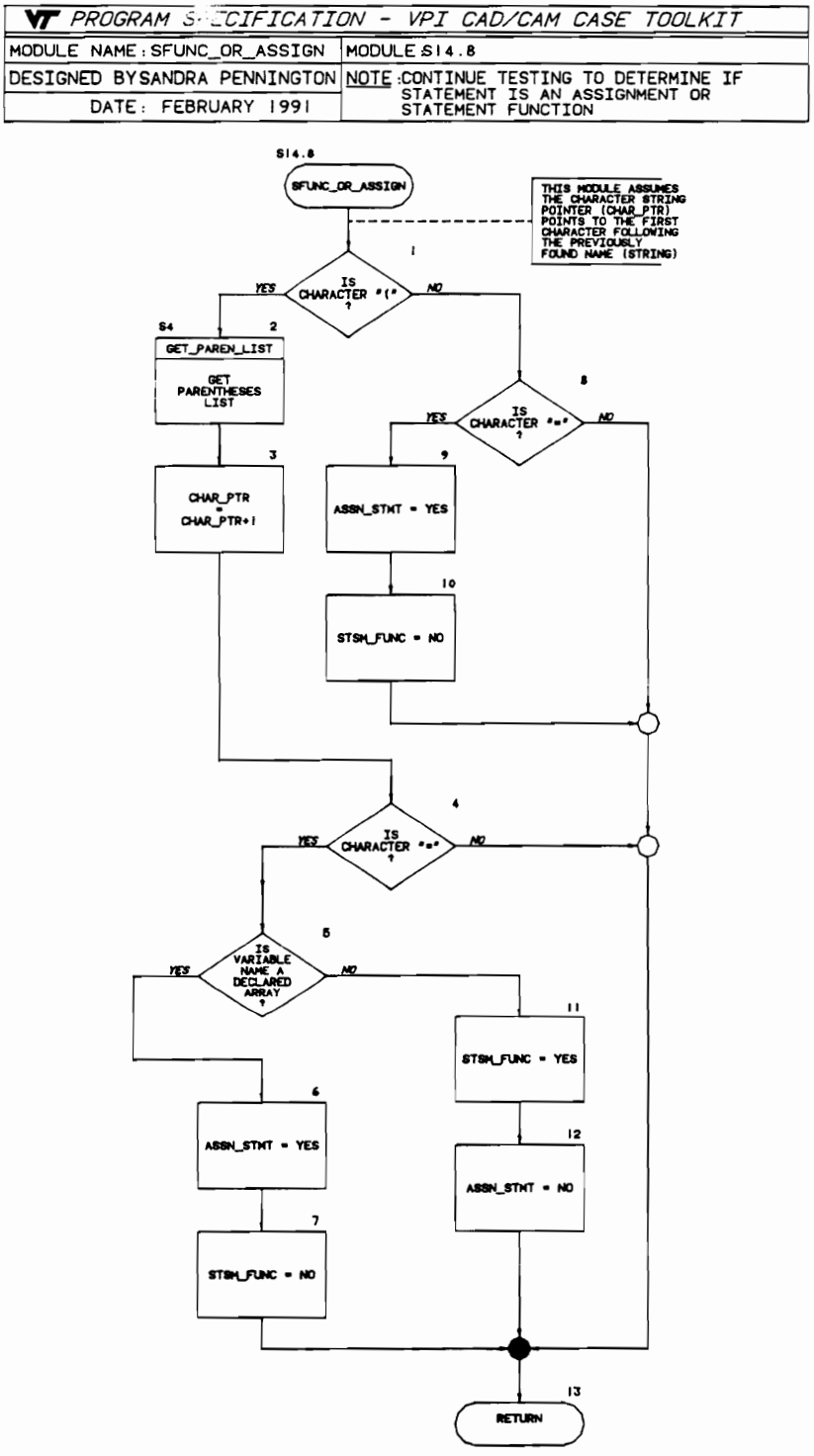


Figure 98. Module S14.8 - SFUNC_OR_ASSIGN

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: GET_CHAR_CONST	MODULE: S15
DESIGNED BY: SANDRA PENNINGTON	NOTE: EXTRACT CHARACTER CONSTANT
DATE: FEBRUARY 1991	

