USING MIS TECHNIQUES WITHIN INSTRUCTIONAL PROGRAMS:

DATA FLOW DIAGRAMS TO DESIGN INSTRUCTION

and

EXPERT SYSTEMS TO DELIVER EXPERTISE

by

Mary Jane Garrett

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Counseling,
Educational Psychology,
and Special Education

1988

ABSTRACT

USING MIS TECHNIQUES WITHIN INSTRUCTIONAL PROGRAMS:
DATA FLOW DIAGRAMS TO DESIGN INSTRUCTION
and
EXPERT SYSTEMS TO DELIVER EXPERTISE

by

Mary Jane Garrett

This dissertation discussed the increased need for efficient and effective instruction of high level intellectual skills. It was noted that learning theories and instructional design theories had been developed that dealt with cognitive processing, but that the tools currently used by instructional designers did not readily facilitate the teaching of cognitive processes. Problems with a current design technique were discussed, and a need was determined for instructional design techniques that permit developers to apply learning and instructional theories more effectively, especially in the indicated problem areas.

It was noted that business and industry had also been affected by the Information Age. The use of data flow diagrams in instructional design was demonstrated and the following advantages were found: (a) they help to locate the points in the learning process where teachers' help may be needed for students to master the cognitive task; (b) they show the type and position of information needed for each process within the instructional system; (c) they aid

in analyzing the cognitive skill to be taught, identifying the cognitive processes for higher order thinking skills, and presenting the cognitive skill as a whole; (d) they show the interrelationships among topics and processes of instruction at various levels from course to lesson; and (e) they provide a convenient way to explain a course to interested, content-area-naive, parties.

The use of expert systems to deliver expertise was discussed. The limitations of expert systems and the differences between expert systems developed for industry and instruction were discussed. The steps necessary to develop an expert system for the delivery of expertise within instructional programs was discussed and an example of the development of an expert system to help students learn to debug computer programs was given. Finally some cautions about the use of expert systems as a part of instruction were presented.

ACKNOWLEDGMENTS

I wish to express my appreciation to Dr. Lawrence Alexander, chairman of my Doctoral Committee, for his encouragement and understanding. His counsel and judgement were most influential in my years at Michigan State University, and his interpretation of often confusing policies and practices was critical in the completion of my degree. The amount of time he willingly devoted to the advancement of my knowledge in instructional design and cognitive psychology was staggering. I wish there were some way to repay him for his kindness other than remembering his generosity when working with my students.

I wish also to express my gratitude to Dr. Paul Slocum, Dr. Robert Poland, and Dr. Richard McLeod, members of my committee, who took time from their extremely busy schedules to attend meetings, review papers, and offer advice. I feel special gratitude to Dr. Dale Davis, from the University of Michigan, Flint, for being willing to provide expertise in Management Information Systems, providing support in my approach to the topic, and for being part of my committee at Michigan State University.

Two additional members of Michigan State University's faculty have more than earned my gratitude: Dr. Joe Byers and Dr. Stephen Yelon. Neither of these dedicated professors had any obligation to provide support for me in the

completion of my program, yet Dr. Byers' door was also open for some friendly advice and Dr. Yelon even reviewed portions of my dissertation and made numerous suggestions for improvements.

I am also grateful to two former colleagues, Frank Gregory and Paul Heavilin, who not only provided encouragement and emotional support, but also proofread papers and offered constructive criticism. Also, without the willingness of Pat Gilbert to allow me to adjust my school schedule to attend classes at Michigan State University, my advanced education would have remained a dream.

Finally no words can express my indebtedness to my parents, Andrew and Irma Tesner, whose love and confidence in me gave me the desire to learn more, and my children, Patrick and Paula Garrett, whose willingness to often "do without their Mother" made this endeavor possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

STATEMENT OF THE PROBLEM AND THE PURPOSE OF THIS STUDY

With the advent of the Information Age new knowledge is being produced at an ever increasing rate. Consequently, efficient and effective education is becoming increasingly important as a means for communicating this new knowledge. This has placed an added burden on teachers who are expected to teach more content at higher skill levels. In addition, schools are expected to provide instruction in sex, drugs and careers, areas that, in the past, were never their responsibility. To meet these demands, educators need better tools and techniques for planning and delivering instruction. Resnick (1963, p. 439) suggests "Programmed instruction offers a means of coping with the universally acknowledged teacher overload". Reigeluth (1983) states that if we can develop highly effective instructional resources, then we can free some of the teacher's time to work on the social, psychological, emotional, and moral development of our children and prepare them better for living in the Information Age.

Business has an analogous problem as a consequence of our entry into the Information Age, viz, to manage more information, in more areas, at a higher level of application so that not only are day-to-day transactions processed, but

all levels of management are supplied with information and supported in the use of technological tools for decision making. To alleviate these problems, industry has adopted the techniques of Management Information Systems (MIS), a branch of computer science/data processing that is concerned with the organization and coordination of information.

MIS has developed two techniques to deal with the need for better planning of information systems and more efficient delivery of organized information. These are structured design methodology, such as data flow diagrams, and expert systems. Structured design methodology uses systems concepts to analyze and describe an information system as functional subsystems and to define the boundaries of and interfaces between each subsystem. Expert systems are computer programs that simulate the cognitive processes of a human expert.

Because both education and business systems have similar problems, involving efficient handling of large amounts of information, it is reasonable to inquire whether MIS techniques of data flow diagrams and expert systems might be used profitably in education. Perelman (1987) comments that recent changes in instruction have been facilitated by the use of advanced information technology to design, manage, and deliver instruction. However, adaptation and transfer of techniques from other fields to the field of instruction

should not be done without careful consideration of the unique constraints in the field of education.

Although current instructional design techniques apply well to the teaching of learning objectives, such as knowledge, comprehension, and application, from the lower levels of Bloom's (1950) taxonomy, they do not apply well to the teaching of higher level cognitive skills, such as analysis, synthesis, and evaluation. In addition, current design techniques have several problems that a designer of instruction should consider when attempting to improve instruction: (a) inadequate consideration for need for expertise and knowledge, (b) insufficient focus on needed information, (c) limitations created by expressing cognitive skills in a task format, (d) lack of consideration for the interrelationships among topics and processes in the instruction, and (e) lack of convenient ways to explain a course to interested, content-area-naive, parties. The tools used in designing instructional systems affect the efficiency of the instruction produced. Instructional design tools need to be analyzed and improved. These problems are discussed in more detail in Chapter II.

The purpose of this dissertation is to investigate the adaptability of MIS techniques to the design of instructional systems. Two MIS techniques, data flow diagrams and expert systems, are examined to determine the potential utility of these techniques in the design and delivery of

instructional systems. To examine this potential utility, a demonstration is provided of how they might be used in a particular instructional system, namely teaching programming in a secondary level vocational data processing course. A second demonstration of the use of data flow diagrams in the design of an interactive video course on pain control in veterinary medicine is also made.

QUESTIONS CONSIDERED IN THIS STUDY

Broad Questions to be Addressed

1). **Might the MIS design tool, data flow diagrams, be used to provide information about the relationships between major chunks of instruction, that is courses, units, chapters, and lessons; or the major cognitive processes within the chunks?**

2). **Might the MIS design tool, data flow diagrams, indicate what information might be needed by students at various points of instructional programs?**

3). **How might the use of data flow diagrams to design MIS systems be modified to use data flow diagrams to design instructional systems?**

4). **When the objective is to teach complex cognitive processes such as problem solving, might data flow diagrams indicate specific points in an instructional program where individualized expert help might be provided by an expert system?**

5). **How might the MIS techniques for the development and use of expert systems be modified to use these techniques for the development of expert systems for instruction?**

Specific questions to be answered

1. How might the use of data flow diagrams in the design of a particular secondary vocational data processing course provide information about the relationships between major chunks of instruction, that is courses, units, chapters, and lessons; or the major cognitive processes within the chunks?
2. How might data flow diagrams used to design a particular secondary vocational data processing course indicate what information might be needed by students at various points of the instructional program?
3. How was the use of data flow diagrams to design MIS systems modified to use data flow diagrams to design a particular data processing course?
4. How might data flow diagrams, used to design a particular instructional program to teach the problem solving skill, computer programming, indicate potential points in the instruction where expert systems might be used to provide students with expert help?
5. How were steps used to develop MIS expert systems modified to develop an expert system to help teach students in the data processing course how to debug Basic computer programs?

SCOPE AND LIMITATION OF THIS STUDY

This feasibility study will seek to demonstrate: (a) that data flow diagrams might be used in instructional design to provide information that is useful in the analysis, design, and development of instruction; (b) that data flow diagrams might indicate where expert systems may be appropriate in an instructional program involving the teaching of higher order intellectual skills; and (c) that expert systems might be used to deliver expert instruction for secondary students. To show that these benefits are possible, specific examples using a vocational data processing program and an interactive video course on pain control in veterinary medicine were developed.

The study presented in this paper has four major limitations: First, only two examples of the use of data flow diagrams were presented and only one expert system was developed. The limitations of generalizations from one or two examples is obvious. The two demonstrations presented involved computer applications. Since the tools used in the instructional design were adapted from the field of computer science, the question of the influence of common domain specific characteristics must be raised. Third, the study was concerned with feasibility. There was no attempt to test empirically the efficiency of the use of data flow diagrams or the effectiveness or generalizability of the

specific expert system designed. Fourth, only the development of a data processing curriculum was examined to determine the problems with instruction developed with the limitations of the current instructional design and analysis tools. These issues will be left for additional study.

OVERVIEW OF THE DISSERTATION

This dissertation is concerned with the feasibility of transferring system design techniques from MIS to the design of instructional systems. It does not involve any empirical test of these techniques. Consequently, the format differs from the format followed by the typical empirical research dissertation. First, the inadequacies of current instructional design techniques used to develop the State of Michigan's Vocational Data Processing Curriculum Guide are presented. Then each of the two techniques from MIS that are being considered for transfer to instructional design are presented, along with a demonstration of how each might be used in the design of instruction. A discussion of the modifications that this researcher found necessary to transfer the techniques to the instructional design of the two systems considered and the benefits of the techniques when used in those instructional designs are presented. Specifically:

Chapter 2 discusses the problems with the current instructional system analysis and design techniques that were found when the development of a data processing curriculum was examined.

Chapter 3 describes the MIS structured design technique, data flow diagrams, indicates how they are used in information systems design, and demonstrates how they might

be used to design a vocational data processing instructional system. In addition, a brief summary is given of the use of data flow diagrams in the design of instructional courseware using interactive video. A demonstration is presented of the use of data flow diagrams in the development of interactive video courseware for pain relief in veterinary medicine.

Chapter 4 describes the MIS delivery technique, expert systems, indicates the different types of expert systems and their uses, and indicates how expert systems are developed and used by industry. Then the business use of expert systems is contrasted with potential instructional usage and the design and development of an expert system for use in a vocational data processing instructional system is demonstrated.

Chapter 5 presents the conclusions and suggestions for further study.

A Glossary of terms used may be found before the appendices.

CHAPTER 2

PROBLEMS WITH THE CURRENT INSTRUCTIONAL ANALYSIS TECHNIQUES

Introduction

Education has always sought to provide effective and efficient instruction, but with the entry of our society into the Information Age, pressure increased for education to communicate new knowledge at higher levels and increasing rates. Instruction, according to R. Gagne (1985), deals with the deliberate arrangement of events in the learner's environment for the purpose of making learning happen effectively. Therefore, to meet the instructional challenges of the Information Age, educators must develop efficient and effective ways of arranging the instructional events in the learner's environment, including those events that will lead to the development of higher order thinking skills.

Efficient instruction is dependent not only on the sophistication of underlying theories, but also on the technology for applying these theories. Although behavioral learning theories predominated in American educational psychology for most of this century, in recent years learning theories incorporating cognitive theories have been developed. These theories provide descriptions of effective instruction that include information processing and higher order thinking skills (Ausubel, 1968; Bruner, 1960;

R. Gagne, 1985). Instructional design, the technology that applies learning theory to instruction, has also shifted from a behavioral science orientation, where the emphasis was to promote a student's overt performance by the manipulation of stimulus materials, to cognitive science orientation, where the emphasis is to promote cognitive processing (Merrill, Kowallis & Wilson, 1981). Currently there are several good instructional design theories that include techniques for dealing with higher order intellectual skills (Collins & Stevens, 1983; Gagne & Briggs, 1979; Landa, 1982; Merrill, 1983; Reigeluth, 1983; Reigeluth & Stein, 1983; Scandura, 1983).

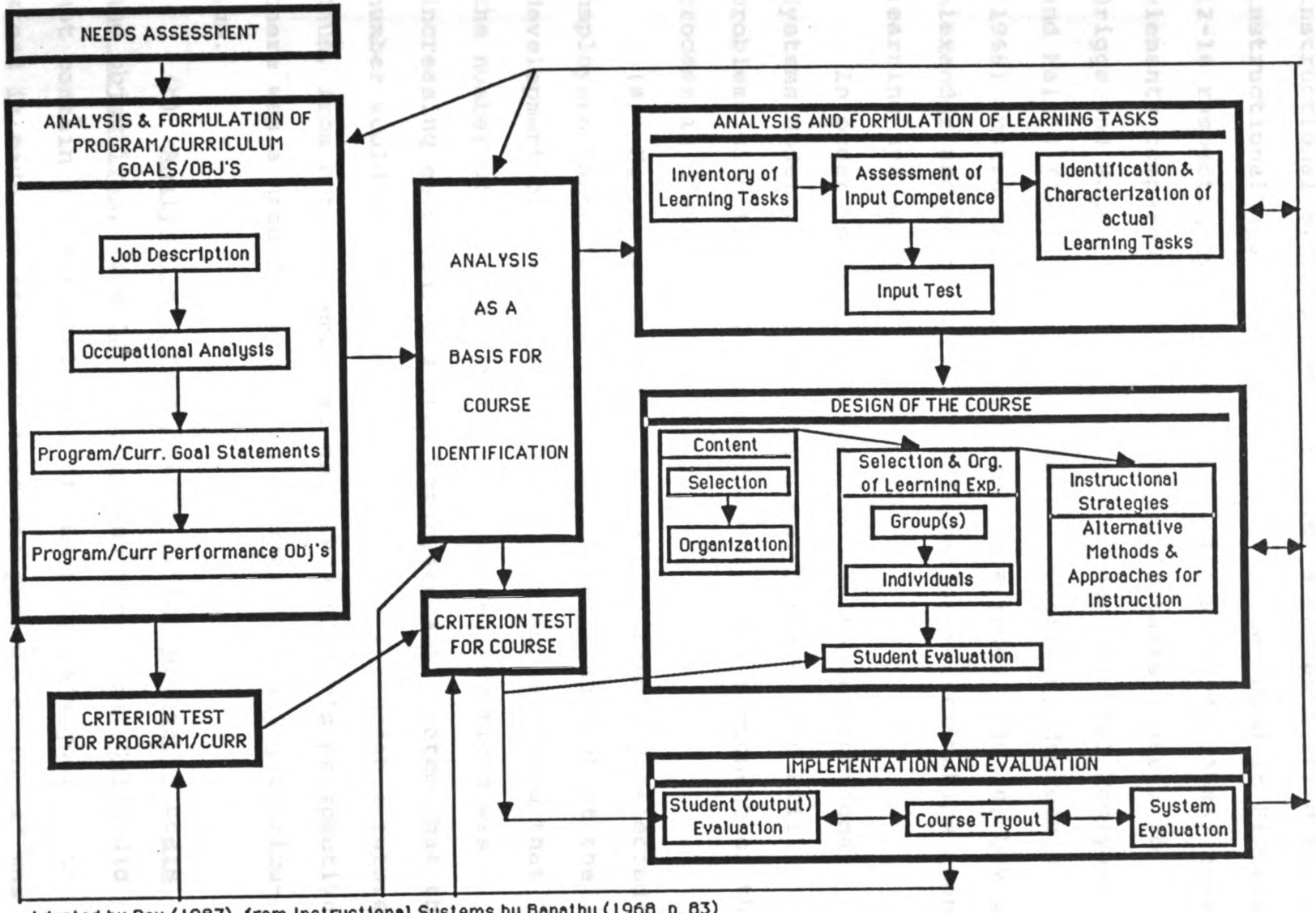
Since good learning and instructional design theories exist, the problems encountered in attempting to design efficient instruction appear to lie not in the theories, but in the tools and techniques developed to apply those theories. As Merrill, Kowallis and Wilson (1981) noted, "models for the development of instruction are surprisingly lacking in prescriptions that suggest how to execute the various steps in the model" (P. 300).

To examine more clearly the limitations of the tools used by current instructional design techniques, this chapter discusses the development of the Michigan Department of Education's Data Processing Curriculum Guide using an instructional systems model modified for vocational courses. As a vocational data processing instructor in Michigan, this

researcher took part in the development of the Curriculum Guide. After discussing the process of developing the Guide, selected portions of the curriculum produced are examined and the following problems are discussed: (a) inadequate consideration for need for expertise and knowledge, (b) insufficient focus on needed information, (c) limitations created by expressing cognitive skills in a task format, (d) lack of consideration for the interrelationships among topics and processes in the instruction, and (e) lack of convenient ways to explain a course to interested, content-area-naive, parties. Finally this chapter discusses a possible source of instructional analysis and design tools and techniques that may alleviate these problems.

The Development of the Michigan Department of Education's Data Processing Curriculum Guide

The development of the Michigan Department of Education's Data Processing Curriculum Guide is described by the Instructional Systems Model shown in Figure 1. Note that the main components in that model are (a) needs assessment, (b) analysis and formulation of the program's goals and objectives, (c) occupational analysis, (d) analysis and formulation of learning tasks, (e) design of the course, and (f) implementation and evaluation. To indicate that the Instructional System's Model is typical of current



Adapted by Ray (1987), from Instructional Systems by Banathy (1968, p. 83)

Figure 1: Instructional Systems Model

ins

ins

12-

ele

Bri

and

(19

Ale

lea

Sys

pro

pro

emp

dev

the

incr

numb

Thus

ther

lum.

and

not

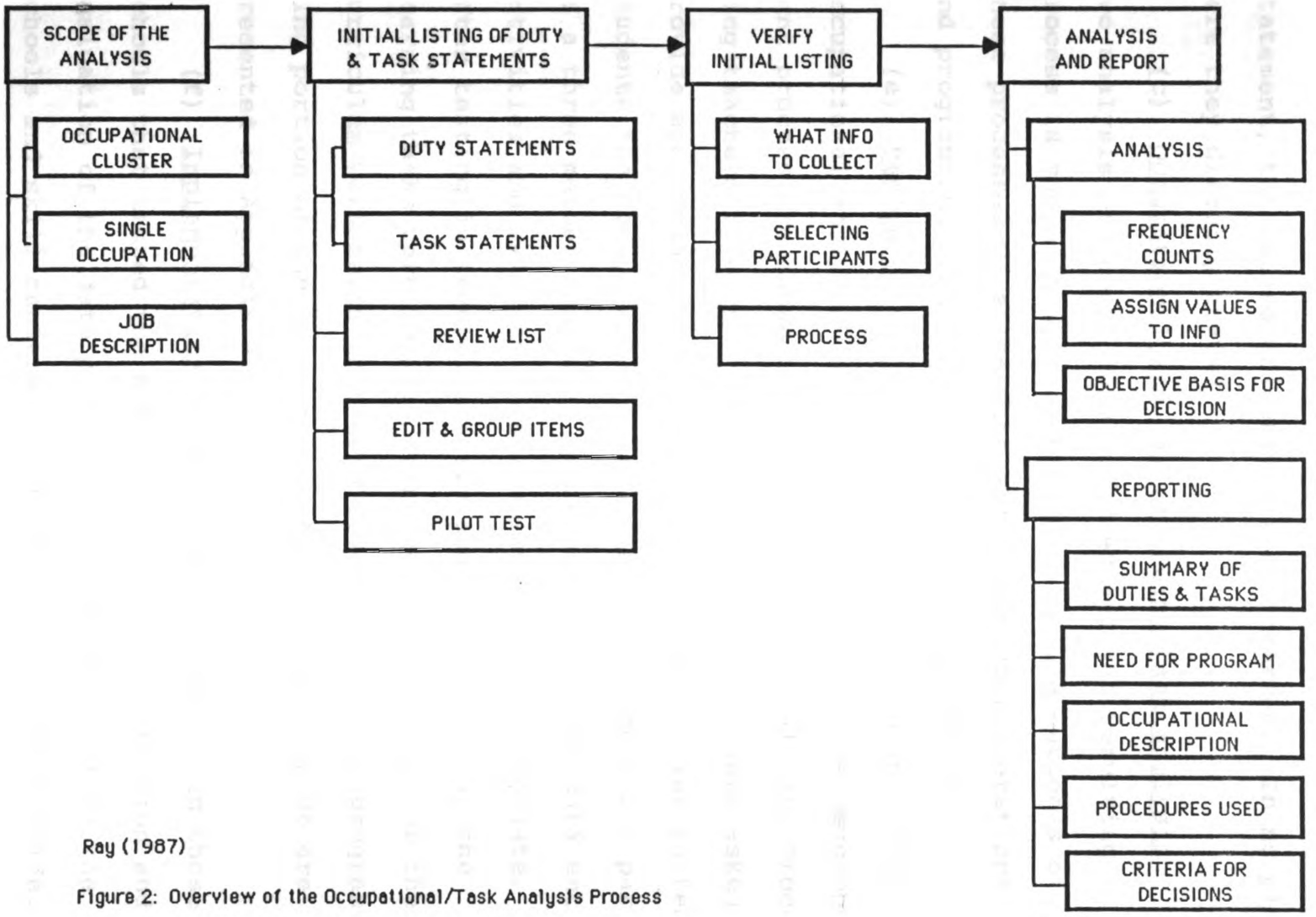
stea

instructional systems design models, a representative set of instructional systems design models can be found in Figures 12-16 respectively in Appendix A: (a) Richey's (1986) core elements compared to Andrews/Goodson's tasks (1980); (b) Briggs and Wagner's (1981) stages of design; (c) Roblyer and Hall's (1985) courseware design model; (d) Hamreus's (1968) instructional systems procedure model; and (e) Davis, Alexander and Yelon's (1974) information relationships among learning system design procedures.

In order to show that even though the Instructional Systems Model was followed completely, there were still problems with the curriculum produced, each component of the process is discussed in detail:

(a) Needs assessment. Surveys of randomly selected employers taken for the needs assessment component of the development of the Data Processing Curriculum showed that the number of new jobs in the data processing field was increasing each year and that employers anticipated that the number would continue to increase in the foreseeable future. Thus, from both the employee and the employer's prospective, there was a need for a vocational data processing curriculum.

(b) Analysis and formulation of the program's goals and objectives. The Data Processing Curriculum Guide did not contain a clear statement of the program's goal. Instead it made the statement that the program's "focus" was



Ray (1987)

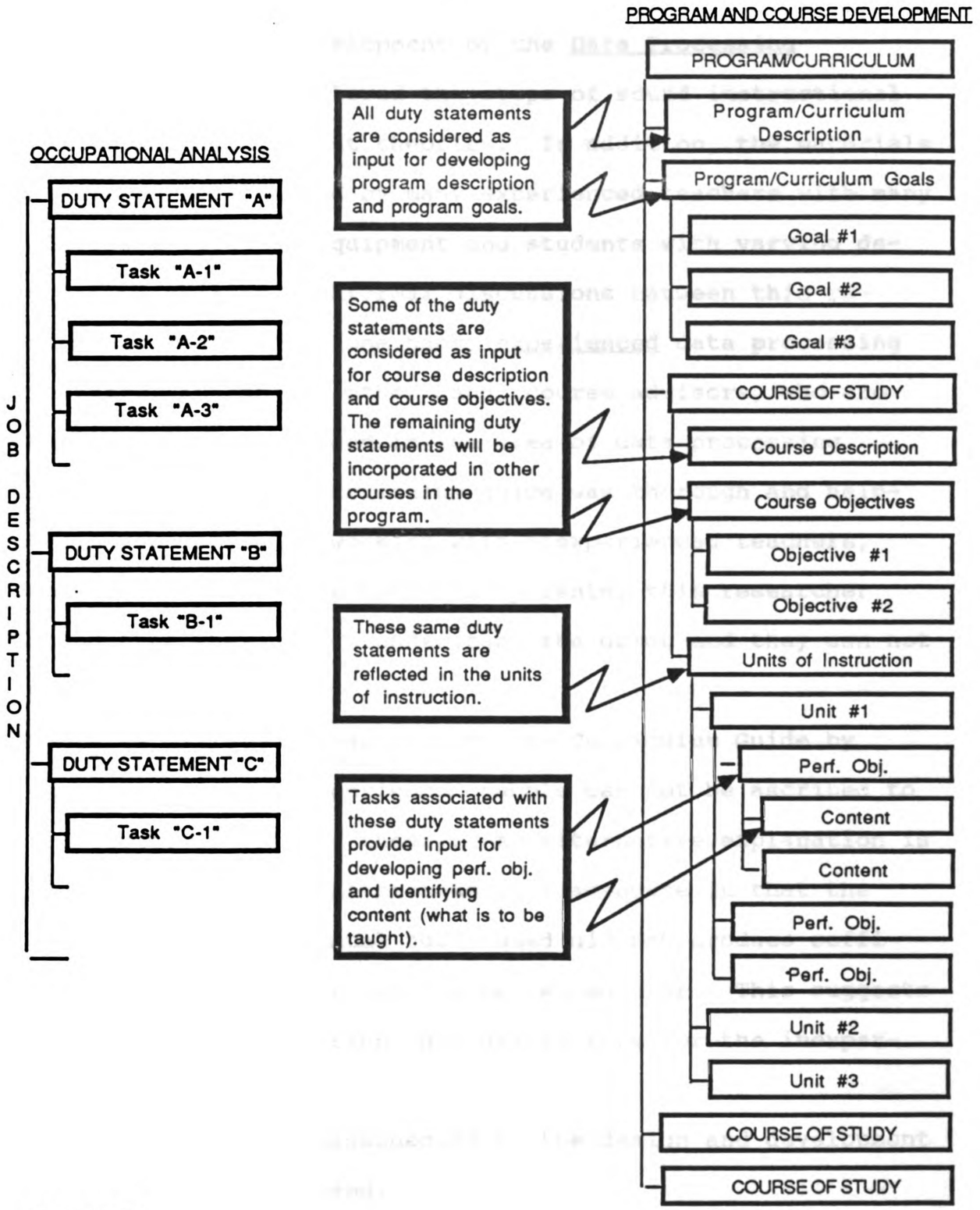
Figure 2: Overview of the Occupational/Task Analysis Process

occupational preparation in the areas of data entry, computer operations, and computer programming. From this, statement, the course objectives were inferred. In no place were they directly stated.

(c) Occupational and (d) learning task analysis. The two analysis processes were combined and the resulting process is shown in detail in Figure 2. The outcomes of these processes were the data entry, computer operations, and programming task lists presented in Appendix A.

(e) Design of the course. The details of how the occupational analysis was then used in the course development process are shown in Figure 3. Vocational Data Processing teachers throughout the State of Michigan were asked to provide appropriate learning and teacher activities to teach students the occupational tasks. This researcher was part of a three member team that reviewed some of the data entry activities and selected those believed most appropriate. After testing in several schools and skill centers, the learning tasks were revised where necessary. Some of the curriculum worksheets that were developed for the programming portion of the data processing curriculum guide are presented in Appendix A.

(f) Implementation and evaluation. Except in those schools that tested the materials, the implementation and evaluation of the data processing course was left to the schools and skill centers that used the curriculum guide.



Ray (1987)

Figure 3: Relationships between Occupational Analysis and Program & Course Development

Clearly the development of the Data Processing Curriculum Guide followed the steps of sound instructional design and development theories. In addition, the materials developed were tested by many experienced teachers with many different types of equipment and students with varying degrees of ability. Informal discussions between this researcher and, on the one hand, experienced data processing teachers, and on the other hand, course advisors, who are businessmen experienced in the area of data processing, indicated that the curriculum guide was thorough and helpful. However, when working with inexperienced teachers, counselors, administrators, and parents, this researcher found that they do not understand the guide and they can not properly use it.

The different reactions to the Curriculum Guide by experienced and inexperienced people can not be ascribed to inadequate underlying theory. An alternative explanation is that the developmental process is inadequate in that the design and developmental tools used did not produce sufficient information for the inexperienced user. This suggests the need for information in a usable form for the inexperienced user.

The specific inadequacies of the design and development tools are now discussed.

Inadequate Consideration for need for Expertise and Knowledge

Most cognitive psychologists agree that experts and novices in any field differ in the amount of domain-specific knowledge they possess and in the strategies that they apply to the knowledge base in solving problems (E.D. Gagne, 1985; Glaser, 1984; Mayer, 1983; Merrill, Kowallis, & Wilson, 1981). During the learning processes, practice with successive presentations of novel problems is necessary for students to develop progressively more expert cognitive strategies and teachers must provide students with feedback as to the efficiency and effectiveness of the strategies used (Collins & Stevens, 1983; Landa, 1982; Reigeluth & Stein, 1983). But this implies that when practice exercises for cognitive strategies are developed, instructional designers must attempt to locate the points in the instructional process where the typical learner must be supplied with expert help to efficiently complete the problem solution. Thus, it is necessary to satisfy the student's need for both expert knowledge and problem-solving strategies during the learning process.

Notice that the curriculum worksheets for programming in the Guide attempt to tell teachers how to do this. In addition, note the use of the action verbs in the teacher activities: assign readings, provide examples, describe the process, hand out samples, explain the use, lead a

discussion, demonstrate, place the solution on the chalkboard. For experienced teachers, this listing of activities with action verbs is sufficient. But for inexperienced teachers attempting to teach cognitive skills such as programming, presenting the activities alone misses the critical information of when to provide students with expert help as students are attempting to perform the task.

However, when most inexperienced teachers attempt to teach students behavioral skills such as data entry, then having only activities with action verbs is not a problem. For example, consider the tasks for data entry. The majority of the tasks begin with "manipulate". The few that begin with "analyze" are really only comparing a given activity to a list of possible activities and picking the ones that match. These are skills that can easily be described and taught using behavioral models and techniques.

It is difficult to develop an instructional guide for teaching cognitive strategies. One difficulty stems from the fact that cognitive operations are unobservable and automated to the extent that their performers are often unaware or only partially aware of them (Resnick, 1987). Thus, although vocationally certified teachers have adequate practical content area experience, they often lack experience in teaching the cognitive skills involved. In fact, they are often unaware of the cognitive skills that are involved in programming, and usually are unaware of the

points within the instructional process at which students need the teachers' help in developing those cognitive skills. Yet nowhere do the curriculum worksheets tell teachers of that need.

A second difficulty stems from the fact that students are not passive receptors of knowledge, but actively interpret incidents in the learning situation with a large set of preexisting ideas that impose some kind of meaning on what they are learning (Norman, 1980; Resnick, 1976). Therefore, instructional methods must be adapted to the individual student's interpretations of that instruction. Experienced teachers know the points at which to watch for student misinterpretations of the instruction. Inexperienced teachers often fail to note the occurrence of student misinterpretations that will make learning more difficult for the students as they progress to more challenging material. Designing instruction that takes into account the points at which students need help and students' idiosyncratic interpretations of the instruction is one of the greatest challenges for educators.

But, despite these difficulties, some researchers have developed instructional methods that provide students with expert help as students develop cognitive strategies (Baker & Brown, 1984; Bereiter, 1985; Palincsar, Brown, & Martin, 1987). In addition, Collins and Stevens (1983) have

developed a theory of instruction, called Inquiry Teaching, that provides students with expert help when it is needed. A major characteristic of inquiry teaching is that a descriptive theory of expert performance, that is a detailed report of what, how, and why something is done, is in fact a prescriptive theory of instruction, that is a set of directions for the learner to follow to reach the desired goal of learning, for the non-expert performer.

These theoretical developments are a further indication that the problem is not the fault of instructional design theory, but that instructional design tools and techniques are needed that systematically indicate the specific points in an instructional program where expert help is likely to be required. Unfortunately, even if inexperienced teachers know where students are likely to require expert help, often they can not find time to provide this help. Instructional delivery methods are needed that can make that help available to students in a timely manner.

Insufficient Focus on Information

In today's Information age society, there is simply too much to know for people to keep current on everything. Instead of attempting to "know" everything about a given subject, it makes sense to consult up-to-date references (Adams & Imhof, 1987). For example, modern data processing

shops provide their programmers, who are professionals, with manuals, standards, and libraries of existing code.

However, consider how the vocational Data Processing Curriculum Guide handles the information that students must have in order to program. Note the key phrases concerning needed information in the programming tasks described on the worksheets in appendix A: "using a knowledge of data processing equipment and techniques" (CP-2), "using a knowledge of programming techniques" (CP-5), and "using a knowledge of the specific language" (CP-11 and CP-15). These phrases imply that it is necessary for students to know this information in order to perform the task, that is, knowing this information is prerequisite to completing the tasks. In other words, students must know this information instead of being provided with the information when it is needed.

Carroll (1976), Scandura (1983), and Snelbecker (1983) all stressed the need in instruction to consider the information students need in order to process new knowledge and apply existing knowledge. Moreover, when information is presented within the context of its use, students are likely to store that information in a manner consistent with the subject matter structure (Bruner, 1960). Therefore, it may be inferred that not only is it not necessary for students to know all the information necessary to complete tasks before they attempt to complete the tasks, learning the information before it is used may even be detrimental to

cognitive processing. However, nowhere in the Data Processing Curriculum Guide is there any indication of what information should be available to the students as they attempt to program.

Limitations created by expressing cognitive skills in a task format

Olson (1973) pointed out that cognitive skills such as making discoveries, speaking convincingly, writing effectively, and social and ethical skills--cannot be taught explicitly because the algorithms underlying them are not known, or where known, are too complex to communicate effectively. Yet certainly these important skills can be taught by using instructional techniques such as "thinking out loud", modeling, demonstrating, and providing feedback in practice sessions (Baker & Brown, 1984; Bereiter, 1985; Gagne, 1985; Palincsar, Brown, & Martin, 1987; Resnick, 1987). A common characteristic of these instructional techniques for teaching cognitive skills is that they teach the skill as a whole.

However, the Curriculum Guide lists tasks that are components of cognitive skills without any indication of how they work together to form a complete skill. This may not be a problem for experienced teachers, but inexperienced teachers may teach only the component tasks instead of the skill as a whole.

Experienced teachers know that for cognitive skills, instruction should start with a broad description of the skill and then, level by level, should establish the component subskills within the context of the complete skill until eventually the students are presented with the lowest level of tasks. To help inexperienced teachers teach this way, instructional design should use techniques that analyze the cognitive skill level by level, focusing on information needed, cognitive processes, that is points where students store, retrieve, organize, apply, or transfer information, and the relationships among the processes. In addition, instructional guides thus developed should convey this information to teachers. This level by level approach is the type of instructional design technique prescribed in Reigeluth and Stein's (1983) Elaboration Theory. If the Curriculum Guide were to convey component skill information using a technique that clearly showed the part that each task played in forming the complete skill, then inexperienced teachers might more effectively teach complex cognitive skills such as programming.

Lack of Consideration for the Interrelationships Among Topics of Instruction

Most of the learning and instructional design theorists consider interrelationships among the topics of instruction, the structure of the subject matter, and previous knowledge

of the students very important (Ausubel, 1968; Bruner, 1960; R. Gagne, 1985; Landa, 1982; Knirk & Gustafson, 1986; Scandura, 1983; Reigeluth & Stein, 1983). Bruner (1960) stated the importance of relationships in instruction most succinctly with the comment that "To learn [subject matter] structure, in short, is to learn how things are related" (p. 7). Therefore, instructional design should use techniques that clearly show the interrelationships among the topics and processes.

Yet nowhere in the Data Processing Curriculum Guide are any of the relationships among the topics and processes shown. There is no indication that data entry, computer operation, and programming skills share common components. There is no indication of the sequence in which the tasks should be taught. Experienced teachers know the importance of such interrelationships and convey them to their students. Determining the interrelationships should be an important part of the instructional design process and conveying those interrelationships to teachers should be a routine part of instructional development.

Lack of Convenient Ways to Explain a Course to Interested, Content-Area-Naive, Parties

In the section of this chapter where the development of the Curriculum Guide was discussed, this researcher noted that inexperienced teachers, counselors, administrators, and

parents do not understand the Guide and can not properly use it. But these parties are the stakeholders in the course. Inexperienced teachers need to present instruction to students before they can become experienced. Counselors must help students decide if they should take the vocational data processing course. Administrators must make both resource allocation decisions and decisions about what course offerings should be made available to the students of their districts. Parents need not only to help their children make intelligent choices as to what courses they should take, but they must also support their children if they encounter difficulties in learning the skills presented in the course.

