INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106
Shen, Yan

ADVICE: AN EXPERT SYSTEM TO HELP EVALUATE GRADUATE STUDY PLANS OF SYSTEMS & INDUSTRIAL ENGINEERING STUDENTS

The University of Arizona

M.S. 1986

University Microfilms International 300 N. Zeeb Road, Ann Arbor, Mi 48106
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark \(\checkmark\).

1. Glossy photographs or pages 
2. Colored illustrations, paper or print 
3. Photographs with dark background 
4. Illustrations are poor copy 
5. Pages with black marks, not original copy 
6. Print shows through as there is text on both sides of page 
7. Indistinct, broken or small print on several pages 
8. Print exceeds margin requirements 
9. Tightly bound copy with print lost in spine 
10. Computer printout pages with indistinct print 
11. Page(s) 28 lacking when material received, and not available from school or author.

12. Page(s) seem to be missing in numbering only as text follows.
13. Two pages numbered . Text follows.
14. Curling and wrinkled pages 
15. Dissertation contains pages with print at a slant, filmed as received 
16. Other

__________________________________________________________________________
__________________________________________________________________________

University Microfilms International
ADVICE: AN EXPERT SYSTEM TO
HELP EVALUATE GRADUATE STUDY PLANS OF
SYSTEMS & INDUSTRIAL ENGINEERING STUDENTS

by

Yan Shen

A Thesis Submitted to the Faculty of the
DEPARTMENT OF SYSTEMS AND INDUSTRIAL ENGINEERING
In Partial Fulfillment of the Requirements
For the Degree of
MASTER OF SCIENCE
WITH A MAJOR IN SYSTEMS ENGINEERING
In the Graduate College
THE UNIVERSITY OF ARIZONA

1986
STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: 

APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

A. TERRY BAHILL

A. Terry Bahill

December 4, 1986

Date

Professor of Systems and Industrial Engineering
To my parents
ACKNOWLEDGEMENTS

I am deeply indebted to Dr. Terry Bahill for giving me the chance to work for him. Thanks for the challenging projects, and the guidance and advice necessary for completing them. Also, thanks for the time spent in discussing and reading this thesis.

I am also grateful to Dr. Wayne Wymore and Dr. Sidney Yakowitz for their time spent in reviewing the thesis and participating on my graduate committee.

Special thanks is extended to William Ganoe, Tom Hassett, and Mousa Mitwasi for their willingness to help when I needed it. Thanks to the many students who helped evaluate system performance and provided valuable suggestions.

Finally, I want to express my appreciation to Robert Drabek for his encouragement and support during the time of my graduate study and thesis completion.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ vii

LIST OF ILLUSTRATIONS .................................................................................. viii

ABSTRACT ................................................................................................................ ix

1. INTRODUCTION ................................................................................................. 1
   1.1 Background .................................................................................................... 1
   1.2 Problem Statement ....................................................................................... 2
      1.2.1 Consideration of Expert Systems ....................................................... 2
      1.2.2 Expert System Shell Selection ............................................................ 3
      1.2.3 Problem Domain .................................................................................. 5
   1.3 Scope ............................................................................................................. 6

2. SYSTEM DEVELOPMENT .................................................................................... 9
   2.1 System Requirement ..................................................................................... 9
      2.1.1 Technology Requirement .................................................................. 9
      2.1.2 Special Features Requirement ............................................................ 10
      2.1.3 Performance Requirement ................................................................ 10
   2.2 Concept Development ................................................................................. 11
      2.2.1 Current Advising Procedures .............................................................. 11
      2.2.2 The Overall Architecture .................................................................. 13

3. TECHNICAL CHARACTERISTICS ................................................................... 16
   3.1 The Inference Engine ................................................................................... 16
   3.2 The Knowledge Base .................................................................................... 17
   3.3 The Database File ......................................................................................... 19
   3.4 External Functions ....................................................................................... 19
      3.4.1 M.1 External Function Interface ......................................................... 20
      3.4.2 C-to-DBASE Interface ....................................................................... 22
      3.4.3 User-Written Functions ..................................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.4 Preparing External Functions</td>
<td>25</td>
</tr>
<tr>
<td>3.5 Hardware Description of ADVICE</td>
<td>29</td>
</tr>
<tr>
<td>4. ARCHITECTURE - PROCESS DESCRIPTIONS</td>
<td>31</td>
</tr>
<tr>
<td>4.1 Startup</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Main Options</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Input/Output Process</td>
<td>34</td>
</tr>
<tr>
<td>4.4 Shutdown</td>
<td>37</td>
</tr>
<tr>
<td>5. SYSTEM EVALUATION</td>
<td>38</td>
</tr>
<tr>
<td>5.1 Verification</td>
<td>38</td>
</tr>
<tr>
<td>5.1.1 Knowledge Base Debugging</td>
<td>38</td>
</tr>
<tr>
<td>5.1.2 Program Proving</td>
<td>40</td>
</tr>
<tr>
<td>5.2 Validation</td>
<td>40</td>
</tr>
<tr>
<td>6. CONCLUSIONS</td>
<td>47</td>
</tr>
<tr>
<td>APPENDIX A: MAKING EXECUTABLE FILES</td>
<td>50</td>
</tr>
<tr>
<td>APPENDIX B: MAKING DBASE II FILES</td>
<td>51</td>
</tr>
<tr>
<td>APPENDIX C: INSTRUCTIONS FOR USERS OF ADVICE</td>
<td>52</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>54</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1-1: Information about COGITO and DCD ........................................ 4
Table 1-2: Hardware/Software requirement of ADVICE ............................... 7
Table 2-1: Designated process options of ADVICE .................................... 15
Table 3-1: ADVICE knowledge base representations .................................. 18
Table 3-2: M.1 external interface files used by ADVICE ............................ 20
Table 3-3: M.1 external interface functions used by ADVICE .................... 21
Table 3-4: C-to-dBASE functions used by ADVICE .................................. 22
Table 3-5: Database file structure description for ADVICE ......................... 23
Table 3-6: Specifications of user-written external functions ....................... 24
Table 4-1: Expressions for collecting personal data in ADVICE .................. 32
Table 4-2: Acceptable legal value types in M.1 ........................................ 35
Table 4-3: Database file structure of ADVICE .......................................... 36
Table 5-1: Comparison of system requirements and performance ................. 42
Table 5-2: Human advisor checked ADVICE's result ................................ 45
Table 5-3: ADVICE reviewed approvals of human advisors ....................... 46
LIST OF ILLUSTRATIONS

Figure 2-1: Major steps in present advising routine .................................. 12
Figure 2-2: The overall architecture of ADVICE ........................................ 14
Figure 3-1: Overview of preparing external functions ............................... 28
Figure 3-2: The IBM PC to DECTalk cable ............................................... 30
Figure 4-1: Output interconnections with conclusions .............................. 36
Figure 6-1: An overview of the ADVICE expert system .......................... 49
ABSTRACT

An expert system, ADVICE, has been built to help evaluate the study plans of Systems & Industrial Engineering graduate students. This thesis introduces the system requirement, the program architecture, the technical characteristics, and the system evaluation of this expert system. ADVICE consists of four components: a knowledge base, an inference engine, a database and an external interface file. A novel feature of ADVICE is its interface to a DECTalk voice synthesizer that talks to the user. The knowledge base was built with production rules, facts, and external functions. The external functions are linked with the C-to-dBASE Interface to receive information from the dBASE II data files where the attributes of each course are stored. ADVICE’s inference engine was M.1, an expert system shell which runs on IBM-compatible personal computers.