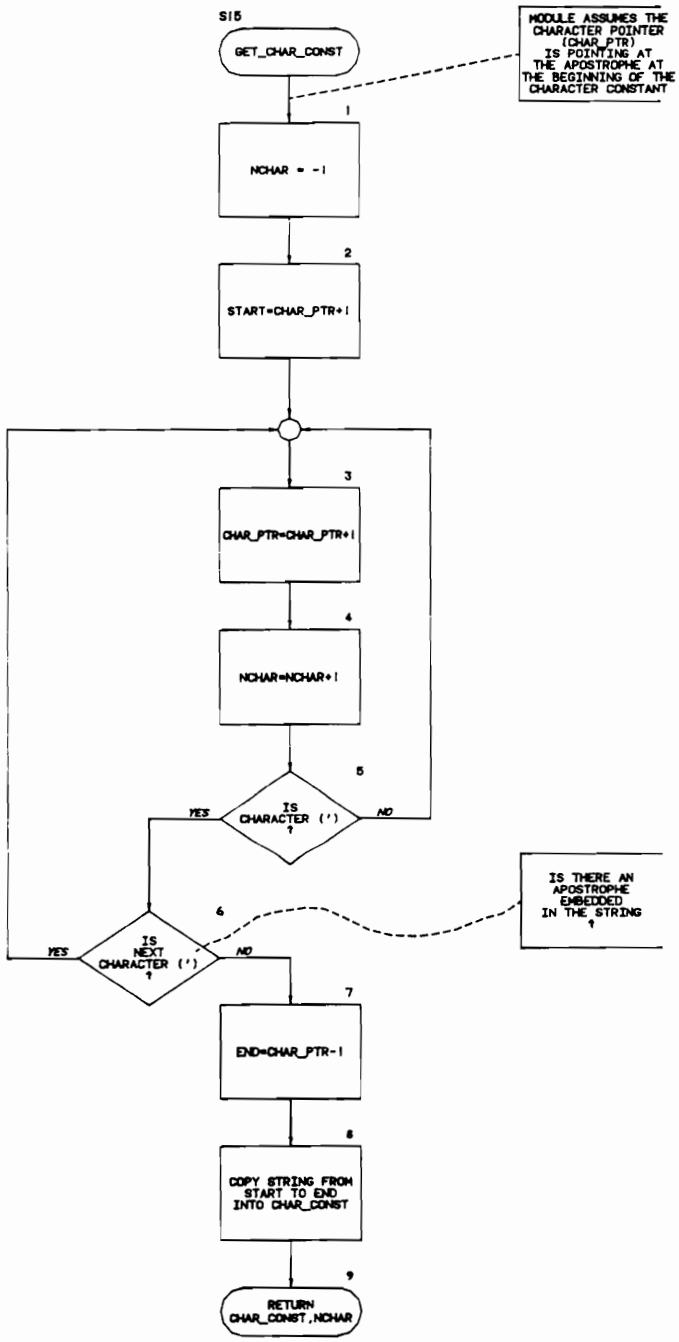


Figure 99. Module S15 - GET_CHAR_CONST

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TEST_NAME	MODULE: S16
DESIGNED BY: SANDRA PENNINGTON	NOTE: TEST STRING FOR VALID NAME OR INTEGER CONSTANT
DATE: FEBRUARY 1991	