Each of these stakeholders must be able to understand what is involved in a vocational data processing course. Yet a thick curriculum guide is an improbable medium for explaining the course to these stakeholders. Moreover, the brief "focus" statement and the task lists available in the Guide are almost meaningless to a person with no experience in data processing. Even experienced teachers who do understand the Curriculum Guide have difficulty explaining the course to people who have limited knowledge of the typical data processing professional's vocabulary or cognitive style. The designers and developers of the Data Processing Curriculum Guide made no attempt to provide users with a reasonable way of understanding the course.

Summary and Conclusion

In this chapter the development of the Michigan Department of Education's Data Processing Curriculum Guide was discussed as an example of a typical instructional development process. Then selected portions of the Guide were examined and the following problems were discussed: (a) the Guide did not provide adequate information about points within the instructional program where students typically required expert help, (b) the Guide did not indicate the location and type of information that was needed by the students as they attempted to learn cognitive skills, (c) the Guide did not indicate the relationships among the topics and processes in the instruction, (d) The Guide listed tasks in a way that might limit the instruction presented by some inexperienced teachers, and (e) the Guide did not give interested persons, who had little knowledge of data processing, any convenient way to understand the data processing curriculum. After examining several instructional theories the conclusion was reached that the problems above were not caused by the underlying theories, but were the result of the limitations of the tools and techniques used in designing and developing the curriculum.

To alleviate these limitations, instructional design must incorporate techniques that allow developers to apply learning and instructional theories effectively. As many leaders in the field of instructional theory have stated,

the practice of designing and implementing instruction is greatly influenced by the tools used by instructional designers (Glaser, 1976; Gropper, 1983; Kowallis & Wilson, 1981; Mayer, 1983). The tools used by instructional designers affect the implementation of the instruction. Instructional designers need to examine the tools they now use and determine if an improvement in the tools used to analyze and design instruction might not lead to improvements in the implementation of instruction, especially in the areas presented as problem areas in this chapter.

A Possible Source of Instructional Analysis and Design Tools

It is not sufficient merely to recognize that the Information Age has created a need for improved instructional design tools and techniques; the improvements must be made. In doing this educators should note that business and industry have also been affected by the Information Age. Therefore, one source of possible improvement in the tools used to design instruction for the Information Age is the information management sector of the business world. Recent changes in instruction have been facilitated by the use of advanced information technology to design, manage, and deliver instruction customized for the individual learner (Perelman, 1987). But everything known about technology, innovation, and productivity in other fields needs to be reinterpreted in light of this cardinal fact: Education is

the only business where the consumer does most of the work (Perelman, 1987, p. ES-1).

The next chapter examines some tools currently being used by Management Information Systems (MIS) that may have potential to alleviate problems with current instructional analysis and design techniques. However, as these MIS tools are examined, care will be taken to "reinterpret" their use in the light of Perelman's warning.

CHAPTER 3

INSTRUCTIONAL DESIGN USING DATA FLOW DIAGRAMS

Introduction

In the preceding chapter, several problems with current instructional design techniques were discussed. In this chapter the alleviation of those problems through the use of data flow diagrams in the instructional design process is discussed.

First, the purpose, conditions, methods, and outcomes of instructional analysis and design using data flow diagrams are discussed. Second, the use of data flow diagrams in Management Information Systems is reviewed. Third, an example is presented of instructional analysis and design using data flow diagrams within a vocational data processing course. Fourth, an example is presented of instructional design using data flow diagrams for an interactive video course in veterinary medicine. Last, the advantages of the use of data flow diagrams in instructional analysis and design are discussed.

Instructional Analysis and Design Using data flow diagrams

A data flow diagram is a graphic tool that could be used in instructional design to alleviate some of the problems that may be found with instruction designed with the

current design tools. Data flow diagrams might do this in many ways: (a) they might help to locate the points in the learning process where teachers' help may be needed for students to master the cognitive task; (b) they might show the type and position of information needed for each process within the instructional system; (c) they might aid in analyzing the cognitive skill to be taught, identifying the cognitive processes for higher order thinking skills, and presenting the cognitive skill as a whole; (d) they might show the interrelationships among topics and processes of instruction at various levels from course to lesson; and (e) they might provide a convenient way to explain a course to interested, content-area-naive, parties.

Since data flow diagrams are used for many different types of systems in MIS, data flow diagrams probably could be used for designing instruction for many levels of knowledge. However, since current instructional design techniques are adequate for behavioral and lower level intellectual skills, the technique would probably be most efficient when applied to instructional design involving higher order thinking skills, such as problem solving, or when applied to the development of computer aided instruction. In problem solving, the complexity of the task and the difficulty in designing instructional guides for teaching cognitive strategies would warrant the extra effort necessary to use data flow diagrams. In computer aided

instruction, such as interactive videodisc courseware, the structure of the data flow diagrams developed in designing the instruction corresponds to the structure necessary for programming the instructional system.

This researcher's experience in developing data flow diagrams as an instructional design technique has shown that the designer must have certain information: (a) clearly stated goals of the course or lesson; (b) access either to subject matter content experts or content analysis in the form of job, task, or cognitive analysis information; and (c) access to expert instructors in the subject matter content area. Also, the designer must know the level of detail desired by the teachers or instruction developers.

The methods for using data flow diagrams in instructional design closely resemble the methods employed in the design of Management Information Systems. Using the goals as a guide, the designer analyzes existing real world processes, consults reference and/or research information, and seeks the advice of content and instruction experts to determine the major components of the system and their interrelationships.

The levels of the analysis/design should be determined in consultation with the developer or instructor to determine the appropriate level of detail, but should normally start at the course level to establish the major interrelationships. Other possible levels of detail are (a)

curriculum, (b) program, (c) course, (d) section, (e) unit, (f) chapter, (g) topic, (h) subtopic, (i) lesson, (j) lesson component. Each level should be constructed using data flow diagrams and should include: (a) the starting and ending points; (b) no more processes, that is points where information is organized, transformed, or applied, than the number that could be stored in short term memory at one time; (c) an indication of what strategies and knowledge are necessary for each process point; (d) points where inputs from outside the instructional program, such as expert help, are needed; and (e) descriptions of the information passed from process to process to the level of detail needed by the developer or instructor[1]. For example, prerequisite job skills may be a sufficient identifier for an experienced developer or teacher, but for a different developer or teacher the designer may need to specify prerequisite abilities, such as able to read at the fifth grade level, able to write or print legibly, able to communicate with supervisors and other workers, able to follow simple directions, able to tell time and monitor time on task, etc. At each level at least one expert content specialist, one expert teacher, and one potential teacher should review the analysis with the

---

[1]If the instructor is not available or the instruction is to be designed for more than one instructor, random samples should be taken of potential instructors to determine what level of detail is needed. When in doubt, designers should select more, not less detail/levels.

designer and modifications should be made, if necessary, before the next level is analyzed.

The outcome of the data flow diagram design technique is a graphical representation of the instructional system at as many levels as desired by the teacher. Each level graph shows the major information processes involved, the information and knowledge necessary for these processes, the interrelationships among the processes, a potential sequencing of the processes, and points where the student is likely to need expert help in mastering the instruction.

A checklist of this researcher's instructional design process using data flow diagrams is presented in Appendix D.

Data Flow Diagrams in Management Information Systems

Using data flow diagrams to analyze systems is not unique to instructional design. It addresses the broader problem of analyzing a process or system to determine what is needed to manage information. People involved with management and Management Information Systems (MIS) have been analyzing systems since the early part of this century. Recent emphasis in MIS has focused on the data to be collected and the information to be manipulated (Lord, 1983). The data flow diagram is one of the most commonly used analysis tools for this task (Davis & Olson, 1985). Data flow diagrams have proven effective in industry to emphasize the sequence, connectivity, and type of data and information

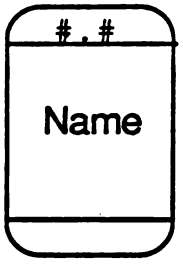
utilized by the system. Therefore it would appear that they have potential to enhance the instructional design process as well.

Gane and Sarson (1979) provide a detailed explanation of data flow diagrams and how they are used in analyzing information processing systems. The following description summarizes the technique, showing the basic symbolism, with the intention of providing only sufficient information for the reader to understand how data flow diagrams might serve as an instructional design tool.

Data flow diagrams show the information flow in a system by using four major symbols: (a) a shadowed square to indicate an entity external to the system, (b) a rounded rectangle to show processes, (c) arrows to show the flow of information, and (d) open rectangles to show data stores. [See Figure 4]. When the same symbol is redrawn in a diagram to simplify the diagram, extra bars are placed in the symbol to show that it is repeated. All data in the diagram are represented in a data dictionary that describes the data in detail.

To use the data flow diagram technique in instruction, this researcher modified the meaning of the symbols. The data flow diagrams show the design of the instructional system by using the four major symbols:

(a) a shadowed square to indicate points that can not be completely created in advance of the delivery of the



Label for the Process

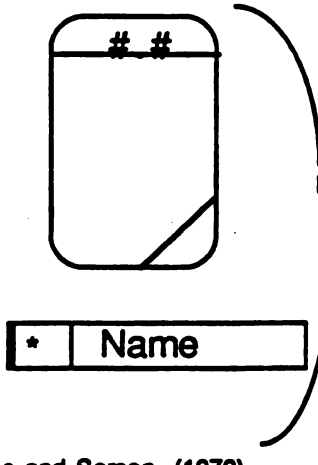
Process, often can be broken downinto subprocesses.

Physical location

Entity external to the system but necessary at least under some circumstances. Can be start and exit points of the system, other systems that contribute, outside consultants, outside evaluations, etc.

Flow of information from one point to another. Label should be explained in detail in data dictionary.

Data store. Could be files, reference materials, computer programs, etc.



An indication that a symbol has been redrawn to make the diagram less complicated. Both symbols represent the same single entity. More lines are added for additional symbols repeating.

Gane and Sarson (1979)

FIGURE 4: Commonly used symbols in data flow diagrams

instruction. For example, a point at which expert, individualized instruction is likely to be required, such as an experienced teacher helping a student find and correct errors in a program the student is attempting to write;

(b) a rounded rectangle to indicate processes, for example, points where the student needs to organize, interpret, or transform information or instruction;

(c) arrows to show the passing of information and instruction between processes; and

(d) rectangles to show stored information, either externally, as in a reference book, or internally as in an acquired strategy.

All information that passes from one processes to another is listed in an instruction dictionary that describes the information in more detail. For example, the diagram may indicate input standards, but the instruction dictionary would specify those standards.

An Example of Data Flow Diagrams in Instructional Design

In the preceding chapter, the development of the vocational Data Processing Curriculum Guide using instructional systems design techniques was discussed. To show how data flow diagrams could be used in instructional design, the same vocational data processing course will be analyzed by this researcher using data flow diagrams. The State of Michigan's Data Processing Curriculum Guide (1983) was used.

The instructional design process given in Appendix D was followed.

In designing the instruction, this researcher performed two other functions: subject matter content expert and expert instructor. This researcher's expertise in these areas is indicated by: (a) certification by the Institute for the Certification of Computer Professionals in Data Processing (CDP), Systems (CSP), and Computer Programming (CCP); (b) eight years of experience teaching vocational data processing; and (c) a nomination for teacher of the year by the researcher's District.

To design the vocational data processing course using data flow diagrams as a tool, first the course goals were established and verified. The goals for this vocational data processing course were (a) to prepare students for entry level positions in the data processing field; (b) to prepare students for post secondary instruction in the data processing field.

Using content matter and instructional expertise, this researcher then determined the course components. In this case, these were: (a) the operation of computers; (b) business applications of the use of the computer, including the entry of data into the computer; (c) the programming of a computer; (d) the role of computers in business and society; (e) the skills necessary to obtain and maintain a job in the data processing industry.

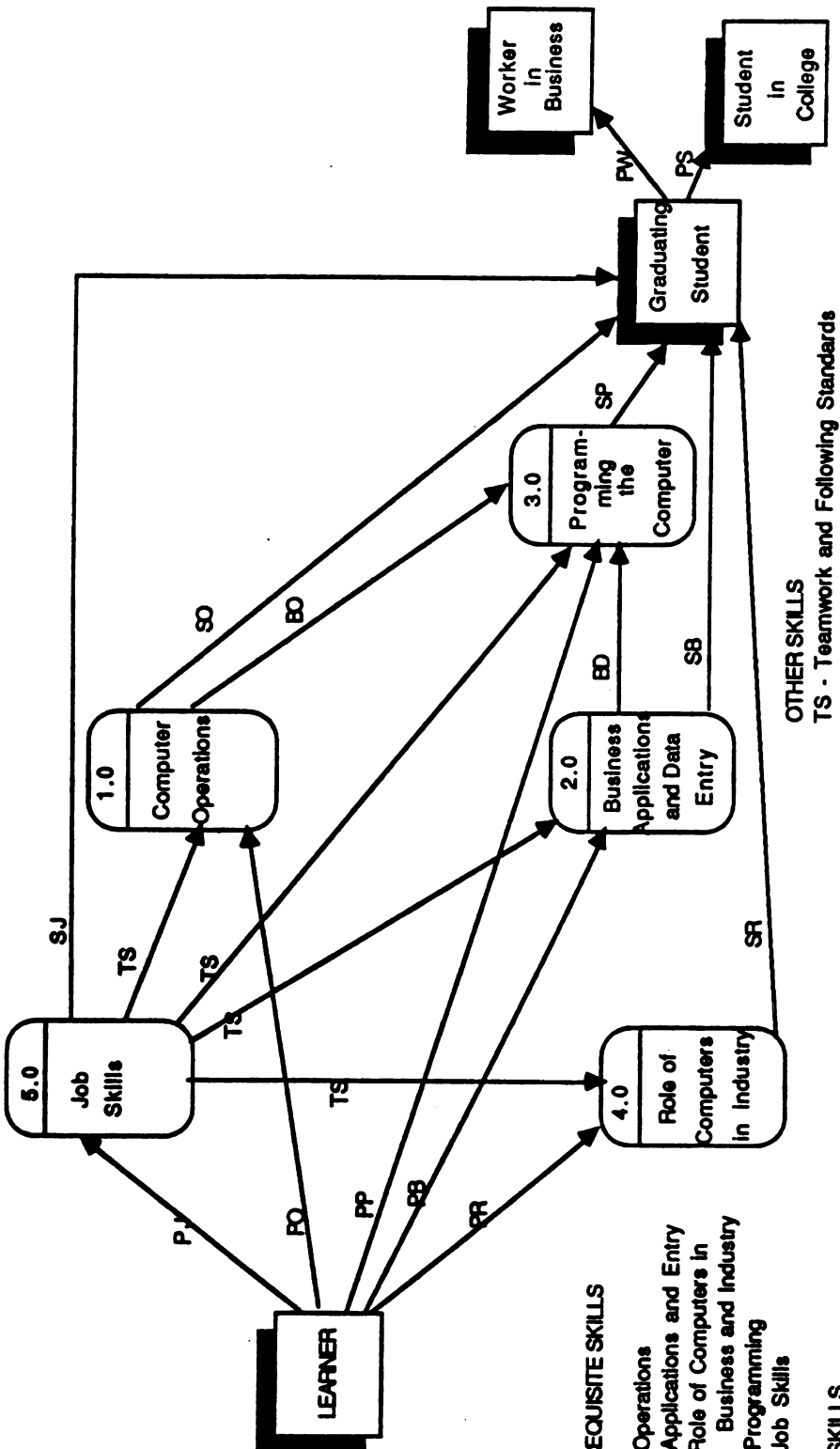


FIGURE 5: Analysis of the Vocational Data Processing Program: An Overview of the System

Once the goals and course components were established, they were represented in a data flow diagram. [See Figure 5]. Note that at this overview design stage, no attempt was made to identify specific information needed or the sources of that information since the diagram would then be so complex that it would no longer be a useful tool. Also note that the numbers used to label the processes served only as a means for linking each process to the more detailed data flow diagram that might be made of that process. Although in MIS the process labels normally designate processing sequence, when this researcher developed the data flow diagrams included in Figures 5 through 29, the numeric process labels were not used to indicate a fixed sequence.

The course overview diagram shown in Figure 5 can be used to show the interrelationships between the course goals and the major course components. For example, by looking only at the major symbols, that is the external entities and the processes, one can see that the major goals of the course are to prepare a student for a job or for higher education. One can also see that there are five major components in the system.

Note that using data flow diagrams, all the information that would be available to the designer using the current technology is still available to the designer. However, notice the additional information that this diagram provides: (a) there are prerequisite skills for each of the

five major components of the course; (b) the job skills of teamwork and following standards are a part of each of the other four components; (c) there are computer operations skills, job skills, programming prerequisites, and business application and data entry skills that are inputs into the programming process; (d) the choice of work or school after completing the course is a matter of personal preference.

To follow the format of the use of data flow diagrams in MIS, a further explanation of the related information in the form of an instructional dictionary should now be developed for each point where information is passed from one item to another on the data flow diagram. For example, the prerequisite programming skill, "understanding elementary algebraic concepts", would be listed in the instructional dictionary as containing the following components: (a) understanding the concept variable; (b) knowing the symbols for and the usage of the relational operators (less than, greater than, equal, greater than or equal, less than or equal, and not equal); (c) knowing the symbols for arithmetic operations (+, -, /,*,), and ^); (d) knowing the rules of order for arithmetic operations; (e) knowing how to solve simple algebraic equations with one unknown; and (f) knowing how to use scientific notation for numbers.

The complete instructional dictionary has not been included in this example for the sake of brevity. In practice, this researcher would include elements in an

instructional dictionary only if the target teachers had some question about the element. For example, most educators interested in teaching computer programming at the secondary level probably have very similar concepts of what "understanding of elementary algebraic concepts" means and the designer would probably not define the term. However, if parents, students, or even counselors wanted clarification, the design tool provides a mechanism for more precise specification, and the designer would be able to add this level of detail to the design without altering the existing structure.

One of the greatest strengths of a data flow diagram design tool is the hierarchical nature of its development. The designer can proceed to the lowest level and specify the smallest fact needed as a part of the instructional package, or can stop at a much higher level if that level is sufficient for the instructional program to be developed and implemented.

The next step in the development process, after the components of the course are identified, is to analyze each of the components of the course and to construct data flow diagrams for each. For the sake of brevity and simplicity, we will focus only on the programming component. Programming was selected since it entails higher order thinking skills of problem solving.

AF - Algorithms and facts
CO - Correction specification
CP - Coded program
CR - Code rules
CS - Coding standards
DA - Detail process algorithms knowledge
DI - Debugging information
DQ - Debugging questions
DS - Detail process strategies
EC - Existing code
EG - Existing graphic representation
EI - Existing input formats
EO - Existing output formats
EP - Existing process specifications
ES - Evaluation criteria subjective
EV - Evaluated program
GR - Graphic representation of process specifications
GS - Graphic standards
IP - Information about problem
IS - Input specifications
JS - JCL statements
LE - Logical error
LR - Language rules
MI - Missing information
NI - Needed Information
OS - Output specifications
PE - Process specification errors
PM - Program error
PP - Problem process definition
PQ - Process specifications
PR - Problem
PS - Problem solving strategies
PT - Processing standards
PW - Program with errors
RF - Rules for operating
RI - Rules for input formats
RO - Rules for output formats
SC - Specific evaluation criteria
SP - Specific processes
TD - Test data
TP - Tested program
UP - Unknown process

Figure 6: Legend

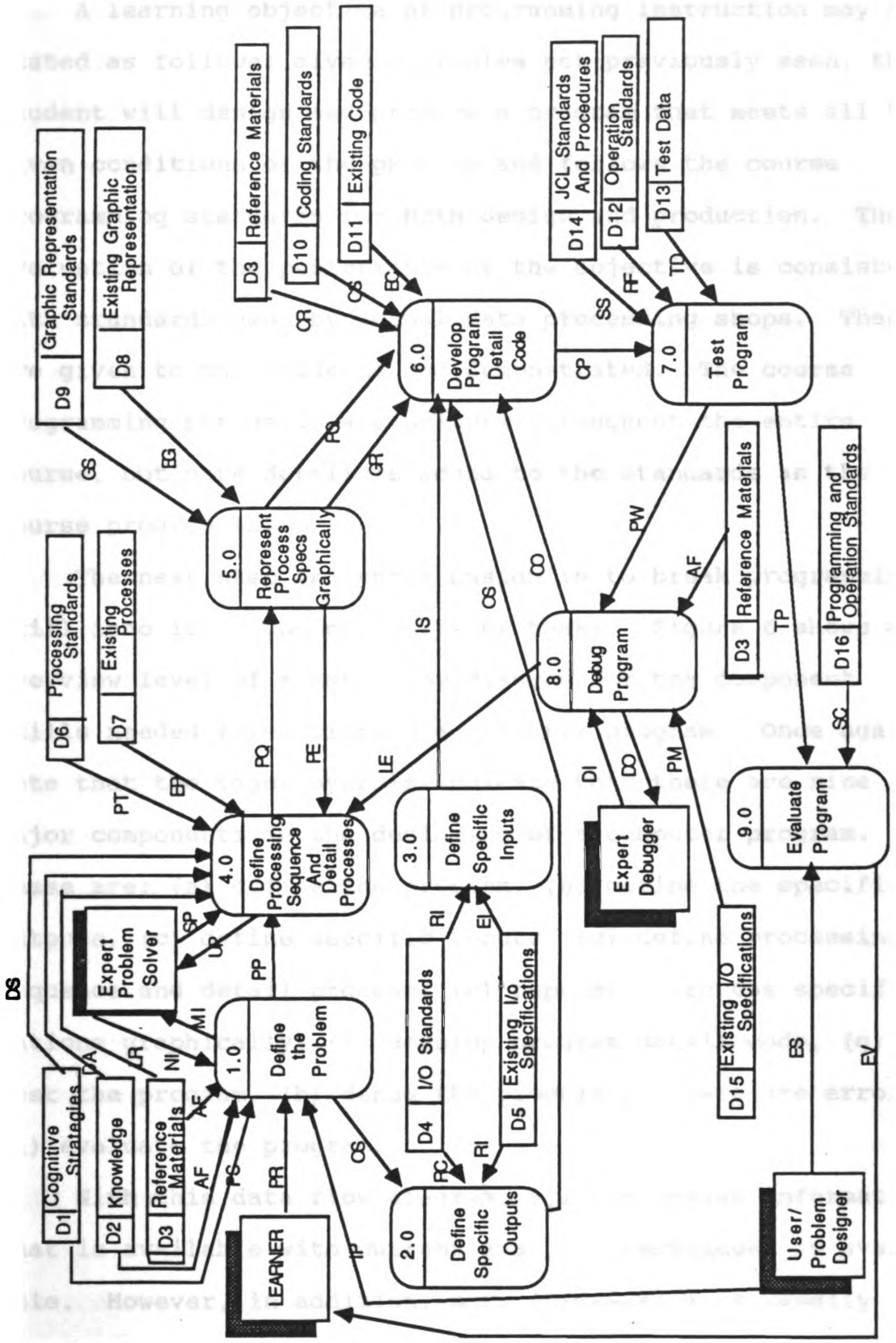


FIGURE 6: Designing a Computer Program

A learning objective of programming instruction may be stated as follows: given a problem not previously seen, the student will design and produce a program that meets all the given conditions of the problem and follows the course programming standards for both design and production. The evaluation of the performance of the objective is consistent with standards used by actual data processing shops. These are given to the students and demonstrated. The course programming standards are uniform throughout the entire course, but more detail is added to the standards as the course progresses.

The next step in system design is to break programming skill into its component parts or tasks. Figure 6 shows an overview level of a data flow diagram for the component skills needed for designing a computer program. Once again note that the major symbols indicate that there are nine major components in the designing of a computer program. These are: (a) define the problem, (b) define the specific outputs, (c) define specific inputs, (d) define processing sequence and detail process, (e) represent process specifications graphically, (f) develop program detail code, (g) test the program, (h) debug the program if there are errors, (i) evaluate the program.

With this data flow diagram, all the design information that is available with current analysis techniques is available. However, in addition, more information is readily

available. For example, there are 16 different types of information the programmer will need, some more than once. Expert problem solver, expert debugger, and user/problem designer are entities external to the programmer's problem solving process that contribute to the programmer's design and development of the program. As can be seen from the numerous arrows between processes, some of the processes are interrelated. For example, the arrows from 2.0, 3.0, 5.0, and 8.0 respectively pass to process 6.0 the information: output specifications, input specifications, process specifications and graphic representation of process specifications, and correction specifications. In addition processes are interrelated in that several information stores provide information to more than one processes. For example, D4 and D5 both provide information to processes 2.0 and 3.0; and D3 provides information to 1.0, 6.0, and 8.0. The data flow diagram also shows that the program design process is not linear, but contains loops as shown by arrows from process 4.0 to process 5.0 and from 5.0 to 4.0 as well as the arrow from 8.0 to 4.0 and 8.0 to 6.0.

Although the additional information provided by the overview of the instruction might be enough so that the instruction could be implemented without further analysis, this is unlikely. This researcher would anticipate the need to provide at least one more level of diagrams to clarify the instructional needs for each of the major processes.

AA - Algorithims

DP - Problem process definition

IC - Information about special conditions

II - Information about input available & desired

IN - Information

IP - Information about problem

IS - Input strategies

IT - Information about process

MI - Missing information

NI - Needed information

OO - Information about output

OS - Output strategies

PA - Process algorithims

PD - Problem input definition

PO - Problem output definition

PP - Purpose of program

PR - Problem

PS - Processing strategies

RR - IPO rules

SP - Problem Strategies

Figure 7: Legend

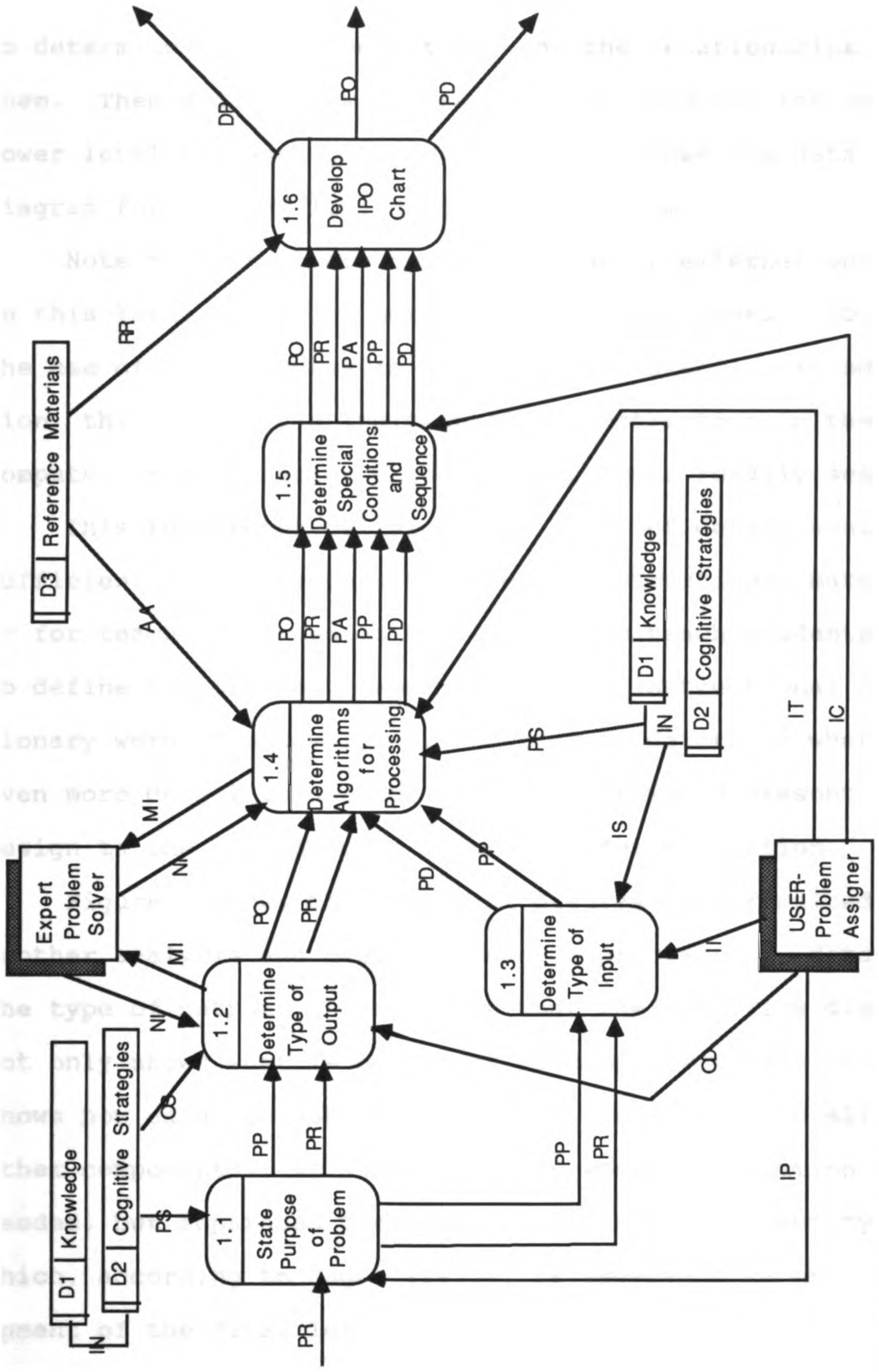


FIGURE 7: DEFINE THE PROBLEM 1.0

Each of the processes shown at this level is analyzed to determine the component tasks and the relationships among them. Then a data flow diagram is developed for the next lower level for each process. Figure 7 shows the data flow diagram for process 1.0, define the problem.

Note that there are no data stores or external entities on this level that were not on the previous level. However, the use of those stores is shown in more detail. In addition, the role of the individualized instruction in the computer program design process can be more readily seen.

This researcher believes this level of detail would be sufficient for developers to develop instructional materials or for teachers to design the lessons to teach students how to define the problem, especially if an instructional dictionary were also developed. However to determine whether even more detail is needed, the designer would present the design to the developer or instructor for evaluation.

Figure 8 shows the data flow diagram that resulted from another analysis and design level for process 1.2, determine the type of output. Again, note that the data flow diagram not only shows what is necessary to complete a task, it also shows how each subtask, or component, is related to all the other components. It shows not only where information is needed, but separates the information into different types, which, according to Gagne (1977), is important to the development of the final lesson.

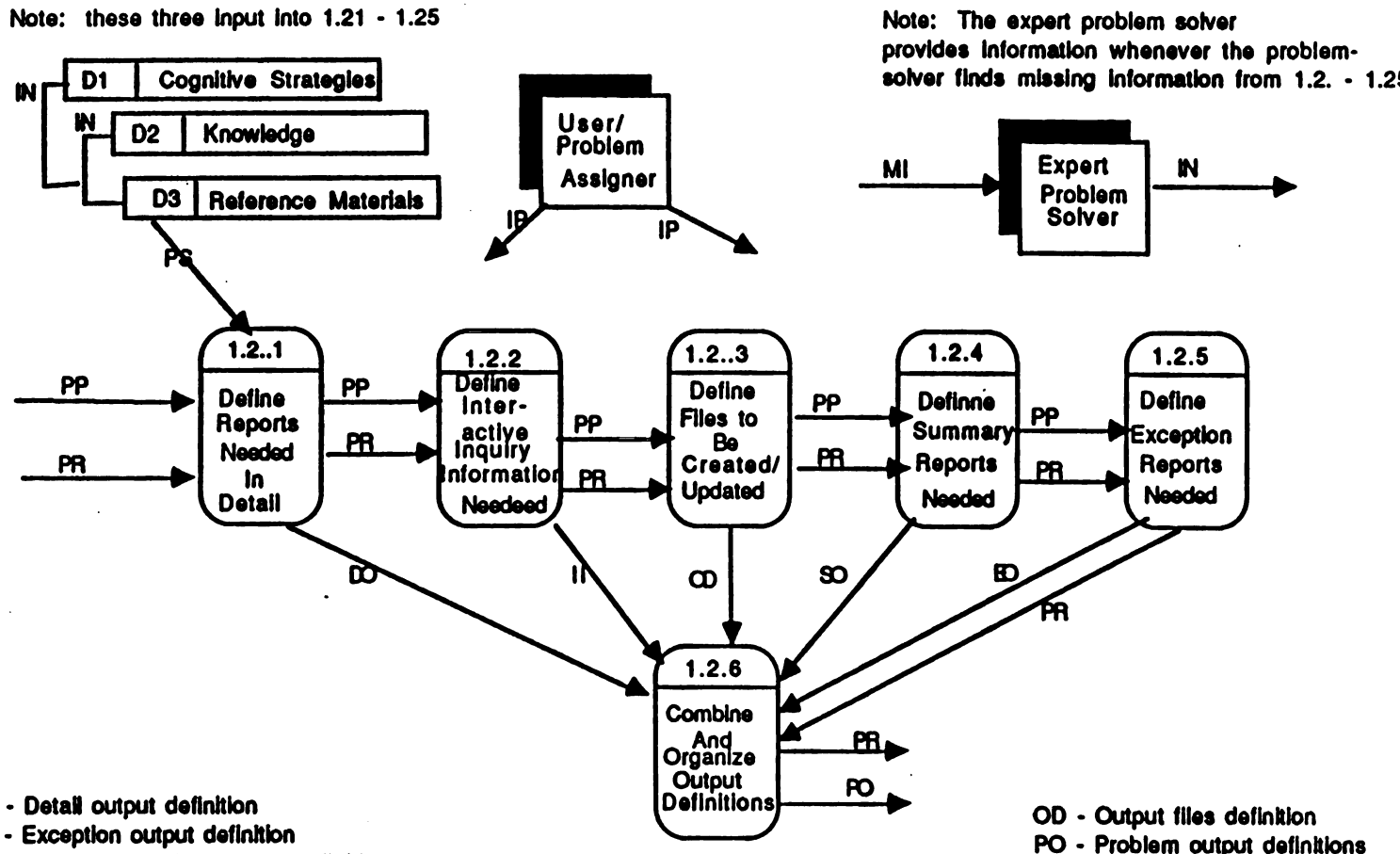


FIGURE 8: Determine type of output 1.2

This level-by-level analysis and design should continue until the point where the person who actually develops or implements the lesson for the task has as much information as is needed to create or teach the lesson. The set of data flow diagrams developed for experienced teachers would probably not need to show as many levels as a set of data flow diagrams developed for inexperienced teachers. If the experience level of the target teachers is not known or the instruction is designed for teachers of varying degrees of experience, the instruction should be designed to the lowest level that may be needed by the target teachers.

Appendix B contains the complete example of using data flow diagrams to analyze and design the programming portion of the vocational data processing class to the second level.