Verification and validation was performed to ensure the accuracy and reliability of the system. ADVICE saved time for both the students and the professors, and provided more accurate and more comprehensive advice for the students.
1. INTRODUCTION

The objective of this thesis is to define the system development, technical characteristics, architecture, and evaluation of ADVICE, an expert system that helps evaluate SIE graduate-student plans of study.

1.1 Background

The Department of Systems & Industrial Engineering offers the Masters of Science degree in Systems, Industrial, and Reliability Engineering with five available options. The degree program, however, does not specify precisely the courses a student must take, but it requires a particular mix of areas under department Degree Requirements. After being admitted to the M.S. program each student must formulate a Graduate Study Plan. The Graduate Study Plan is made with the help of an advisor, and is reviewed by the Graduate Committee of the department and recommendations are made for strengthening it if deemed necessary [1].

Previously, departmental study plans were based on four major resources: department degree requirements, course restrictions, professors' suggestions, and student interests. Professors gave suggestions and advice in order for a study plan to satisfy requirements in the major and option. Plans must meet all outside restrictions, be of interest to the student, and be challenging but not overwhelming. The advising responsibilities have become heavier for the professors since the enrollment has nearly doubled during the last two years, and students have a tendency to depend on a professor's opinion and to ignore what could be done independently based on the other three
resources. In addition, the traditional method has become obsolete as knowledge-oriented problems can be solved by today's computer technology.

We need an intelligent computer system to assist advising and to help evaluate student Graduate Study Plans. Students could express their study interests and abilities by interacting with the computer system. This special type of computer system is called an "expert system" or a "knowledge-based system".

1.2 Problem Statement

1.2.1 Consideration of Expert Systems

Today's most successful knowledge systems are capable of solving problems in which the knowledge needed to solve that problem is already understood, and that problem could be solved by an expert in a reasonable amount of time [2]. The reasons for choosing an expert system instead of conventional computer program are explained in the following statement by F. Hayes-Roth, D.A. Waterman, and D.B. Lenat [3]:

First, most of the difficult and interesting problems do not have tractable algorithmic solutions since many important tasks originate in complex social or physical contexts, which generally resist precise description and rigorous analysis.

The second reason for emphasizing knowledge rather than formal reasoning methods is pragmatic: human experts achieve outstanding performance because they are knowledgeable. If computer programs embody and use this knowledge, then, they too should attain high levels of performance.

The third reason for focusing on knowledge recognizes its intrinsic value. Knowledge is a scarce resource whose refinement and reproduction creates wealth.

Professors were considered "experts" in knowing what study plans best suited the students. An expert system was deemed reasonable to solve the
advising problem. Some primary expectations about the expert system were: it would provide an environment in which the students would be able to discuss their study plans with the intelligent computer advisor; it would check the plans and give recommendations; it would provide a plan for final approval by a human advisor. Finally, I decided to gather the knowledge from (1) Dr. A. Terry Bahill, the graduate committee chairman who was also the director of this project; (2) the Department Handbook (1985-1986), and (3) the University General Catalog (1985-1987). My next step was to choose an appropriate expert system shell from the available shells.

1.2.2 Expert System Shell Selection

The available shells in the department were M.l of Teknowledge and OPS5 of Carnegie-Mellon University (Forgy and McDermott). My job was not to determine which shell was better, but to determine which shell would be more suitable for our problem.

To learn more about these two shells, I studied two expert systems that were recently built in the department. One system was written in M.l [4] and the other was written in OPS5 [5]. Table 1-1 describes these systems.

COGITO was built with M.1, version 1.0 in 1985. Since the memory of version 1.0 was restricted, COGITO's knowledge base was written in six parts. The extra time spent loading each part and saving cache constantly slowed down the speed of processing. DCD was built with OPS5, version 5.0, but its information gathering program was written in C.

OPS5 is a production-system programming language [6]. It is a symbolic language with a highly-restricted format and is hard to learn. In contrast,
Table 1-1: Information about COGITO and DCD.

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COGITO</td>
<td>It helps to install the UNIX operating system on a VAX computer. It uses M.1 as its inference engine, and runs on IBM compatible personal computers. Its users are computer systems managers.</td>
</tr>
<tr>
<td>DCD</td>
<td>It directs trouble-shooting in activities of connecting a peripheral device to a computer or local area network. It uses OPS5 as its inference engine, and runs on a VAX computer. Its users are systems engineers.</td>
</tr>
</tbody>
</table>

M.1's rules are written in English-like phrases, and the inference engine is primarily designed for the non-specialist computer programmer. Thus in M.1 the user does not need to learn the LISP-like code of OPS5 with special syntax. The notation of OPS5 did not bother the author of DCD since he was an experienced programmer. But why was part of DCD written in C? The author explained as follows, since "all the necessary data can be collected before we even start with the inference process, we were able to use an independent module written in C [5]". In addition, the C program helped overcome the user interface problems of OPS5. The author of DCD then concluded the following, "The problem at hand could have been solved by M.1 as well, and M.1 is much easier to learn than OPS5..., and the human/computer interface of M.1 is better than that of OPS5".

It seemed that M.1 was a better choice for ADVICE than OPS5 because the IBM PC was more available than the VAX, the M.1 language was easier to learn, and M.1's inference engine was more friendly. Furthermore,
M.1 could be connected to the database management system dBASE II where a large number of facts could be stored.

I decided to use M.1 and built the first model with version 1.0 of M.1 at the end of 1985. Because of the memory limitation, this early version of ADVICE had to be broken into ten parts. And because the C86 C compiler was not available in the department at that time, the C-to-dBASE Interface was not used and all the course attributes were stored in the knowledge base. It took great effort to keep the subroutines working together.

In May 1986, Teknowledge released a new version of M.1, written in C, providing fewer restrictions on rules and greater dynamic memory capacity. This new version had an improved facility to interact with external C programs and the dBASE II software package. We then purchased the C86 C compiler, redesigned the knowledge base, and stored course attributes in database files. The data saved in the database would be sent to the knowledge base through external function calls. Being a knowledge-representation language, M.1 allowed rules and facts to be written with symbolic variables that could be replaced with values during a consultation [2]. This powerful feature allowed ADVICE to keep calculations simple and fast. More importantly, this method reduced the dynamic memory usage.

1.2.3 Problem Domain

ADVICE was hence designed and created to help evaluate SIE graduate student plans of study. It interacted with students. It would prompt a student for personal information and a study plan. For the study plan, it would validate degree requirements, all course prerequisites, and assure that
the courses were to be offered during an appropriate semester. It would encourage students to be more independent in making their own decisions. It would provide an opportunity for students to learn more about the M.S. program while using the expert system. It would help students to make flexible plans of study. Consequently, it would free up professors’ time and provide more accurate advise.

1.3 Scope

In order to connect ADVICE to M.1, dBASE files, and the DECtalk voice synthesizer, I had to create an ADVICE external function file containing some of the M.1 external functions, C-to-dBASE Interface functions, user-written functions, and various C libraries functions. An executable file M1AUX.EXE was made by compiling and linking the ADVICE external function file. The procedures to make an executable file are listed in Appendix A. Table 1-2 below shows the specified hardware and software requirements for making an executable file.

ADVICE was designed for the Department of Systems and Industrial Engineering at the University of Arizona. Therefore, the rules were based on the SIE department requirements only. In the future, ADVICE could be modified to suit requirement of other departments, or more rules could be added for other departments if M.1’s memory capability were increased to an appropriate level. The current version of M.1 allows up to 1000 rules [2].