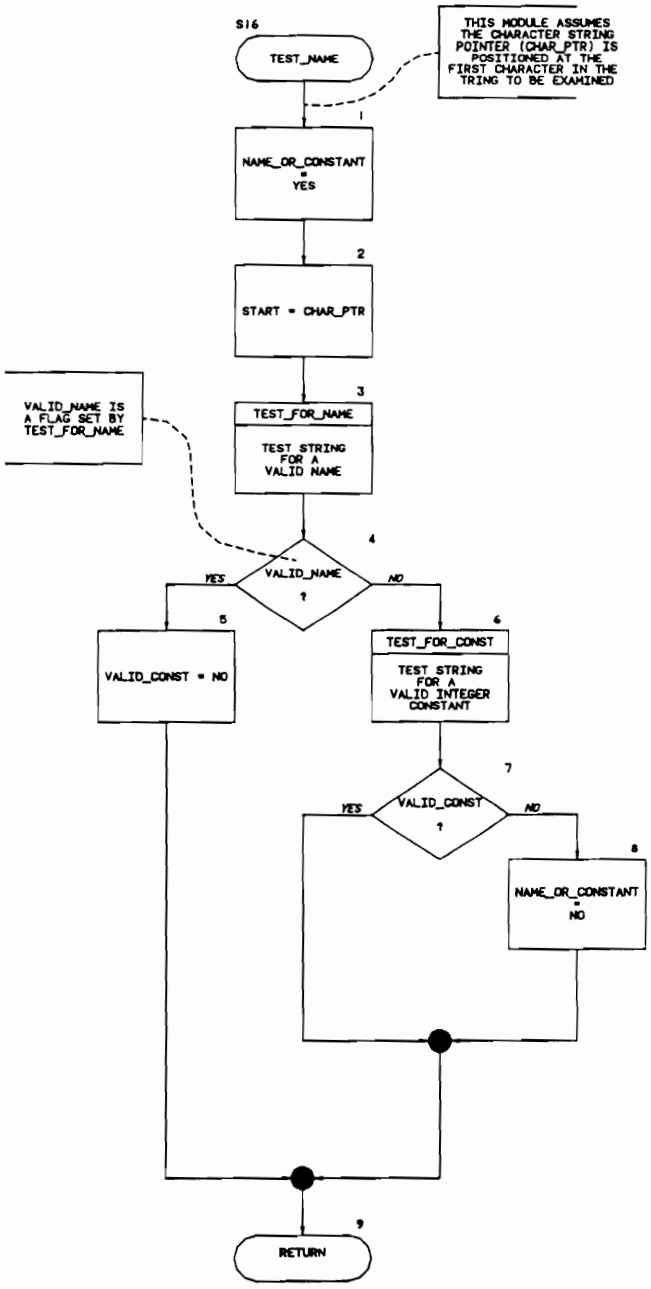


Figure 100. Module S16 - TEST_NAME

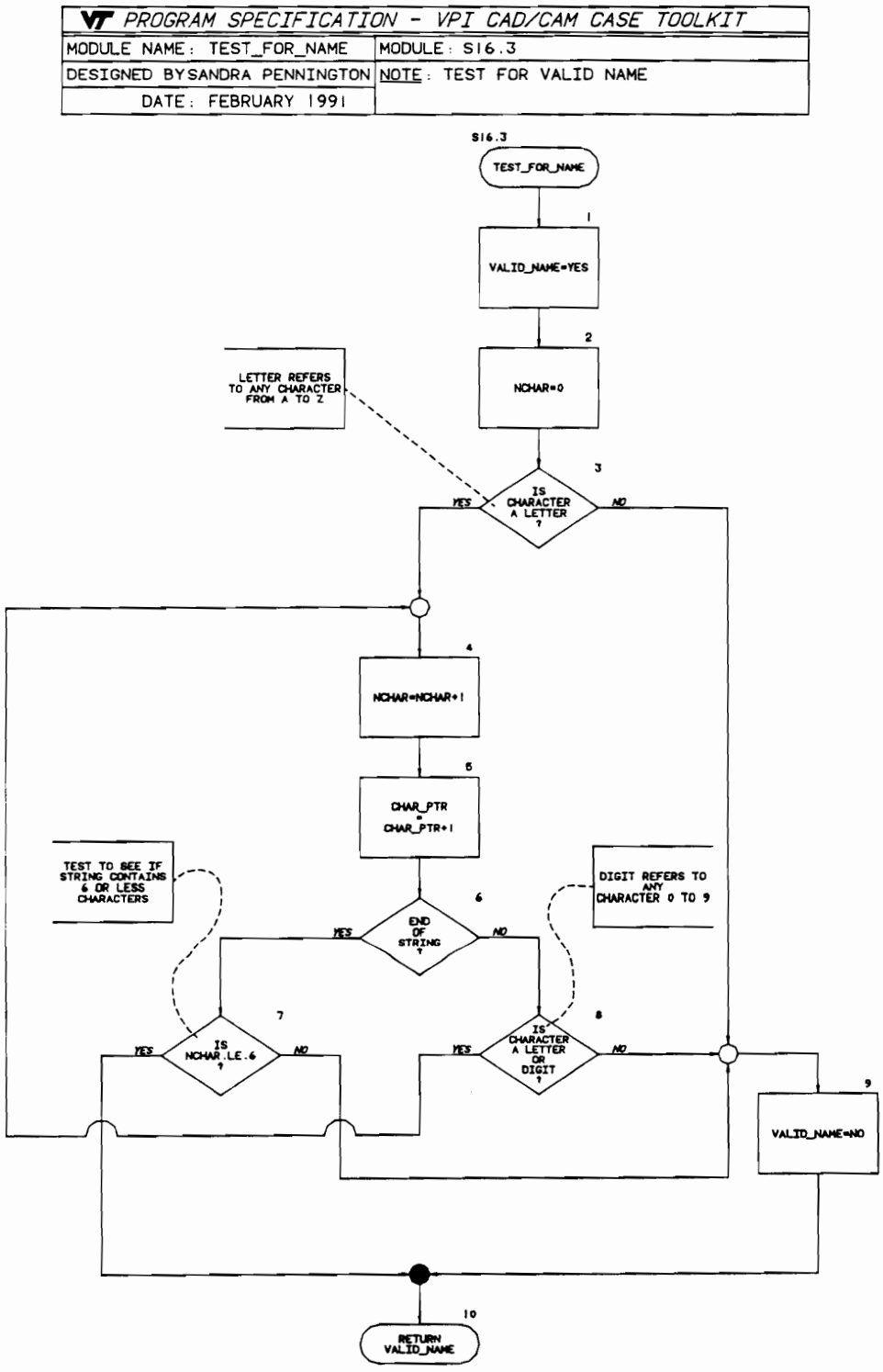


Figure 101. Module S16.3 - TEST_FOR_NAME

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TEST_FOR_CONST	MODULE: S16.6
DESIGNED BY: SANDRA PENNINGTON	NOTE: TEST STRING FOR VALID INTEGER CONSTANT
DATE: FEBRUARY 1991	