Using data flow diagrams to Design Interactive Video Courseware

The preceding sections of the chapter demonstrated how data flow diagrams could provide useful information when the technique was used to design a course including problem solving in computer programming. Data flow diagrams could also provide useful information when used to design interactive video courseware. Figure 9 shows the first level data flow diagram this researcher's design group developed for an interactive video course for teaching the role of pain and its control in veterinary medicine. Although the course was

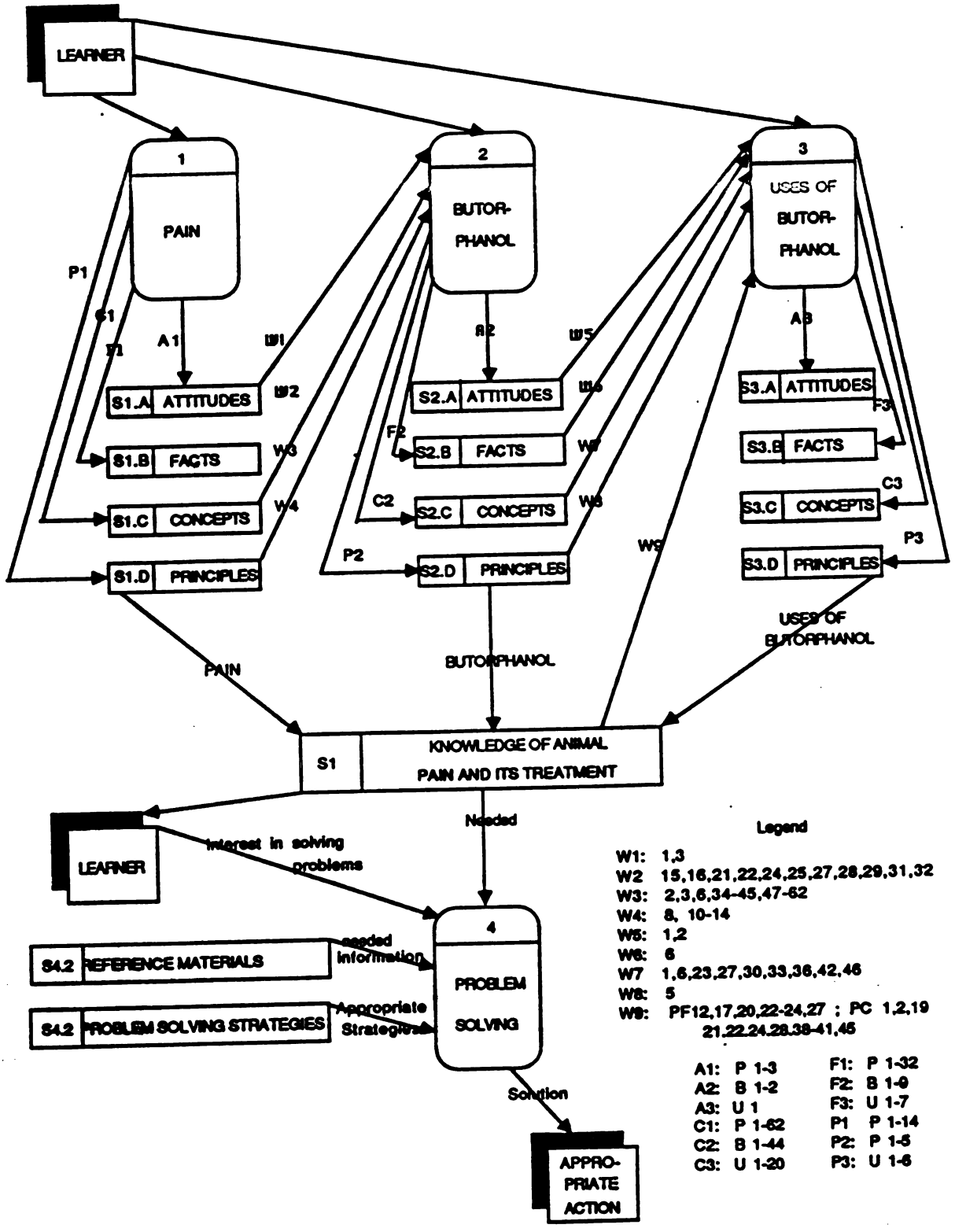


Figure 9: METHOD OF CONTROLLING PAIN IN VETERINARY MEDICINE

still under development at the time of the writing of this dissertation, the data flow diagrams had been presented to the client who not only approved of their accuracy, but indicated approval of the level-by-level design approach by saying "I can see the whole course now, layer by layer," and "I knew where I wanted to go with this material, but now I feel as if I have been given a road map showing me how to get there". In addition, the menus that will be used to control the presentations have been developed from the data flow diagrams and an estimate of the amount of time necessary to the complete the project has been make.

Note the different kinds of information that are provided at the first level data flow diagram. [See Figure 9]. The diagram shows that learners can enter the course at any of the four major course processes. The first three processes are designed to add to the learner's knowledge base, while the fourth process requires that the learners <u>use</u> their knowledge bases, along with cognitive strategies and reference materials, to solve problems. The information that the learners will process in order to add to the their knowledge bases is separated into attitudes, facts, concepts, and principles since each of these major types of information needs to be presented to learners using different instructional techniques (Gagne, 1985; Gropper, 1983; Landa, 1982; Reigeluth & Stein, 1983; Scandura, 1983). If the course is to have learner control of the pace and

sequence, the instruction will need to provide learners access to information from the previous sections. In the first level data flow diagram shown in Figure 9, the information that is presented in one section and then used to build the knowledge in another section is shown by the connecting arrows. For example, note that arrow W4 between S1.D, stored principles from process 1, and process 2, shows that principles 8, 10, 11, 12, 13, and 14 are presented in the process 1, PAIN, but the learner will also need those principles to process information presented in process 2, BUTORPHANOL. By using this data flow diagram, the developer of the instruction knows where the courseware must be able to either branch back to previous information or provide the previously presented information in a different format. The principles represented by W4 are listed in the instructional dictionary for the course in Appendix E.

Since information provided by the first level data flow diagram includes the number of major processes, the number of instructional events of each type, and the number of links between sections that must be provided, it can be used by the instructional designer to make rough estimates of the cost and time necessary to develop the course. However, sophisticated estimates of development costs can not be made until the content expert provides information as to the importance and difficulty of each event in the instructional dictionary.

Appendix E contains the data flow diagrams for the first section of the course, Pain, and the instructional dictionary for the section. When the course is authored, each of the processes from the data flow diagrams will become a menu selection for the course. Appendix F contains the procedures developed by the CAI group at Michigan State University for using data flow diagrams to develop interactive video courseware. The complete process, including the relationship between the instructional dictionary and the developmental worksheets is being readied for separate publication and will be available in the future.

Advantages of the Use of data flow diagrams

The preceding chapter presented several problems with the current instructional design techniques. In the preceding sections of this chapter, data flow diagrams were discussed and a demonstration was made of their use in instructional design and analysis. In this section we will focus on the advantages of using this technique.

Data flow diagrams help locate the points in the learning process where help may be needed. In the data flow diagram developed for computer programming, three external entities, expert problem solver, expert debugger, and user/-problem designer, contributed to the process of the design and development of the program that solves the problem. These entities indicated the points of the process where

students often need help when they are learning to program computers. Even an inexperienced teacher using this data flow diagram would be able to determine where special effort may be required to help students learn to program properly. By using symbols to designate information sources within the instructional system that are different from the symbols used for information sources outside the system, data flow diagrams make it easy to see points where teachers should consider making expert help available for their students.

Data flow diagrams show the type and position of information needed for processes within the instruction. Since the data flow diagrams provide graphic indications of the points within the instruction where information is needed and, if necessary, the instructional dictionary shows specifically what is needed, it is easier for teachers to provide students with the information during the instructional process instead of requiring students to know the information before they attempt the task. In addition, teachers have a concrete way of determining what media would be most appropriate for presenting the information to the students. The potential of existing instructional packages to meet students' instructional needs could be measured by comparing the information the packages deliver to the information the instructional dictionary specified. Hamreus (1968) stated that if the instructional technologist is to get maximum use from media in improving learning outcomes, he must be able

to determine how, what, and when media can most effectively be employed. By focusing on the information and sequencing, data flow diagrams can help designers and teachers determine where some of the teaching might be relegated to self-paced, programmed, and computer aided instruction.

Data flow diagrams aid in analyzing the cognitive skill to be taught. By stressing which data, information, and strategies are needed at different points of the process, data flow diagrams help the instructional designer focus on all the information and interrelationships as a complete skill. This allows the designer, developer, and instructor to work at a higher cognitive level. Compare the information about the programming process provided by the data flow diagram in Figure 6 with the task list on computer programming provided in Appendix A. Using the data flow diagram, even inexperienced teachers could see the part each of the tasks in computer programming played in making up the complete cognitive skill, yet they could also see the entire, complete task as an organized whole. Thus they would be more likely to present the instruction to their students as a complete cognitive processes, not merely a collection of behaviorally oriented tasks.

Derry (1984) pointed out that in teaching cognitive strategies, there must be a taxonomy of curriculum-relevant component strategies. The data flow diagram can be a tool to help the instructional designer see the taxonomy of

strategies and also see where those strategies are needed in the instructional program. Therefore the probability is increased that those strategies will be correctly addressed by the instructional program and will be developed by the student.

Higher level thinking skills are represented by data flow diagrams by focusing on goals, information, and needed expertise instead of focusing on specific tasks that make up the skill. By examining the different levels of the process, the instructional designer can see where cognitive strategies or diagnostic knowledge are required.

Data flow diagrams show the interrelationships among topics and processes of instruction. When data flow diagrams are used to design the instruction, even inexperienced teachers can become more aware of sequencing alternatives in the presentation of the components. In the discussion of Figure 6, the specific ways that the data flow diagram showed the interrelationships among the topics and processes of instruction were indicated. Since the graphical representation of the instruction system produced with data flow diagrams contains lines showing the passing of information between components, it is easy to see interrelationships among components of the system, to identify components that require the same information, and to note any necessary sequencing of components.

Data flow diagrams provide a convenient way to explain a course to interested, content-area-naive, parties. One of the strengths of data flow diagrams stems from the graphic nature of its representation. Polhemus (1987) states that the use of graphics provides a very efficient medium for exchange of information because graphics are easily explained, have strong impact, and permit multiple dimensions to be displayed concurrently.

Using data flow diagrams to represent a system graphically allows even a relatively inexperienced person to recognize the critical elements of the instructional design. Thus more of the stakeholders in the instructional program could have a better understanding of courses designed using data flow diagrams. Since the design provides multiple levels of representation of the system, each level provides an advanced organizer for the following level. In addition, since data flow diagrams show one complete level of a system at a time, an examination of the system could range from looking at the first level for an overview of the entire system to carefully analyzing the lowest level of a module for a detailed analysis of the components, depending on the needs or desires of the person using the data flow diagrams.

Summary

In this chapter the purpose, conditions, methods, and outcomes of instructional analysis and design using data

flow diagrams were discussed. The use of data flow diagrams in Management Information Systems was reviewed. The instructional analysis and design of a vocational data processing course using data flow diagrams was developed using the techniques that were presented in Appendix D. Where appropriate, the results were compared to the results from Chapter 2. Figure 10 summarizes the major differences. An example was presented of an instructional design using data flow diagrams for an interactive video course in veterinary medicine. The response of the client and the strengths of the technique were discussed. Finally, the advantages of the use of data flow diagrams in instructional analysis and design were discussed. The following advantages were noted: (a) the location of the expertise needed by students to learn higher order thinking skills is presented; (b) the type and position of information needed for each process within the instructional system is indicated; (c) the components of the cognitive processes for higher order thinking skills are presented as a whole; (d) the interrelationships among topics and processes of instruction are graphically shown at as many levels as are needed; and (e) a convenient way to explain a course to interested, content-area-naive, parties is available.

Thus, using data flow diagrams to analyze and design an instructional process provides more information in a readily usable form than the current systems design technology. In

addition, since the symbols used to designate information sources within the system are different from the symbols used to designate information sources outside the system, data flow diagrams also indicate points where expert systems could be of use to deliver needed information. Expert systems will be discussed in the next chapter.

Information about the teaching of programming provided by the Curriculum Guide	Information about the teaching of programming provided by data Flow diagrams
tasks that make up programming	the processes that make up programming
	the points within the instructional program where students may need expert help
	the type and location of information needed within the instructional program
	a presentation of the cognitive skill as an organized whole
	the interrelationships among the processes
	a convenient way to explain the course to content-area-naive stake holders

Figure 10: Curriculum Guide compared to Data Flow Diagrams

CHAPTER 4

EXPERT SYSTEMS

Introduction

In the preceding chapter the use of data flow diagrams for instructional design was discussed. It was demonstrated that data flow diagrams not only emphasize information, processes, and interrelationships, but also show points in the instructional program where individualized expert help may be needed by typical students. In this chapter the use of expert systems to provide that help is discussed, followed by a discussion of some concerns educators might have when considering the use of expert systems within instructional programs.

First, expert systems are defined and their use by business and industry is discussed. Next, the limitations of expert systems are discussed, followed by a discussion of some reasons for developing expert systems, despite their limitations. Examples are presented of expert systems that have proven to be valuable within instructional programs. Then the theoretical potential for the use of expert systems in education is discussed, followed by a discussion of steps involved in developing expert systems in both business and education. Finally, an example of the development of an

expert system within a vocational data processing program is discussed.

Definitions

Expert systems are computer programs that make the expertise of an experienced worker available to a novice. An expert system consists of three parts: (a) a knowledge base composed of production rules, semantic networks, frames, relational database, and inference rules; (b) an inference engine, which consists of the procedures that generate the consequences, conclusions or decisions from the existing knowledge; and (c) a working memory, which is a temporary storage area to hold the immediately relevant portions of the knowledge base, the user's responses, and the current inference. An expert system must also have an user interface, a way of communicating with the user (Boose, 1986; Hayes-Roth, Waterman, & Lenat, 1983).

One of the simplest types of expert systems to develop is the production-rule system. F. Hayes-Roth (1985) cites many advantages to using rule-based systems. First, since they incorporate practical human knowledge in conditional if-then rules, it is possible to develop production-rule systems to help supply novices with the expertise of experienced workers in any situation where the expertise could be expressed by algorithms. Second, since the skill necessary to design production-rule systems increases in direct

proportion to the size of the knowledge bases, relatively inexperienced programmers could develop small systems containing the most critical rules and then add the more complex rules as their programming skills improve and they gain more knowledge about the content area. Third, by determining the best sequence of rules to execute, selecting relevant rules, and then combining the results in appropriate ways, production-rule systems could solve a wide range of complex problems. In addition, production-rule systems could help the novice understand the rules the expert used by explaining their conclusions, retracing their lines of reasoning, and translating the logic of each rule employed into natural language.

Since production-rule expert systems are relatively simple to develop, but still have many advantages, the development of this kind of expert system should be considered in situations where the expertise needed could be expressed in a production-rule format.

Classes of Expert Systems

When considering the development of expert systems, it is important to realize that not all expert systems are designed to do the same type of tasks. Expert systems are applicable to three generic classes of problems: analysis, synthesis, and a combination of analysis and synthesis (Hayes-Roth, Waterman, & Lenat, 1983).

The simplest class consists of analysis problems. Analysis problems require classifying a set of objects based on their features. Typical expert systems designed to solve analysis problems classify items, provide help in debugging processes, aid in deriving diagnosis, identify unknown items, or aid in the interpretation of output from some system. Currently most of the cost effective expert systems used by businesses are designed to solve analysis problems (McCorduck, 1987).

More complex than analysis problems are problems of synthesis. Synthesis problems are generative problems in which a solution must be built up from component parts. Typical expert systems designed to solve synthesis problems determine the configuration of systems, design systems, and develop plans for courses of action. Although production-rule expert systems that are capable of forward chaining have been developed to solve synthesis problems, most production-rule expert systems solve analysis problems.

The most complex problems for which expert systems are used consist of a combination of analysis and synthesis. Typical expert systems designed to solve combination problems are used for command and control, making management decisions, monitoring complex physical processes, predicting future events based on current and past events, and determining components needing repair.

In developing expert systems, it is important that the type of expert system chosen to solve the problem matches the class of the problem. For example, if the problem involves synthesis, an expert system that could only do backward chaining, working from the goal toward the given data, could not possibly solve the problem. A general heuristic for planning the development of an expert system is that the more complex the class of problem, the more resources will be required to develop the expert system.

Limitations of Expert Systems

The discussion of the limitations of expert systems in this chapter is not designed to be either detailed or exhaustive. Rather the intention is to provide some insight into the problems that may occur with the use of expert systems. A detailed discussion of the limitations of expert systems and the concerns that potential developers of expert systems should consider is provided by Hayes-Roth, Waterman, and Lenat (1983).

To be effective, expert systems must contain a substantial amount of domain expertise organized for efficient search (Klahr & Waterman, 1986). To model the thinking of experts, expert systems must be able to use heuristics, that is, rule-of-thumb knowledge that enables experts to make educated guesses when necessary, and to deal with incomplete or inconsistent information (Boose, 1986).

Expert systems must be able to contain the problem-solving knowledge of the expert, whatever its form or degree of complexity (Hayes-Roth, 1985). For example, doctors doing a diagnosis of a patient must not only classify symptoms of the patient, but must also infer the most probable cause of those symptoms. Experts in the area of finance must not only know what types of investments are available, they must also have preferred strategies for reducing the risk of the investment. They must know how to eliminate the uncertainty of the effects of future events. Expert repair persons must not only know how to repair the product, they must also know the likeliest places to look for relevant information about the cause of the problem.

In general, experts in all fields must observe events and make specific inferences based on the observations. In addition they make abstractions from which they form generalizations and categorizations (Hayes-Roth, 1985). Larkin, McDermott, D. Simon, and H. Simon (1980) provide an example from solving algebra problems. Suppose the problem

> A board was sawed into two pieces. One piece was one-third as long as the whole board. It was exceeded in length by the second piece by 4 feet. How long was the board before it was cut (p. 207)?

was given to both novice and expert algebraic problem solvers. The typical novice sets up the equation $x/3 + (x/3 + 4) = x$ and solves for the value of x. However, the expert problem solver almost immediately states that the

answer is 12. If pressed to explain the reasoning used, the expert will note that if one piece was 1/3 of the board, the other piece 2/3 of the board. Since the second piece, 2/3 of the board, was the length of the first (1/3 of the board) plus 4 feet, 1/3 third of the board was 4 feet long and the entire board was 3 times 4 or 12 feet long. Thus by making abstractions, generalizations, and categorizations instead of following the "rules for solving word problems", the expert could almost instantly "see" the answer while the novice was still attempting to express the problem using mathematical symbols. Experts used the formal "rules for solving word problems" only when they could not "see" relationships within the problem that readily lead to the solution. Thus, experts not only know many ways to reach a desired goal such as the solution to an algebraic problem, they also know which ways are necessary and sufficient conditions for goal achievement under the given conditions (Hayes-Roth, 1985).

One of the most complex types of knowledge that experts possess is the knowledge of the limitations of their knowledge (Hayes-Roth, 1985). For example, expert doctors not only know how to make diagnosis and inference of probable cause, they also know when to tell the patient to seek the advice of a specialist, an even more knowledgeable expert.

From the preceding discussion it could be seen that the expertise possessed by experts could represent a wide range

of complexity of many different types of knowledge. As noted by B. Hayes-Roth (1985), the difficulty of modeling many of those forms of knowledge within an expert system is obvious.

In addition to difficulty of modeling many types of knowledge possessed by experts, most expert systems have other limitations (Hayes-Roth, Waterman, & Lenat, 1983). Although many expert systems could list the rules used by the system to reach the solution, most could not explain their results in a manner suited to the particular audience using the expert system. The novice then could not get the same benefit from the answer provided by the expert system that could be obtained from a human expert-even if they both provided identical solutions. This may limit the usefulness of the expert system as a training tool.

Also, human experts learn from their experiences. Most expert systems could not learn from the problems they solve, stretch their own rules, or restructure their knowledge (Hayes-Roth, Waterman, & Lenat, 1983). Thus, most expert systems could not improve their performance. To improve the performance of the expert system, a human expert must determine what changes must be made and the expert system programmer must then modify the system. Perhaps the greatest limitation of expert systems stems from their lack of self knowledge (Boose, 1986). Expert systems are not able to determine the relevance of their knowledge to the current

problem and they can not degrade gracefully at problem boundaries. Thus it is quite likely that a novice, giving an expert system a problem that was not the type for which the system was developed, might still get a plausible solution that was very much in error. This could, of course, cause many problems for the novice.

The final limitation to be considered stems from the inability of expert systems to engage in the higher levels of reasoning (Boose, 1986). Expert systems could not reason about space and time, or reason from underlying problem domain principles, or reason analogically. These higher order reasoning abilities are currently only produced by human experts.

Reasons for Developing Expert Systems and Examples

Despite their limitations, there are many situations in which it is useful to develop expert systems (Boose, 1986). Within the business environment, expert systems may be developed to help the replacement worker when experts retire, taking their knowledge with them. In some areas, such as determining the best configuration for a computer network, expertise may be scarce or not available. Often when experts are available, their expertise may be very expensive to obtain. When expertise is scarce, an expert system may help to deliver a product or service in a more timely manner than would be done by a human expert, because the human

expert is not always immediately available. One of the most common reasons for developing an expert system is that experts, being human beings with human limitations, are not always consistent (Boose, 1986).

McCorduck (1987) has researched the current uses and effectiveness of expert systems in business. She visited companies in both Japan and the United States and only considered systems that were actually being used. She cited numerous examples of large productivity gains or cost reductions effected by the use of expert systems. She summarized the information she had gathered in six good reasons for developing and implementing expert systems:

1. Expert systems could be developed for codification, replication and distribution of expertise. If the expertise of a key person is captured in an expert system, the reasoning is coded in a systematic manner that will give consistently identical output for identical input. The expert's reasoning could then be used by more than one decision maker in more than one geographical area.

Examples of this effective use of expert systems are: (a) Nippon Kokan Steel developed an expert system to "clone" their best blast furnace controller and allow inexperienced controllers to use his experience; (b) Dupont offered early retirement to experienced workers to cut costs only to discover that they needed their expertise! The experts were brought back and expert systems were developed to not only

recapture that lost expertise, but also to increase its continuity in the company; (c) Texas Instrument was able to keep an old and "temperamental" machine running by developing an expert system to help the repair persons quickly to repair the machine when it broke down.

2. Expert systems could be developed for combining expertise from many sources. It is not uncommon for business decisions to require input from more than one expert. If an expert system is developed to aid in the decision making, it could include the expertise of more than one expert in both its knowledge base and its inference engine. Thus the combination of the experts' reasoning could be used without the difficulty of assembling all the experts in one place.

Examples of this effective use of expert systems: (a) Digital Equipment Corporation developed the EXpert CONfiguration system EXCON to determine computer configurations. They also used EXCEL to help salespersons to present the product more effectively. DEC estimated that the use of these two systems provided an annual savings of $25,000,000. (b) Kajima Construction Co. developed an expert system to determine how to sink the pylons for their construction projects. The expert system not only saves $200,000 per year, but also improves the quality of the work.

3. Expert systems could be developed to amplify expertise and thus produce dramatic gains in productivity or in

the creation of new tasks, products, and services. Almost all experts in business are called upon to provide expertise at a variety of levels. If an expert system could be built to provide expertise at some lower levels, a resident expert might have more time to use his expertise at a higher levels. This, in turn would increase the expert's productivity and allow more time for work with new developments.

Examples of this effective use of expert systems: (a) Cannon used expert systems to set up simulation runs to test new camera lens. As a result, one person now could do in two weeks the design testing that in the past took several experts months of work. (b) Arthur Anderson developed expert systems to help with the auditing of financial systems, allowing high level consultants to aid in more consultations. (c) American Express developed an expert system that could authorize credit limits in under 90 seconds based on information in 13 different data basses. It is estimated that the use of this expert system saved American Express millions of dollars in credit card fraud each year and also reduced the time pressure on workers when making credit decisions.

4. Expert systems could be developed to avoid significant costs in wasted resources and capital. Texas Instruments developed an expert system to help engineers prepare presentations for documenting capital investment procedures. This new system took one twelfth of the time of the previous

system and saved the company an estimated 500,000 dollars annually.

5. Expert systems could be developed to generate complementary streams of revenue. In Japan, a bank developed an expert system for investment portfolios that analyzed the customer profile and recommended investments based not only on current and projected market trends, but also on the personal philosophy of the investor. The pleased investors spoke so highly of the investment service that customers without investment portfolios started using the bank because it was perceived as modern and good. Three hundred thousand yen was added to the average amount invested with the bank annually.

6. Expert systems could be developed to solve a variety of problems that are mainly tractable. Dupont developed a system that helped local managers to determine how to pay taxes across different geographical regions. Because the tax codes are readily available, this was not a difficult task, but was so time consuming that managers often took the easiest options, even when other options might have been more cost effective. With the expert system, it is estimated that the corporation saved $2,000,000 per year. IBM developed an expert system that greatly increased the productivity of their engineers by helping them complete the documentation required. Another expert system used by IBM, which had a very high payoff in cost savings, helped move

capital equipment from one geographical region to another. These expert systems solved problems that were more a matter of expertise in knowing what was wanted than expertise in higher order thinking skills. Such tractable problems tend to consistently have high returns on investment when expert systems are developed to solve them.

The use of expert systems by business continues to grow. Coopers & Lybrand of New York recently surveyed the Nation's leading insurance companies and found that 65 per cent of the companies are either currently using or are developing expert systems, twice as many as were using expert systems a year ago. David Shpilberg, their head of Decision Support Services, says "insurers are realizing that expert systems technology offers a way to harness knowledge efficiently in an industry that is extremely knowledge-intensive" ("Insurance embraces", 1988, p. 39).

The reasons for developing expert systems in the business environment also apply for developing expert systems within the educational environment. Yazzdani (1987) pointed out that after an expert system has been developed, the line of reasoning, knowledge, and organization of the expert could made available for observation by trainees. In school, the expertise of the teacher is continually available, but it must be allocated among many students within a limited time period. By providing students another source of expertise, the computer program, not only is the

available time teachers have to work with students, a scarce resource, increased, but students have access to the expertise whenever they need it during the learning process.

Examples that Illustrate Use for Expert Systems in Education

As in business, instruction is knowledge intensive, and the development of instructional technology has shown a steady advancement in the level of skills that could be taught using technology. The evolution of present-day teaching programs consists of: (a) Pressy's linear frames (1926), (b) Crowder's scrambled textbooks or branching frames (1962), (c) adaptive systems that branch based on a history of responses, and (d) generative systems that use algorithms to generate problems and answers (Gable & Page, 1980). At present, specialized programs designed specifically for education that use artificial intelligence and expert systems to aid in student acquisition of some elements of an expert's knowledge, could be added to the list.

Currently several intelligent tutorial systems are being used primarily as experimental vehicles in instruction (Sleeman & Brown, 1982). Two examples that use artificial intelligence in the instructional process are SOPHIE, which is an operational intelligent computer aided instructional (ICAI) system designed to provide tutoring in the domain of electronic troubleshooting, and BUGGY, which is an instructional game that helps students and inexperienced teachers

understand the student's misconceptions in mathematics, by analyzing errors the student is making (Suppes, 1979).

SOPHIE provides students with simulations in electronic troubleshooting. Students could enter a board configuration and the measure of energy to be applied to the board into SOPHIE and SOPHIE will tell them what output will result. Students could "experiment" with different board configurations and energy inputs without the risk of damaging the board components and without expending the large amounts of time required physically to construct the boards. In addition, SOPHIE could present students with boards that have "bugs" in them and require the student to determine what component is malfunctioning. Students could use different inputs or even change components on the board as they seek to find the causes of the malfunctions. Students using SOPHIE took less time to gain a higher level of expertise in troubleshooting electronic circuits than students who worked only with actual circuits.

BUGGY trains student teachers to examine surface manifestations of students misconceptions in mathematics and use the results of that examination to determine the deep-structure misconceptions in a student's knowledge. When BUGGY was used with students, the students began to study and understand the underlying procedural skill instead of merely performing the behavioral operation (Brown & Burton, 1978).

Such programs as BUGGY and SOPHIE come closer to instructing students with a degree of the skill of a good teacher than the mere branching capabilities of programed instruction. One of the most important of these skills is the teacher's ability to synthesize an accurate "picture," or model, of a student's misconceptions from the meager evidence inherent in the student's errors. If instructional programs are designed to provide the student with the expert help that teachers normally supply, they will also need to be designed to respond appropriately to errors that students make.

As one example of such an instructional program, Sebrechts, Shooler, LaClaire and Soloway (1987) created a computer-based expert system called GIDE which interprets students' statistical errors. GIDE is a goal-driven diagnostic system that analyzes answers to statistical problems and presents that analysis in a form that is designed to be used by the student to improve performance in working that type of problem. Currently, it could evaluate a student's solution to a statistical problem specifically requiring a repeated-measures T test. Testing showed that the program was capable of interpreting over 90% of the student solutions and print out a report of the "bugs", that is, errors in computation or solution approach. In comparing the expert system to the work of teaching assistants, the GIDE program was found to do a consistently better job than the

T.A.'s in identifying all the areas of error in the solution. Although teaching assistants "interpreted" all the solutions, over 10% of the solutions graded by the teaching assistants fell into the category of "I do not understand what you were attempting to do" as the T.A.'s evaluation.

One of the fastest areas of growth in the development of expert systems is the use of expert system shells. Shells are expert systems for which the inference engine has been designed, but the knowledge base and the rules are left blank so that applications could be developed by entering the domain specific application knowledge base and rules. Lee (1987) reported the use of an expert shell written in the PROLOG computer language to help students learn concepts, think analytically, and test ideas. The shell created a versatile way for the students to explore knowledge about number theory, or other subjects, and encouraged students to learn actively and to think logically. Lee taught the students to program the domain they were exploring using PROLOG and then to see what other information was true, given the information that they placed in the domain. Unfortunately, the fact that the students must have communicated with the expert system using PROLOG decreased the usefulness of the system as an instructional tool, since learning to use this complex language effectively took a substantial amount of time.

Expert systems are also being designed to teach problem solving skills. Many areas addressed by emerging technology in the area of expert systems are those very areas defined as important areas of problem solving (Newell & Simon, 1972). For example, consider this information processing definition of problem solving by Anderson (1985, pp.198-199) which states:

> Problem solving is defined as a behavior directed toward achieving a goal. Problem solving involves decomposing the original goal into subgoals and these into subgoals until subgoals are reached that can be achieved by direct action...Problem solving can be conceived of as a search of a problem space. The problem space consists of physical states or knowledge states that are achievable by the problem solver. The problem-solving task involves finding a sequence of operators to transform the initial state into a goal state, in which the goal is achieved...The working-backward method for problem solving involves breaking a goal into a set of subgoals whose solutions logically imply solutions of the original goal...The knowledge underlying problem solving can be formalized as a set of productions that specify actions that will achieve goals under particular conditions.

Boose (1986); Hayes-Roth, F. Hayes-Roth (1985); and Waterman, & Lenat (1983); all described expert systems that use proposition-rules to make available to novices the domain specific problem-solving knowledge of an expert. Many of these systems use backward chaining and most provide explanations to the user as to the reasoning used by the system. O'Shea and Self (1983) observed that expert systems could be used for imparting knowledge from an expert in a

field to a large number of trainees. Waterman (1986) stated the diagnostic function of a human tutor seems ready to be attacked with the well-established methodology of expert systems.

The Theoretical Potential for the Use of Expert Systems in Education

In addition to the four examples of how expert systems are being used in educational research and a look at how expert systems could be useful in problem solving, it is equally important to note that the theoretical body of knowledge of expert systems also supports their potential use in the instructional environment.

The theoretical body of knowledge of expert systems. Wensley (1987) pointed out that there are two broad functions for expert systems: expert replacement and user support. But the potential function of the expert system as a training tool should not be overlooked. Our current method of training experts, extensive experience under apprenticeship, has not changed in centuries. With expert systems, the experience of an expert is imbedded in the system. Thus the novice should be able to use the system and gain expert experience at an accelerated rate (Abdolmohammadi, 1987; Wensley, 1987). Thus while replacing an expert or providing user support, the expert system could also provide training for a novice to become an expert.

As regards the support function, Wensley suggests the following uses that would be applicable to education: First, expert systems could serve as an "intelligent assistant" to experts to check their conclusions. If the conclusions arrived at by the expert are different from the conclusions produced by the expert system, the human expert is alerted to double check the reasoning by which the conclusion was reached.

Second, expert systems could be used to investigate, expand, and develop expertise. The process of developing an expert system requires that the expert or team of experts analyze their expertise closely and actually determine how that expertise is applied. In the process of developing the expert system, experts often expand their expertise and develop expertise at a higher level. As an added benefit, the experts might be able to spend less time checking their conclusions, leaving them more time to work on more difficult problems.

As a third support function, expert systems could provide users with advice. For example, IBM's expert system that aided in troubleshooting customer complaints provided advice to the troubleshooting consultant as to probable cause and remediation. The expert consultant then used the expert system's recommendation as one of the factors to be considered when giving advice to the customer.

Lastly, expert systems could serve in a support function by being one of a set of learning tools to help develop expertise in a domain. For example, an expert system developed to help students correct errors in a computer program could not only help the student learn to correct the errors, it could also show students the steps that an expert uses when correcting the errors.

In summary, the theoretical body of knowledge of expert systems indicates that a broad function of expert systems is to provide user support. In the discussion of the user support function of expert systems it was noted that they help in the development of expertise, provide checks for conclusions, provide insight into expertise, and provide advice. Note that most of these "user" support functions are similar to the types of support systems that good teachers provide in an educational environment. Thus, the theoretical body of knowledge concerning the use of expert systems supports their potential use in the instructional environment.

The theoretical body of knowledge of learning theory. According to cognitive psychologists, the study of higher order thinking skills must include both the observable actions of students and their cognitive processes (Anderson, 1985; Ausubel, 1968; Mayer, 1987). For example, how students perceive and reason about the subject matter domain in which they are learning must be considered along with what

they are learning. Experts and novices differ in their cognitive processes. Experts do not simply have more knowledge than novices, they structure the knowledge differently and apply their knowledge using different problem-solving strategies (Anderson, 1985). Both Ausubel (1968) and Bruner (1960) stressed the importance of structuring the subject matter in instruction. Keravnou and Johnson (1986) noted that expert systems include the structure of the body of knowledge and the domain strategies. Thus, expert systems not only contain the knowledge of an expert, but also the higher order rules by which that knowledge is manipulated by the expert. Therefore, it is reasonable to attempt to use an expert system within an instructional program to train novices to develop both the expert's knowledge and understanding of the structure of the subject matter.

The body of knowledge of inexperienced problem-solvers is relatively unstructured. As a person gains experience in problem-solving in the knowledge domain, he gradually imposes more structure on this knowledge. The structure reflects the strategies and heuristics which have been demonstrated as effective by experience. On the other hand, experienced and competent problem-solvers have a knowledge structure and a set of compatible problem-solving heuristics. They have structured their factual knowledge in the most efficient and effective way for the subject matter domain.

Therefore, the potential use expert systems in education is supported by the theoretical body of knowledge about learning theory because expert systems contain both the knowledge of a domain expert and the heuristics that a domain expert would use to search the knowledge base for the correct solution to a problem. Thus, a properly designed expert system might enable novices to structure their knowledge bases in a more effective and efficient manner and decrease the amount of experience that a novice would need to become an expert.

Guidelines for Selecting Points for Expert System Usage

Since developing an expert system as a learning tool or for another purpose takes time, requires additional software, and restricts the use of the computers for other purposes, before an attempt is made to develop an expert system, it should be determined that the expertise represented in the expert system fills a need in the complete process for which an expert system could be developed. In industry or business, locating the point in the process where an expert system could be used is not conceptually complex: use and organizational chart to locate an expert within the process and determine whether the expert's contribution could be increased or replaced by an expert system. However, Bosse (1986), presented several points that

should be considered in the actual application of this principle:

1. The end user of the expert system should be an apprentice in the same field who would take three to five years to become an expert. Since expert systems contain the structured knowledge of the expert, they also use the special vocabulary of the domain expert. Since this specialized vocabulary is often meaningless jargon to people with little or no knowledge of the field, it is necessary that the users of the expert system already have some experience with the subject matter domain before using the expert system. Also, if training the novice is not a lengthy process taking years to complete, it would be more cost effective to merely train the novice than to develop and use an expert system as an aid to learning to be an expert.

2. There should be someone who knows how to solve the problem already. Expert systems capture the expertise of an expert. If there is no existing expert, there is no source for obtaining the expertise to place in the system.

3. The problem should not require team effort or more than a single expert who could currently solve the problem. Although expert systems could be developed to hold the expertise of more than one expert, such systems are very complex. A team approach to solving a problem may be used when the problem is so ill-defined that it requires a group effort to determine what is needed to solve the problem. As

a rule, the development of an expert system should not be attempted unless the problem is well defined and sufficiently limited so that a single expert could solve it.

4. The expert should be available to consult with users. Expert systems, even extremely well constructed systems, are computer programs and thus severely limited in their ability to communicate with human beings. As users work with the expert system, their skill will increase in using the system, but there will be many times when the inexperienced user will not be able to determine what input to give to the expert system. Contact with the expert is, therefore, essential, to provide the support that only a human expert could give to a novice.

5. There should be a clear justification, based on monetary, liability, or safety considerations, for developing an expert system. Expert systems should not be built simply because the expert is available. The development of an expert system should meet the same cost-benefit criteria that other projects must meet.

6. The expert should be available for a sufficient period of time during development and testing of the expert system. If the developer of the expert system does not have sufficient access to the expert, the information extracted from the expert will probably be incomplete and improperly structured. In addition, during the testing stages of the

system, there will need to be frequent comparisons of the output of the expert system with the output of the expert.