In the Summer of 1986, the Digital Equipment Corporation donated a voice synthesizer, DECtalk, to the Speech and Hearing Science Department at the University of Arizona, for use in the SIE Computer Systems Laboratory. I
Table 1-2: Hardware/Software requirement of ADVICE.

<table>
<thead>
<tr>
<th>Type</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine</td>
<td>IBM compatible personal computer</td>
</tr>
<tr>
<td>DOS</td>
<td>Version 2.0 or above</td>
</tr>
<tr>
<td>Memory</td>
<td>512K bytes</td>
</tr>
<tr>
<td>Linker</td>
<td>IBM PC Version 2.3 or higher</td>
</tr>
<tr>
<td>Compiler</td>
<td>Computer Innovation C86 C</td>
</tr>
<tr>
<td>Database</td>
<td>Ashton-Tate dBASE II</td>
</tr>
</tbody>
</table>

connected it to the IBM PC and wrote external code to interface it with ADVICE. DECtalk has brought a unique feature to the expert system. As experiments showed, the students appreciated the friendliness of the machine and were more than happy to follow the talker’s instructions. Further research should be conducted in (1) relations between screen directions and voice instructions and (2) the main effects of using voice during advise processing.

ADVICE assumed that students had some knowledge about the department, faculty, and courses they were interested in. At the same time, ADVICE provided ample instructions to direct students on how to make study plans. Students who did not have clear ideas about their plans often found ADVICE to be helpful in making up their minds. In addition, ADVICE pro-
vided automatic saving, checking, and printing facilities, and the students could freely revise their plans by following ADVICE's on-line instructions.
2. SYSTEM DEVELOPMENT

This section describes the system requirements, system development, and overall architecture of the expert system.

2.1 System Requirement

2.1.1 Technology Requirement

ADVICE used M.l as its inference engine. In order for the ADVICE expert system to run, the hardware and software requirement described earlier in Table 1-2 must be satisfied. For example, our AT&T 6300 PC was only capable of running this system after the installation additional memory.

M.l supports external functions for Computer Innovation C86 C, MicroSoft C and assembly language. But ADVICE only used the C86 C compiler because the same compiler was also required by the C-to-dBASE INTERFACE. Related external functions and C functions will be introduced in later sections.

M.l was implemented in the C programming language [2]. It was designed for engineers without prior experience in expert system technology. However, this does not imply that no computer knowledge was needed. Since ADVICE required the use of external functions and a data-base system, familiarity with the following was necessary: the IBM PC, MS-DOS, C compilers, dBASE II, interfacing of these components, and the C programming language.

A good editor was also needed. The PC-Write editor version 2.5 supported by Quicksoft, for example, could not be used on a file larger than 60K
bytes. The editor used for ADVICE on the IBM PC was MicroEMACS version 3.6 written by David Conway. When many changes were involved, I would transfer the program to the UNIX/VAX system to use the "vi" editor. I personally preferred using the "vi" editor because it was faster, and its "spell" program could help detect typing errors.

2.1.2 Special Features Requirement

The following features of the ADVICE system were specified by the faculty members during the system development:

1. Deliver instructions in each stage of consultation.
2. Provide revising facilities.
3. Provide help facilities.
4. Provide necessary default answers for "unknown" meta-facts.
5. Save and print personal data and plans.
6. Show logical reasoning.
7. Use dBASE II to store course information.
8. Use DECtalk voice synthesizer to mention names and comments.
9. Validate students' plans and print result of analysis.
10. Periodically write information on disk to prevent data loss in event of power failure.

The ADVICE system should satisfy these requirements in order for it to be approved as an acceptable expert system for the SIE department.
2.1.3 Performance Requirement

Extending the specific features described in the previous section, ADVICE should be an intelligently designed, logically oriented system. Efficient rules and notices should prevent any unexpected termination during consultation. The system should be user-friendly. It should provide clear instructions to students. It should provide accurate analysis of results to professors. And it should provide reliable advice to benefit both students and advisors.

2.2 Concept Development

In computer science, programmers use "algorithms" to describe a set of steps to be taken for a problem to be solved. In systems design, systems engineers and knowledge engineers use "concept development" to describe a procedure being used to solve a problem. This procedure includes understanding the problem, constructing an abstract algorithm, and developing a model from which a real system can be built.

2.2.1 Current Advising Procedures

The present advising routine involves seven major steps as shown in Figure 2-1. To implement this algorithm, I first found that the students would come to see an advisor under three conditions:

(1) Need advise for study directions, such as major, option, etc.
(2) Need consultation for course selections.
(3) Need discussion with advisor for approval of study plans.
Figure 2-1: Major steps in present advising routine.
Secondly I found that these students could be divided into three groups regarding to their familiarity with the M.S. program:

(1) Totally unfamiliar. Unable to make any decision.
(2) Limited familiarity. Able to decide some courses.
(3) Familiar. Able to fill out the study plan.

The purpose of the ADVICE expert system here was to lead the students from a lower level to a higher level. ADVICE would encourage students of groups one and two to read the handbook, study the catalog, and join the third group.

2.2.2 The Overall Architecture

The ADVICE system had three principal levels: startup, options, and quit. The heart of this architecture was the options section, which showed a main menu of what was available on the system. Figure 2-2 shows the overall architecture of the system.

Rules were grouped for each particular level. Multiple interaction with existing conclusions was permitted. Each component in the architecture is represented by a if-then rule that is composed of facts, meta facts, or external meta-propositions. Table 2-1 lists the designated processes for each option.
Figure 2-2: The overall architecture of ADVICE.
Table 2-1: Designated process of options of ADVICE.

<table>
<thead>
<tr>
<th>Option</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>collect personal data, accept study plan, check prerequisites and offering time, identify deficiencies, update total units and areas</td>
</tr>
<tr>
<td><strong>Load</strong></td>
<td>load previous plan for revising</td>
</tr>
<tr>
<td><strong>Review</strong></td>
<td>display current plan</td>
</tr>
<tr>
<td><strong>Check</strong></td>
<td>check degree requirement, plan validity, save and print</td>
</tr>
<tr>
<td><strong>AddC</strong></td>
<td>accept a new course to an existing semester, check prerequisite and time for this course</td>
</tr>
<tr>
<td><strong>AddS</strong></td>
<td>accept and save a new semester in the plan, check prerequisite, offering time, units and area</td>
</tr>
<tr>
<td><strong>DeleteC</strong></td>
<td>delete a course from the plan, recalculate units and area</td>
</tr>
<tr>
<td><strong>DeleteS</strong></td>
<td>delete a semester from the plan, recalculate units and area</td>
</tr>
<tr>
<td><strong>HelpM</strong></td>
<td>show help commands for M.1</td>
</tr>
<tr>
<td><strong>HelpA</strong></td>
<td>show help commands for ADVICE</td>
</tr>
<tr>
<td><strong>Quit</strong></td>
<td>call DECTalk, close database file and exit ADVICE</td>
</tr>
</tbody>
</table>
3. TECHNICAL CHARACTERISTICS

ADVICE has four components: an inference engine, a knowledge base, a database and an external interface file. Each component will be discussed explicitly in the following sections.

3.1 The Inference Engine

ADVICE used the M.1 inference engine with a backward-chaining reasoning process to reach conclusions. At the heart of this inference engine is the mechanism for finding values for expressions. Input data and facts must be found to match the values on the right side of an expression in order for a conclusion to be true. The following example illustrates the process of backward chaining:

rule-3:

/* 1 */ if prerequisite = yes
/* 2 */ and time = yes
/* 3 */ then course = checked.