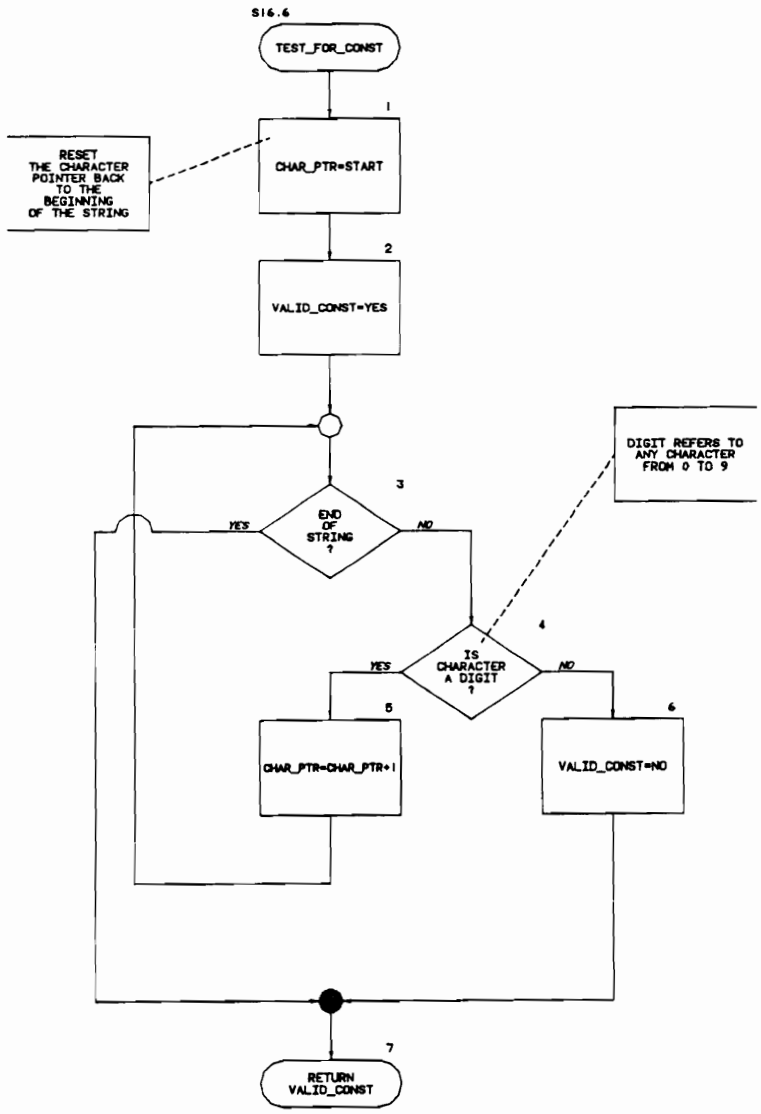


Figure 102. Module S16.6 - TEST_FOR_CONST

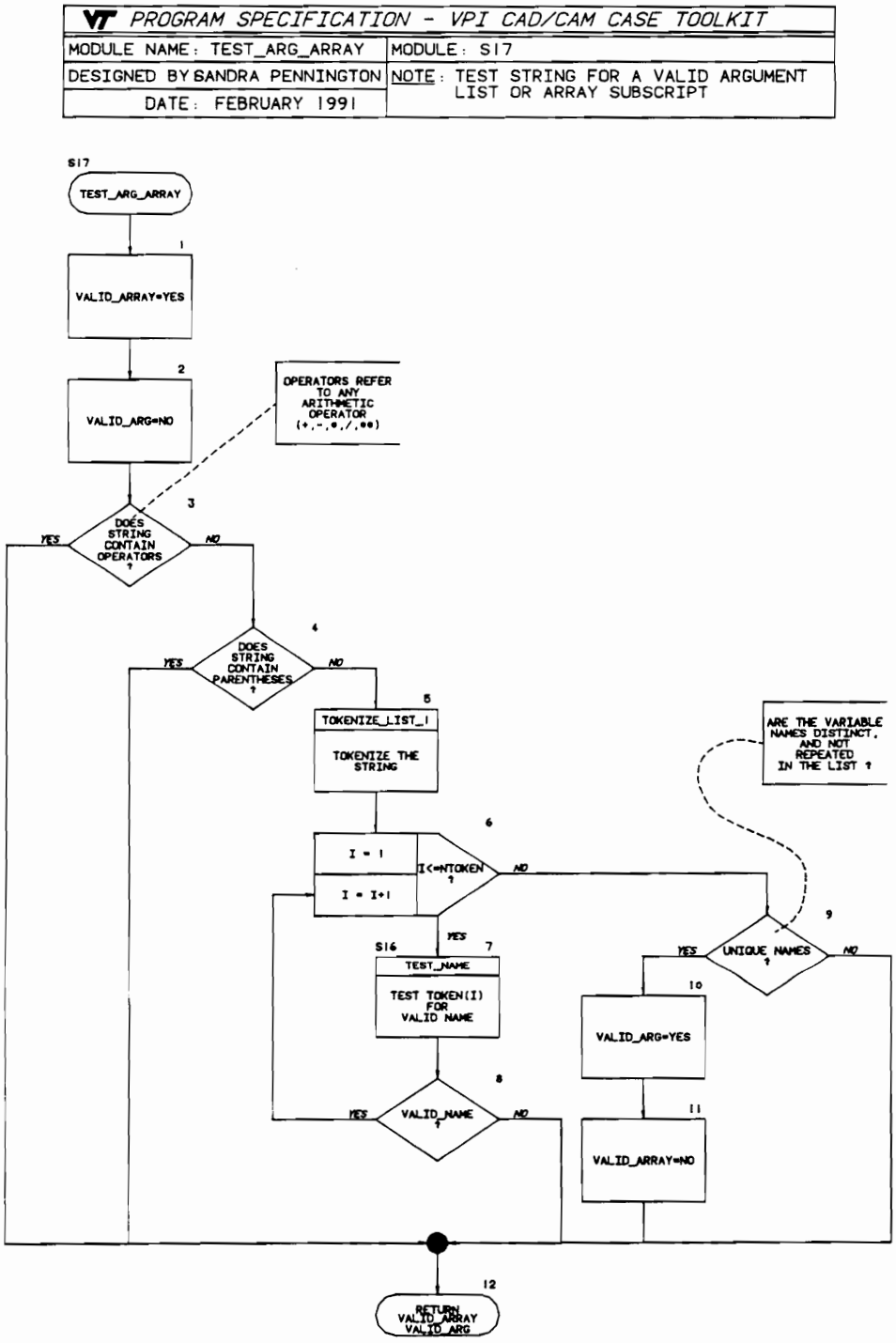


Figure 103. Module S17 - TEST_ARG_ARRAY

PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: TOKENIZE_EXPR	MODULE: S18
DESIGNED BY: SANDRA PENNINGTON	NOTE: TOKENIZE EXPRESSION