7. The expert should have a collection of real cases readily available. Such cases are necessary for both the development and testing of the expert system because an expert's expertise often is difficult to capture. If the developer could observe the expert solving real cases, the developer could gain insight into the expert's problem solving tactics. Also, the alpha testing of a system requires many cases to test all aspects of the system.

Chaturvedi (1987) takes a structured approach to the problem of selecting points in processes to develop expert systems. He proposes two broad categories of problems with which expert systems could be used: (a) those that could be solved with an analysis approach, in which a complex problem could be successively broken down into a number of subproblems until the subproblems could be solved directly; and (b) those that could be solved with a synthesis approach, in which the system starts with solutions to subproblems and combines those solutions to get a solution to the complex problem.

Analysis is used for problems such as classification, interpretation, evaluation, assessment or diagnosis. Examples of the types of problems requiring synthesis are design problems and program construction problems. The primary problem with the synthesis approach is that it is

necessary to search through all possible paths, which could become explosively large in number as the number of starting subproblems increases even slightly. Consequently solutions for problems involving synthesis must involve heuristics.

Current expert system technology has had very limited success with solving problems involving synthesis. Such solutions rely very heavily on estimations and the mathematics of probabilities and combinations. Currently there are no mathematical rules for combining the probabilities of events which are not independent. Until there are, there probably will not be many successes with systems requiring synthesis.

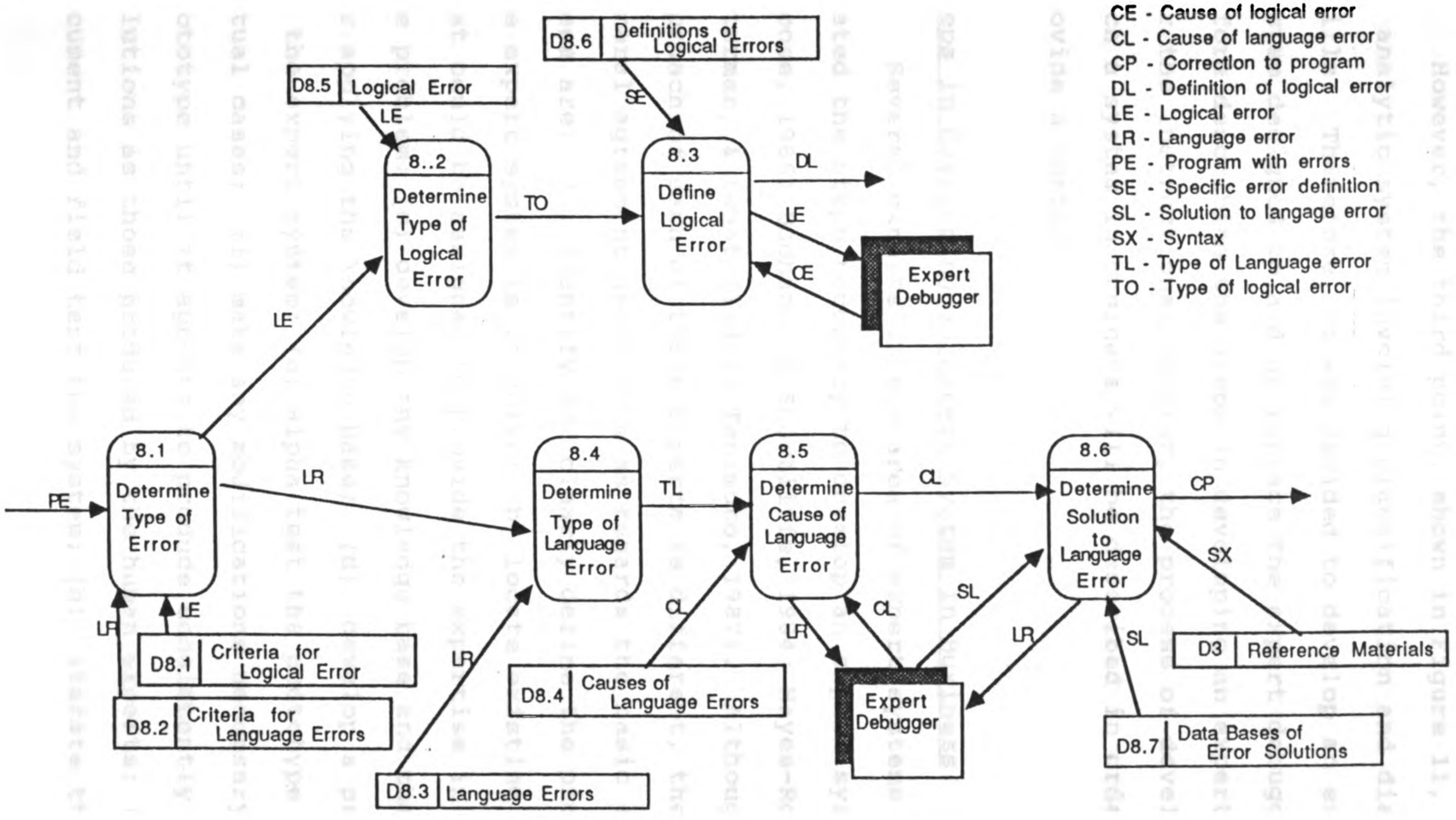
Guidelines for Selecting Points for Expert System Usage in Instruction

Business versus instructional use. Researchers such as Boose (1986), Chaturvedi (1987), and Davis (1987) provide criteria for determining where the use of an expert system would be beneficial within the business environment. However, there is currently little research on where the use of expert systems is appropriate within the instructional environment. In business the function of the expert is to reach some goal. In education the function of teachers, as experts, is to help students reach some goal, not to reach the goal for them. As Perelman (1987) points out, in education the student must do the work. Thus, subject matter

expertise must be integrated into the instructional program to aid students to arrive at solutions, not to provide solutions. Such integration could be done by analyzing the instructional task to determine both where and what type of expertise is needed.

A specific example from vocational data processing. In the previous chapter it was shown how the analysis technique of data flow diagrams might help the instructional designer determine the points within the instructional system at which an expert system might be useful. By looking at successive levels of the analysis, the characteristics of the expert system could be determined.

For example, the overview of the instructional unit for teaching students to design a computer program, [see Figure 6], indicated three points at which an expert might have input into the system: (a) in defining the problem; (b) in defining the processing sequence and detail process; and (c) in debugging the program. The more detailed analysis shown in Figures 7 and 8 indicated that expert systems for the first two points would be synthesis systems or a combination of analysis and synthesis. Since synthesis systems are very difficult to develop and have been shown in the past to be of only limited success, it seemed that an attempt to develop either of these two systems would not have been desirable.



Note: Data store D15 Has been broken down into its component parts and listed here as D8.1 - D8.7

FIGURE 11: Debug Program 8.0

However, the third point, shown in Figure 11, indicated an analytic system involving classification and diagnostic skills. Therefore, it was decided to develop an expert system designed to aid or replace the expert debugger. Before describing the steps in developing an expert system for the instructional program, the process of developing such a system in business will be described in order to provide a context.

Steps in Developing an Expert System in Business

Several experts in the area of expert systems have stated the steps necessary to develop an expert system (Boose, 1986; Buchanan & Shortliffe, 1984; Hayes-Roth, Waterman, & Lenat (1983); Tanimoto, 1987). Although the approach of each of these experts is different, there is general agreement among them as regards the basic steps. These are: (a) Identify and clearly define the problem that the expert system is to solve; (b) locate existing experts that could be assigned to provide the expertise in solving the problem; (c) develop the knowledge base and the rules for applying the knowledge base; (d) develop a prototype of the expert system; (e) alpha test the prototype using actual cases; (f) make any modifications necessary to the prototype until it appears to produce consistently the same solutions as those produced by the human experts; (g) document and field test the system; (h) iterate through

steps (a) - (g) as often as necessary; (i) release the system for general use.

The Use of Shells in Developing Expert Systems

The many complex steps in the development of an expert system cause most developers to attempt to simplify the process by the use of an expert system shell. A shell for an expert system refers to all the system's procedures but not the application-specific knowledge: for example, the procedures for accessing the knowledge and procedures that generate the consequences, conclusions or decisions from the existing knowledge base (Tanimoto, 1987). Shells are used for rapid prototyping and could facilitate the construction of knowledge bases. However work of formulating the knowledge requires extensive interactive dialogues between a knowledge-base building tool and one or more human experts. Waterman (1986) listed many of the commercially available shells and explained the differences in the way they were designed.

Because of the complexity of analyzing and obtaining expertise, most developmental experts agree that if there is an expert shell available that will meet the requirements for the system to be designed, that shell is the best choice for developing the expert system. Ruth (1987) suggested that two good shells for instructional systems are EXUS and FIRST CLASS because both are goal driven, rule-based shells

with backward and forward chaining. Boose (1986) provided a relatively simple description of this type of expert system. In a simple goal driven backward chaining system, the user picks a goal, with or without the help of the system. The inference engine looks through all the rules that lead to that goal and queries the user about the most direct rules leading to that goal. This process is repeated until the truth of the goal is established.

In a simple data driven forward chaining system, the user gives the system certain facts and the system draws whatever conclusions that it could from those facts. The queries to the user are based on the traits or attributes characteristic of the given problem domain which are stored in the knowledge base. Also stored in the knowledge base are appropriate goals, the heuristics of what to query, and the desired sequencing. This information is loaded into the expert system shell through the cooperative efforts of the knowledge engineer and the domain expert. The knowledge engineer must not only elicit rules from the expert, he must also determine all the traits of the propositions and their characteristics as well as the accompanying control information.

Potential sources of expert knowledge for the expert system include human experts, textbooks, data bases, examples, case studies, and personal experience. The transfer of knowledge from human experts to knowledge bases is called

expertise transfer. The knowledge transferred consists of facts, procedures, and judgmental rules in the problem domain.

In some situations, it may be difficult to obtain knowledge from experts because their explanations may be in terms of idealized verbal descriptions. Also much expertise seems to be compiled in heuristics that are accumulated over the years and it may be difficult for experts to talk about these compiled patterns. In such situations, knowledge extraction techniques may be necessary.

A detailed discussion of knowledge extraction techniques was presented by Waterman (1985) from Clancey (1981) and reported by Boose (1986). They are summarized as follows: (a) observe the expert as he actually solves problems on the job; (b) talk to the expert about the problem and what he is doing as he solves it; (c) develop an extensive description of the problem, with or without the help of the expert, and have the expert react to and improve the description; (d) do a detailed task analysis of the problem; (e) develop a prototype of the system and refine it until the expert and the user agree that it produces the same results as the expert; (f) examine the system objectively to determine whether there are any weaknesses, and (g) validate the system by comparing the system's results with the expert's results in existing documented cases, or by having other outside experts use the system and compare its results

with the results they would produce for the same problem input. Most of these techniques are usually needed when an expert system is to be developed for a complex problem.

Steps in Developing an Expert System in Instruction

In developing an expert system, the purpose must be kept clearly in mind. As discussed previously, the purpose of an expert system in the instructional environment is to provide students with the expertise of an expert so that students could both accomplish some task requiring expertise and could also acquire the expertise of the expert. Thus, an expert system in instruction must not only provide the student with expertise, but also with an understanding of the structure of the knowledge of the expert and the heuristics used by the expert. At any point in the use of the system, the student should not only be able to query the system to find what knowledge is necessary, but also to find why it is necessary, and how it fits into the whole problem.

Major steps of development. Drawing from the steps used by business to develop expert systems and considering the unique needs of instruction, this researcher proposes the following steps in the development of an expert system. First, the steps will be described generally and then they will be applied specifically to the Vocational Data Processing course.

1. Select the point in the instructional program where an expert system might be of use. Using an analytical technique such as data flow diagrams, determine the points in the instructional program at which the student may need individualized expertise. Carefully examine the characteristics of the students who will use the system and analyze the instructional program to determine the points where expertise is likely to be needed. These points are the points where an expert system is likely to be effective.

2. Determine if the problem is appropriate for the use of an expert system. Determine if the expertise needed by the student requires analysis, that is classification, debugging, diagnosis, identification, or interpretation. As discussed earlier, expert systems from the analysis class are the simplest systems to develop and proposition-rule expert systems are the simplest types of analysis systems to develop. Because of the time and cost constraints of a typical instructional program, educators should be wary of attempting to design an instructional expert system that is not from the analysis class and proposition-rule based. If an attempt is made to develop a complex expert system, the costs are likely to be very high and the success is less likely.

3. Select an appropriate shell for the expert system. As previously stated, the shell should have procedures for accessing the knowledge base and inference procedures that

are appropriate for the intended application. Appropriateness of shells is estimated by comparing existing applications using the shells with the task for which the expert system is being developed. The more characteristics the systems have in common, such as type of analysis, structure of the subject matter, apparent heuristics for accessing the knowledge base, the more likely it is that the shell is appropriate. Because of time and cost constraints of a typical instructional program, developers should be wary of attempting to design an instructional expert system for which there is not an appropriate shell. Expert systems could be developed most efficiently if access and inference procedures already exist.

4. Establish sources for the expertise to be loaded into the expert system. Locate available reference materials which indicate how to perform the task under consideration. Also locate at least one expert instructor who has extensive successful experience with providing expert help to students attempting to learn the task under consideration, and who is willing to spend time working in the development of the expert system. If more than one expert is to be used, and if there is likely to be disagreement as to how to do the task, determine in advance the criteria to be used to determine which advice to follow.

Be certain that the expert agrees as to the correctness of any reference materials that may be used as a guide.

However, if the expert differs from the reference material, follow the advice of the expert. If the expertise of a human expert is loaded into the expert system, the expert system is most likely to provide the type of instructional support that the human instructor would give.

5. Create the rule base. Work with the expert and the reference materials to construct the rule-base in the if-then format. Not only should the expert be asked, but, if possible, the expert should be observed actually instructing students in the task. Experts often do not know, or could not tell, the heuristics that they use in performing a complex task or instructing students in that task. By observing the expert, comparing the expert's actions to the reference materials and the expert's stated rules, and querying the expert on the actual instructional content, the rule base of the expert system is much more likely to be accurate.

6. Create a prototype of the system and test it for programming errors. Once the system has been programmed and modified until there are no syntax errors, test the program by having the expert run the program to see if the expert agrees with the instructional aids provided by the system, and make any changes the expert recommends. Then ask another expert to test the system. Discuss the recommendations of the second expert with the developmental expert.

When the developmental expert approves of the system, test it on a representative sample of the target students who will be using the system and correct any problems that they may have with the user interface. Check to see that the results that the students get from the expert system are the same as the results that the expert said they should get. When the system appears to be working properly, test the system, comparing the outcome with a control group receiving instruction from the expert. Make any necessary adjustments to the system. By prototyping, testing, and modifying the system, the expert system is most likely to be effective as an instructional aid.

7. Continue to monitor the expert system. Use the system with additional students and add rules or modify existing rules as indicated. The design of expert systems is rarely considered to be finished because the situations for which they were designed tend to change over time. Therefore, there should be periodic reviews built into the use of expert systems in instruction so that the systems will be kept current.

Developing an Expert System for Debugging Programs Within the Vocational Data Processing Program

Introduction and first two steps. The steps in the development of an expert system within instruction could be more clearly seen in their application to the program

debugging example. Having determined where in the instructional program expertise was necessary (see Figures 6-8,11) and having determined that the expertise needed to aid the student in learning to debug computer programs was an analytic, diagnostic system (see A specific example from vocational data processing within the section Guidelines for Selecting Points for Expert System Usage in Instruction), the next step was to find a shell that would be both reasonably priced and effective.

Selecting a shell. Since the objective of the system was to help the user to debug computer programs as an expert would, it was necessary to find a shell that would permit the designer to use production rules that followed closely the top down, goal driven, breath-first problem solving techniques used by an expert programmer (Anderson, 1985). The top down, goal driven approach is represented by backward chaining. Therefore, the shell to be chosen needed to be capable of backward chaining and be able to group choices into logical arrangements similar to the breath first techniques used by expert programmers. In addition, the shell had to allow explanations as to the reasoning behind the choices, because the objective was not only for students to be able to debug like an expert debugs, but also to understand the line of reasoning involved so that the knowledge gained would be properly structured.

Considering these requirements, this researcher chose the shell INSIGHT, published by Level Five Research, which was recommended by Blaisdell (1985). Although INSIGHT was limited to only backward chaining, it was reasonably priced and met all other necessary requirements. Also Level Five Research was willing to give permission to use an educational version of the program for testing with no additional cost.

Locating expertise and creating the rule base. The goals and rules were determined by this researcher with the aid of IBM's basic manual, and with the aid of a student from a vocational data processing course who had graduated at the top of the class. The functions of subject matter content expert and expert vocational data processing instructor were performed by this researcher. Three major goals within the area of logical errors and fifteen major goals within the area of language errors were identified. The production-rules were based on the expertise of this researcher.

Creating a prototype. A prototype was developed by this researcher and the graduate student in about forty hours of programming time. The prototype was initially tested for appropriateness by two graduate students who were very competent programmers. Subgoals were added as soon as the primary goals were working properly. The prototype was modified as new points were noted. An additional ten hours

were needed to test and modify the system. After the system was tested with students, the final count was sixty-eight goals.

The rules were grouped so that related goals would appear on the same screen and multiple possibilities as to the error conditions would also be displayed on one screen. To make the system as useful as possible for students, the system was loaded into the background of the computer's memory as students programmed in the foreground. If students encountered problems for which help was desired, they switched to the background expert system, found the desired help, and switched back to the foreground programming session to correct the error.

Testing the expert system. The prototype was tested with secondary vocational data processing students who were learning to develop programs using the BASIC programming language. Before going to the computer room, students used Input-Process-Output sheets, layout sheets, and program flowcharts to design their programs. In a typical computer session, the student placed a system disk containing the expert system, Double Dos, which is a program which partitions computer memory and permits more than one program to run on the same computer concurrently, and BASIC into the system drive of an IBM compatible computer and turned on the computer. The auto-execute file on the disk loaded Double Dos. The Double Dos program partitioned the computer's

memory so one part contained 250,000 bytes of memory and the other portion contained whatever additional memory the computer had available. Double Dos then loaded the expert system into the 250,000 byte portion of the memory and made that portion inactive, or background. Double Dos loaded BASIC into the other portion of the computer's memory, made that portion active, set the switch between memory portions to be the combination of the ALT and ESC key, executed BASIC so that the ready prompt was displayed, and turned control of the computer over to the student. This entire process took approximately one and one-half minutes. Loading BASIC directly took the typical student approximately one minute.

If students were continuing from earlier computer sessions, they would load their existing BASIC program and continue their work. If students were starting new programs, they would start typing a new program into the computer. When students reached the point where they were ready to test their programs, they typed RUN, causing one of three actions to occur: (a) the program ran as the student anticipated; (b) the program ran, but did not run as the student anticipated; or (c) the program would not run. To clarify the interface between the student and the expert system, two examples of how a student might interact with this expert system are given. The first example will be of a logical error and the second example will be of a syntax error.

1. Having entered a portion of the program to test, the student typed RUN and pressed enter. However, instead of running properly, the program continued to execute and did not stop. Being inexperienced and not knowing how to handle this problem, the student activated the expert system by pressing ALT and ESC together, a key combination the student often used in other computer classwork to change between applications. The student was asked by the expert system to indicate whether the problem was a logic error or a language error.

For each question the expert system asked, there were expand screens that could give the student more information. These expand screens could be activated from a menu of function keys that always appeared at the bottom of the student's screen. In addition, for each question the student could indicate that the answer to the question was not known, and the expert system would then ask a series of questions to help the student answer the question.

For this example, the student either indicated directly, or by answering a series of determining questions, that the error was a logic error. The expert system then displayed information about logic errors and their typical causes. The student was asked to indicate whether or not the program stopped. In this case the student indicated that the program did not stop. The expert system asked if the program had a menu or used the KEYWORD INPUT. If the

student indicated that there was no menu and INPUT was used, the expert system displayed information about the use of the INPUT statement and the function of trip data. Then the expert system asked the student if the trip data check was the line of code immediately following the INPUT line. If the student indicated that it was not, a screen was displayed that explained the function of the check for a trip data and provided two examples of the proper use of trip data. The student was then told a logic error in which the program did not stop had occurred and to correct the error by checking for trip data input. If the student indicated that there was a trip record in the program, the expert system displayed two screens explaining the use of GO TO statements and explained how to use the trace function to locate the line numbers that were being executed in a loop. The student was then told that a logic error in which the program does not stop had occurred and to correct the error by changing the GO TO statement.

Having been told the probable cause of the error, given examples of the proper use of the KEYWORD causing the error, and told how to correct the error, the student pressed the ALT and ESC key together and returned to the basic program, which was still running. The student then made the correction suggested by the expert system and executed the program again.

2. Having entered a portion of the program to test, the student typed RUN and pressed enter. However, instead of running properly, the program executed for a short time and then printed the message "out of data in line xxx." The student followed the same procedures followed by the student in example 1, except that for this example, the student either indicated directly, or by answering a series determining questions, that the error was a language error. The expert system then displayed information about language errors and their typical causes. The student was given a list of language errors and asked to indicate the one given by the program in error. In this case, the student selected the out of data error. The computer then displayed a screen containing an explanation of the six most common programming mistakes that could cause this error. The expert system then asked the student to indicate which of these six errors had caused the error message. If the student selected an error, two screens were displayed which explained the function of the READ statement and how to check for trip data. Examples were provided of the proper use of READ and DATA statements or of matching the DATA and READ variables. The student was then told that a language error in which an out of data message had occurred and to correct the error for whichever of the six methods the student had indicated caused the problem.

If the student indicated that the cause of the out of data error was not known, the expert system displayed two screens that explained the characteristics of the six most common programming mistakes that could cause this error. The expert system then asked the student a series of questions that helped determine which specific error had caused the problem. For example, first it asked if there was a check for trip data after the READ statement. Although trip data was explained on the screen that described the most probable cause of the out of data error, expand screens were available so that if the student asked for help, the term trip data would be explained and several examples of the proper use of trip data with a READ would be given. If the student indicated that there was a check for trip data, the expert system asked the student if the trip data were in the DATA statement. If the student indicated that trip data were in the DATA statement, the expert system asked the student if the value of the data in the check trip data statement matched the value of the data in the DATA statement. If the student indicated that it did, the expert system asked the student if the trip data were being assigned to the same variable that was being used in the trip data check. If the student indicated that it was, the expert system asked the student if for every variable in the READ statement there was a value in the data statement. If the student indicated that there was, the expert system

asked the student if the variables in the READ statement matched the data in the DATA statement.

If the answers to any of the above questions would have been no, the error would have been located and the expert system would have displayed a screen telling the student how to correct that particular error. The student would then have been told that a language error in which there was an out of data message had occurred and to correct the error by whichever of the six methods the student had indicated caused the problem.

Having been told the probable cause of the error, given examples of the proper use of the KEYWORD causing the error, and told how to correct the error, the student pressed the ALT and ESC key together and returned to the basic program. The student then made the correction suggested by the expert system and executed the program again.

However, if the student indicated that the last question to be asked did not indicate the error, the student was told that a language error in which there was an out of data message had occurred but that the expert system could not determine the correction for the error. The student could then ask the expert system what reason it had for asking all the questions. The expert system would then step through each of the inference rules the system had used and list the goal to which it was directed. In this example the explanation would be: The question "the program runs is true or

false" was asked to determine that "the error belongs to language errors." The list of possible syntax errors was presented to determine that "the error was out of data". The question "the error is no check for trip data after the READ" was asked to determine "correct error by checking for trip data after the Read." The question "the error is no trip data in the Data statements" was asked to determine "correct error by adding trip data to DATA statements". This display of the questions asked and the conclusion that would have been given if the answer had been no would continue until all the questions asked by the system had been displayed. Then the expert system would have printed "the information provided was not sufficient to reach a conclusion." The student would have gone through this presentation of questions and their possible conclusions to see if the answer given had been incorrect or if the cause of the error could be determined by merely using the expert system's line of reasoning. If the student could not determine the error after asking the expert system to explain its reasoning, the student would need to ask a human expert for help.

Monitoring the expert system. As students used the system, some situations occurred for which the expert system did not have rules or where the rules were unclear. In such situations, new rules were added or existing rules modified. The final system required seventy-two rules. In addition,

several points were discovered where the students did not understand the system's questions and needed more information. Expand screens were made available for these more complex situations. Displays were added for each goal and subgoal, that explained the rationale underlying the solution. The modifications took approximately ten additional hours of programming time, making the total programming hours approximately sixty. A printout of latest version of the expert system is given in Appendix C.

CHAPTER FIVE

DISCUSSION AND RECOMMENDATIONS

Summary of the Study Reported

This dissertation discussed the increased need for efficient and effective instruction of high level intellectual skills in order to meet the challenges of the Information Age. It was noted that learning theories and instructional design theories had been developed that dealt with cognitive processing, but that the tools currently used by instructional designers did not readily facilitate the teaching of cognitive processes. Several specific problems with one current design technique were identified and discussed. These were: (a) inadequate consideration for need for expertise and knowledge, (b) insufficient focus on needed information, (c) limitations created by expressing cognitive skills in a task format, (d) lack of consideration for the interrelationships among topics and processes in the instruction, and (e) lack of convenient ways to explain a course to interested, content-area-naive, parties. Thus, the conclusion was reached that there is a need for instructional design techniques that permit developers to apply learning and instructional theories more effectively, especially in the indicated problem areas.

Next it was noted that business and industry had also been affected by the Information Age. The suggestion was made that data flow diagrams and expert systems, two tools and techniques developed by MIS to handle information, might also be of use within instructional design. The broad questions were addressed within the discussion of the specific questions. The five questions about the use of these tools were addressed as follows:

1. How might the use of data flow diagrams in the design of a particular secondary vocational data processing course provide information about the relationships between major chunks of instruction, that is courses, units, chapters, and lessons; or the major cognitive processes within the chunks? By using data flow diagrams to develop part of the design of a secondary vocational data processing course and examining several levels of the design, the answer was determined that data flow diagrams indicated the passing of information between the units of instruction and processes within the programming unit as well as indicating the sequencing among the units and processes. This information was not available in the Michigan Vocational Data Processing Curriculum Guide designed using current design techniques.

Before attempting to address the first broad question, part of a set of data flow diagrams used to design an interactive video unit for veterinary medicine was examined. It was noted that the data flow diagram of the entire course

structure showed that three units of the course would add to the students' knowledge bases, while a fourth unit would require students to access information from their knowledge bases. In addition, it showed that information developed in some units of the course was needed in other units. Thus it showed some of the ways that the units were interrelated.

After examining these two demonstrations in which data flow diagrams were used to design very different types of instruction, the answer for the first broad question was determined: the MIS design tool, data flow diagrams, might provide information in the instructional design process.

2. How might data flow diagrams used to design a particular secondary vocational data processing course indicate what information might be needed by students at various points of the instructional program? The data flow diagrams that were developed to answer the first specific question also indicated the location and various types of information that might be needed by students for each of the major processes and also indicated where the same information was needed for more than one process.

When the data flow diagrams for the design of the veterinary medicine courseware were examined they provided: a structure that corresponded to the presentation format used in instruction developed for interactive video; a graphic representation of points where the instruction needed to provide opportunities for remediation; a way of

classifying the instructional events as attitudes, facts, concepts, and principles for appropriate presentation formats; and the data necessary to estimate the potential cost of developing the system.

After examining these two demonstrations in which data flow diagrams were used to design very different types of instruction, the answer for the second broad question was determined: the MIS design tool data flow diagrams could indicate what information might be needed by students at various points of the instructional program. In addition, data flow diagrams could provide information that might be used to develop interactive video courseware.

3. How was the use of data flow diagrams to design MIS systems modified to use data flow diagrams to design a particular data processing course? To use the data flow diagram technique to design the course this researcher modified the technique to use: (a) the shadowed square to indicate points that were dependent on the individual students; (b) the rounded rectangle to indicate points where the students organize, interpret, or transform information; and (c) rectangles to show internally stored information as well as external stores. In addition, this researcher did not use the instructional dictionary if the information it contained was probably known to the developer or teacher.

The third broad question was discussed in the context of how data flow diagrams were used in business and how they

might be used in instructional design. One modification that might be made is that the symbols used in business to show activities that were not directly a part of the information system might be used in instruction to show points that are dependent on the individual students or where expert help might be needed. A second possible modification might be that the data dictionary used by business is always developed to show the specific details about the information, but the instructional dictionary might not always be used. For example where the need for cognitive strategies was shown in the design of the instruction for the cognitive skill programming, the instructional dictionary was not used, but the dictionary was very important when it contained the specific instructional events that were a part of the interactive video course in pain control. A third possible modification might be that when the purpose of the instruction is to use interactive video courseware to add to students' knowledge bases, the information stores might not be grouped by topic, but might be grouped by type of instructional events to facilitate the development of the presentation modules and to estimate developmental costs.

4. How might data flow diagrams used to design a particular instructional program to teach the problem solving skill, computer programming, indicate potential points in the instruction where expert systems might be used to provide students with expert help? Through the development

and examination of data flow diagrams for the problem solving skill, computer programming, the answer was determined that points in the instructional program were indicated where expert systems might be used to provide the expert help needed by the students. From this example the answer to the fourth broad question was determined: when the objective is to teach complex cognitive processes such as problem solving, data flow diagrams might indicate points in an instructional program where individualized expert help typically might be provided.

Having discussed the use of data flow diagram in instructional design, the second tool from business, expert systems, was examined to address the fifth specific question:

5. How were steps used to develop MIS expert systems modified to develop an expert system to help teach students in the data processing course how to debug BASIC computer programs? After discussing several examples of expert systems in both business and education, the differences between expert systems designed for businesses and expert systems designed for instruction were discussed. The steps required to develop expert systems in business were discussed and a specific example of the development of an expert system for debugging BASIC computer programs was presented. The development of this expert system took seven steps: select the point in the instructional program where

an expert system might be of use; determine if the problem is appropriate for the use of an expert system; select an appropriate shell for the expert system; establish sources for the expertise to be loaded into the expert system; create the rule base; create a prototype of the system and test it for programming errors; and continue to monitor the expert system.

By comparing the development steps of this expert system for debugging to the general steps for developing an expert system for business, the answer to the fifth broad question was determined. The development of the expert system used as a tool within an instructional program might be different from the development of an expert system for business use in several ways:

1. In business applications, the problem is clearly defined and the expert is located as the first steps. In instruction, the need for expert help is not as discernable and the first steps are to locate the need and see if the problem is appropriate for the use of an expert system.

2. In business applications, expert systems are built using whatever techniques the developers think are appropriate. In instruction, the financial constraints suggest the selection of an expert shell for typical classroom usage.

3. In business, expert systems can be built that have extensive knowledge bases and rules for applying the

knowledge. In instruction, financial and time constraints might limit the development of expert systems to systems that can be built from rule bases.

4. In business, the expert system must be extensively tested and documented before being released for general use. In instruction, the expert system supplements rather than replaces the expert teacher so that the testing process might be less extensive before the system is used.

5. In business, the expert system is released for use. In instruction, continued monitoring of the system might be established to keep the system current.

Recommendations

Despite the limitations to this study, it was demonstrated that an instructional developer can examine and improve the tools and techniques currently used to design instructional systems. The discussion and examples presented suggested that data flow diagrams and expert systems might provide some improvement. Several areas of study might prove to be beneficial. One of the greatest research needs in this area is to determine the impact of data flow diagrams and expert systems in a wide variety of instructional systems, especially in those that do not involve computers. Specifically:

1. Do data flow diagrams provide equally useful information for more structured subject matter areas than

for less structured subject matter areas; for example, mathematics/science, on the one hand, versus language arts and the social sciences on the other hand?

2. Do data flow diagrams provide useful information to teachers of different levels of subject matter experience and expertise?

3. Do data flow diagrams provide useful information for instruction for students of different grade levels?

4. Can teachers with limited experience with systems analysis techniques use data flow diagrams without extensive training in their use?

5. Can instructional designers with limited experience with systems analysis techniques develop data flow diagrams without extensive training in their use?

6. Will the information provided in data flow diagrams enable teachers to determine appropriate media for the delivery of the instructional events?

7. Will non-instructional stakeholders understand data flow diagrams of instructional programs? If so, will their understanding depend on the subject matter content or with their familiarity with the subject matter content?

8. Will inexperienced teachers teach higher order cognitive skills more effectively if they are given data flow diagrams of the known processes that compose these skills than if they are given only conventional information

about the skills, such as task lists and lists of objectives?

9. Will students that are provided help by an expert system learn to do the task as well as students that are provided help by a human expert? Will there be an interaction with grade level?

10. Will students want to get help from an expert system or will they want their help to come from a person?

11. Will students using expert systems learn the processes involved in a higher order cognitive skill in addition to demonstrating the behavioral manifestation of the skill?

12. When applied to different subject matter areas, will data flow diagrams indicate points at which students require help?

13. Will expert systems provide help in diagnostic areas less structured than computer program debugging, for example in debugging an English composition or a history report?

A second major area for additional study concerns the effectiveness and efficiency of the data flow diagram technique in instructional design and of the expert system as an instructional delivery tool. Specifically:

1. Will the extra time needed to develop data flow diagrams as a part of instructional design improve the instruction sufficiently to justify the additional cost?

2. How effective and efficient is the use of data flow diagrams to design instructional systems compared to other MIS structured system design techniques (Davis & Olson, 1985) such as top-down design with stepwise refinements (graphically illustrated with hierarchy charts), Structured Analysis and Design Technique (SADT), and Hierarchy-Input-Process-Output (HIPO), or with the new Computer Aided Software Engineering techniques (CASE)?

3. Will teachers use data flow diagrams to make instructional decisions?

4. Will providing students with expert systems be cost effective? If it costs more to provide students with expert systems than it did to provide them with competent tutors, would there be any advantages to using expert systems?

5. Would parents, administrators, teachers, and students be willing to accept the "expertise" of a machine?

6. Would teachers feel threatened by the use of expert systems to teach higher order thinking skills?

The use of data flow diagrams in the design of interactive video instruction seems especially promising, but many questions still require more study. Specifically:

1. Will data flow diagrams consistently show where access to previously presented materials needs to be made?

2. Will the instructional designer be able to make estimates of the cost and time necessary to develop the courseware from the data flow diagrams?

3. Will the developer be able to maintain the flow of the instruction when the instructional dictionary treats each instructional event independently?

4. Will such factors as the importance and difficulty for each instructional event provide specific information about the time, cost, and most appropriate media to use in developing the courseware?

5. Will experienced designers, developers, and authors be able to use the data flow diagrams effectively when designing interactive video courseware?

The discussion of problems with current instructional design techniques was primarily based on the curriculum produced by one technique in one subject area. More instructional design and development systems need to be examined to see if other systems have problems similar to the ones found in the development of the data processing curriculum.

One of the unanticipated findings of this study in the use of data flow diagrams was that the instructional dictionary was different for the two examples given, which represent the design of different types of instruction, which represented different cognitive processes. If the same flexible design tool were used for a variety of instructional design problems and the differences among the different designs were studied, perhaps some insights into different cognitive processes might be found. Many

different types of courses need to be designed using data flow diagrams and the results compared to see if there are any observable differences in the data flow diagrams that suggest possible differences in cognitive processing.

Cautions

Even if expert systems prove to be efficient and effective instructional tools, their use still should be carefully researched. Wensley (1987) pointed out that there are psychological as well as cognitive characteristics of experts. Expert systems currently can only attempt to model the cognitive characteristics. Adams & Hamm (1987) stated: expert systems work, but they miss much of the subtle, experience-based wisdom of human experts even as they allow for acquisition of some elements of an expert's knowledge. Unfortunately, the best expert systems that could be made within instructional budgets give only a mechanistic rule-governed simulation of the lowest stage of an expert's cognitive skill. Anything approaching higher levels of human thought is still on the technological horizon.

GLOSSARY

GLOSSARY OF TERMS IN THIS DISSERTATION

Agricultural Age - a period of time in which the economy is built on the growing of food and the majority of the workers have jobs working on farms.

Alpha testing: preliminary testing of a computer program by the developers of the program. Alpha testing is designed to find any cases that are not properly addressed by the system so that the system can be modified to handle all possible cases.

Artificial intelligence (AI) - a field of study that uses computers to model the information processing characteristics of humans.

Authoring - the process of writing instructional computer recognizable code using a language specifically designed for the development of instructional materials.

Backward chaining: working from the goal toward the given data.

Breath-first problem solving technique: a technique where a computer programmer finishes programming portions of a program that are similar to portions of programs previously done before even starting other portions of the program which may be less familiar.