This rule would be invoked when the value of "course" needed to be determined. If either of line 1 or line 2 failed, this rule would fail. If the rule failed, M.1 would scan the knowledge base seeking another rule with the expression "course." If M.1 found no more rules for this expression, a conclusion would be made and saved in cache as "course was sought, but no value was concluded." This conclusion would then terminate the session and the inference engine
would not be able to continue searching for the expression "course."

The naming convention for the system expression and variables followed the general English meaning presented in the knowledge base. Therefore, the rules in ADVICE were readable by non-programmers.

### 3.2 The Knowledge Base

The knowledge base was derived from three sources of knowledge: professors’ suggestions, department degree requirements, and university regulations. The knowledge was interpreted into rules, facts, meta-facts, and meta-propositions. Table 3-1 shows the definitions and examples.

Two major programming tools were used in constructing the knowledge base: symbolic variables and recursive rules. Symbolic variables were used to allow automatic substitutions with values during a consultation and to permit similar rules to be collapsed into more general expressions. ADVICE also used symbolic variables to perform simple calculations. One must notice that variables in M.I are not the same as variables in conventional programming languages [2]. In a conventional language, variables are labels or names for memory locations that hold data during program execution. For example, an assignment statement "a = b" retrieves the number residing in the location named "b", and stores the result in another location named "a". In M.I, variables are not used as labels, as named memory locations. In contrast, M.I would treat "a" as an expression (not a variable), and assign a value "b" to it. The statement "a = b" in M.I means: "b" is the value of the expression "a", and "b" itself may not have another value. Similarly, a statement "blue = pink" means: "pink" is the value of the expression "blue". If "b" has a value
Table 3-1: ADVICE knowledge base representations.

<table>
<thead>
<tr>
<th>ADVICE Knowledge Base Representation</th>
<th>Type</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rules</td>
<td>relationships between incoming information, intermediate concepts and the final advice</td>
<td>if PREMISE, then CONCLUSION</td>
</tr>
<tr>
<td></td>
<td>facts</td>
<td>assertion, proposition, static, non-judgmental knowledge</td>
<td>EXPRESSION=VALUE</td>
</tr>
<tr>
<td></td>
<td>meta-facts</td>
<td>facts about the knowledge base or the consultation</td>
<td>goal, initialdata, explanation, configuration, questions, legalvals</td>
</tr>
<tr>
<td></td>
<td>meta-propositions</td>
<td>propositions about the state of consultation, knowledge base</td>
<td>external calls, do, cached, display, EXPRESSION is STATUS</td>
</tr>
</tbody>
</table>

itself and you want to assign b's value to "a", then the rule should be: if b = X then a = X.

Recursive rules were used for repeated checking of the same task. The combination of variables and recursive rules provided significant economies in the construction of the knowledge base. The following example demonstrates such usage:

rule-1:

/* 1 */ if number semester = N
and semester(N) = checked
then all semester = checked.

rule-2:

if N = 1
or (N-1 = X and semester(X) = checked)
and number courses = Y
and semester(N)-course(Y) = checked
then semester(N) = checked.

In this example, rule-2 was called N times. N was assigned a new value at the end of every iteration. The first time it was called, M.1 would remember that "semester(N) is being considered." If N is not equal to 1, then "semester(N-1) is being considered." This process would continue until N was equal to 1 and line 6 and line 7 were executed. These two lines then would be repeatedly executed 2 * N times until every semester was checked.

3.3 The Database File

The database file was called attrib.dbf. It contained 140 records with 7 fields per record. Table 3-2 lists the database file structure and the field description (with c = character and n = numeric for type convention). Course information based on the General Catalog of the University of Arizona, 1985-1987 was stored in each record.
Table 3-2: Database file description for ADVICE.

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Type</th>
<th>Width (Char Length)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>course</td>
<td>c</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>time</td>
<td>c</td>
<td>006</td>
</tr>
<tr>
<td>3</td>
<td>pre1</td>
<td>c</td>
<td>010</td>
</tr>
<tr>
<td>4</td>
<td>pre2</td>
<td>c</td>
<td>010</td>
</tr>
<tr>
<td>5</td>
<td>units</td>
<td>c</td>
<td>001</td>
</tr>
<tr>
<td>6</td>
<td>siegrad</td>
<td>n</td>
<td>001</td>
</tr>
<tr>
<td>7</td>
<td>grad</td>
<td>n</td>
<td>001</td>
</tr>
<tr>
<td>8</td>
<td>area</td>
<td>c</td>
<td>002</td>
</tr>
</tbody>
</table>

3.4 External Functions

I constructed an external interface file to connect ADVICE to M.1, dBASE files, and the DECtalk voice synthesizer. Functions in this external interface file came from the following sources: M.1 External Interface functions, the C-to-DBASE Interface functions, and user-written functions. An
executable file M1AUX.EXE was made from this external interface file. This file was automatically executed when M.1 loaded the ADVICE system. I will now discuss the three types of functions and their connections.

3.4.1 M.1 External Function Interface

The M.1 External Function Interface is supported by TEKNOWLEDGE. It was written for MicroSoft C, Computer Innovation C86 C, and assembly language. ADVICE used the interface for Computer Innovation C86 C since it was the only one that could interact with the C-to-dBASE Interface. Table 3-3 lists the files from the M.1 external function interface, and Table 3-4 lists the functions from the M.1 interface library that were used by ADVICE.

One restriction on using the M.1 interface was that the functions extInit() and extExit() must be included in the ADVICE interface file. The M.1 external function interface provided great flexibility allowing the programmer to write additional functions other than those in the M.1 external function interface. It was necessary for me to add four user-written functions to the interface file of ADVICE.

3.4.2 C-to-dBASE Interface

The C-to-dBASE Interface was designed to assist operations on dBASE II data and index files [7]. It is a library of more than 100 C functions. Any of these functions could be added to the ADVICE external interface file. Table 3-5 lists the C-to-dBASE functions used by ADVICE.
Table 3-3: M.1 external interface files used by ADVICE.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extif.h</td>
<td>C code include file, symbol definitions</td>
</tr>
<tr>
<td>extlibs.lib</td>
<td>interface library, small model</td>
</tr>
<tr>
<td>ccsml.bat</td>
<td>batch file to compile, small model</td>
</tr>
<tr>
<td>linksml.bat</td>
<td>batch file to link, small model</td>
</tr>
<tr>
<td>extester.exe</td>
<td>a utility program for testing external functions</td>
</tr>
<tr>
<td>setparas.exe</td>
<td>a utility program for setting memory requirement for external functions</td>
</tr>
</tbody>
</table>

3.4.3 User-Written Functions

I wrote four functions to let ADVICE interact with the database and DECTalk. These functions were included in the ADVICE external interface file where other M.1 and C-to-dBASE external functions were selected and defined for the ADVICE system. Table 3-6 is a list of the user-written functions.

The function descriptions are given as following:

1. **Findattrib**: imports the course number to the database, finds the identified record, and returns "ok" if successful.
Table 3-4: M.1 external interface functions used by ADVICE.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFunction</td>
<td>It tells M.1 how to initialize the function subroutine</td>
</tr>
<tr>
<td>extInit</td>
<td>It is called when external code is originally loaded into memory</td>
</tr>
<tr>
<td>extExit</td>
<td>It is called when M.1 is exited normally</td>
</tr>
<tr>
<td>export</td>
<td>It returns arguments from an external function to M.1</td>
</tr>
<tr>
<td>import</td>
<td>It passes arguments from M.1 to an external function</td>
</tr>
<tr>
<td>setErrornum</td>
<td>It passes an error code from an external function to M.1 if an error occurs in the external function</td>
</tr>
</tbody>
</table>

2. Talk: calls another C program, adtk.c, that reads the input file "adinput," and sends output to the communication port where the DECTalk voice synthesizer is connected.