DATE: FEBRUARY 1991	

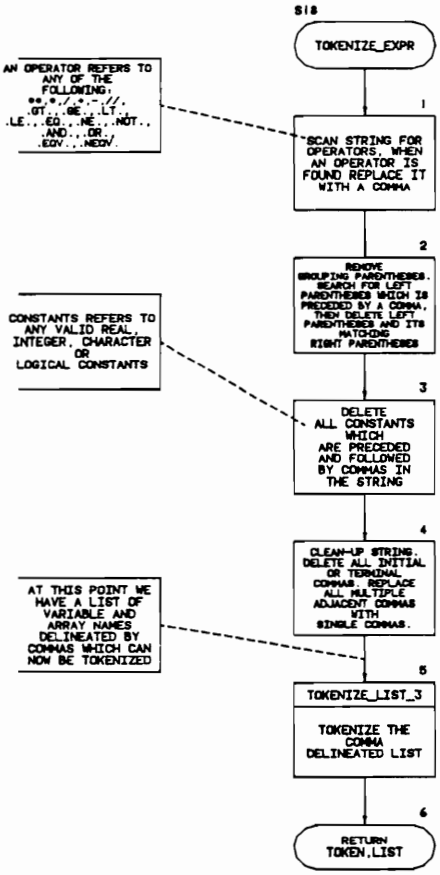


Figure 104. Module S18 - TOKENIZE_EXPR

VT PROGRAM SPECIFICATION - VPI CAD/CAM CASE TOOLKIT	
MODULE NAME: PROC_SUB_ARG	MODULE: S19
DESIGNED BY: SANDRA PENNINGTON	NOTE: PROCESS SUBROUTINE ARGUMENTS
DATE: FEBRUARY 1991	