Chunk: A collection of interrelated facts, concepts, principles, attitudes, images, sounds, and feelings that can be manipulated cognitively as a single unit. "A maximal familiar substructure of the stimulus (Simon, 1981, p. 80). "any configuration that is familiar to the subject and can be recognized by him (Simon & Newell, 1972, pp. 780-781).

Computer aided instruction (CAI) - any instruction that uses a computer for the delivery of the instruction. (see programed instruction)

Data flow diagrams - A manner of graphically documenting the flow of data and the procedures used within an information system using four symbols to show: (1) source or destination of data, (2) flow of data, (3) processes which transform flows of data, and (4) storage of data.

Debugging - in computer programming, the correction of logical or syntactic errors in a computer program or system. In problem solving, the process of locating and correcting errors in a solution.

Expertise transfer is the transfer of knowledge from human experts to expert system knowledge bases.

Expert system - a computer program that simulates the cognitive processes of a human expert. Expert systems are an applied form of artificial intelligence.

Expert System Shells: expert systems for which the inference engine has been designed, but the knowledge base and the rules are left blank so that applications can be developed by entering the domain specific application knowledge base and rules.

Forward chaining: a method of problem solving that starts with the given or input and moves toward the goal.

Heuristics: rule-of-thumb knowledge that enables experts to make educated guesses when necessary, and to deal with incomplete or inconsistent information (Boose, 1986).

If-then or if-then-else format: Rules in the form of proposition, truth inference, negation inference. For example: If the program runs, the error is an error in logic else the error is a language error.

Industrial Age - a period of time in which the economy is built on the production of goods and a majority of the workers have jobs directly involved with the production of goods.

Inference engine: procedures that generate the consequences, conclusions or decisions from the existing knowledge base.

Information Age - a period of time in which the economy is based primarily on information processing and a majority of the workers have technical, managerial, and clerical jobs.

Information system - a system to organize data in a meaningful fashion to produce information. The basic model consists of input, process, storage, and output, but in recent years distribution has come to be considered a part of the system. (see Management Information Systems)

Instructional Design - see instructional systems design.

Instructional Systems Design: the total set of procedures that are followed in planning, developing, implementing, and evaluating instruction. The procedures are derived from knowledge of human learning relevant for instruction and from the results of empirical data obtained during tryouts of preplanned instruction. (Aronson & Briggs, 1983, p. 99)

Intelligent Computer Aided Instruction (ICAI)- a form of computer aided instruction where the characteristics of human tutoring are incorporated into the program allowing instruction to adjust to different individual student aptitudes. (see computer aided instruction)

Knowledge base: production rules, semantic networks, frames, relational database and inference rules critical to a specific domain application.

Loading an expert system shell: The process on entering the knowledge base and rules into a computer program that already has developed procedures for accessing the knowledge base and applying the rules.

Natural language is a language that has developed over time as part of the process of human interaction.

Management information systems (MIS) - a branch of computer science/data processing concerned with the organization and coordination of the information owned by a company so that not only are day-to-day transactions processed, but all levels of management are supplied with information and supported in the use of technological tools for decision making.

Problem solving - a higher order intellectual skill consisting of the ability to make decisions or choices by using multiple pieces of information.

Production-rule expert system: a relatively simple expert system that consists of productions and rules in the IF...THEN...Else format.

Programed instruction - instruction designed to present learning in a step/sequence order, accompanied by reinforcement. In recent years, programed instruction has often implied drill and practice types of computer aided instruction.

Prototyping: a method of development of computer programs or systems that avoids extensive analysis and design phases in development by creating working models of the program or system. These models are tested and expanded or modified until the complete system is finished. This technique is usually significantly faster than any other design techniques but risk omission of important functions that were not considered during initial development. Usually systems built by the prototype method require frequent modification after installation.

Prototype: a working model of a system based on information about the functions of the system. It is the product of prototyping.

Rule-base: the collection of rules that indicate how decisions are made or conclusions are reached in a subject matter domain.

Shell: See Expert System Shell

Structured design methodology - the use of systems concepts to decompose an information system into functional subsystems and to define the boundaries and interfaces of each subsystem.

Technological society - a society where the use of technology has a major role in the performance of routine activities.

User interface: a way of communicating with the user. In expert systems, the user interface should close to the user's natural language.

Working memory: for an expert system, a temporary storage area to hold the immediately relevant portions of the knowledge base, the user's responses, and the current inference.

APPENDIX A

APPENDIX A

CURRENT INSTRUCTIONAL SYSTEMS AND DEVELOPMENTS

This appendix contains:

FIVE INSTRUCTIONAL SYSTEMS MODELS

Core Elements	Andrews/Goodson Tasks
Determine learner needs	Assessment of need, problem identification occupational analysis, competence, or training requirements Characterization of learner population
Determine goals and objectives	Formulation of broad goals and detailed subgoals stated in observable terms Analysis of goals and subgoals for types of skills/learning required Sequencing of goals and subgoals to facilitate learning
Construct assessment procedures	Development of pretest and post-test Matching goals and subgoals
Design/select delivery approaches	Formulation of instructional strategy to match subject-matter and learner requirements Selection of media to implement strategies Development of courseware based on strategies Consideration of alternative solutions to instruction
Try-out instructional system	Empirical try-out of courseware with learner population, diagnosis of learning and courseware failures, and revision of courseware based on diagnosis
Install and maintain system	Formulation of system and environmental descriptions and identification of constraints Development of materials and procedures for installing, maintaining, and periodically repairing the instructional program Costing instructional program

Richey (1986,p 96) and Andrews and Goodson (1980)

Figure 12: Comparison of Core Elements and Common Tasks

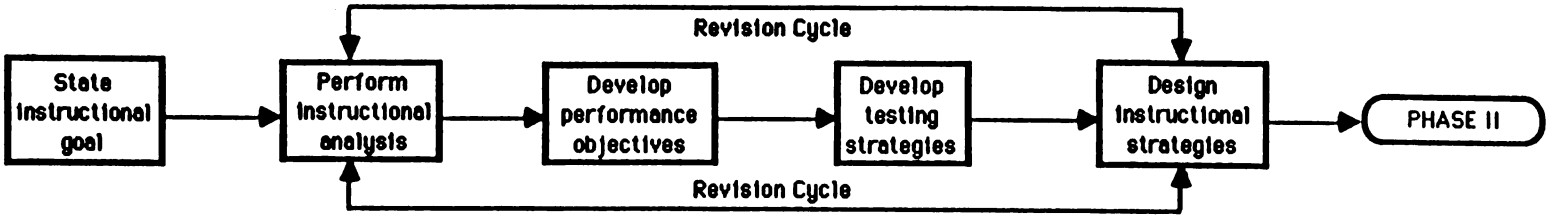
Stages of Instructional Design

1. Assessment of needs, goals, and priorities
2. Assessment of resources and constraints, and selection of a delivery system
3. Identification of curriculum and course scope and sequence
4. Determination of gross structure of courses
5. Determination of sequence of unit and specific objectives
6. Definition of performance objectives
7. Analysis of objectives for sequencing of enablers
8. Preparation of assessments of learner performance
9. Designing lessons and materials: (a) instructional events; (b) media; (c) prescriptions (utilizing appropriate conditions of learning)
10. Development of media, materials, activities
11. Formative evaluation
12. Field tests and revisions
13. Instructor training
14. Summative evaluation
15. Diffusion and operational installation

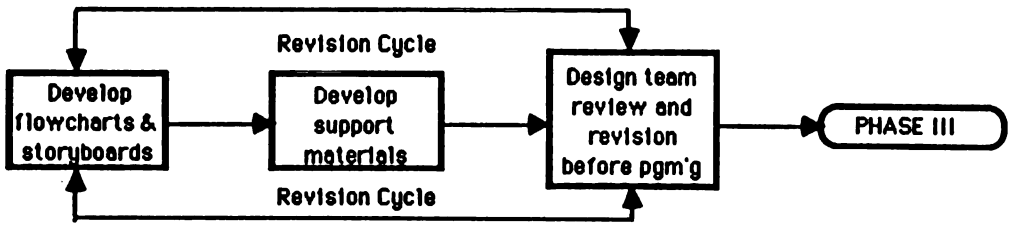
Briggs and Wager (1981, 5)

Figure 13: Stages of Instructional Design

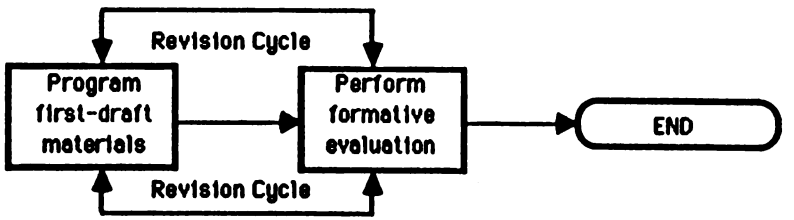
PHASE I: DESIGN



PHASE II: PRE-PROGRAMMING DEVELOPMENT



PHASE III: DEVELOPMENT/EVALUATION



Roblyer and Hall (1985)

Figure 14: Courseware Design Model

A - Various curriculum committees state goals in broad terms

B - Instructor meets with Instructional Specialist

C - Instructor assesses course limits, number of students, available finances, materials, etc.

D - Evaluation Specialist joins Instructor and Instructional Specialist to assist in description of specific objectives, content and behavoir

E - Instructor and Evaluation Specialist develop testing situations which measure defined behavior

F - Instructor and Instructional Specialist compile completed input information

G - Instructor and Instructional Specialist decide on group size, teacher student ratio contact, communication methods, experience factore, ect., based on theroy of instruction

H - Instructor, Instructional Specialist, Media Specialist, and other resource persons decide on information sources and exemplars

I - Instructor and Media Specialist determine best models based on perception and learning theory

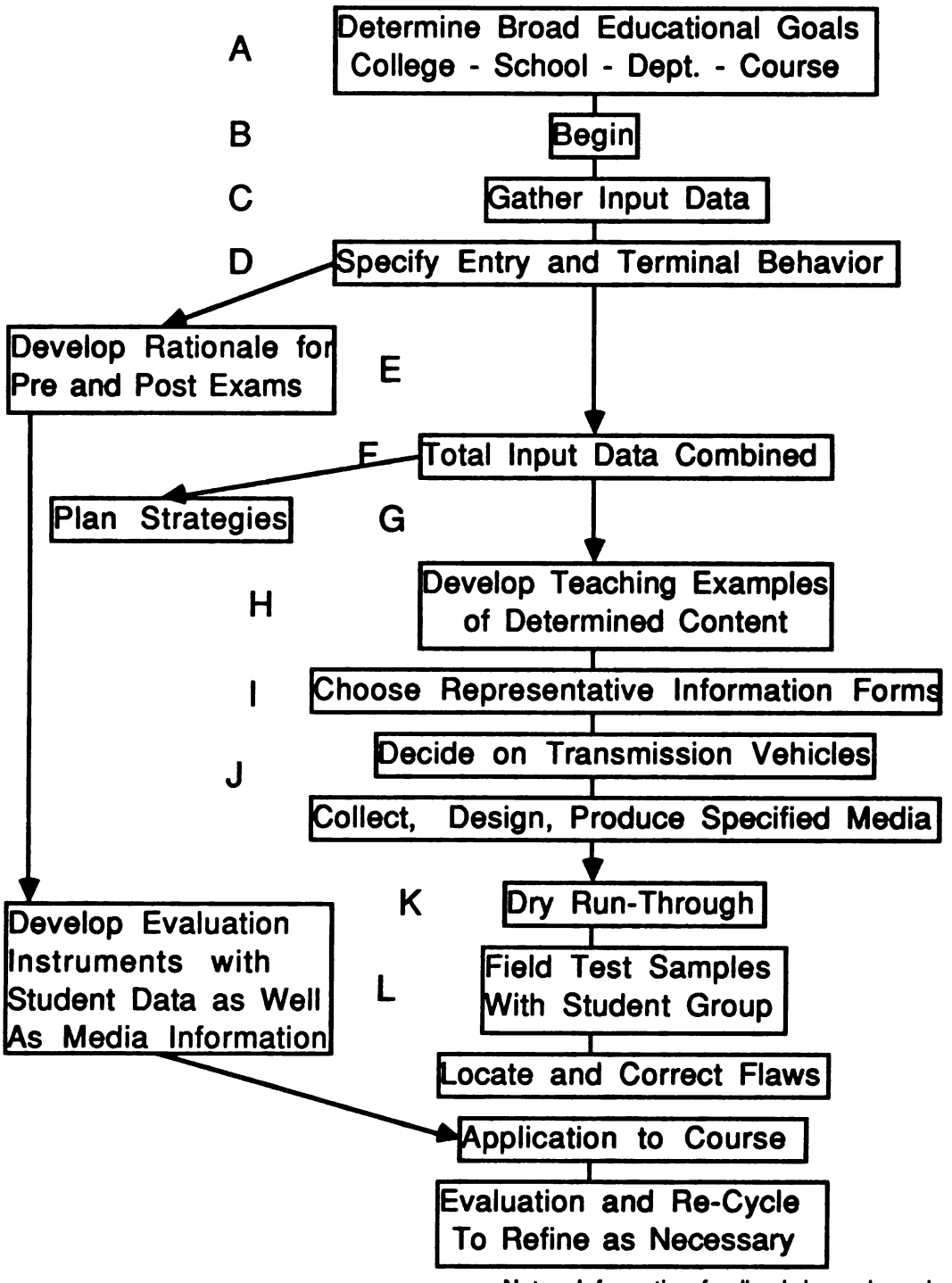
J - Instructor and Media Specialist determine which of various media is called for at points within the system

K - Instructor, Evaluation Specialist, and Instructional Specialist conduct representative dry runs of system packages

L - Instructor, Instructional Specialist and Media Specialist check feasibility of system with live audience and related test samples

Hamreus, 1968, p. I-59

Figure 15: Legend



Note: Information feedback loops have been deleted from this illustration.

Hamreus, 1968, p. I-59

Figure 15: Michigan State University Instructional Systems Procedure Model.

Design Procedure	Information Required for Practice	When Information is Used
1. Describe current status of system	What resources do I need? What resources do I have to work with? What limitations and constraints must I consider? (Examples: time; student entry skills; administrative rules; space)	Deriving and writing objectives Designing instructional procedures Formulating and implementing evaluation plan
2. Derive and write learning objectives	Goals Referent situation analysis Subject matter areas Resources and limitations	Formualte evaluation plans. Describe and analyze tasks. Instructional procedures. Feedback to referent and learning system description.
3. formulate evaluation plans	Objectives Resources and limitations	Implement eval. plan: assessment of student achievement. Redesign of learning system. Feedback to obj.
4. Describe and analyze tasks	Objectives Referent situation analysis	Design and implement instructional procedures. Feedback to objectives and evaluation.
5. Design and implement instructional procs.	Learning principles Task description and analysis	Implement evaluation plan: analysis of planned, executed, and result
6. Implement evaluation plan	Evaluation plan Instructional procedures design plan	Assess student achievement. Identify instructional problems for redesign.
7. Redesign	Evaluation data Learning principles Instructional procedures design plan Learning system description	Redesigned system

Davis, Alexander, & Yelon, 1974, p. 315

Figure 16: Information Relationships Among Learning System Design Procedures

Data Entry Task

Task No.	Task
DE-1	Manipulate switches, keys and levers using a knowledge of the device operations to activate a data entry device.
DE-2	Manipulate switches, keys and levers using a knowledge of the device operations to de-activate a data entry device.
DE-3	Analyze device malfunctions using a knowledge of the device operations to identify the cause of the malfunction as to whether it can be corrected by the operator or is to be reported to a service firm.
DE-4	Manipulate switches, keys and levers using a knowledge of the device operations to correct malfunctions on a data entry device.
DE-5	Manipulate switches, keys and levers using a knowledge of the device operations to execute standard service functions, such as changing ribbons, cleaning screens, emptying chips and other such services.
DE-6	Manipulate switches, keys and levers using a knowledge of the specific device to create a program to control record format operations such as skipping and duplicating.
DE-8	Manipulate switches, keys and levers using a knowledge of the specific device to load a program to control record format operations such as skipping and duplicating.
DE-9	Manipulate switches, keys and levers using a knowledge of the specific device to load media onto or into a device for keying data.
DE-10	Manipulate switches, keys and levers using a knowledge of the specific device to select a program format for a specific record.
DE-11	Manipulate switches, keys and levers using a knowledge of the specific device to enter data onto a media.

Task No.	Task
DE-12	Manipulate switches, keys and levers using a knowledge of the specific device to verify by machine the data previously entered into a media by a keying device.
DE-13	Analyze a data field using a knowledge of media coding to verify visually the data previously entered onto a media by a keying device.
DE-14	Manipulate switches, keys and levers using a knowledge of the specific device to duplicate data from one record to another.
DE-15	Manipulate switches, keys and levers using a knowledge of the specific device to update data contained in an existing record.
DE-16	Write job information using a knowledge of recording procedures to create a log of completed work.
DE-17	Analyze written descriptions in technical manuals using a knowledge of the device to resolve problems with device operations.
DE-18	Place media in a storage area using a knowledge of storage techniques to maintain a media file library in a logical sequence.
DE-19	Place media in protective containers using a knowledge of media storage techniques to store media.

Computer Operations Task

Task No.	Task
CO-1	Arrange a group of jobs to be executed on a computer using a knowledge of computer system performance to minimize the operational steps and overall run time.
CO-2	Analyze operating instructions using a knowledge of media used with a specific system to obtain the media needed for a specific job to be executed on the computer.
CO-3	Read operation instructions using a knowledge of computer operations procedures to identify the sequence of steps to be performed in completing a specific job on the computer.
CO-4	Inform persons who are assuming your duties of the status of operations using a knowledge of computer console operations and messages to maintain uninterrupted operations.
CO-5	Analyze operation instructions using a knowledge of peripheral equipment to identify the media to be used with a specific job.
CO-6	Analyze operating instructions using a knowledge of peripheral equipment to identify the media to be used with a specific job.
CO-7	Place media in a storage area using a knowledge of storage techniques to maintain a media file library in a logical sequence.
CO-8	Manipulate protective containers using a knowledge of data storage techniques to store media in containers.
CO-9	Manipulate switches and keys on a computer console using a knowledge of computer console operations to power up and initialize a computer system.
CO-10	Manipulate switches and keys on a computer console using a knowledge of operating system commands for a specific system to reassign control functions such as priorities and memory allocations.

Task No.	Task
CO-11	Manipulate switches and keys on peripheral devices using a knowledge of specific devices to load media onto the device.
CO-12	Select job control commands using a knowledge of job control language to run a specific job on a computer system.
CO-13	Analyze messages produced on a console using a knowledge of messages produced by a specific operating system to identify the status of a job being processed by a computer.
CO-14	Analyze written descriptions in technical manuals using a knowledge of a specific job control language to interpret descriptions of job control statements for a specific computer system.
CO-15	Compare computer produced output to a job description sample using a knowledge of computer media output to verify the accuracy of a computer run.
CO-16	Write information in a log using a knowledge of computer operations to record information about completed jobs.
CO-17	Analyze machine generated output using a knowledge of computer operations to identify the cause of processing errors.
CO-18	Manipulate switches and keys on a computer console using a knowledge of a specific operating system to restart a program execution in which an abnormal halt has occurred.
CO-19	Manipulate levers, keys and switches on media devices using a knowledge of the specific devices to adjust operating performance of the device.
CO-20	Write information on labels using a knowledge of cataloging descriptions to identify specific media as to the data entry thy contain.

Task No.	Task
CO-21	Manipulate levers, switches and keys on media devices using a knowledge of the specific device to complete routine servicing normally performed by the operator.
CO-22	Interpret device performance indicators using a knowledge of the specific device to identify a machine malfunction.
CO-23	Report verbally or in writing the malfunction of a device using a knowledge of the reporting procedure to notify service firms of the malfunction.
CO-24	Analyze operating instructions using a knowledge of back-up and restorations procedures to identify files that need back-up.
CO-25	Interpret system performance using a knowledge of system performance to identify system performance inefficiencies.

Computer Programming Task

Task No.	Task
CP-1	Analyze a program request using a knowledge of programming techniques to prepare an estimate of the time involved in preparing an executing a computer program.
CP-2	Analyze a program request containing specifications using a knowledge of data processing equipment and techniques to prepare program documentation that describe the parameters necessary to produce the desired output.
CP-3	Analyze program specifications using a knowledge of file organizational types to prepare file specifications to be described in the program.
CP-4	Analyze program specifications using a knowledge of data characteristics to identify specific data characteristics to be described in the program.
CP-5	Analyze program specifications using a knowledge of programming techniques to construct a guide to code the logical steps of a program.
CP-6	Analyze written descriptions in technical manuals using a knowledge of a specific language to interpret descriptions of language statements and operating characteristics of a specific computer system.
CP-7	Write language statements using a knowledge of the specific language and program documents to produce a source program coded in a specific language.
CP-8	Review a source program using a knowledge of the specific language to identify coding errors within the source program prior to compilation.
CP-9	Key source statements into a device using a knowledge of the device to produce computer readable source statements.

Task No.	Task
CP-10	Write job control statements using a knowledge of job control language and the specific operating systems to produce the instructions necessary to compile a source program into an object program.
CP-11	Analyze error notation produced by a computer using a knowledge of the specific language to identify syntax errors in the source program.
CP-12	Write source statements using a knowledge of the specific language and an error notation listing to produce corrected statements in a source program.
CP-13	Design data using a knowledge of data characteristics to test the logical correctness of a program.
CP-14	Write job control statements using a knowledge of job control language and the specific operating system to produce the instructions necessary to execute an object program.
CP-15	Analyze output data produced by an object program using a knowledge of the test data to identify logical errors in the program.
CP-16	Write source statements using a knowledge of the specific language and output test data to produce a corrected logical sequence in a source program.
CP-17	Analyze output of a test run with a user using a knowledge of the program requirements to obtain approval that the program is producing the desired results.
CP-18	Write user instructions using a knowledge of the program logic and specifications to produce a user's manual.
CP-19	Collect program listings, test data, and other specified documents using a knowledge of documentation to produce documentation of the design, coding and operation of the program.
CP-20	Analyze program documentation using a knowledge of programming techniques to produce a list of coding changes to a computer program.

Task No.	Task
CP-21	Write operating instructions using a knowledge of the program logic and specifications to produce an operations manual.

CURRICULUM WORKSHEET

Duty No. CP Task No. 2

Duty: Performing Activities Related to Computer Programming

Task: Analyze a program request containing specifications using a knowledge of data processing equipment and techniques to prepare program documentation that describe the parameters necessary to produce the desired output.

Pre-Test (Same as Achievement Indicators):
The learner:

1. Analyze the program request to determine description of input/output data and major processing parameters

2. Prepared a written general description of the input files, output files and major processing parameters and/or logic

---

References & Resources:
See Bibliography Nos. 1, 8, 9, 10, 11, 12, 13, 14, and 19 at end of section.

Duty/task Number CP-2

STUDENT LEARNING ACTIVITIES	TEACHER ACTIVITIES
1. Participate in a class discussion on how to analyze a program request to determine description input/output data and major processing parameters.	1. Take the class through a simple program request showing them how to write a general description of the input and output files and major processing parameters and/or logic.
2. Analyze a program request.	2. Assign readings and/or handouts to accomplish this task.
3. Prepare a written general description of input/output files and major processing parameters and/or logic.	3. Assign readings and/or handouts to teach system flowcharting.
4. Read assignments and handouts to understand how to accomplish the task.	4. Provide flowcharting templates.
5. Use a flowcharting template and flowcharting instructions to correctly flowchart a program request.	5. Describe the input Files.
6. Student can describe the files necessary to produce the desired output.	6. Describe the output files.
	7. Hand out a sample narrative to the class.

Duty/Task Number CP-2

TOOLS AND/OR EQUIPMENT	CONDITIONS
Central processing unit (CPU) capable of COBOL, RPG, and BASIC or access to a time-sharing computer. Cathode Ray Tubes (CRT) devices for input and output. Printer -- on-site. An adequate number of work-stations should be available so that one half of the class is involved in hands-on activities at any one time. Other off-line data entry equipment: key-to-tape, key-to-diskette (floppy disk), key-to-card.	The activity of writing general descriptions for input/output files and major processing parameters could be accomplished most any place. These are the kind of activities students should be involved in when they can't get on a work-station.

Criteria: Competence in the task will be recognized when to program documentation describes the data and logic necessary to prepare file specifications, data characteristics and a logic guide for the program.

Post-Test

Goal Statement #1 - You will produce documentation that describes the parameters necessary to produce a desired output.

Performance Objective #1 - Given a program request, you will demonstrate the following tasks: (a) analyze the input/output request and (b) prepare a written description of the input, output, and required processing. Performance will be accepted when documentation describes the data and logic completely, enabling the file specifications to be written without error.

Evaluation/Feedback

CURRICULUM WORKSHEET

Duty No. CP Task No. 5

Duty: Performing Activities Related to Computer Programming

Task: Analyze program specifications using a knowledge of programming techniques to construct a guide to code the logical steps of a program.

Pre-Test (Same as Achievement Indicators):

The learner:

1. Analyze the specifications to determine the organization of the logic necessary to produce the desired output.

2. Prepared a guide to the logical structure of the program by constructing a design document accepted by the industry.

References & Resources:
See Bibliography Nos. 10, 11, 12, 13, 14, 15, and 19 at end of section.

Reference Manuals

Duty/task Number CP-5

STUDENT LEARNING ACTIVITIES	TEACHER ACTIVITIES
1. Student can read the text material.	1. Explain uses of flowcharting symbols (if used).
2. Student can diagram (flowchart) problems.	2. Explain a situation requiring program to student.
3. Student can write the logic steps in an orderly fashion (IOP chart).	3. Lead a class discussion about detailed program flowcharting.
4. Students can exchange logic guides and check for each others errors.	4. Provide sample flowchart.
5. Participate in a class discussion about program flowcharting.	5. Provide flowcharting templates.
6. Write a detailed flowchart.	6. Demonstrate the programming cycle.
7. Determine the necessary calculations or processing steps in a program.	7. Place solution to the problem on chalkboard or overhead projector.
8. Determine the logical steps in a program.	

Duty/Task Number CP-5

TOOLS AND/OR EQUIPMENT	CONDITIONS
Central processing unit (CPU) capable of COBOL, RPG, and BASIC or access to a time-sharing computer. Cathode Ray Tubes (CRT) devices for input and output. Printer -- on-site. An adequate number of work-stations should be available so that one half of the class is involved in hands-on activities at any one time. Other off-line data entry equipment: key-to-tape, key-to-diskette (floppy disk), key-to-card. Flowcharting template. Overhead projector. Chalkboard.	This is an individualized and/or group activity which could be accomplished in or out of the classroom with no equipment needed. An appropriate supply of card layouts and printer spacing charts are necessary. An appropriate supply of flow-charting templates and coding sheets are necessary.

Criteria: Competence in the task will be recognized when the design of the logic describes the processing needed to produce the output according to techniques used within the industry.

Post-Test

Goal Statement #1 - You will construct a guide to code the logic steps of the program.

Performance Objective #1 - Given a program statement and the necessary tools, the student will construct a guide to code the logic of the problem. Performance will be accepted when the guide correctly describes the logic necessary to solve the problem and is completed according to techniques used within the industry.

Evaluation/Feedback

CURRICULUM WORKSHEET

Duty No. CP Task No. 11

Duty: Performing Activities Related to Computer Programming

Task: Analyze error notation produced by a computer using a knowledge of the specific language to identify syntax errors in the source program.

Pre-Test (Same as Achievement Indicators):

The learner:

1. Matched each error in the error listing with the source statement

2. Noted the nature of the error in the source statement according to the specific language structure

---

References & Resources:
See Bibliography Nos. 10, 11, 12, 13, 14, and 19 at end of section.

Reference Manuals

Duty/task Number CP-11

STUDENT LEARNING ACTIVITIES	TEACHER ACTIVITIES
1. Know when, where, and how to get program diagnostics.	1. Describe the common errors encountered in similar programs and enumerate the possible reasons for the errors.
2. Participate in a group discussion on debugging programs.	2. Help individual students find particularly difficult errors.
3. Be able to debug a program without assistance.	3. Demonstrate how to debug a program.
4. Correctly use available debugging tools or devices.	4. Provide a handout of a program for debugging.
	5. Provide and demonstrate whatever debugging tools and devices that are available.

Duty/Task Number CP-11

TOOLS AND/OR EQUIPMENT	CONDITIONS
Central processing unit (CPU) capable of COBOL, RPG, and BASIC or access to a time-sharing computer. Cathode Ray Tubes (CRT) devices for input and output. Printer -- on-site. An adequate number of work-stations should be available so that one half of the class is involved in hands-on activities at any one time. Other off-line data entry equipment: key-to-tape, key-to-diskette (floppy disk), key-to-card. Debug template.	A classroom situation should be available that allows students the opportunity to identify and correct syntax errors. Reference manuals and debugging templates should be available.

Criteria: Competence in the task will be recognized when all errors have been identified as to type.

Post-Test

Goal Statement #1 - You will identify syntax errors in the source program.

Performance Objective #1 - Given a source listing and an error listing produced by the compiler, you will identify the syntax errors in the source listing. Performance will be accepted when all errors have been identified.

Evaluation/Feedback

CURRICULUM WORKSHEET

Duty No. CP Task No. 15

Duty: Performing Activities Related to Computer Programming

Task: Write source statements using a knowledge of the specific language and output test data to produce a corrected logical sequence in a source program.

Pre-Test (Same as Achievement Indicators):

The learner:

1. Selected the coding sheets for the specific language
2. Wrote the statements to correct the logical errors using a source listing

---

References & Resources:
See Bibliography Nos. 10, 11, 12, 13, 14, and 19 at end of section.

Reference Manuals

Duty/task Number CP-15

STUDENT LEARNING ACTIVITIES	TEACHER ACTIVITIES
1. Students can write the language statements in corrected form, to eliminate the errors.	1. Provide handouts to cover the procedure for correcting a printout.
2. Using system input device to correct errors in the source program.	2. Demonstrate techniques for horizontal and vertical centering.
3. Recompile and execute the job control statements.	3. Ensure that the student understands why the original statements were in error.

TOOLS AND/OR EQUIPMENT	CONDITIONS
Central processing unit (CPU) capable of COBOL, RPG, and BASIC or access to a time-sharing computer. Cathode Ray Tubes (CRT) devices for input and output. Printer -- on-site. An adequate number of work-stations should be available so that one half of the class is involved in hands-on activities at any one time. Other off-line data entry equipment: key-to-tape, key-to-diskette (floppy disk), key-to-card. Spacing rulers.	In a laboratory situation the students will retrieve their printout, proofread for logical errors and correct these errors. Students should use vertical and horizontal rules discussed in the class. Printer spacing charts should be provided for student use.

Criteria: Competence in the task will be recognized when the object program is error free and executes as described in the program specifications.

Post-Test

Goal Statement #1 - You will write corrected source code to eliminate logical errors in your source program.

Performance Objective #1 - Given a source listing with logical errors identified, and an output listing with erroneous data, the student will write the source code correctly to eliminate all errors. Performance will be accepted when the object program produces an output listing with correct data.

Evaluation/Feedback

APPENDIX B

APPENDIX B

DATA FLOW DIAGRAMS FOR DEVELOPING A COMPUTER PROGRAM

This Appendix contains the data flow diagrams showing the design for the instruction to teach students how to develop computer programs to solve problems.