3. String: imports 3 string arguments, calls IBM PC function com_rst() to set port, baud rate, parity, stop bit, and word length, finally sends output to communication port where DECTalk is being connected.

4. Adtk: reads an input data file and sends the content to DECTalk.
### C-to-dBASE Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dalloc()</td>
<td>allocate memory for the header structures</td>
</tr>
<tr>
<td>dopen()</td>
<td>open the dBASE data and index files</td>
</tr>
<tr>
<td>dclose()</td>
<td>close the dBASE data and index files</td>
</tr>
<tr>
<td>dread()</td>
<td>read data record into memory</td>
</tr>
<tr>
<td>derror()</td>
<td>print stderr message</td>
</tr>
<tr>
<td>dfree()</td>
<td>release all run-time memory space associated with the master control structure</td>
</tr>
<tr>
<td>dget_rec()</td>
<td>locate to a record and call another function to read the record information into memory</td>
</tr>
<tr>
<td>dpanic()</td>
<td>print panic message and abort the program</td>
</tr>
<tr>
<td>findsk()</td>
<td>search recursively</td>
</tr>
<tr>
<td>upper()</td>
<td>convert a string to upper case</td>
</tr>
</tbody>
</table>

3.4.4 Preparing External Functions

Preparing the external functions for ADVICE involved the 6 steps shown in Figure 3-1 on page 28. Explanations for each step will be given in the
Table 3-6: Specifications of user-written external functions for ADVICE.

<table>
<thead>
<tr>
<th>User-Written External Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function Name</strong></td>
</tr>
<tr>
<td>findattrib</td>
</tr>
<tr>
<td>talk</td>
</tr>
<tr>
<td>string</td>
</tr>
<tr>
<td>adtk</td>
</tr>
</tbody>
</table>

following paragraphs.

1. Evaluate the Need: There were two reasons for using external functions for ADVICE: (1) M.1 has limited memory, but ADVICE needs a large memory to store course attributes. So I decided to use a database file for this purpose. However I now needed programs to interface M.1 to the database file. External functions could be used for this purpose since the C-to-dBASE Interface was available. (2) ADVICE needed to connect to DECtalk to satisfy one of the systems requirements. And this could only be accomplished with the use of external functions.

2. Define the Function: Select the name of the function, determine the function identifier, and define inputs, outputs, and side effects. The function identifier was the name used within the ADVICE knowledge
base to identify which function was being called. These functions will be called from ADVICE by external meta-propositions.

3. Prepare the External Meta-Proposition: The input and output within the meta-propositions had the following form:

   \[
   \text{external}(\text{FUNCTION_ID}), [\text{IN_ARGS}] = [\text{OUT_ARGS}]
   \]

   The following is an example of such form:

   if \text{external}(\text{open, [attrib, attrib]]) = [\text{ok}]
   then data_base_open.

   In this example, "open" was a function from the C-to-dBASE Interface. It was used to open a database file, and "attrib" was the name for both the database file and the index file.

4. Prepare Source Code for External Interface File: Write the appropriate C language interface code for the M.1 external functions and the C-to-dBASE interface functions. Write the user-defined functions.

5. Prepare Executable External Interface File and Test: First you must compile and link the ADVICE external interface file to create the executable file M1AUX.EXE. Next the program "setparas" must be run on M1AUX.EXE to allocate memory. A program "extester", supplied by TEKNOWLEDGE, was used to test M.1 external code. After the program was started, extester asked for the input values for that function, then it gave these values to that function and if the correct values were returned, then you knew the function worked. Revisions could be made if the returned values were not correct. Every external function in ADVICE was tested by using the test program "extester" before being installed in ADVICE's knowledge base. This
tester saved time during the system development and was a valuable tool. Procedures for preparing executable files are listed in Appendix A.

6. Link Knowledge Base with External Interface File and Test: Sample knowledge bases were built to test the meta-propositions that called the external functions. The sample knowledge bases were especially useful because by doing so, the ADVICE knowledge base itself did not have to be loaded each time during testing.
3.5 Hardware Description of ADVICE

The entire expert system was installed on an IBM personal computer XT’s hard disk where the C86 C compiler was also stored. The input device for the system was the keyboard, and the output devices were the terminal screen, a parallel printer and the DECtalk voice synthesizer. These devices are described in the following paragraphs.

Printer: One parallel printer was attached to the IBM PC XT, port LPT1. The mode for the printer was set to a maximum of 80 characters per line and six lines per inch.

COM1: The output sent from the expert system to the DECtalk voice synthesizer went through the serial communications port, COM1. Several parameters were associated with this port. These parameters defined how fast and in what form data would be transmitted [8]. The following settings were used for the IBM PC communications parameters:

- Port: COM1
- Baud rate: 9600
- Parity: none
- Databits: 7
- Stopbit: 1

These parameters could be set by typing on the keyboard or with the C86 external function "com_rst()" [9]. In ADVICE this function was defined in the "talk" external function.
DECtalk: The DECtalk DTC01, a voice synthesizer supplied by the Digital Equipment Company, was used by the ADVICE knowledge system. Generally speaking, the DECtalk voice synthesizer does not work by itself [10]. It must be connected to a terminal, a telephone, or a computer. In our case, it was connected to an IBM PC XT. Connecting the DECtalk to the PC XT consisted of simply setting the PC XT communication parameters as above and physically connecting the PC XT to the voice synthesizer’s Cable Connector using an RS232 cable. Figure 3-2 shows the pin connections for this cable. The short connections between pins 4, 5, 6, 8, and 20 on the DECtalk side were found to be unnecessary but not harmful.

![DECtalk to IBM PC cable diagram](image)

Figure 3-2: The IBM PC to DECtalk cable.
4. ARCHITECTURE - PROCESS DESCRIPTIONS

4.1 Startup

Starting up the ADVICE system requires invoking the M.1 inference engine, loading the ADVICE knowledge base, and loading the external code file M1AUX.EXE. This process takes 3 to 4 minutes. At the end of this step, brief instructions on using ADVICE would be given and the main options would be shown on the screen.

Early versions of the ADVICE did not include the main menu section. It was added after I discovered that students were wondering about the ADVICE options. The main menu helped the students understand what was available on the system, and helped the students give quicker responses with fewer mistakes.

4.2 Main Options

There are eleven (11) options in ADVICE; they can be grouped into five parts: Input, Revise, Check, Help, and Quit.

Input: This part included two options "input a new plan" and "load a plan." The first option "input a new plan" collected the student’s academic background data and the student’s graduate study plan. The second option "load a plan" loaded the cache file that a student had saved from the previous uses of ADVICE. The student’s academic background data was collected by asking questions about the eight expressions shown in Table 4-1:
Table 4-1: Expressions for personal data in ADVICE.