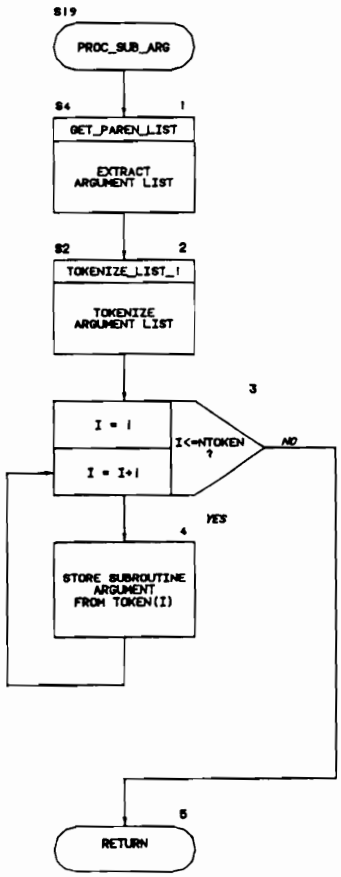


Figure 105. Module S19 - PROC_SUB_ARG

Appendix C. GLOSSARY

Beta Testing: Initial software testing performed by a small group of end-users

BOM: Bill of Materials

CAD: Computer Aided Design

CADAM: CAD/CAM software system for drafting, analysis and NC part programming, also supports 3-D wireframe model creation

CADAM ISD: Optional module of the CADAM software system which provides interactive solid model design capabilities

CADAM ACCESS: Optional module of the CADAM software system which allows users to access the CADAM database via custom-written application programs

CAD/CAM: Computer Aided Design / Computer Aided Manufacturing

CAM: Computer Aided Manufacturing

CASE: Computer Aided Software Engineering

CATIA: CAD/CAM software system for drafting, analysis and NC part programming. Also supports 3-D wireframe model creation and solid models

CATIA IUA: Same as CADAM ACCESS for the CATIA software system

CMM: Coordinate Measuring Machine

CIM: Computer Integrated Manufacturing

Database: A collection of persistent data used by application programs

Data Dictionary: A set of definitions of all data appearing on a data flow diagram in data stores

DBMS: Database Management System, software that handles all access to a database

EC Markup: Engineering Change Markup

HVAC: Heating, Ventilating and Air Conditioning

Framework: see IPSE

IGES: Initial Graphics Exchange Specification, a standard format specification for transferring data between dissimilar CAD/CAM systems

IPSE: Integrated Project Support Environment - provides an infrastructure for the integration of CASE toolkits within a CASE workbench

NC: Numerical Control

PDES: Product Data Exchange Specification (successor to IGES)

PHIGS: Programmer's Hierarchical Interactive Graphics System, the 3-D international standard for graphics

SDLC: Software Development Life Cycle, see section 2.3.2

Tool (CASE): A software tool that automates a particular task of the SDLC

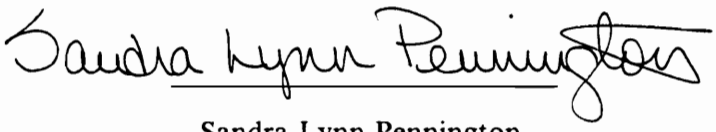
Toolkit (CASE): A set of integrated CASE tools that have been designed to work together and to automate a phase of the SDLC

Workbench (CASE): A set of integrated CASE tools that have been designed to work together and to automate the entire SDLC

Vita

The author, Sandra Lynn Pennington, was born in Jacksonville, Florida on November 21, 1957. She graduated from Edward H. White High School in Jacksonville, Florida in June 1975. In September of that same year she entered Florida State University, located in Tallahassee Florida, and graduated in June of 1979 with a Bachelor's Degree in Fine Arts. To accomodate her changing interests and goals, she began work on a Bachelor of Science degree in mechanical engineering at Florida Atlantic University in Boca Raton, Florida in the spring of 1980. Following graduation in May 1984, she entered Virginia Polytechnic Institute and State University in Blacksburg, Virginia where she completed her Master of Science degree in mechanical engineering in 1986, with emphasis in Computer Aided Design. Upon completion of the Master's degree, the author continued her studies in mechanical engineering and Computer Aided Design in the doctorate program of study at VPI & SU. She taught as an instructor within the department for four terms and has most recently worked as a research associate on a large project sponsored by the IBM Corporation. After graduation she will join the Engineering

Graphics Planning & Strategy organization of the IBM Corporation in Kingston, New York.



Sandra Lynn Pennington