AF - Algorithms and facts
CO - Correction specification
CP - Coded program
CR - Code rules
CS - Coding standards
DA - Detail process algorithms knowledge
DI - Debugging information
DQ - Debugging questions
DS - Detail process strategies
EC - Existing code
EG - Existing graphic representation
EI - Existing input formats
EO - Existing output formats
EP - Existing process specifications
ES - Evaluation criteria subjective
EV - Evaluated program
GR - Graphic representation of process specifications
GS - Graphic standards
IP - Information about problem
IS - Input specifications
JS - JCL statements
LE - Logical error
LR - Language rules
MI - Missing information
NI - Needed Information
OS - Output specifications
PE - Process specification errors
PM - Program error
PP - Problem process definition
PQ - Process specifications
PR - Problem
PS - Problem solving strategies
PT - Processing standards
PW - Program with errors
RF - Rules for operating
RI - Rules for input formats
RO - Rules for output formats
SC - Specific evaluation criteria
SP - Specific processes
TD - Test data
TP - Tested program
UP - Unknown process

Figure 6: Legend

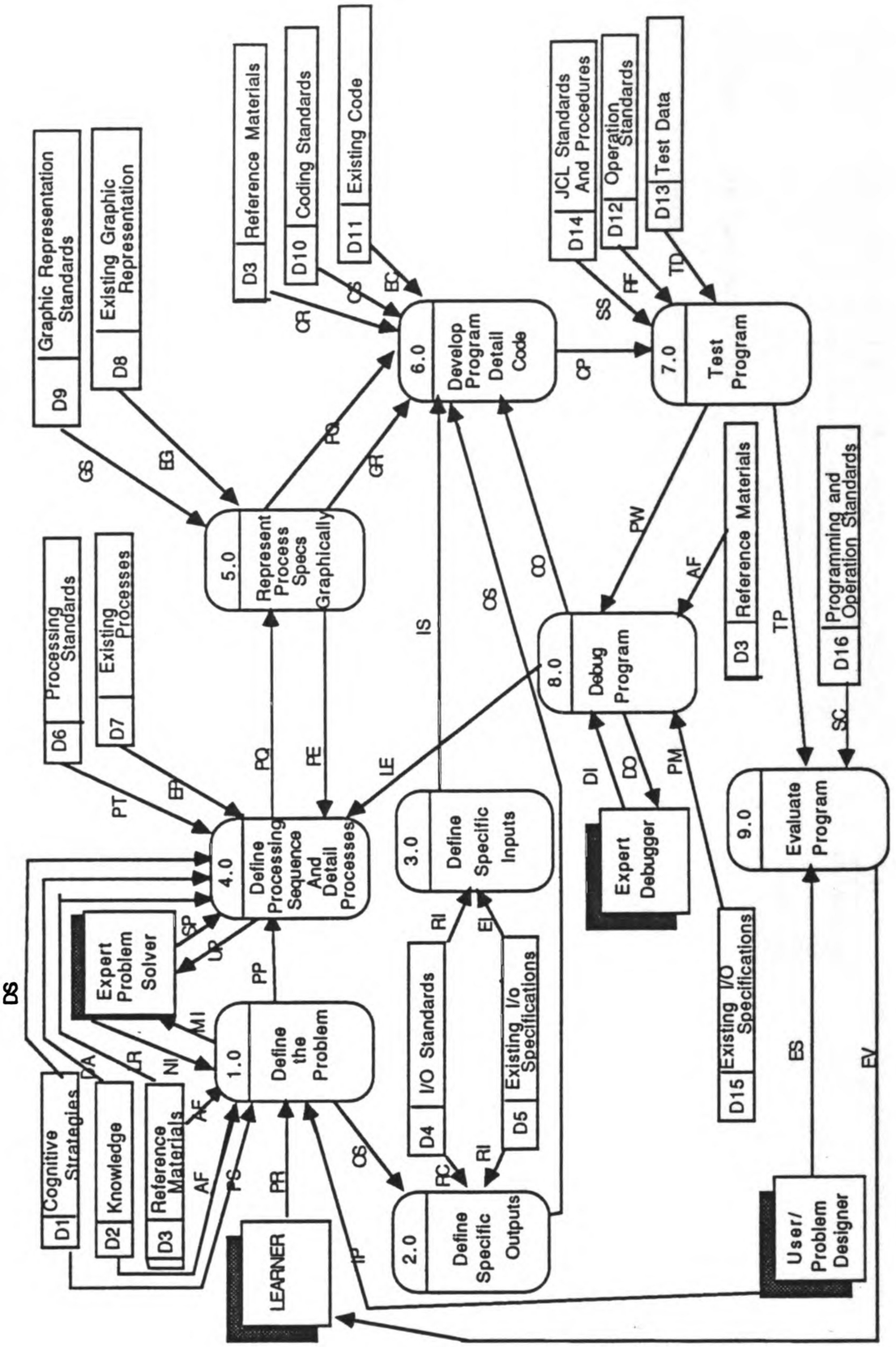


FIGURE 6: Designing a Computer Program

AA - Algorithims

DP - Problem process definition

IC - Information about special conditions

II - Information about input available & desired

IN - Information

IP - Information about problem

IS - Input strategies

IT - Information about process

MI - Missing information

NI - Needed information

OO - Information about output

OS - Output strategies

PA - Process algorithims

PD - Problem input definition

PO - Problem output definition

PP - Purpose of program

PR - Problem

PS - Processing strategies

RR - IPO rules

SP - Problem Strategies

Figure 7: Legend

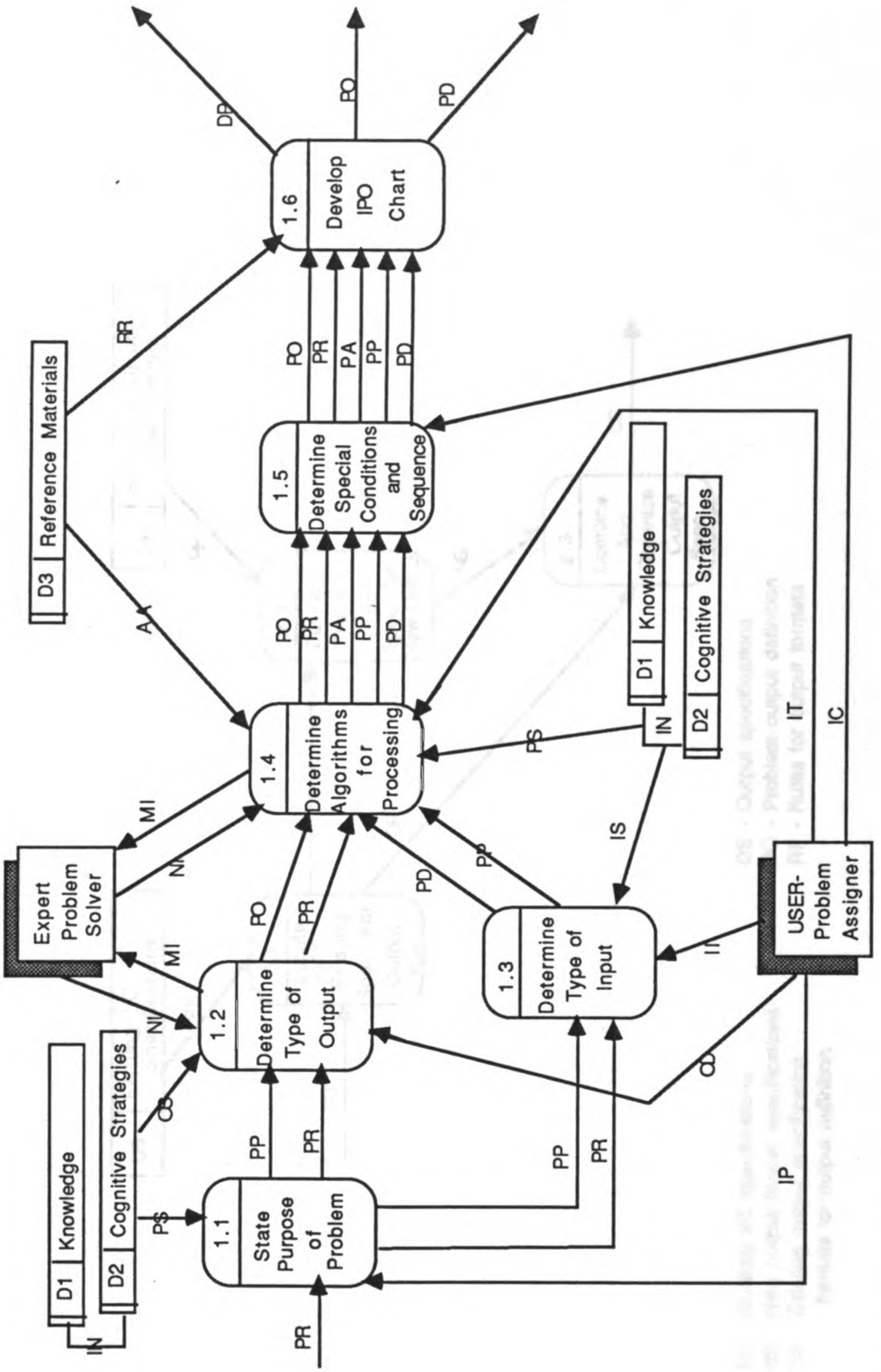


FIGURE 7: DEFINE THE PROBLEM 1.0

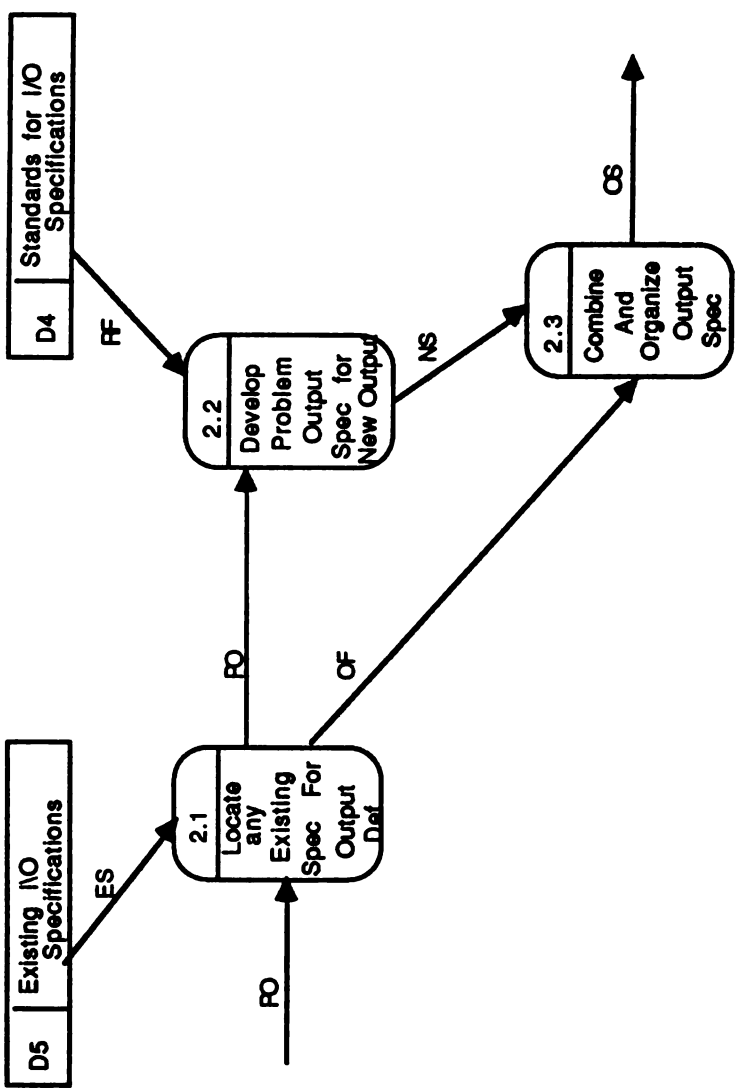


FIGURE 17: Define specific outputs 2.0

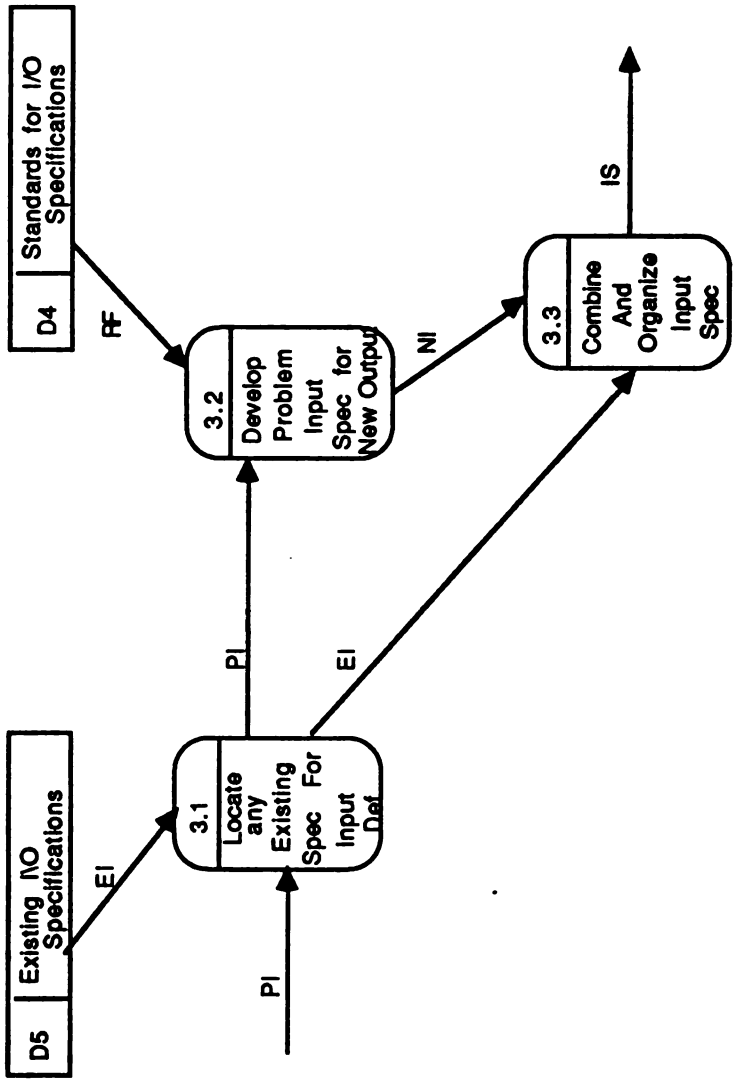


FIGURE 18: Define specific inputs 3.0

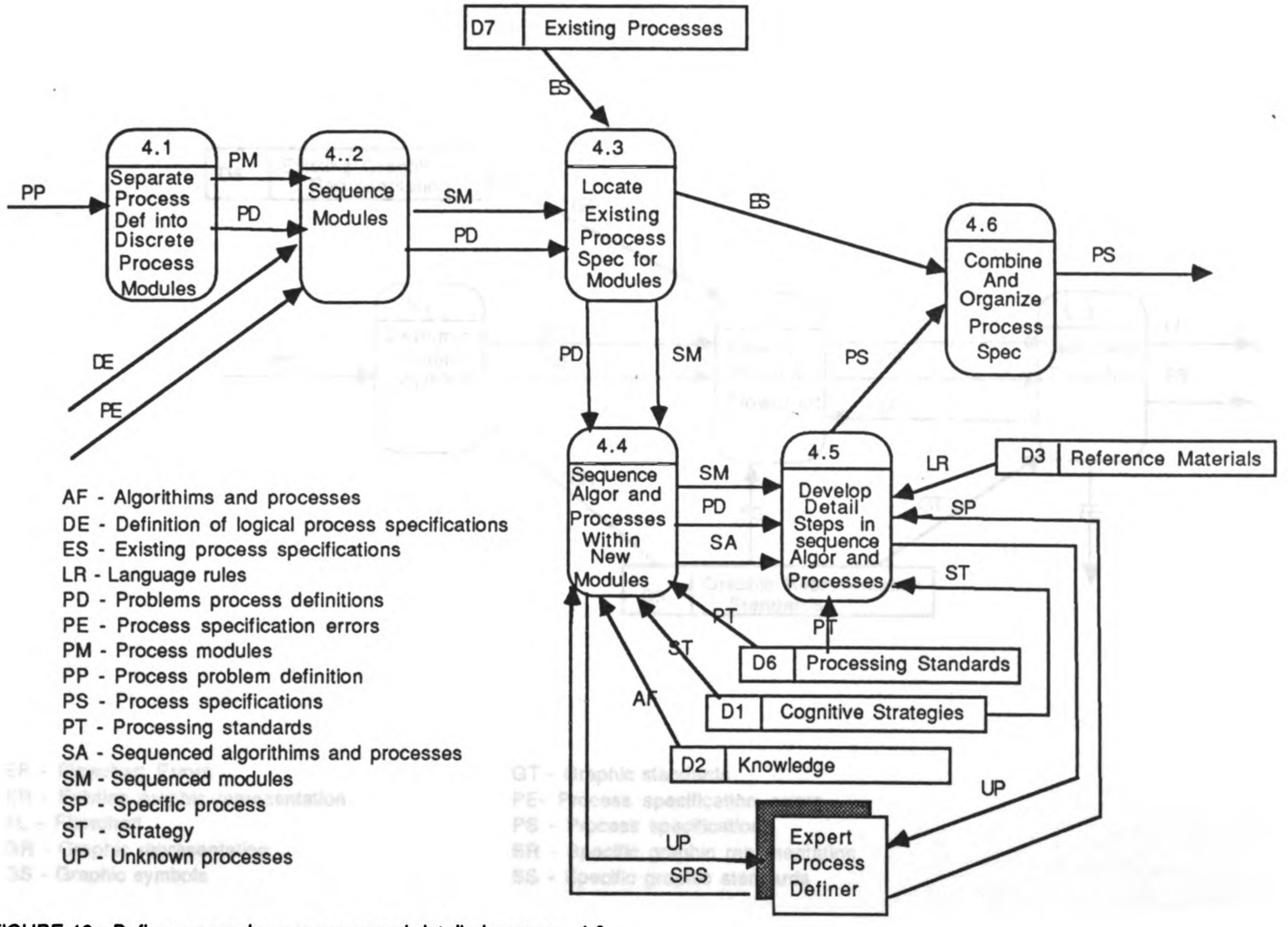
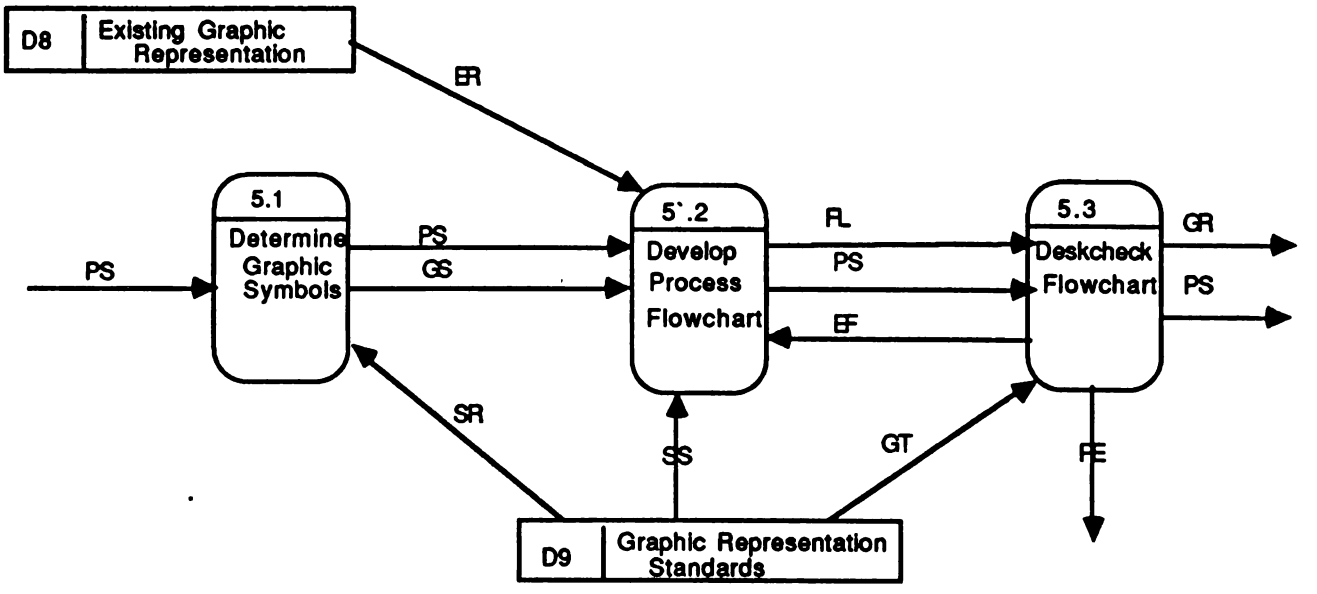


FIGURE 19: Define processing sequence and detailed process 4.0



EF - Flowchart Errors
ER - Existing graphic representation
FL - Flowchart
GR - Graphic representation
GS - Graphic symbols
GT - Graphic standards
PE- Process specification errors
PS - Process specifications
SR - Specific graphic representation
SS - Specific graphic standards

FIGURE 20: Represent process specifications graphically 5.0

CC - Correction code

CD - Coding standards

CE - Code with errors

CI - Coded input specifications

CO - Coded output specifications

CP - Coded program

CR - Correction Specifications

CS - Coding syntax

CT - Coding output syntax

EC - Edit code

EI - Existing input code

EO - Existing output code

EP - Existing process code

GR - Graphic representation of program logic

IC - Input coding syntax

IE - Input edit code

IS - Input specifications

IT - Input standards

OS - Output specifications

OT - Output standards

PC - Process code

PO - Process code standards

PM - Process module code specifications

PR - Process coding syntax

PS - Process specifications

Figure 21: Legend

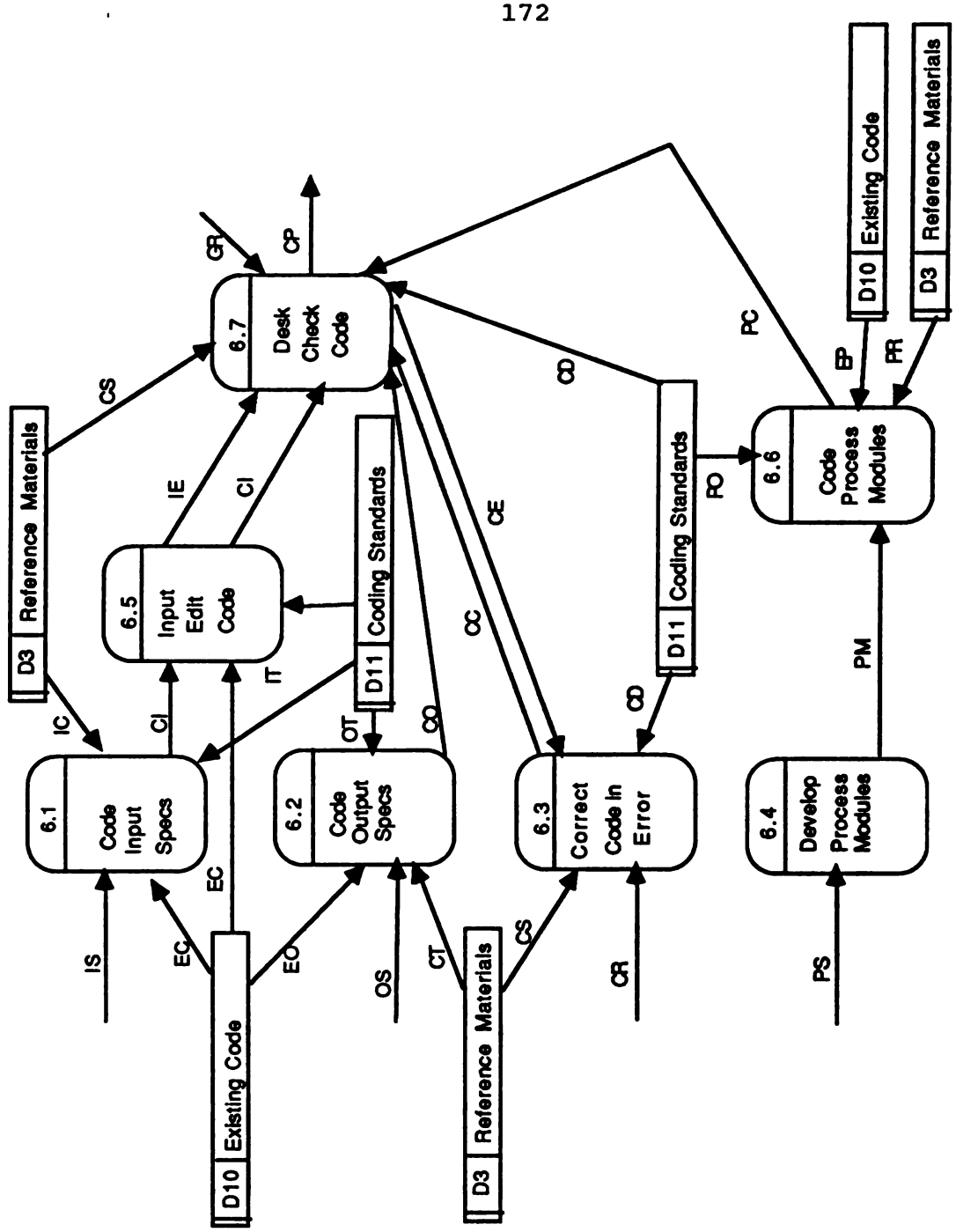
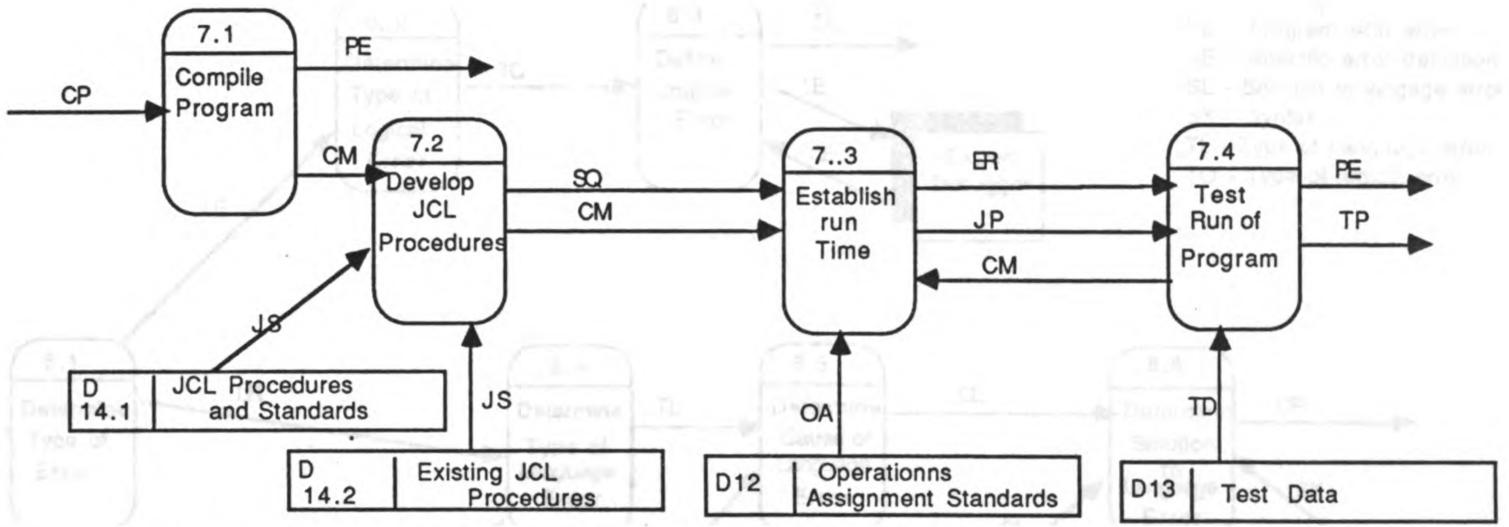


FIGURE 21 Develop program detail code 6.0

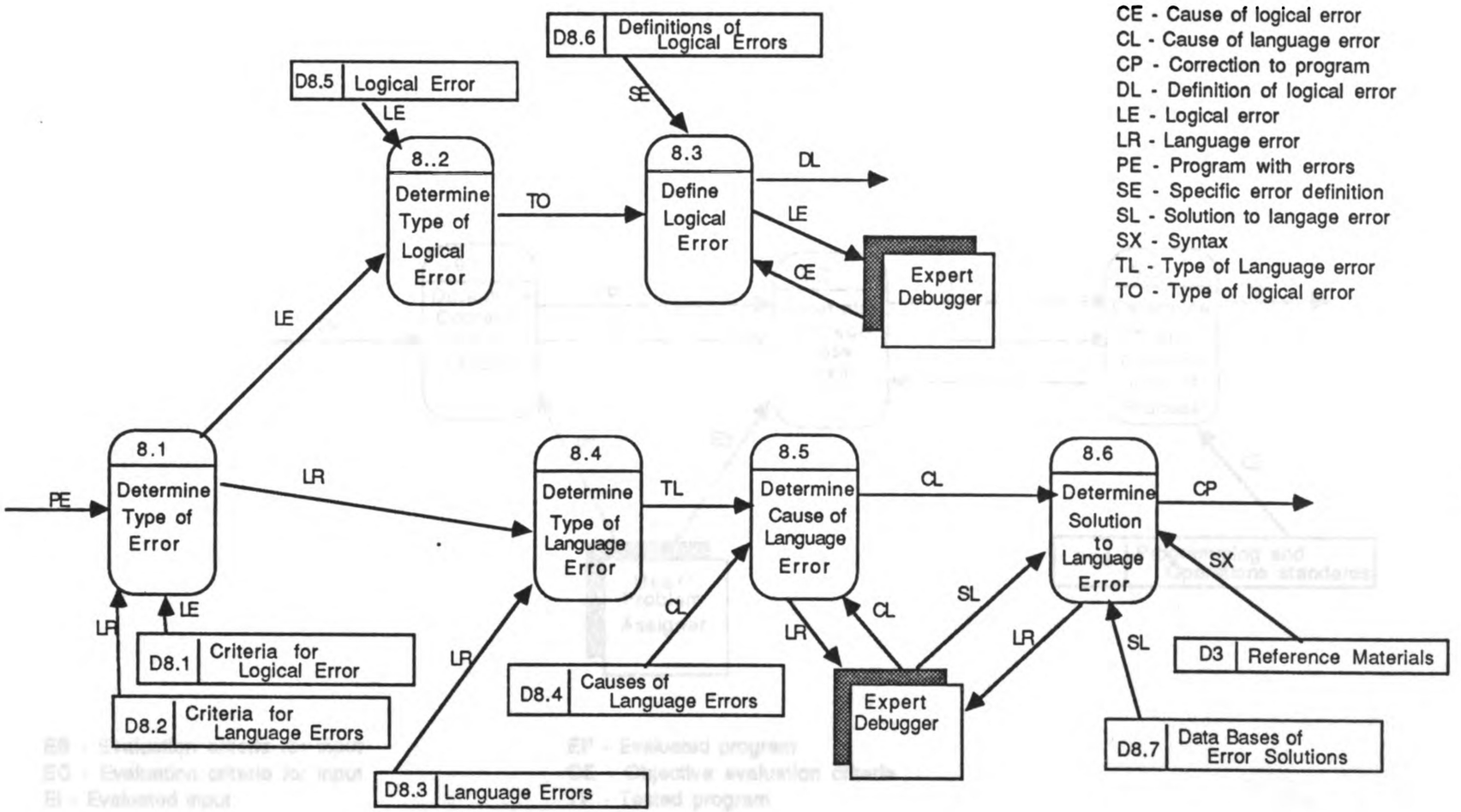


Note: D14 has been broken down into its component parts

CP - Coded program
CM - Compiled program
ER - Established run time
JP - JCL procedures
JS - JCL statements

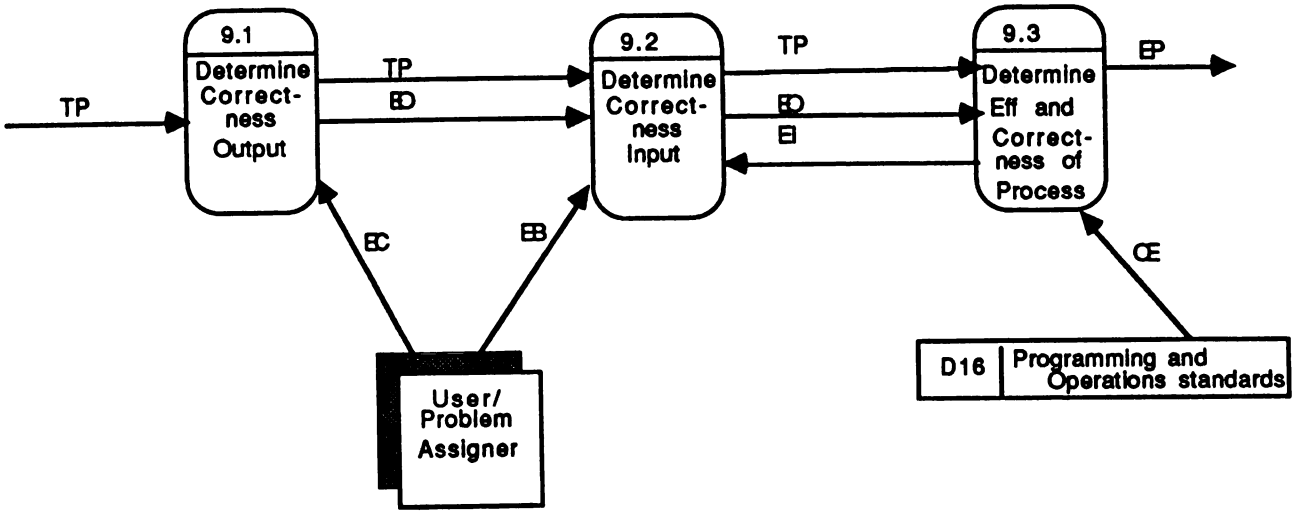
OA - Operation assignment
PE - Program with errors
TD - Test data
TP - Tested program

FIGURE 22: Test program 7.0



Note: Data store D15 Has been broken down into its component parts and listed here as D8.1 - D8.7

FIGURE 11: Debug Program 8.0



EB - Evaluation criteria for input
EC - Evaluation criteria for input
EI - Evaluated input
EO - Evaluated output

EP - Evaluated program
OE - Objective evaluation criteria
TP - Tested program

FIGURE 23: Evaluated Program 9.0

APPENDIX C

APPENDIX C

AN EXPERT SYSTEM AID TO DEBUGGING BASIC PROGRAMS

```
!-----------------------------------------------------------!
!        S A M P L E   K N O W L E D G E   B A S E          !
!                                                           !
!                        D E B U G                          !
!                                                           !
!             an aid to debugging basic programs            !
!                                                           !
!-----------------------------------------------------------!
!
TITLE Debugging Aid to basic programs DISPLAY
!
```

D E B U G

DEBUGGING AID FOR BASIC PROGRAMS

Debugging aid to basic programs is a knowledge base designed to guide the introductory basic student through the process of identifying unknown errors in a basic program. Debug will assist the student in:

- identifying the type of error in the program
- determining the type of language or logical error
- determining the cause of the error
- determining the solution to the error

DEBUG is a prototype system that is restricted in scope to a number of errors commonly made by introductory basic students.

```
     NEXT without FOR          Syntax error
     RETURN without GOSUB      Out of data
     Illegal function call     Undefined line number
     Subscript out of range    Duplicate Definition
     Division by zero          Type Mismatch
     can't continue            Missing operand
     Device Timeout            Device Fault
     FOR without NEXT          Out of paper
     WHILE without WEND        WEND without WHILE
     Advanced Feature          FILE handling ERRORS

The data contained in DEBUG has been extracted from the
Microsoft Corp. BASIC IBM Personal Computer Hardware
Reference Library, and based on the experience of the
Shiawassee County Vocational Data Processing Instructor:
Mary Garrett.
!
!
THRESHOLD = 60
CONFIDENCE OFF
!
1.  The error belongs to Logic errors
    1.1  error is program output in error
         1.1.1 correct error by reformatting output
         1.1.2 correct error by reassigning algorithms
         1.1.3 correct error by adding more output lines
               1.1.3.1 correct error by adding print lines
                       to condition
               1.1.3.2 correct error by adding total print
                       lines
         1.1.4 correct error by adding a GO TO line
    1.2  error is program does not stop
         1.2.1 correct error by changing the GO TO
         1.2.2 correct error by checking for trip data input
         1.2.3 correct error by adding end option to menu
    1.3  error is program stops but has no output
         1.3.1 correct error by adding print (output) lines
         1.3.2 correct error by moving print (output) lines
!
2.  The error belongs to language errors
    2.1  NEXT .... FOR error DISPLAY fornext
         2.1.1 correct error by matching FOR and NEXT
               variables
         2.1.2 correct error by proper nesting of FOR
               ...NEXTs
         2.1.3 correct error by matching FOR and NEXT
               statements
         2.1.4 correct error by removing GO TO to a line
               inside the loop
         2.1.5  correct error by clearing FOR..NEXT variable
    2.2  Undefined line number DISPLAY undefined line number
    2.3  RETURN without GOSUB DISPLAY RETURN without GOSUB
```

2.4 Out of data DISPLAY out of data
    2.4.1 correct error by checking for trip data after READ
    2.4.2 correct error by adding trip data to DATA statements
    2.4.3 correct error by matching check with trip data
    2.4.4 correct error by matching variable in check with READ
    2.4.5 correct error by adding to the trip data
    2.4.6 correct error by matching the DATA with the READ variables
2.5 can't continue DISPLAY can't continue
2.6 Advanced Feature DISPLAY advanced feature
2.7 Subscript out of range DISPLAY subscript out of range
    2.7.1 correct error by adding a dimension statement
  2.7.2 correct error by fixing a logic error
    2.7.3 correct error by increasing dimension statement subscipt
    2.7.4 correct error by moving the dimension statement