<table>
<thead>
<tr>
<th>Collecting Personal Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>first</td>
</tr>
<tr>
<td>last</td>
</tr>
<tr>
<td>degree</td>
</tr>
<tr>
<td>under-major</td>
</tr>
<tr>
<td>gpa</td>
</tr>
<tr>
<td>major-in-masters</td>
</tr>
<tr>
<td>option</td>
</tr>
<tr>
<td>advisor</td>
</tr>
</tbody>
</table>

The values of these items could be changed using the two steps shown below:

step 1: `>> reset EXPRESSION`

step 2: `>> set EXPRESSION = NEWVALUE`

The following is an example to change option from course to thesis:

step 1: `>> reset option`

step 2: `>> set option = thesis`
This method was useful for people who wanted to change gpa, major, option or advisor for which the "unknown" answers were available. The default values were set for these expressions as listed in the above table. An "unknown" answer is not allowed for the first four expressions. For example, if ADVICE received "unknown" as an answer for the student's name, it would prompt the user with an error message and would ask the same question again. The DECTalk voice synthesizer spoke twice during the "input plan" process. First to welcome the student, and second to suggest that the advisor who was chosen by the student would be pleased to have the student as an advisee.

Revis: This part included five options: "review plan," "add course," "add semester," "delete course," and "delete semester." A description for each option was given earlier in Table 2-1. The "review plan" option shows the current plan in cache and provides instructions on how to display the plan during consultation. The other four options let the students show what courses or semester they want to add or delete. Then ADVICE recalculates the units, area, and updates the plan.

Check: This option printed the plan, checked the validity, printed the checking analysis, and saved the cache file. The checking was based on the following SIE department requirements for Masters studies:

(1) Total graduate units.
(2) Total SIE graduate units at the 500 level or above.
(3) Requirements for specific majors.
(4) Requirements for specific options.
(5) Deficiencies for students with non-engineering backgrounds.

(6) Whether the course is offered in desired semester and whether the student has the required prerequisites.

Help: This part included two options: help for M.1, and help for ADVICE.

Quit: This option included four processes: send message to DECTalk, close database files, reset cache, and exit the system peacefully.

4.3 Input/Output Process

Input data could come from either the users or a database file. ADVICE collected data from users by asking questions. Acceptable responses were displayed on the computer terminal screen for users to choose from. Selected questions allowed "unknown" as an answer. Users could use M.1 facilities such as "option" or "panels" as described in the "help" options.

Answers to questions were restricted by a "legalvals" (legal values) statement. Acceptable legal value types for M.1 are shown in Table 4-2. In addition to specifying legal value types, the knowledge engineer could also list specific legal values. For example, the legal values for masters option are thesis, paper, report, course, and exam.

The database input was from a dBASE II data file "attrib.dbf" (see Table 3-5). In addition to the data structure introduced in chapter 3, more specific information about each field is shown in Table 4-3. Course information stored in the data file was based on the General Catalog of the University of Arizona, 1985-1987. These attributes were the input to the ADVICE knowledge base when the external function "findattrib" and "read" were invoked. It would take more than 10K bytes of dynamic memory if these
Table 4-2: Acceptable legal value types in M.1.

<table>
<thead>
<tr>
<th>Type</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>a sequence of digits. range: ((-2^{31})) to ((2^{31})-1).</td>
</tr>
<tr>
<td>real</td>
<td>floating point numbers. 6 decimal digits of precision.</td>
</tr>
<tr>
<td>number</td>
<td>either integer or real.</td>
</tr>
<tr>
<td>string</td>
<td>any lower case alpha-numeric sequence.</td>
</tr>
</tbody>
</table>

attributes were stored in the knowledge base.

Conclusions could be displayed on the terminal screen, printed by a printer, or spoken by the DECTalk voice synthesizer. Conclusions of cached expressions were saved in a file on a floppy disk. Figure 4-1 shows the interconnection.

The DECTalk voice synthesizer received two types of messages (1) from an input file read by a C program, and (2) arguments passed from ADVICE to DECTalk through the external interface. The external function talk(), defined in the ADVICE external interface, read an input file. The procedures of talk() included (a) setting up the baud rate, parity, stop bits, and word length for the IBM PC serial communications, and (b) reading in the data file and sending it to the communication port.
Table 4-3: Database file structure of ADVICE.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>course</td>
<td>combination of a dept. abbr and a course number</td>
<td>sie550</td>
</tr>
<tr>
<td>time</td>
<td>semester course is offered: fall, spring, or both</td>
<td>fall</td>
</tr>
<tr>
<td>pre1</td>
<td>the first prerequisite for this course</td>
<td>sie350</td>
</tr>
<tr>
<td>pre2</td>
<td>the second prerequisite for this course</td>
<td>sie270</td>
</tr>
<tr>
<td>grad</td>
<td>number of grad. units this course carries</td>
<td>3</td>
</tr>
<tr>
<td>siegrad</td>
<td>number of units for SIE 500 level and above</td>
<td>3</td>
</tr>
<tr>
<td>area</td>
<td>area covered, based on SIE graduate Handbook</td>
<td>se</td>
</tr>
</tbody>
</table>

SCREEN

DISK — CONCLUSIONS — PRINTER

DECTALK

Figure 4-1: Output interconnections with conclusions.

I defined a function "string()" to pass the arguments to the DECTalk voice synthesizer. Recall the external meta-proposition form

\[
\text{external(FUNCTION_ID, [IN_ARGS]) = [OUT_ARGS].}
\]

The maximum number of input arguments was three, and the maximum
number of output arguments was one. The type for each argument was defined to be an M.1 atom, which was either an alphanumeric sequence or a string. An alphanumeric sequence was a variable containing a value of an expression, and a string was a sequence of characters bounded by single quotes such as 'Welcome to the SIE department.' The function "string()" then sent the atoms to the communication port that had been connected to the DECTalk machine.

4.4 Shutdown

As described in an earlier section, the main menu provided an option "quit" to shutdown the system. The process "quit" in the main menu consists of four basic functions:

a. Call the DECTalk voice synthesizer to say "good bye".

b. Close database file and index file.

c. Reset cache.

d. Stop and exit the system.

The original version of ADVISE had a step to "save cache" before exiting the system. This work was changed to be done automatically after each semester was completed to minimize lose in case of equipment failure. The function "reset cache" cleared up the dynamic memory. And the last function was to call the top level command "exit" that quit the system. The M.1 top level command "exit" or "abort" might also be used to exit the system. However, they would either cause information loss or use up extra memory. Students should use the "quit" option in the main menu to exit the system.
5. SYSTEM EVALUATION

5.1 Verification

To ensure that the expert system ADVICE would give its users accurate advice and correct analysis, two verification schemes were used (1) checking that the knowledge base was correct, and (2) verifying that the program would interpret and apply the information correctly.

5.1.1 Knowledge Base Debugging

This process was used to check if the knowledge base was correct and complete. It involved testing and refining the knowledge in order to discover and correct a variety of errors that arose during construction. The debugging process was a continual process for the author of the expert system. The final system was tested by (1) the expert who was the graduate committee chairman, and (2) various students.

Problems arose during the transformation of the expert's knowledge to the knowledge base. The continued testing and debugging focused on the following areas:

1. Complete and consistent knowledge: Rules generated for ADVICE depend on professors' knowledge, department degree requirements, and university regulations. ADVICE would generate accurate advice only if the rules representing that knowledge were complete and consistent. Buchanan and Shortliffe pointed out "When knowledge is represented in production rules, inconsistencies in the knowledge base..."
appear as conflict, redundancy, and subsumption [11]." In the knowledge base debugging process, "missing rules" seem to be the most common errors detected. Professor Terry Bahill had read the knowledge base to check if the knowledge was represented correctly and consistently. Another way to detect errors in the knowledge base was to check the result with the human expert.