    2.7.5 correct error by fixing a typing error
2.8 Division by zero DISPLAY division by zero
2.9 Duplicate Definition DISPLAY duplicate definition
    2.9.1 correct error by removing a definition
    2.9.2 correct error by moving a definition
    2.9.3 correct error by changing a goto
2.10 Missing operand DISPLAY missing operand
2.11 Type mismatch DISPLAY type mismatch
    2.11.1 correct error by putting quotes around a string constant
    2.11.2 correct error by putting a $ at the end of a variable name
    2.11.3 correct error by changing print using image
    2.11.4 correct error by changing variable type in a function
    2.11.5 correct error by converting a type
        2.11.5.1 convert a string to number DISPLAY string to number
        2.11.5.2 convert a number to string DISPLAY number to string
2.12 WHILE .... WEND error DISPLAY whilewend
    2.12.1 correct error by matching WHILE and WEND statements
    2.12.2 correct error by removing GO TO to between WHILE and WEND
2.13 Illegal function call DISPLAY illegal function
    2.13.1 correct error by defining PRINT USING image
    2.13.2 correct error by changing values in a string function
        2.13.2.1 assign a value to string variable in function

```
                2.13.2.2 shorten string length variable in
                         string function
          2.13.3 correct error by changing argument in a
                 function
          2.13.4 correct error by changing negative or large
                 subscript
          2.13.5 correct error by not assigning a negative
                 record number
          2.13.6 correct error by not raising a negative
                 number to a power
          2.13.7 correct error by not trying to delete a
                 nonexistent line
     2.14 Device error or out of paper DISPLAY device error
          2.14.1 correct error by turning on printer
          2.14.2 correct error by pressing select button on
                 printer
          2.14.3 correct error by connecting or replacing
                 printer cable
          2.14.4 correct error by putting paper in the
                 printer
     2.15 Syntax error DISPLAY syntax error
          2.15.1 correct error by changing READ and/or DATA
          2.15.2 correct error by adding a space
          2.15.3 correct error by fixing the spelling
          2.15.4 correct error by changing variable name
          2.15.5 correct error by matching parenthesis
          2.15.6 correct error by fixing punctuation
                 2.15.6.1 correct error by using , with READ
                 2.15.6.2 correct error by using ; with  PRINT
                          USING
                 2.15.6.3 correct error by using ; with  INPUT
                          with message
          2.15.7 correct error by correctly assigning values
          2.15.8 correct error by adding a variable after an
                 AND or OR
          2.15.9 correct error by adding missing term in a
                 function
!
!  RULES FOR DETERMINING LOGICAL ERROR
!
RULE  For determining belongs to Logic errors
IF    The program runs
THEN  The error belongs to Logic errors
AND   DISPLAY CHARACTERISTICS OF THE LOGIC ERROR
ELSE  The error belongs to language errors
AND   DISPLAY CHARACTERISTICS OF THE LANGUAGE ERROR
!
RULE  for differentiating Logic error
IF    The error belongs to Logic errors
AND   the program IS has output
THEN  error is program output in error
!
```

```
RULE  for differentiating program output in error
IF    error is program output in error
AND   the program output IS has the correct values
AND   the output does not match the planned output
THEN  correct error by reformatting output
!
RULE  for differentiating program output in error
IF    error is program output in error
AND   the program output IS does not have the correct values
THEN  correct error by reassigning algorithms
AND   DISPLAY algorithms
!
RULE  for differentiating program output in error
IF    error is program output in error
AND   the program output IS some expected output is missing
THEN  correct error by adding more output lines
AND   DISPLAY add print lines to condition
!
RULE  for differentiating program output in error
IF    error is program output in error
AND   the program output IS there is only one line of detail
      output
THEN  correct error by adding a GO TO line
AND   DISPLAY add a GO TO line
!
RULE  for differentiating correct error by adding output
IF    correct error by adding more output lines
AND   the program total lines are not missing
THEN  correct error by adding print lines to condition
AND   DISPLAY add print lines to condition
ELSE  correct error by adding total print lines
AND   DISPLAY adding total print lines
!
RULE  For differentiating Logic error
IF    The error belongs to Logic errors
AND   the program IS does not stop
THEN  error is program does not stop
!
RULE  For differentiating program does not stop
IF    error is program does not stop
AND   the program has a main menu
AND   the menu does not have an end option
THEN  correct error by adding end option to menu
AND   DISPLAY end option to menu
!
RULE  For differentiating program does not stop
IF    error is program does not stop
AND   the program uses INPUT to get new data
AND   after the INPUT does not check for trip data
THEN  correct error by checking for trip data input
AND   DISPLAY check for trip data
!
```

```
RULE  For differentiating program does not stop
IF    error is program does not stop
AND   the program checks for trip data
AND   there is no menu or a menu with an end option
THEN  correct error by changing the GO TO
AND   DISPLAY GO TO
!
RULE  For differentiating Logic error
IF    The error belongs to Logic errors
AND   the program IS no output
THEN  error is program stops but has no output
!
RULE  For differentiating program stops but has no output
IF    error is program stops but has no output
AND   the program does not have print (output) lines
THEN  correct error by adding print (output) lines
AND   DISPLAY add print lines
ELSE  correct error by moving print (output) lines
AND   DISPLAY move print lines
!
!  RULES FOR DIFFERENTIATING LANGUAGE ERROR
!
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE cross
THEN  correct error by proper nesting of FOR ...NEXTs
AND   DISPLAY proper nesting of FOR ... NEXTs
!
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE do not cross
AND   the FOR variable IS does not match connected NEXT
      variable
THEN  correct error by matching FOR and NEXT variables
AND   DISPLAY match FOR ... NEXT variables
!
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE do not cross
AND   the FOR variable IS is not connected to a NEXT
      statement
THEN  correct error by matching FOR and NEXT statements
AND   DISPLAY match FOR and NEXT statements
!
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE do not cross
AND   the FOR variable IS extra NEXT is connected to the FOR
      statement
THEN  correct error by matching FOR and NEXT statements
AND   DISPLAY match FOR and NEXT statements
!
```

```
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE do not cross
AND   the FOR variable IS matches the connected NEXT
      variable
AND   the program has IS a GO TO to a line inside the loop
THEN  correct error by removing GO TO to a line inside the
     loop
AND   DISPLAY removing branch to inside loop
!
RULE  For differentiating NEXT .... FOR error
IF    NEXT .... FOR error
AND   the lines connecting FOR ... NEXT ARE do not cross
AND   the FOR variable IS matches the connected NEXT
      variable
AND   the program has IS no GO TO to a line inside the loop
THEN  correct error by clearing FOR..NEXT variable
AND   DISPLAY clearing FOR..NEXT variable
!
RULE  For differentiating Out of data
IF    Out of data
AND   the error IS is no check for trip data after the READ
THEN  correct error by checking for trip data after READ
AND   DISPLAY correct Out of Data
!
RULE  For differentiating Out of data
IF    Out of data
AND   the error IS is no trip data in the DATA statements
THEN  correct error by adding trip data to DATA statements
AND   DISPLAY correct Out of Data
!
RULE  For differentiating Out of data
IF    Out of data
AND   the error IS is check for trip data not same as in
      DATA
THEN  correct error by matching check with trip data
AND   DISPLAY correct Out of Data
!
RULE  For differentiating Out of data
IF    Out of data
AND   the error IS is variable READ is not variable in Check
THEN  correct error by matching variable in check with READ
AND   DISPLAY correct Out of Data
!
RULE  For differentiating Out of data
IF    Out of data
AND   the error IS not enough data after the trip data
THEN  correct error by adding to the trip data
AND   DISPLAY correct Out of Data
!
```

```
RULE  For differentiating Out of Data
IF    Out of data
AND   the error IS what is being READ doesn't match the DATA
THEN  correct error by matching the DATA with the READ
      variables
AND   DISPLAY matching the DATA with the READ variables
AND   DISPLAY correct Out of Data
!
RULE  For advanced feature
IF    the error message is advanced feature
THEN  advanced feature
AND   DISPLAY advanced feature
!
RULE  For can't continue
IF    the error message is can't continue
THEN  can't continue
AND   DISPLAY can't continue
!
RULE  FOR undefined line number
IF    the error is undefined line number
THEN  undefined line number
AND   DISPLAY undefined line number
!
RULE  For differentiating Subscript out of range
IF    Subscript out of range
AND   there is a dimension statement IS False
THEN  correct error by adding a dimension statement
AND   DISPLAY adding a dimension statement
!
RULE  For differentiating Subscript out of range
IF    Subscript out of range
AND   there is a dimension statement IS True
AND   the value of subscript in error < 1
THEN  correct error by fixing a logic error
AND   DISPLAY subscript out of range by logic error
!
RULE  For differentiating Subscript out of range
IF    Subscript out of range
AND   there is a dimension statement IS True
AND   the value of subscript in error > value in the
      dimension statement
THEN  correct error by increasing dimension statement
      subscript
AND   or correct error by fixing a logic error
AND   DISPLAY picking the value for a dimension statement
AND   DISPLAY subscript out of range by logic error



!
```

```
RULE  For differentiating Subscript out of range
IF    Subscript out of range
AND   there is a dimension statement IS True
AND   the value of subscript in error <= value in the
      dimension statement
AND   DISPLAY typing errors on subscripts
AND   the dimensioned variable IS matches the variable in
     error
THEN  the dimension is being cleared by a CLEAR or other
      statement
AND   correct error by moving the dimension statement
AND   DISPLAY moving the dimension statement
!
RULE  For differentiating Subscript out of range
IF    Subscript out of range
AND   there is a dimension statement IS True
AND   the value of subscript in error <= value in the
      dimension statement
AND   the dimensioned variable IS does not match the
      variable in error
THEN  correct error by fixing a typing error
!
RULE  For differentiating duplicate definition
IF    Duplicate Definition
AND   DISPLAY how to find array being defined twice
AND   duplicate definition IS caused by an array being
      defined twice
THEN  correct error by removing a definition
AND   DISPLAY other than removing a definition
!
RULE  For differentiating duplicate definition
IF    Duplicate Definition
AND   duplicate definition IS caused by an OPTION BASE
      command
THEN  correct error by moving a definition
AND   or correct error by moving the OPTION BASE command
AND   DISPLAY moving the OPTION BASE command
!
RULE  For differentiating duplicate definition
IF    Duplicate Definition
AND   duplicate definition IS caused by something else
AND   DISPLAY the two other causes of duplicate definition
AND   the array was IS set to the default size of 10 before
the definition
THEN  correct error by moving a definition



!
```

```
RULE  For differentiating duplicate definition
IF    Duplicate Definition
AND   duplicate definition IS caused by something else
AND   the array was IS not set to the default size of 10
      before the definition
THEN  correct error by changing a goto
AND   or some other method
!
RULE  For differentiating WHILE .... WEND error
IF    WHILE .... WEND error
AND   the lines drawn ARE each WHILE does not connect to a
WEND
THEN  correct error by matching WHILE and WEND statements
AND   DISPLAY match WHILE and WEND statements
!
RULE  For differentiating WHILE .... WEND error
IF    WHILE .... WEND error
AND   the lines drawn ARE each WHILE connects to a WEND
AND   the program has IS a GO TO to a line inside the loop
THEN  correct error by removing GO TO to between WHILE and
      WEND
AND   DISPLAY removing branch to between WHILE and WEND
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS a PRINT USING image
AND   the PRINT USING image is not defined
THEN  correct error by defining PRINT USING image
AND   DISPLAY PRINT USING image
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS a string function
THEN  correct error by changing values in a string function
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   correct error by changing values in a string function
AND   the value assigned to the function's string variable
      is null
THEN  assign a value to string variable in function
AND   DISPLAY value string



!
```

```
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   correct error by changing values in a string function
AND   the value of the length variable in the function is
      wrong
THEN  shorten string length variable in string function
AND   DISPLAY length string
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS a function other than a
      string
THEN  correct error by changing argument in a function
AND   DISPLAY argument
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS contains an arrayed
      variable
THEN  correct error by changing negative or large subscript
AND   DISPLAY negative subscript
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS a GET or PUT statement
THEN  correct error by not assigning a negative record
      number
AND   DISPLAY GET/PUT
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS a variable raised to a
      power
THEN  correct error by not raising a negative number to a
      power
AND   DISPLAY  negative power
!
RULE  For differentiating Illegal function call DISPLAY
      illegal function
IF    Illegal function call
AND   the error line contains IS command to delete a line or
     lines
THEN  correct error by not trying to delete a nonexistent
      line
AND   DISPLAY delete line
!
```

```
RULE  For differentiating Device error
IF    Device error or out of paper
AND   the printer IS not turned on
THEN  correct error by turning on printer
!
RULE  For differentiating Device error
IF    Device error or out of paper
AND   the printer IS on, but the select button is not on
THEN  correct error by pressing select button on printer
!
RULE  For differentiating Device error
IF    Device error or out of paper
AND   the printer IS on and the select button is on
THEN  correct error by connecting or replacing printer cable !

RULE  For differentiating out of paper
IF    Device error or out of paper
AND   the printer IS the paper light is on
THEN  correct error by putting paper in the printer
AND   DISPLAY out of paper
!
RULE  For differentiating syntax error
IF    Syntax error
AND   the syntax error is on a DATA statement
THEN  correct error by changing READ and/or DATA
AND   DISPLAY correct DATA syntax error
!
RULE  For differentiating syntax error
IF    Syntax error
AND   DISPLAY a space may be needed
AND   there is no space between two variables/commands
THEN  correct error by adding a space
!
RULE  For differentiating syntax error
IF    Syntax error
AND   you mistyped the name of a command
THEN  correct error by fixing the spelling
!
RULE  For differentiating syntax error
IF    Syntax error
AND   DISPLAY keywords can cause syntax errors
AND   there is a reserved word in the variable name
THEN  correct error by changing variable name
!
RULE  For differentiating syntax error
IF    Syntax error
AND   there is not an equal number of "(" and ")"
THEN  correct error by matching parenthesis
AND   DISPLAY matching parenthesis
!
```

```
RULE  For differentiating syntax error
IF    Syntax error
AND   error line has READ or PRINT USING or INPUT with
      message
THEN  correct error by fixing punctuation
!
RULE  For differentiating punctuation errors
IF    correct error by fixing punctuation
AND   the error line IS has a READ statement
THEN  correct error by using , with READ
AND   DISPLAY using a , with READ
!
RULE  For differentiating punctuation errors
IF    correct error by fixing punctuation
AND   the error line IS has a PRINT USING statement
THEN  correct error by using ; with PRINT USING
AND   DISPLAY using a ; with PRINT USING
!
RULE  For differentiating punctuation errors
IF    correct error by fixing punctuation
AND   the error line IS has an INPUT with message statement
THEN  correct error by using ; with INPUT with message
AND   DISPLAY using a ; with INPUT with message
!
RULE  For differentiating syntax error
IF    Syntax error
AND   there is an expression to the left of an equal sign
THEN  correct error by correctly assigning values
AND   DISPLAY how to correctly assign values
!
RULE  For differentiating syntax error
IF    Syntax error
AND   there is constant to the left of an equal sign
THEN  correct error by correctly assigning values
AND   DISPLAY how to correctly assign values
!
RULE  For differentiating syntax error
IF    Syntax error
AND   the error was on an IF statement
AND   DISPLAY how to check AND and OR
AND   an AND or OR is missing a value
THEN  correct error by adding a variable after an AND or OR
!
RULE  For differentiating syntax error
IF    Syntax error
THEN  correct error by adding missing term in a function
AND   or correct error by other
AND   DISPLAY correct syntax error by other
!
```

```
RULE  For differentiating type mismatch
IF    Type mismatch
AND   type of values you are working with IS strings
AND   you are assigning a constant
AND   the constant does not have quotes around it
THEN  correct error by putting quotes around a string
      constant
!
RULE  For differentiating type mismatch
IF    Type mismatch
AND   type of values you are working with IS strings
AND   the variable being assigned the string does not end
      with a $
THEN  correct error by putting a $ at the end of a variable
   name
AND   DISPLAY put $ on all
!
RULE  For differentiating type mismatch
IF    Type mismatch
AND   error was with a PRINT USING
THEN  correct error by changing PRINT USING image
AND   DISPLAY correcting PRINT USING image
!
RULE  For differentiating type mismatch
IF    Type mismatch
AND   error from function (SWAP, LEFT$, ...) except IF and
      LET
AND   type of values you are working with IS numbers
THEN  correct error by changing variable type in a function
AND   DISPLAY changing variable type in a function
AND   DISPLAY number to string
!
RULE  For differentiating type mismatch
IF    Type mismatch
AND   error from function (SWAP, LEFT$, ...) except IF and
      LET
AND   type of values you are working with IS strings
THEN  correct error by changing variable type in a function
AND   DISPLAY changing variable type in a function
AND   DISPLAY string to number
!
RULE  For differentiating type mismatch
IF    Type mismatch
AND   error from function (SWAP, LEFT$, ...) except IF and
      LET
AND   type of values you are working with IS mixed
THEN  correct error by changing variable type in a function
AND   DISPLAY changing variable type in a function
AND   DISPLAY string to number
AND   DISPLAY number to string
!
```

```
RULE  For differentiating type mismatch
IF    Type mismatch
AND   type of values you are working with IS mixed
THEN  correct error by converting type
!
RULE  For differentiating converting type
IF    correct error by converting type
AND   You are using IS string with numbers
THEN  convert a string to number
AND   DISPLAY string to number
!
RULE  For differentiating converting type
IF    correct error by converting type
AND   You are using IS number with strings
THEN  convert a number to string
AND   DISPLAY number to string
!
DISPLAY syntax error
A syntax error is what the computer uses to say that it has
no better way of describing the error.  One of the most
common causes of a syntax error is a typing error.  Make
sure that all commands on the line with the error are
correctly spelled.
!
DISPLAY matching parenthesis
If the number of left parenthesis isn't the same as the
number of right ones, then you must find where the extra
or missing one is.  When nesting commands and subscripted
variables it is easy to forget to add a ).  Start from the
inside and work your way out, writing down the steps on
paper if needed.  This way you can find where you are
missing or have an extra parenthesis.  Here is an example:

      (Y*((3+4)/S(X))/5

          (3+4)                start from the inside
          (3+4)/S(X)           and work your way out
         ((3+4)/S(X))
        Y*((3+4)/S(X))
      (Y*((3+4)/S(X))/5   <-- Missing a right ).  We can
      (Y*((3+4)/S(X)))/5      then add the missing ).
!
DISPLAY using a , with READ
There are two ways to get syntax error with a READ.
One is to just have a READ without specifying what to READ
the data into.  The other is to use the wrong type of
punctuation with the READ command.  Here is an example:

  10 READ A$;B$;C$    should be   10 READ A$,B$,C$
!
DISPLAY using a ; with PRINT USING
The first value after a PRINT USING statement is the
```

format in the form of a string. The format must be separated from the other values to be printed using a ;. If you use a comma, it will result in a syntax error. Here is an example of the correct way to use PRINT USING:

```
   100 PRINT USING"    ####.##     \      \";N,A$

            or

   10  I$ = "    ####.##     \      \"
    .
    .
    .
   100 PRINT USING I$;N,A$
!
```

DISPLAY using a ; with INPUT with message
Whenever you use the INPUT with Message form of INPUT, you must place a ; between the message and the first variable that will be input. If you omit the ; or use a comma, the result will be a syntax error. Here is an example of the correct way to use INPUT with a message:

```
   100 INPUT "what is your age"; A


   200 INPUT "Enter starting date , ending date";SD,ED
```

!
DISPLAY how to correctly assign values
A common error to beginning students is to place things on the wrong side of the equal sign. Whatever is on the left side of the = is replaced with what is on the right side after being evaluated. You can not normally have a command on the left side of the = with the exception of LET and MID$. Example:

```
  10 Y + Z = X
```

This will not work. If you switch the left and right sides you will get:

```
  10 X = Y + Z
```

and this will cause the computer to replace the old value of X with the sum of Y and Z when line 10 is executed. You also can not assign a value to a constant.
!
DISPLAY how to check AND and OR
If you are using AND or OR, then you might be using it incorrectly. Here is an example of how it can be used incorrectly to cause a syntax error:

```
   10 IF A<B OR >C THEN 100
```

There must be an expression on both sides of an AND and OR. The line above could be corrected by changing it to:

```
   10 IF A<B OR A>C THEN 100
```

```
!
DISPLAY correct syntax error by other
If none of the other possibilities are true, then you need
to look up the command that the error occurred on, and check
the syntax of the command.  It is also possible that you
put two commands on one line without properly separating
them with a :.

One common mistake is leaving out a term from a command.
Here is an example:
    100 IF LEFT$(A$) = "Y" THEN 10
This should be corrected as follows:
    100 IF LEFT$(A$,1) = "Y" THEN 10

!
DISPLAY correct DATA syntax error
A syntax error on a data statement is caused by trying to
read a number, when the information on the DATA line
contains a non-number.

You should first find out the location of the READ
statement that is giving the problem.  Use TRON/TROFF to
help you find it.

A common cause of this error, is getting the DATA out of
sync with the READ.  Check the data statements, and if a
string you wish to read contains either a "," or a ":" then
enclose that part of the data in quotes.  Example:

   10 Read T$
  100 DATA "Subject: Computers"
!
DISPLAY a space may be needed
BASIC requires a space between most variables, keywords
and numbers.  If you run things together, the computer will
not be able to pull them apart and break it down.  A space
is not needed after a number, and after a special symbol
such as <>()=+-*/^;.  Check the line with the error and see
if you need a space.

Here are a few examples:
   Incorrect:                  Minimum spacing:
   10 FORA=1TO3                10 FOR A=1TO 3
   20 PRINTA;A*A               20 PRINT A;A*A
   30 NEXTA                    30 NEXT A
!
DISPLAY keywords can cause syntax errors
Sometimes if you have a reserved word inside of a variable
name, it will cause a syntax error.  In some instances it is
allowable to have a reserved word in a variable name.  A
reserved word in a variable will give the most trouble if
it starts the variable name.  Here is the list of all the
```

reserved words in BASIC. Check and see if any of the variables on the line with the error contains one.

ABS	AND	ASC	ATN	AUTO	BEEP
BLOAD	BSAVE	CALL	CDBL	CHAIN	CHR$
CINT	CIRCLE	CLEAR	CLOSE	CLS	COLOR
COM	COMMON	CONT	COS	CSNG	CSRLIN
CVD	CVI	CVS	DATA	DATE$	DEF
DEFDBL	DEFINT	DEFSNG	DEFSTR	DELETE	DIM
DRAW	EDIT	ELSE	END	EOF	EQV
ERASE	ERL	ERR	ERROR	EXP	FIELD
FILES	FIX	FN	FOR	FRE	GET
GOSUB	GOTO	HEX$	IF	IMP	INKEY$
INP	INPUT	INPUT#	INPUT$	INSTR	INT
KEY	KILL	LEFT$	LEN	LET	LINE
LIST	LLIST	LOAD	LOC	LOCATE	LOF
LOG	LPOS	LPRINT	LSET	MERGE	MID$
MKD$	MKI$	MKS$	MOD	MOTOR	NAME
NEW	NEXT	NOT	OCT$	OFF	ON
OPEN	OPTION	OR	OUT	PAINT	PEEK
PEN	PLAY	POINT	POKE	POS	PRESET
PRINT	PRINT#	PSET	PUT	RANDOMIZE	READ
REM	RENUM	RESET	RESTORE	RESUME	RETURN
RIGHT$	RND	RSET	RUN	SAVE	SCREEN
SGN	SIN	SOUND	SPACE$	SPC(	SQR
STEP	STICK	STOP	STR$	STRIG	STRING$
SWAP	SYSTEM	TAB(	TAN	THEN	TIME$
TO	TROFF	TRON	USING	USR	VAL
VARPTR	VARPTR$	WAIT	WEND	WHILE	WIDTH
WRITE	WRITE#	XOR			

!
!

DISPLAY type mismatch

A type mismatch error is caused by trying to use one type variable when the computer is expecting a different type. The two most common errors, are mixing a string with a numeric value, and mixing a numeric value with a string. Another possible cause is trying to SWAP two variables of different type. If you wish to transfer from one type to another, then use the conversion functions VAL and STR$. Here are some examples that will cause this error.

Incorrect	Correct	Assuming
A = "JOE"	A$ = "JOE"	
B$ = A	B$ = STR$(A)	You want to store the string equivalent of the value in A.
A = 3 / C$	A = 3 / VAL(C$)	C$ in known to contain a number, and you wish to divide 3 by that number.

```
!
DISPLAY put $ on all
You will need to put the $ after all string variables.
Make sure that you add the $ to all locations where you
have string variables in your program.  You can
also use DEFSTR to make a variable into a string without
the $, but if you do this, you must treat the variable
as if the variable had a $ at the end of it.
!
DISPLAY correcting PRINT USING image
A type mismatch error will occur with a PRINT USING if the
format doesn't match the type of variables being printed.
Check your format, and compare the types that it defines
with those of the variables.  Also, some characters have
special meaning, and if you wish to display them, you must
use the _ (underscore) character.
      !                 First character of a string
 \n spaces\             A string of n+2 characters
      &                 Variable length string field
      #                 A digit in a number
      +                 beginning or end of ### format
      -                 beginning or end of ### format
     **                 fill leading spaces with asterisks
     $$                 floating $
     **$                floating $ with leading asterisks
    ^^^^                specify exponential format
      _ (underscore)    print next character exactly

If you wish to print something like:
     You have    $100.00 dollars!
Then your format would be:
    "You have $$#####.## dollars_!"
!
DISPLAY changing variable type in a function
Check the reference manual for the type of values that
function expects.  Then use the conversion routines
VAL and STR$ to convert an incorrect type to the
correct type, or use a command that matches the
type of variable you want.
!
DISPLAY string to number
If you have a number in the form of a string, and wish to
convert it to a number, then use the function VAL.
Here is an example:
    10 INPUT"Enter a number";N$
    20 NUMBER = VAL(N$)
    30 PRINT"Number ="; NUMBER
!
DISPLAY number to string
If you have a number, and wish to convert it to a
string, then use the function STR$.  Here is an
example:
```

```
    10 INPUT"Enter a number"; NUMBER
    20 N$ = STR$(NUMBER)
    30 PRINT"Number ="N$
!
DISPLAY  CHARACTERISTICS OF THE LOGIC ERROR


      CHARACTERISTICS OF A LOGIC ERROR

 The program runs, but the results are not what
 the programmer planned or the user expected.

!


DISPLAY  CHARACTERISTICS OF THE LANGUAGE ERROR

         CHARACTERISTICS OF THE LANGUAGE ERROR

 The program does not run, and the computer prints out
 some sort of error message.

!
DISPLAY end option to menu

A menu should always include an option to end the
program.  This is usually the last item on the menu.
Choosing this option should cause the program to
branch to an end routine or a line number containing
the word end.

!
DISPLAY check for trip data

In a program that gets its data from input statements,
there must be a statement after the input statement
to determine if it is time for the program to stop.

This is done by comparing the input variable to some
predetermined value, called trip data, that signals
that the user is through entering data.  A typical
example is:
100 INPUT"How old are you (enter 0 to end)";X
110 If X = 0 then END

If your program uses arrays, make sure your trip check
uses the same element of the array as the input. A
typical example is:
100 PRINT"Enter 0 to end"
110 PRINT"what is the cost of item"X;:INPUT C(X)
120 If C(X) = 0 then 200 :'processing begins at line 200
```

!
DISPLAY GO TO

GO TO statements that go to a line number smaller than the line number of the GO TO statement may set up an endless loop.

GO TO statements that go to a line number smaller than the line number of the GO TO statement should go to a line number that:
1). Is an INPUT statement
2). Is a read statement
3). Is an EOF(#) statement
4). Is a Menu statement
5). GET # statement
If one of these is not the destination of the GO TO statement, determine which of the above is the proper destination for your GO TO statement and change the line number to match that line number.

If you can not find the GO TO line that is causing the problem, type TRON (TRace ON) and then run your program. Each time the computer runs a line of your program, the line number will appear on the screen. Watch for repeating patterns of line numbers. These will show you where your program is looping. The GO TO that is causing the problem will be one of those numbers (usually the largest). Press CTRL BREAK to stop the program and change the GO TO so it goes to one of the five possible destinations given on the previous page. You will probably want to type TROFF (TRace OFF) to turn off the appearance of the line numbers. If the program still does not stop, repeat the use of TRON to find the new loop and correct it.
!
DISPLAY add print lines to condition

If some of the detail output (prints each time you get a new set of input data) does not print even though some does, the most likely cause is that you have IF...THEN statements for handling some of the input data in different ways and that one or more of those routines do not have the print (output) statements as part of the routine. If you can not find the routine that is missing the print (output) statements, type TRON (TRace ON) and then run your program. Each time the computer runs a line of your program, the line number will appear on the screen. Watch for repeating patterns of line numbers that include the input data lines but do not produce output. These will show you where your program is looping without printing the output. Add the print line to the loop before it goes back to get more data. Type TROFF (TRace OFF) before running the program again.

If the line numbers go by too fast to read, you may find it useful to hold down the CTRL and SHIFT keys and press the PRTSC key to print all the output to the screen to the printer. This will make it easier for you to follow the TRON line number output. Press the three keys again to stop sending the output to the printer.

If your program uses arrays and is missing some of the output, check to see if you are incrementing (adding a value to) the loop parameter (the variable following the FOR and the NEXT). This will cause the output to skip some lines as the variable is incremented by both the program code and the NEXT statement.
!
DISPLAY adding total print lines

After the trip data has been reached, the program should print the total lines before it ends. The easiest way to do this is to change the trip data check line so that instead of ending it will go to a line number which is higher than any of the other program lines and print the total line before ending. For example

```
100 If X = 0 then 900
.
.
.
900 Print"the total XXXXXXX is"TOTAL
910 END
```

!
DISPLAY add print lines

Locate the line that contains GO TO the line that inputs the data to be processed. Add a line before the GO TO line that prints the detail output (out put each time data is input). If you can not find the GO TO line, type TRON (TRace ON) and then run your program. Each time the computer runs a line of your program, the line number will appear on the screen. Watch for repeating patterns of line numbers that include the input data lines but do not produce output. These will show you where your program is looping without printing the output. Add the print line to the loop before it goes back to get more data. Type TROFF (TRace OFF) before running the program again.

!
DISPLAY move print lines

Locate the line that prints your output. Locate the line that has the GO TO the line that inputs the data to be

processed. Move the PRINT line to the line before the GO TO line. Delete it from its old line number. If you can not find the GO TO line, type TRON (TRace ON) and then run your program. Each time the computer runs a line of your program, the line number will appear on the screen. Watch for repeating patterns of line numbers that include the input data lines but do not produce output. These will show you where your program is looping without printing the output. Move the print line to the loop before it goes back to get more data. Type TROFF (TRace OFF) before running the program again.

!

DISPLAY add a GO TO line

After a program prints the output line, it must go back to the get input data line to continue processing until all the input data is processed. Locate the line that prints the output and add a line following it that contains GO TO and the line number of the input line. This makes a loop that allows the program to get the input and print the output until all the input is processed.

!
DISPLAY Algorithms

Algorithms are rules or formulae used to determine values. For example:

```
   200   AREA = 3.1416 * RADIUS * 2
```

is the Algorithm for determining the area of a circle. If your format (layout) of the output is correct, but the numbers are wrong, the most likely cause is that one or more of the algorithms are wrong.

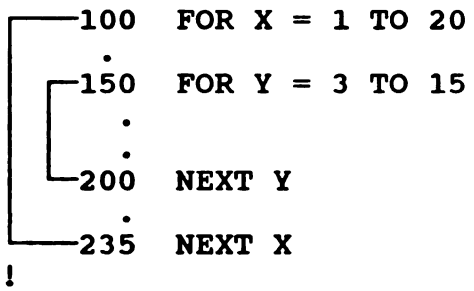
When looking for incorrect algorithms, be certain that you check for variable names that you have spelled wrong. For example:

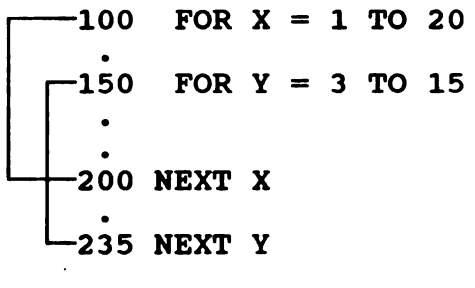
```
   199 RADUS = DIAMETER/2
   200   AREA = 3.1416 * RADIUS * 2
```

will give an AREA of zero since the RADUS and RADIUS are different variables.

!
DISPLAY fornext

Before you attempt to determine the cause of the error, list your program on paper. Draw a line from each FOR statement to its Next statement. Use this listing to answer the questions that follow.



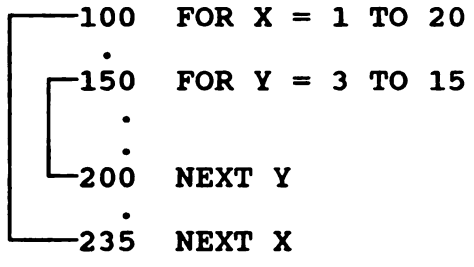


!

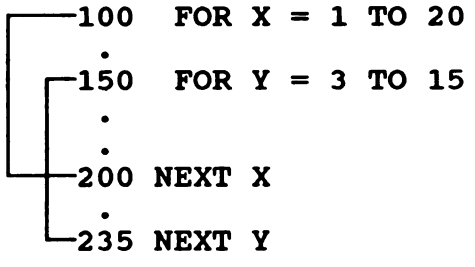
DISPLAY proper nesting of FOR ... NEXTs

When a FOR ... Next loop occurs within another FOR ... NEXT loop, you must be careful that the loops are nicely nested. For example:

NICELY NESTED



IMPROPER NESTING



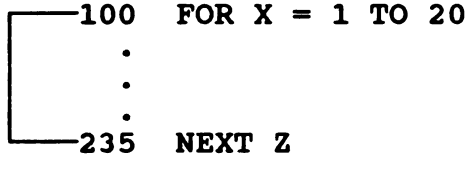
If the lines cross, as in the example of IMPROPER NESTING, change the code so that they are NICELY NESTED.

!

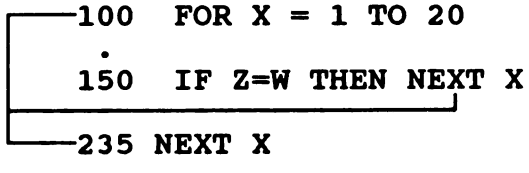
DISPLAY match FOR ... NEXT variables

Every FOR variable name = start value TO end value must have EXACTLY one NEXT variable name. There are two common violations of this rule:

CHANGED VARIABLE NAME



TWO NEXT STATEMENTS



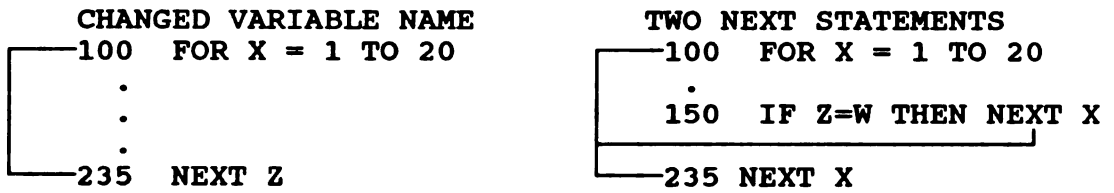
If you have changed variable names, change code so the FOR and NEXT both have the same variable name.

Although some versions of BASIC allow you to have TWO NEXT statements, IBM version 2.0 -3.2 do not. Change your program so the IF statement is THEN GO TO 235 instead of NEXT X.

!

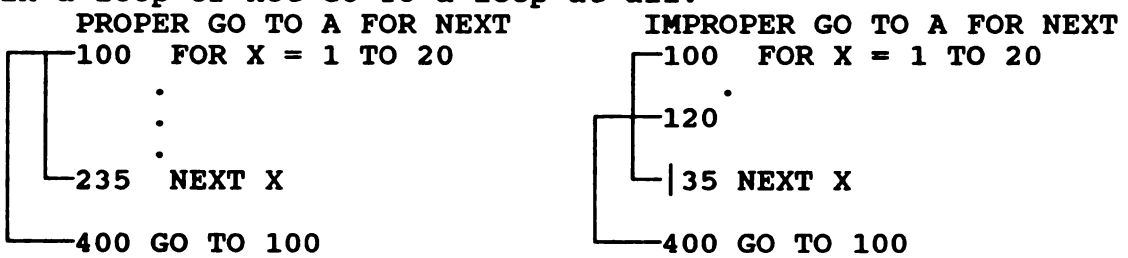
DISPLAY match FOR and NEXT statements

Every FOR variable name = start value TO end value must have EXACTLY one NEXT variable name. There are two common violations of this rule:



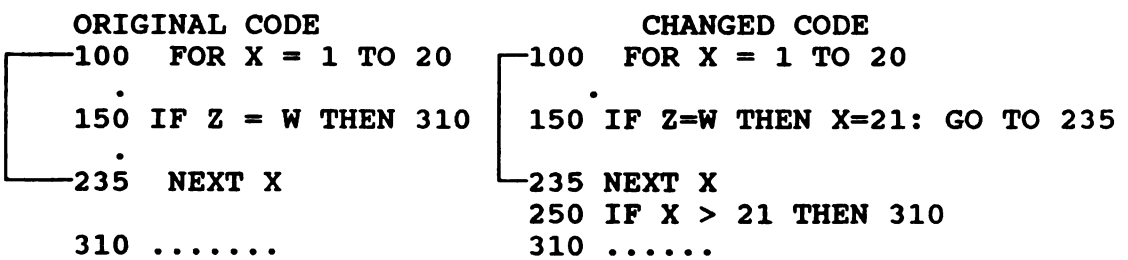
Although some versions of BASIC allow you to have TWO NEXT statements, IBM version 2.0 -3.2 do not. Change your program so the IF statement is THEN GO TO 235 instead of NEXT X.

If you have changed variable names, change code so the FOR and NEXT both have the same variable name.
!
DISPLAY removing branch to inside loop
You must not have a GO TO line outside of the FOR .. NEXT loop in your program that GOes TO a line inside a FOR ... NEXT loop. The GO TO must go to the FOR statement in a loop or not GO TO a loop at all.



If you are not certain if there is a GO TO a line number inside the loop, type TRON (TRace ON) and run your program. The line numbers print on the screen as the program runs and you can see if there is a line number outside the loop before the error occurs. Change the GO TO and type TROFF.
!
DISPLAY clearing FOR..NEXT variable

Sometimes if you have a very long program, even when you have not changed variable names, have not used improper nesting, or do not have two NEXT statements, you may still get either a FOR without NEXT or Next without FOR error. This can occur when you exit from one FOR ... NEXT loop without clearing the variable and move into a new loop. Eventually, the computer runs out of storage places to indicate where to return. To correct this, Change as shown



```
!
DISPLAY RETURN without GOSUB

The program has reached a RETURN statement without
first executing (reaching) a GOSUB.  The usual cause of
this error is not having an END to your program.  Add
an END statement before the first line of the subroutine.

If the error is not corrected by adding an END statement,
type TRON (TRace ON) to display the lines of codes as they
are run.  When the error appears on the screen, the line
numbers of the code run before the error will also appear.
ADD either a GO TO to branch back to the input of data
statement or an END among the lines of code run before the
lines of code that make up the subroutine.  Type TROFF
(TRace OFF) to restore the program to normal run mode.

!
DISPLAY out of data

This problem is normally caused by one of six possible
error conditions:
1). forgetting to put a check for the Trip Data after
    the READ statement
2). forgetting to add the trip data after all the normal
    data to be processed in the DATA statements
3). not having the Trip Data in the data statement match
    the check following the read
4). Not having the variable in the check after the READ
    match the variable in the READ statement
5). Reading into more than one variable at a time, and
    having nothing after the trip data to fill the other
    variables being read in
6). Having an error in the DATA causing it to no longer
    match the READ variables

Check each of these six and correct any that cause a
problem.
!
DISPLAY correct Out of Data

When using the READ....DATA statements, your program should
resemble this code.  Correct your code to follow this
example:

100 READ X$,Y,Z$
110 IF X$ = "END" THEN END
.
.
1000 DATA Joe,34,Doctor,BILL,45,Lawyer,SUE,12,student
1010 DATA Frank,39,teacher,Paul,54,teacher aid
1020 DATA END,-99,END
```

Note that the out of data error would occur if 1020 were 1020 DATA end,-99,END since END does not equal end
!
DISPLAY matching the DATA with the READ variables
You need to first find out how far the DATA matches correctly with the READ. Add some PRINT statements after the READ, then run the program and see at what point incorrect values for DATA start coming up. You can then check the data around the area where the program started reading the incorrect data. Here are some things to look for when checking your DATA:

* Two commas next to each other
* A comma starting or ending a line
* A ":" or a "," as part of a data element you want to read (If you wish to read a : or , then enclose that data element in "quotes."