2. Feasible computer representation: While the expert's knowledge may be accurate and complete, it might not be adequately transferred to the computer-based representation. A human advisor easily bends the rules based on personal feelings for a student or situation. For example, a professor might allow students without prerequisite courses to enroll in his class if they have Grade Point Averages higher than 3.4. But this usually varies depending upon individual cases. When it becomes a rule in the knowledge base, it does not seem as flexible unless there are many associated rules to support it. This shows that the computer system makes decisions based on existing rules with no feelings, no sympathies, or exceptions. And because of this, the computer system is able to generate consistent results for anyone under the same circumstances.

3. Syntax mistakes: Mistakes in spelling or syntax were frequent sources of errors. These mistakes must be corrected immediately because M.1 would ignore any rule containing syntax errors. M.1's two built-in tools "trace" and "panels" were useful for debugging a program. As previously mentioned, I found the program "spell" in the UNIX operating system to be a good one for detecting spelling errors.
5.1.2 Program Verifying

There were two distinct aspects in program verification: program proving and program testing. Each of these is an art in itself. Program proving was to ensure that the program contained consistent rules for the system requirement. Professor Terry Bahill, the graduate committee chairman had read and checked the knowledge base several times during the system development, and provided valuable suggestions. To assure that for all possible combinations of inputs, the program and its specification agree, ADVICE was used by various students with different backgrounds. This wide range of users helped to prove the system. Program testing was performed between the system builder and the human advisor. It involved evaluating the performance of the program and revising it to conform to standards of system requirement. All system requirements were satisfied.

5.2 Validation

It was necessary to assess formally the program's performance during the development of ADVICE. By the end of September, 1986 more than fifteen study plans went through the system. This section will concentrate on "what", "when", and "how" to evaluate the expert system.

What to Evaluate? (1) Advice and Performance: Since accurate, reliable advice is an essential component of an expert system, by the time the system has reached completion, every aspect should have warranted formal assessment. Although the mechanisms for deciding whether the system is appropriate is difficult, it is clear that ADVICE would not be accepted by professors if they failed to be convinced that the
decisions, recommendations and advice given were pertinent and reliable. In addition, ADVICE would not be accepted by students if the system failed to provide enough instruction and logical explanation.

(2) Correct Reasoning: Some rules provide reasoning for users. For example, deficiencies for non-engineering major students would not be waived unless the system found such deficiencies were covered by the existing study plan. The courses that were not covered by the plan would be printed on the output for human advisors to decide whether the student needed to take it or not. ADVICE would explain how such situation was being considered and handled.

When to Evaluate? The evaluation process was a continual one that began with system design, becoming increasingly formal as the system moved toward its final implementation.

How to Evaluate? (1) Comparison of system requirements and system performance. Table 5-1 shows the result.

(2) Comparison of advice quality. This comparison involved having the expert system ADVICE and the human advisor review the results of each other's advising. Accordingly, two experiments were designed and executed to do such a comparison. First, the graduate committee chairman was asked to review and critique seven (7) study plans for which ADVICE had evaluated and offered comments. The graduate committee chairman checked the plans, and wrote his findings, comments, and suggestions for each study plan on a separate piece of paper. Table 5-2 on page 45 shows the results of this experiment. The column personal data contains the students' chosen major area
Table 5-1: Comparison of the system requirements and performance.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>instructions in each stage of consultation</td>
<td>yes</td>
</tr>
<tr>
<td>revising facilities</td>
<td>yes</td>
</tr>
<tr>
<td>help facilities</td>
<td>yes</td>
</tr>
<tr>
<td>default answers</td>
<td>yes</td>
</tr>
<tr>
<td>save and print</td>
<td>yes</td>
</tr>
<tr>
<td>logical reasoning</td>
<td>yes</td>
</tr>
<tr>
<td>dBASE II for database</td>
<td>database &amp; index file</td>
</tr>
<tr>
<td>DECtalk voice synthesizer</td>
<td>yes</td>
</tr>
<tr>
<td>validate plan</td>
<td>yes</td>
</tr>
<tr>
<td>save data</td>
<td>end of each semester completion</td>
</tr>
</tbody>
</table>

for the masters degree, the chosen option and their undergraduate major. The rest of the table is self explanatory. Second, ADVICE reviewed fourteen (14) study plans that had been approved by advisors and the graduate committee. ADVICE provided its findings, comments, and suggestions for each study plan. Table 5-3 on page 46 shows the results of this experiment. More discussions follow.

Results: (1) Table 5-2 shows that ADVICE helped the graduate committee chairman to find out what was missing in the handbook. For example, there was no statement in the handbook to distinguish whether SIE 900 or SIE 910 should be taken for the "paper" option. Another example is shown in case C. Students choosing thesis and paper options were asked to submit one or two pages of description about the goals of their research. But there was no such requirement in the handbook for the report option. The attention was brought to the new graduate committee chairman after he read the result of
ADVICE. He pointed out that the "report option should also require a statement of goals of research."

(2) Table 5-3 shows that ADVICE followed the degree requirement strictly. It detected the courses that were not offered in certain semesters and it expected clear answers to its questions. It created "Unsure prerequisites" because the study plans did not mention all of the prerequisites. ADVICE assumed that all information for SIE courses was stored in the database and new SIE courses could be entered whenever they were available. The findings of case B show this effect. Case C in Table 5-3 was approved by the human advisors but not by ADVICE because the human advisors used the following rule "SIE 437x of Fall 1985 = SIE 536" that was not specified in the handbook. Case G was commented because the course number and semester for thesis option were not specified in the plan. SIE 420 is a required deficiency course for all non-majors. However the SIE Graduate Plan of Study Form does not list this course, so most human advisors never checked for this deficiency. ADVICE does.

The expert system allows no ambiguities in the rules. It forces clear thinking. Things must be stated explicitly. For example, when the course SIE 270 (Computer Methods for Engineering) was added as a deficiency course in 1985, the faculty agreed that this requirement would not be explicitly checked for students with engineering undergraduate degrees. Rather it would be assumed that they had such backgrounds. As an afterthought it was agreeded that SIE 310 (Human Factors Fundamentals) would be treated similarly. This informal, wishy-washy agreement was never written into the graduate hand-
book and it posed a problem for the first expert system. An explicit rule had to be added to cover this situation as shown below.

rule-4:
if under_major = engineering
    and nocheck = deficiency-sie310
    and nocheck = deficiency-sie270
then constraint-d = ok.

In addition, ADVICE is able to check equivalent courses so that course material is not taken twice. An example of this can be that a deficiency course SIE 420 is equivalent to SIE 320b, where the former is for graduate students and the latter is for undergraduate students.

In summary, advantages of the expert system can be seen from the evaluation results. ADVICE did accurate calculation and checking based on what was specified in the degree requirement from the handbook. It satisfied the system requirements and performance requirements. It provided reliable consultation and advice.
Table 5-2: Human advisor checked ADVICE's result.

<table>
<thead>
<tr>
<th>Name</th>
<th>Personal Data</th>
<th>Results of ADVICE</th>
<th>Findings of Human Advisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Systems Paper</td>
<td>Approved</td>
<td>Needs distinction between paper and thesis. ADVICE uses SIE 910 for both, he suggested using SIE 900 for paper.</td>
</tr>
<tr>
<td></td>
<td>Engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Systems Course</td>
<td>Approved</td>
<td>Concurred with ADVICE</td>
</tr>
<tr>
<td></td>
<td>Non-Major</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Systems Report</td>
<td>Approved</td>
<td>Report option should also require a statement of goals of research</td>
</tr>
<tr>
<td></td>
<td>Engineering</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Systems Course</td>
<td>Approved</td>
<td>Concurred with ADVICE</td>
</tr>
<tr>
<td></td>
<td>Non-Major</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>Systems Thesis</td>
<td>Approved</td>
<td>Concurred with ADVICE</td>
</tr>
<tr>
<td></td>
<td>SIE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Industrial Thesis</td>
<td>Disapproved</td>
<td>Concurred with ADVICE</td>
</tr>
<tr>
<td></td>
<td>Non-Major</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Systems Course</td>
<td>Disapproved</td>
<td>Concurred with ADVICE</td>
</tr>
<tr>
<td></td>
<td>Non-Major</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5-3: ADVICE reviewed approvals by human advisors.

<table>
<thead>
<tr>
<th>Name</th>
<th>Personal Data</th>
<th>Findings of ADVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Systems Exam Engineering</td>
<td>A transferred course was assumed to be SIE4xx, SIE455 is not offered in spring</td>
</tr>
<tr>
<td>B</td>
<td>Industrial Exam Engineering</td>
<td>SIE 545 &amp; SIE 613 were not found in database</td>
</tr>
<tr>
<td>C</td>
<td>Industrial Exam Non-Major</td>
<td>Not enough SIE courses at 500 level</td>
</tr>
<tr>
<td>D</td>
<td>Industrial Exam Engineering</td>
<td>Unsure prerequisites</td>
</tr>
<tr>
<td>E</td>
<td>Systems Report Non-Major</td>
<td>No prerequisite or deficiency courses were checked</td>
</tr>
<tr>
<td>F</td>
<td>Systems Course Non-Major</td>
<td>Unsure Prerequisites</td>
</tr>
<tr>
<td>G</td>
<td>Systems Thesis Engineering</td>
<td>Thesis units and time unclear</td>
</tr>
<tr>
<td>H</td>
<td>Industrial Thesis Under-Major unclear</td>
<td>Deficiency courses not checked</td>
</tr>
<tr>
<td>I</td>
<td>Systems Report Non-Major</td>
<td>Inappropriate number, SIE5xx, was used for report option. SIE540 is not offered in spring</td>
</tr>
<tr>
<td>J</td>
<td>Major unclear Report Engineering</td>
<td>No semester or years were listed</td>
</tr>
<tr>
<td>K</td>
<td>Reliability Exam Non-Major</td>
<td>ADVICE could not finish evaluation because too many courses in plan do not match course offering time.</td>
</tr>
<tr>
<td>L</td>
<td>Industrial Report Engineering</td>
<td>Time of research is unclear</td>
</tr>
<tr>
<td>M</td>
<td>Industrial Exam Non-Major</td>
<td>Number of thesis units was not specified</td>
</tr>
<tr>
<td>N</td>
<td>Industrial Exam IE</td>
<td>Concurred with human advisor</td>
</tr>
</tbody>
</table>
6. CONCLUSIONS

ADVICE used three pieces of hardware, four software packages, and three knowledge sources. Figure 6-1 on page 49 gives an overview of the system. ADVICE thinks, writes, and talks. It interacts with students. It has the intelligence to provide instructions, do calculations, check plan validities, and generate conclusions. More importantly, it provides quality advice and it saves time for both students and professors. This system was designed for the professors and graduate students of the Systems & Industrial Engineering Department. But it is possible to extend its usage to other departments by either extending M.1's memory capacity or adding more C programs.

Based on the technology requirements stated initially and observations made during the system development, I would like to make three suggestions for future research. The first suggestion is to extend usage of the external function facilities on expert systems. The use of external functions can enable expert systems to interact with other languages since the available expert system shell's capability is limited. ADVICE is an example of adding more features from other languages and softwares to the M.1 expert system shell. ADVICE is able to store data in the database and recall data from database files to M.1. This leads to my second suggestion to try to send output from M.1 to other packages such as dBASE II database files. ADVICE is able to link the DECtalk voice synthesizer to M.1. This leads to my third suggestion to investigate how users react to the synthesized voice during machine consultation.
From learning concepts of artificial intelligence and fundamentals of knowledge-based systems, to applying the M.1 expert system shell to solve a real world problem, I have gained valuable knowledge and experience in expert system design, human/machine interfacing, and computer programming. I believe that this experience will contribute to my future work and research.
Figure 6-1: An overview of the ADVICE expert system.
APPENDIX A: MAKING EXECUTABLE FILES

These are the steps I used to make the executable file M1AUX.EXE for the ADVICE system.

1. Write the external function file: This file contains functions selected from the M.1 external interface functions, the C-to-dBASE interface functions, and user-written functions.

2. Compile the external function file: Use the Optimizing C86 C compiler from Computer Innovations and the batch file "ccsml.bat" supported by Teknowledge. The command is
   ccsml advfunc

3. Link the external function object code. The order of the libraries is important.
   link advfunc+adtk,m1aux,m1aux/m,extlibs+ibmpcs+c86s2s+c2dbase

4. Run Teknowledge's setparas utility program. This program allocates sufficient memory space for the external function file.
   setparas m1aux 4096

5. Admire your executable file M1AUX.EXE.
APPENDIX B: MAKING DBASE II FILES

To use dBASE II, you will need an MS DOS or PC DOS operating system with at least 128K bytes of internal memory. Follow these steps.

To create a database file:
1. Insert dBASE system disk in drive A.
2. Type dbase then hit RETURN.
3. Type create filename to start "create" mode.
4. Follow the on-line instructions to define each field as described in Table 3-5 of this thesis.
5. Hit [Ctrl] and W to save and exit the "create" mode.
6. Type quit to end the dBASE II session.

To append more records:
1. Type use filename to load the database.
2. Type append to start appending.
3. Hit RETURN to stop appending.

To make an indexed file:
1. Type use filename to load the database.
2. Type index on fieldname to indexname to make an indexed file.
APPENDIX C: INSTRUCTIONS FOR USERS OF ADVICE

For first time users:

1. Turn the IBM PC XT on (do not close the disk-drive doors).
2. Insert a disk (for saving your plan) in drive A, close the door now.
3. Change to the subdirectory where the ADVICE system is stored.
   Type cd c then type cd c86 in CE 324G. Or type cd m1 in CE 220G.
4. At the c:\c\c86 (or c:\m1) prompt, type m1 advice and allow 3.5 minutes for M.1 to load the system.
5. Read and follow the on-line instructions carefully.

For advanced users and trouble shooting:

1. If in the middle of a process, you accidently hit the Ctrl+Break keys (which will suspend the process), and you wish to continue the process, at the "ADVICE>" prompt, type restart.
2. If you don't want to continue inputing courses when you are asked to input another course, respond with ignore.
3. If for some reason you want to stop the program no matter what, type quit.
4. There is an option "c" for checking, saving and printing the result. Please use this option to get an output before quitting.
5. The program will save your files automatically. But you can also save it by typing your last name and first name as follows
savecache a:last.first. This should save the conclusions of your input in a file under your name in drive A.

6. A short way to display your courses on the screen is, at the ">>" prompt, (when you are asked to answer a question) type show semester(A)-course(B).

7. After you turn the printer on, please adjust the page too. The printer is ready only when the three lights (power, ready, and on-line) are all on.

8. You can delete courses after you have entered them.

9. You can change your major, option, advisor, etc. Use the "h" or "n" options to display the on-line help facilities, or see page 32 of this thesis.

10. If you cannot finish the program in one setting, you can come back some other time. But be sure to type in the same name next time you log in.
REFERENCES