!
DISPLAY advanced feature

You have chosen an advanced feature such as PAINT or LINE or CIRCLE or COLOR or some other KEY word or expression that can only be used with advanced BASIC. SAVE your program. Type SYSTEM and press enter. You will now be back at the DOS level. Type BASICA (BASIC with Advanced features) and press enter. LOAD your program and run it again. You will not get an Advanced feature error if you are using BASICA.
!
DISPLAY can't continue

You have typed CONT and the program can not start up again from wherever it had stopped. On IBM basic, you can not press the break key when you are LISTing your program and start it up again by typing CONT. If you want to stop a LIST, hold down the CTRL key and press NUM LOCK. Start up again by pressing ENTER.

If you have pressed CTRL Break to look at your program, you must not type a line number or EDIT. If you do, you will not be able to start your program up again by typing CONT. If you try, you will get a can't continue error.

To correct a can't continue error, just RUN your program again from the start (type RUN and press enter).

!
DISPLAY undefined line number
You have a branching statement (GO TO or GOSUB) that tells the program to branch (GO TO) a line number, but the line number to which it is supposed to go does not exist!!.

List your program and find the correct line number. If you are certain you typed in the line to which the program should branch, but it is not there, you should not only type in the line, but you should also check to see if you accidently typed in the wrong line number when you entered the line. Common typing errors for line numbers include typing a 0 for a 9 (or a 9 for a 0); dropping a repeated digit (4000 instead of 40000); and reversing digits (typing a 1223 instead of 1232). If you have typed the line number incorrectly, be certain to remove the incorrect line and retype any lines it may have replaced before you try to run your program again.
!
DISPLAY subscript out of range
A subscript out of range error can be caused many different things. One cause would be misspelling a function such a LOG(), thus causing the computer to think that you are using a subscripted variable when you wished to use the function. Select the option for a typing error if this is what happened.

Some functions, such as TAB require that there is no space between the function name and the "(". A common error is to include a space after the word TAB. If this is what happened, then select the option for a typing error.

Without using a dimension (DIM) statement, you imply to computer a DIM variable(10). This will allow you to use any integral value from 0-10 as the subscript, unless if you have "OPTION BASE 1" in your program, which will set the lowest allowable subscript at 1.
!
DISPLAY typing errors on subscripts
Double check that the variables in the dimension statement are spelled the same, and have the same number of subscripts within the (...). If an array was defined as:

```
10 DIM A(20,20)
```

You would not be able to access A(X), but you could access A(X,Y). If one of the variables is spelled differently, or has a different number of subscripts, then does not match.
!
DISPLAY adding a dimension statement
If you are using an array, you should include a dimension (DIM) command. The dimension is optional if it is a single subscripted array and will not exceed 10, but it is good programming practice to include one even if the array will not exceed 10.

When determining the size to use, for an array which you do not know the maximum subscript needed, it can be set up to be easy to change as follows:

```
 10 LN=20:OPTION BASE 1
 20 DIM FIRST$(LN),LAST$(LN)
100 N=0:PRINT"Type X,X when finished."
110 N=N+1 : INPUT"Lastname, First";LAST$(N),FIRST$(N)
120 IF LAST$(N)="X" AND FIRST$(N)="X" THEN N=N-1:GOTO 200
130 IF N=LN THEN PRINT"Sorry, no more room.":GOTO 200
140 GOTO 110
200 ...
```

The size of the array is determined in the first line. If you decide to change the size of the array, you need only change the value of LN in line 10.

OPTION BASE 1 is optional, and it's only advantage is that it saves memory by not allowing you access to element 0 in an array. If you have double or triple subscripted arrays, this can save a large amount of memory.

!

DISPLAY picking the value for a dimension statement

You need to either increase the value in the dimension statement for the variable that caused the error, or change your program to keep from going outside the range of the dimension statement. Take the following program as an example:

```
 20 DIM FIRST$(20),LAST$(20)
100 N=0:PRINT"Type X,X when finished."
110 N=N+1 : INPUT"Lastname, First";LAST$(N),FIRST$(N)
120 IF LAST$(N)="X" AND FIRST$(N)="X" THEN N=N-1:GOTO 200
140 GOTO 110
200 ...
```

With the above program, if the person tries to enter in 21 names, the computer will respond with a subscript out of range error. However, by adding two lines and changing a third, we will eliminate this problem, as follows:

```
 10 LN=20:OPTION BASE 1
 20 DIM FIRST$(LN),LAST$(LN)
100 N=0:PRINT"Type X,X when finished."
110 N=N+1 : INPUT"Lastname, First";LAST$(N),FIRST$(N)
120 IF LAST$(N)="X" AND FIRST$(N)="X" THEN N=N-1:GOTO 200
130 IF N=LN THEN PRINT"Sorry, no more room.":GOTO 200
140 GOTO 110
200 ...
```

OPTION BASE 1 is optional, and simply helps to save memory by telling the computer we will not be accessing FIRST$(0), or LAST$(0). The change in line 20 is made so that if the array size ever needs to be changed, it can be changed easily, simply by changing the value of LN in line 10. Line 130 will stop the person from entering too many names, thus keeping N from going outside the bounds of the array.

It is also possible that the error may have been caused by some type of logic error. If so, make use of TRON/TROFF and PRINT statements to find out what is causing it to go outside the limit.
!

DISPLAY subscript out of range by logic error
If your program is trying to access an array element that it should not access to figure out why, here are a few simple steps that you can take to help find the cause of the error.

1. Place a PRINT before each access of the subscript, and print the subscript value, and the contents of the array element if it will help.
2. Place a PRINT after each calculation that is used in calculating the subscript value. This can be useful if you are using some sort of hashing technique, and have an error in your hashing formula.
3. If your program goes through many lines while calculating the value for the subscript, then use TRON and TROFF to turn on and off listing the lines it is accessing.

!
DISPLAY moving the dimension statement
If your array is shrinking in size, then you probably need to move it. Some commands must be given in a certain order. A CLEAR command will clear out all variables, including any definitions for an array. There are other commands that may cause this to happen such as improper use of the COMMON and CHAIN commands or RUN your program from a line other than the start. Also, make sure that you are not using ERASE to delete the array.

If you have a CLEAR statement in your program, make it the first thing to be executed.
!
DISPLAY division by zero
A division by zero error will not stop execution of a program in IBM basic. It is there as a warning which the computer displays if you attempt to divide by zero, or raise 0 to a negative exponent. A division by be zero will cause that portion of the expression to be evaluated as machine infinity (+/-).

You might want to make sure that there isn't a CLEAR, or other command wiping out the contents of a variable causing you to try and divide by zero. You can also put in a check as in the following program:

```
   10 PRINT"When finished enter -99"
   20 INPUT"Number";NUMBER
   30 IF NUMBER = -99 THEN 100
   40 TOTAL = TOTAL + NUMBER
   50 COUNT = COUNT + 1
   60 GOTO 20
  100 IF COUNT = 0 THEN PRINT"You did not enter any numbers
      to average." : GOTO 120
  110 PRINT"The average of the numbers is: "; TOTAL / COUNT
  120 END
```

!
DISPLAY duplicate definition
You have tried to redefine the size of an array. There are a number of different possible causes, which include:

* The same array is defined in two DIM statements (or the same DIM statement is executed twice.)

* The program encounters a DIM statement for an array after the default dimension of 10 is established for that array.

* The program sees an OPTION BASE statement after an array has been dimensioned, either by a DIM statement or by default.

!
DISPLAY how to find array being defined twice
The first thing that I recommend you do is to examine your program and see if you are directly defining the array twice. One way to do this is to save your program in ascii (EXAMPLE: SAVE"PROGRAM",A) and load it into a text editor. You can then use global search commands and look for the key word DIM. All DIM statements should be near the top of the program (after a CLEAR statement if you have one.) If you are defining the size of an array for many variables, you might have accidently typed in the same variable twice, even on the same line.
!
DISPLAY other than removing a definition
If you really want to define the array twice (Perhaps to increase the size of the array) you can use the ERASE command to clear out one or more arrays. Remember that, if you do this, the previous contents of the array will be lost (unless you save the array somewhere else.)
!
DISPLAY moving the OPTION BASE command
Move the OPTION BASE command to the beginning of your program. It must be in front of any DIM statements, or

commands that will use arrays. A clear statement will reset it back to the default of OPTION BASE 0.
!
DISPLAY the two other causes of duplicate definition
There are two other common causes of a duplicate definition error. One is initializing the array to something before the DIM statement as in the following example:

```
10 A$(1)="RED":A$(2)="BLUE":A$(3)="GREEN"
20 DIM A$(3)
```

After the computer encounters A$(1)="RED" it notices that it is a new array, and will assign it a default dimension size of 10, and when it reaches line 20 the computer already has it set up for size 10. To solve this sort of problem, move the dimension statement to the beginning of the program.

The other common error is shown in the example below:

```
10 DIM X$(50)
20 CLS
 .   .   .
1000 INPUT"Do you wish to run this again?";Q$
1010 IF LEFT$(Q$,1)="Y" or LEFT$(Q$,1)="y" THEN 10
```

If the person decided to redo the program, the computer will encounter the same dimension statement twice. There are a couple of ways that this can be fixed. You only need change either line 10, or line 1010 in this case. You could change line 10 to:

```
10 CLEAR : DIM X$(50)
```

or change line 1010 to:

```
1010 IF LEFT$(Q$,1)="Y" or LEFT$(Q$,1)="y" THEN RUN
```

or change line 1010 so it jumps to a point after the DIM.

```
1010 IF LEFT$(Q$,1)="Y" or LEFT$(Q$,1)="y" THEN 20
```

!
DISPLAY missing operand
You have an expression in your program that is missing an operand. Having a * (multiply) or the OR operator with nothing after it, are common causes for this error. Also some commands like LOCATE and POKE will give this error if you do not supply all of the operands.

To fix this error, add the missing operand(s) or fix some sort of typing error that may have caused it. You may need to look up the command or operator in the reference

manual. They will have the format of the command and all required operands for that command.
!
DISPLAY whilewend

To attempt to determine the cause of the error, list your program on paper. Draw a line from each WHILE statement to the nearest WEND, starting with the WHILE and WEND that are closest to each other.

```
┌──100  WHILE X <= 20
│    .
│  ┌─150  WHILE Y = 1  AND  Z < 10
│  │   .
│  │   .
│  └─200  WEND
│    .
└──235  WEND
```

!
DISPLAY match WHILE and WEND statements
Every WHILE expression must have EXACTLY one WEND statement. There are three common causes of this error:

```
   MISSING WEND      MISSING WHILE     TWO WEND STATEMENTS
┌─100 WHILE X = 1  ┌─100               ┌─100  WHILE X <= 10
│      .           │   .               │   .
│      .           │   .               │  150  IF Z=W THEN WEND
│      .           │   .               └──────────────────┐
└─235              └─235 WEND          └─235  WEND
```

If you have left out either a WHILE or a WEND, rewrite the code so that each WHILE matches a WEND.

If you have two WEND statements, change your program so the IF statement is THEN GO TO 235 instead of WEND.
!
DISPLAY removing branch to between WHILE and WEND
You must not have a GO TO line outside of the WHILE...WEND loop in your program that GOes TO a line inside a WHILE .. .. WEND loop. The GO TO must go to the WHILE statement in a loop or not GO TO a loop at all.

```
  PROPER GO TO A WHILE....WEND      IMPROPER GO TO WHILE..WEND
┌──┌─100  WHILE X = 1               ┌─100  WHILE X = 1
│  │     .                          │    .
│  │     .                          │  ┌─120
│  └─235  WEND                      │  └─135 WEND
└──400 GO TO 100                    └──400 GO TO 120
```

If you are not certain if there is a GO TO a line number inside the loop, type TRON (TRace ON) and run your program.

```
The line numbers print on the screen as the program runs
and you can see if there is a line number outside the loop
before the error occurs.  Change the GO TO and type TROFF.
!
DISPLAY illegal function
List the line in which the illegal function Call occurred.

If the line has a PRINT USING statement, print the image.
For Example:  100 PRINT USING I$; X, Y, Z
type ? I$  and press enter.  If you get a blank line and OK,
I$ is not defined!  Either add a line defining I$ or move
the existing line so that it is ALWAYS executed before the
PRINT USING statement.

If there is no PRINT USING image or if it is defined, print
any other functions (see pages 4-17 thru 4-22 in the IBM
BASIC manual) until you find the one that is causing the
illegal function Call.  If the argument [inside the ()], is
a variable, print that also.  For example:
   100 Y = SQR(X)
? SQR(X)
ILLEGAL FUNCTION CALL
? X
-27
!
DISPLAY PRINT USING image

Print the image from the PRINT USING image.
For Example:  110 PRINT USING I$; X, Y, Z
type ? I$  and press enter.

OK

If you get a blank line and OK,
I$ is not defined!  Either add a line defining I$
109 I$ = "##.##    ###    ###.###"

or move the existing line so that it is ALWAYS executed
before the PRINT USING statement.  Make
200 I$ = "##.##    ###    ###.###"
  5 I$ = "##.##    ###    ###.###"
!
DISPLAY  value string

Print the variable in the string function.  For example:
If 100  X$ = RIGHT$(C$,1)    then run the program.  When
ILLEGAL FUNCTION occurs, print the variable

? C$

OK
```

Change the code so that C$ (or the variable from your program) has a value before line 100. If there is a chance that C$ (or the variable from your program) might have no value (might be the null set), make the line

```
100 IF C$ = "" THEN X$ = "" ELSE X$ = RIGHT$(C$,1)
```

!

DISPLAY length string

Print the variable in the string function. For example: If 100 X$ = MID$(C$,S,1) then run the program. When ILLEGAL FUNCTION occurs, print the variable

```
? S,C$
```

If S<=0 or C$ is less than S characters long, you would not be able to find the middle starting with the Sth character! Either the value of C$ (or your program's variable) is not being assigned correctly, or the starting position is not being assigned correctly. Change your code so that both of these values are properly assigned. If you are assigning them properly but some of the test data is in error, change the code to test for valid values:

```
100 IF S<=0 or LEN(C$)<S then X$="" ELSE X$=MID$(C$,S,1)
```

!

DISPLAY argument

Print any functions (see pages 4-17 thru 4-22 in the IBM BASIC manual for a list of possible functions) until you find the one that is causing the illegal function Call. If the argument [inside the ()], is a variable, print that also. For example:

```
   100 Y = SQR(X)
? SQR(X)
ILLEGAL FUNCTION CALL
? X
-27
```

If seeing the value of the variable does not show you what the problem is, look up the function in the BASIC manual and compare the value of the variable in your program with the values that are allowed for the argument. Change the code so that values are being assigned correctly. If some of the test data is in error, change the code to test for valid values:

```
   100 If X < 0 then Y = 0 ELSE Y = SQR(X)
```

!

DISPLAY negative subscript
If there are no functions (see pages 4-17 thru 4-22 in the IBM BASIC manual for a list of possible functions), print any subscripted variables until you find the one that is causing the illegal function Call. If the subscript [inside the ()], is a variable, print that also. For example:

```
   100 Y = SUM(X)
? SUM(X)
ILLEGAL FUNCTION CALL
? X
-27
```

Since you can not have a negative value of a subscript, change the code so that values assigned to the subscript are not negative. If some of the test data is in error, change the code to test for valid values:

```
   100 If X < 0 then Y = 0 ELSE Y = SUM(X)
```

!
DISPLAY negative power
If there are no functions (see pages 4-17 thru 4-22 in the IBM BASIC manual for a list of possible functions) and no arrays, print any variables raised to a power until you find the one that is causing the illegal function Call. Print both the variable and the power:

```
   100 Y = X^Z
? X^Z
ILLEGAL FUNCTION CALL
? X
-27
OK
? Z
1.34
OK
```

Since you can not have a negative value raised to a non integer power, change the code so that values assigned to the variable are not negative or the power is an integer. If some of the test data is in error, change the code to test for valid values:

```
   100 If X < 0 then Y = X^INT(Z) ELSE Y = X^Z
```

!
DISPLAY delete line

List the lines that you are attempting to delete.
If any of them do not list, the lines do not exist.
You can not delete a nonexistent line!

Retype your delete command so that you are using lines that exist.

!

```
DISPLAY get/put

Print the variable indicating the record number.
For example:
   100 GET #1, RECORD.NUMBER
? RECORD.NUMBER
-8
OK
Since you can not have a negative value for a record
number, change the code so that values assigned to the
record number are not negative.  If some of the test data
is in error, change the code to test for valid values:
   100 If RECORD.NUMBER < 1 then PRINT"Record number
incorrect":PRINT"Correct data and rerun the program": END
ELSE GET #1, RECORD.NUMBER
!
DISPLAY device error

A device error occurs when the BASIC program attempts
to communicate with one of the hardware devices such as
the printer, disk drive, modem, or cassette recorder.

The most common error is that the printer is not turned
on or that the select button needs to be pressed for the
printer to be ready, although the problem could be that
the printer cable is bad.

After trying to communicate with a device for a preset
number of seconds, BASIC will issue a DEVICE TIMEOUT
error.

If BASIC can tell that there is something wrong with
the hardware, it will issue a DEVICE FAULT ERROR.
!
DISPLAY out of paper

An out of paper error occurs when the BASIC program
attempts to communicate with the printer and fails to
complete the task.

The most common error is that the printer has run out of
paper, but this error can also occur if the printer is not
turned on or that the select button needs to be pressed for
the printer to be ready, although the problem could be that
the printer cable is bad.

Look at the printer and see if the paper light is on.
If so, put paper in the printer or if it has paper, check
to be certain that the paper opens the paper sensor switch.
You may need to move the paper to keep the sensor open.
!
!
```

EXPAND The error belongs to Logic errors

Before you can determine what is causing an error, you must determine the type of error that has happened. Errors in logic occur when the program runs, but the results are not what you expected. Language errors occur when the code does not follow the rules for basic code. Language errors cause the compiler to print out a message to help you determine the source of the error.

!
EXPAND The error belongs to language errors

Before you can determine what is causing an error, you must determine the type of error that has happened. Errors in logic occur when the program runs, but the results are not what you expected. Language errors occur when the code does not follow the rules for basic code. Language errors cause the compiler to print out a message to help you determine the source of the error.

!
EXPAND there is only one line of detail output

Detail output is the type of output where one line is output for each set of data input. For example:

```
100 READ A,B,C
110 IF A=0 THEN END
120 D=A+B/C
130 PRINT A,B,C,D
140 GOTO 100
```

Line 130 is detail output since it prints each time A,B, and C are read.

!
EXPAND correct error by reassigning algorithms

Algorithms are rules or formulae used to determine values. For example:

```
   200   AREA = 3.1416 * RADIUS ^ 2
```

is the Algorithm for determining the area of a circle. If your format (layout) of the output is correct, but the numbers are wrong, the most likely cause is that one or more of the algorithms are wrong.

When looking for incorrect algorithms, be certain that you check for variable names that you have spelled wrong. For example:

```
   199 RADUS = DIAMETER/2
   200   AREA = 3.1416 * RADIUS ^ 2
```

will give an AREA of zero since the RADUS and RADIUS are different variables.
!
EXPAND after the INPUT does not check for trip data

Trip data is special data (like -99 or END OF FILE or sometimes even 0) that could not be "true" data for a program. It is used to signal the program that processing is finished. A program that uses either a READ statement or INPUT to get the data to process usually has a check for some special data (trip data) in the line following the READ or INPUT lines.

!

EXPAND correct error by checking for trip data input
Trip data is special data (like -99 or END OF FILE or sometimes even 0) that could not be "true" data for a program. It is used to signal the program that processing is finished. A program that uses either a READ statement or INPUT to get the data to process usually has a check for some special data (trip data) in the line following the READ or INPUT lines.

!
EXPAND the program checks for trip data
Trip data is special data (like -99 or END OF FILE or sometimes even 0) that could not be "true" data for a program. It is used to signal the program that processing is finished. A program that uses either a READ statement or INPUT to get the data to process usually has a check for some special data (trip data) in the line following the READ or INPUT lines.

!
EXPAND the value of subscript in error
Look at the line on which the error occurred. Find which variable gave the error. If there is more than one subscripted variable on the line, then use PRINT until you find which one gives the error. Example:

```
   100 A(X)=B(Z) * C(Y*2)
```

You could then do, PRINT A(X), and if it returns a value, that is not the one causing the error. Let's say that you found C(Y*2) to be the one in error, you would then "PRINT Y*2", and enter in the value given by the computer.

!

EXPAND value in the dimension statement
Find the dimension statement that defines the array that is giving the error. If there is a variable in the subscript, be sure to give the value of the subscript, and not the contents of the subscript. Example:

```
   10 LN=100 : DIM A(LN), B(LN), C(LN*2)
```

And C(...) was found to be causing the problem. You would then respond with 200 for this question.
!
END

APPENDIX D

APPENDIX D

Checklist for Data Flow Diagrams in Instructional Design

I. Have the course goals been identified?
II. Have course goals been separated into nine or less logical subgroups called course components?
    A. Have the components been verified by a subject matter expert?
    B. Have interrelationships between the components been identified and verified?
    C. Has a DFD been made of the course components and their interrelationships?
        1. Have information and processes between components been explained to the level required for development and implementation?
        2. Has the correctness of the data flow diagram been verified by
            a). a subject content area expert?
            b). an expert content area teacher?
            c). a potential instructor?
III. Have the goals of each component been determined?
IV. Have the component goals been separated into logical groupings called units?
    A. Have the units been verified by a subject matter expert?
    B. Have interrelationships between the units been identified and verified?
    C. Has a data flow diagram been made of the units and their interrelationships?
        1. Have information and processes between units been explained to the level required for implementation?
        2. have necessary knowledge, strategies, and reference materials been shown and explained?
        3. have points where expert help may be necessary been identified?
        4. Has the correctness of the data flow diagram been verified by
            a). a subject content area expert?
            b). an expert content area teacher?
            c). a potential instructor?
V. Have the goals of each unit been identified?
    A. Have the goals been verified by a subject matter expert?
    B. Have interrelationships between the goals been identified and verified?

C. Has a data flow diagram been made of the goals and their interrelationships?
   1. Have information and processes between goals been explained to the level required for implementation?
   2. have necessary knowledge, strategies, and reference materials been shown
   3. have points where expert help may be necessary been identified?
   4. Has the correctness of the data flow diagram been verified by
      a). a subject content area expert?
      b). an expert content area teacher?
      c). a potential instructor?

VI. Have the enabling objectives been identified?
   A. Have the enabling objectives been verified by a subject matter expert?
   B. Have interrelationships between the enabling objectives been identified and verified?
   C. Has a data flow diagram been made of the enabling objectives and their interrelationships?
      1. Have information and processes between enabling objectives been explained to the level required for implementation?
      2. have necessary knowledge, strategies, and reference materials been shown and explained?
      3. have points where expert help may be necessary been identified?
      4. Has the correctness of the data flow diagram been verified by
         a). a subject content area expert?
         b). an expert content area teacher?
         c). a potential instructor?

VII. Have the tasks involved in reaching the objectives been identified?
   A. Components?
      1. Starting and ending points?
      2. processes?
         a). information needed?
         b). interrelationship between processes?
         c). points were expert help is needed?
   B. Have the tasks and components been verified by a subject matter expert?
   C. Have interrelationships between the components been identified and verified?
   D. Has a data flow diagram been made of the tasks and components their interrelationships?
      1. Have information and processes between tasks and components been explained to the level required for implementation?

2. have necessary knowledge, strategies, and reference materials been shown?
3. have points where expert help may be necessary been identified?
4. Has the correctness of the data flow diagram been verified by
   a). a subject content area expert?
   b). an expert content area teacher?
   c). a potential instructor?

VIII. Does the developer or instructor feel confident that the lessons can be developed (using Gagne's (1985) elements of instruction, Reigeluth's (1982) Elaboration Theory, or Landa's (1982) Snowball approach)?

APPENDIX E

APPENDIX E

PAIN AND ITS TREATMENT IN VETERINARY MEDICINE: AN INTERACTIVE VIDEO COURSE

This appendix contains part of the design of an interactive video course using data flow diagrams.

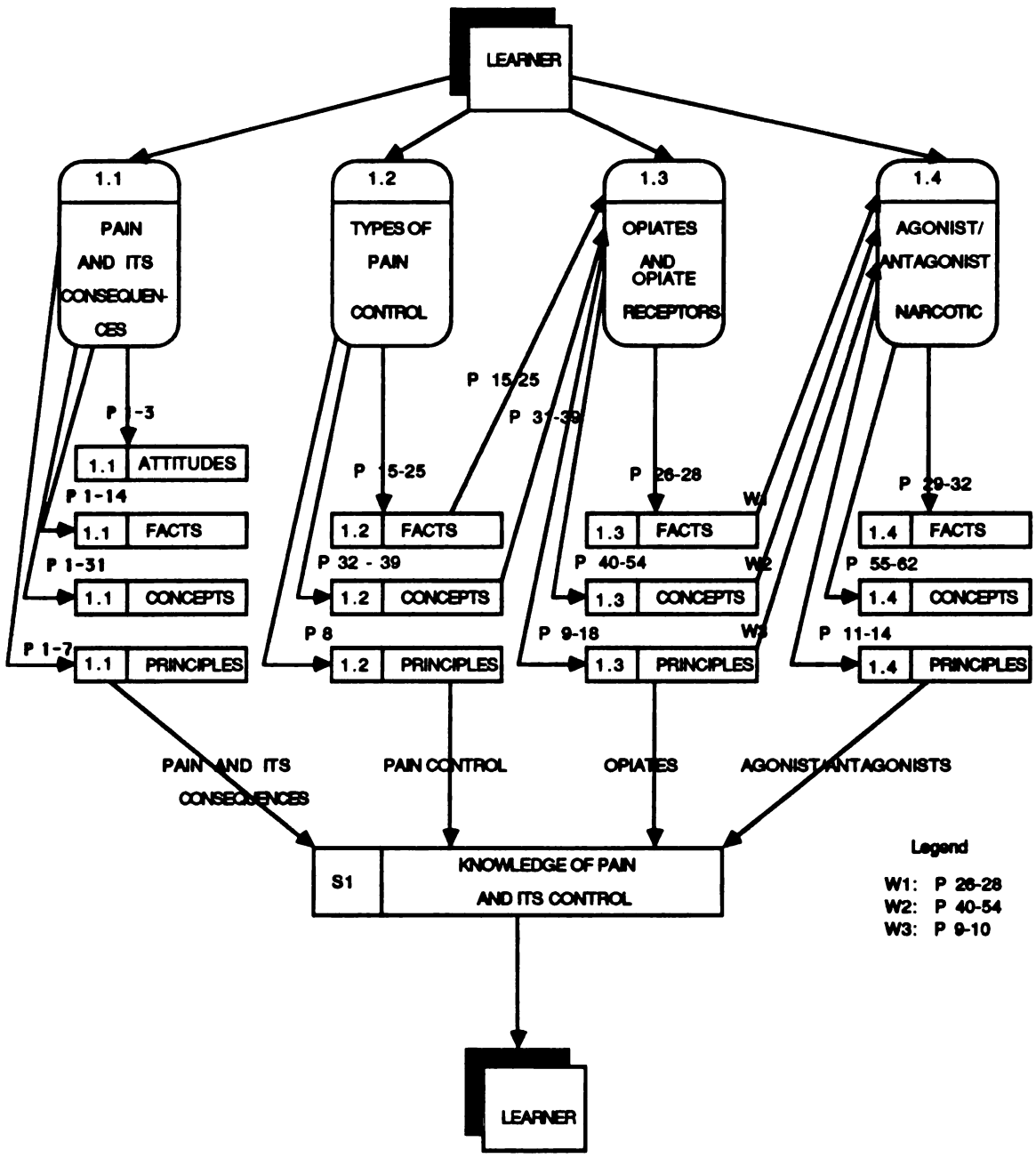


Figure 24: PAIN AND ITS TREATMENT IN VETERINARY MEDICINE 1.0

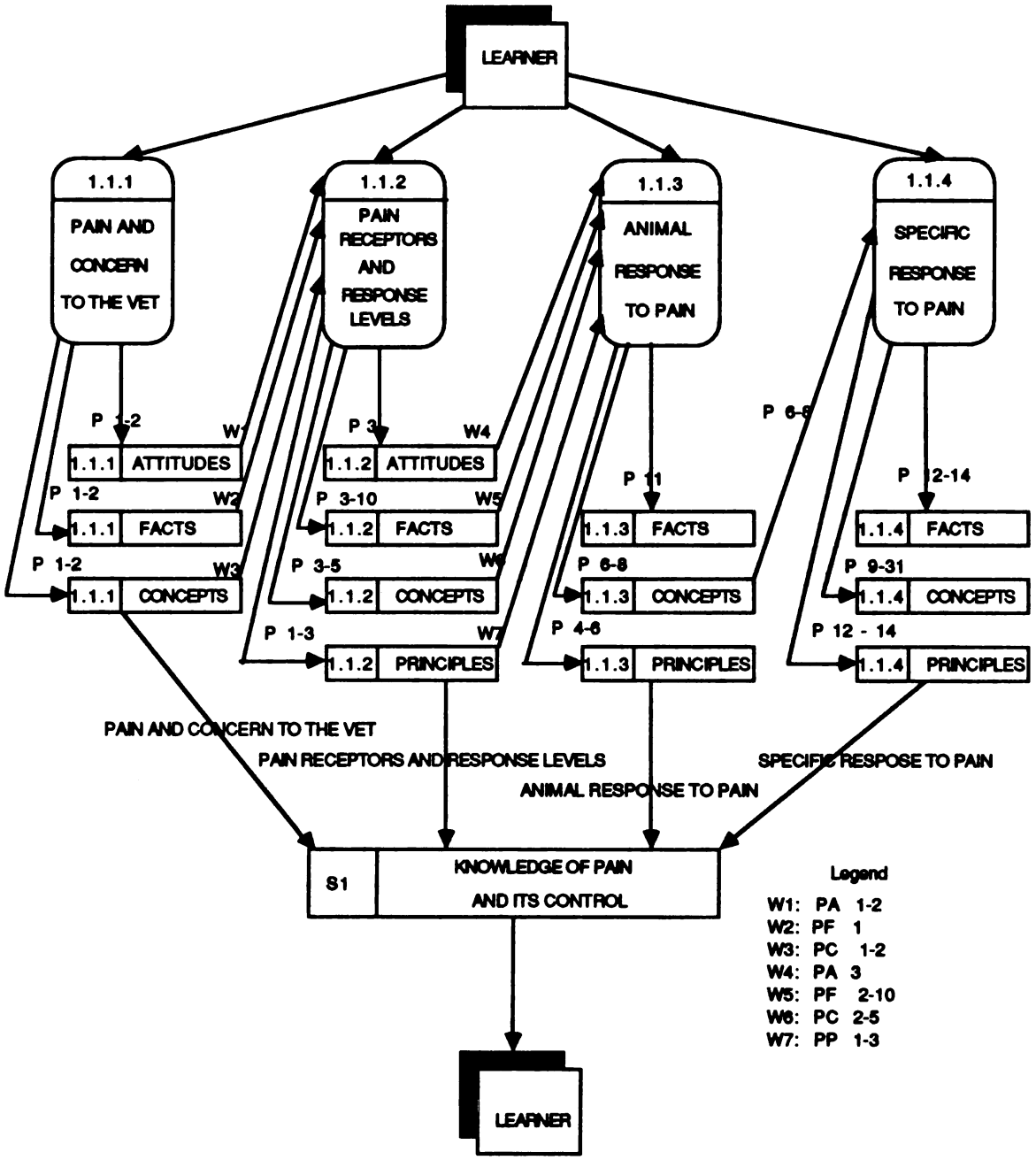


Figure 25: PAIN AND ITS CONSEQUENCES 1.1

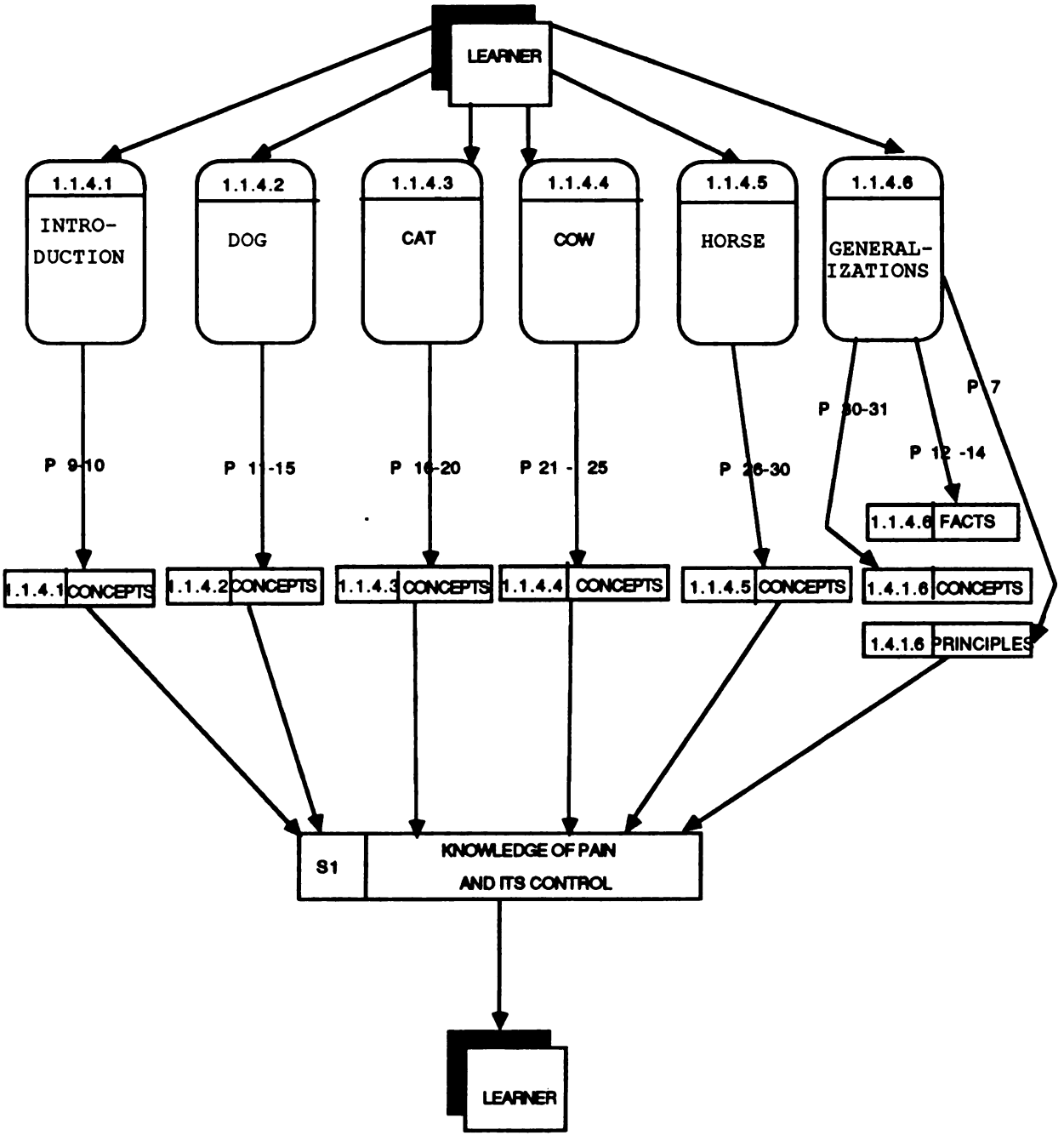


Figure 26: SPECIFIC RESPONSES TO PAIN 1.1.4

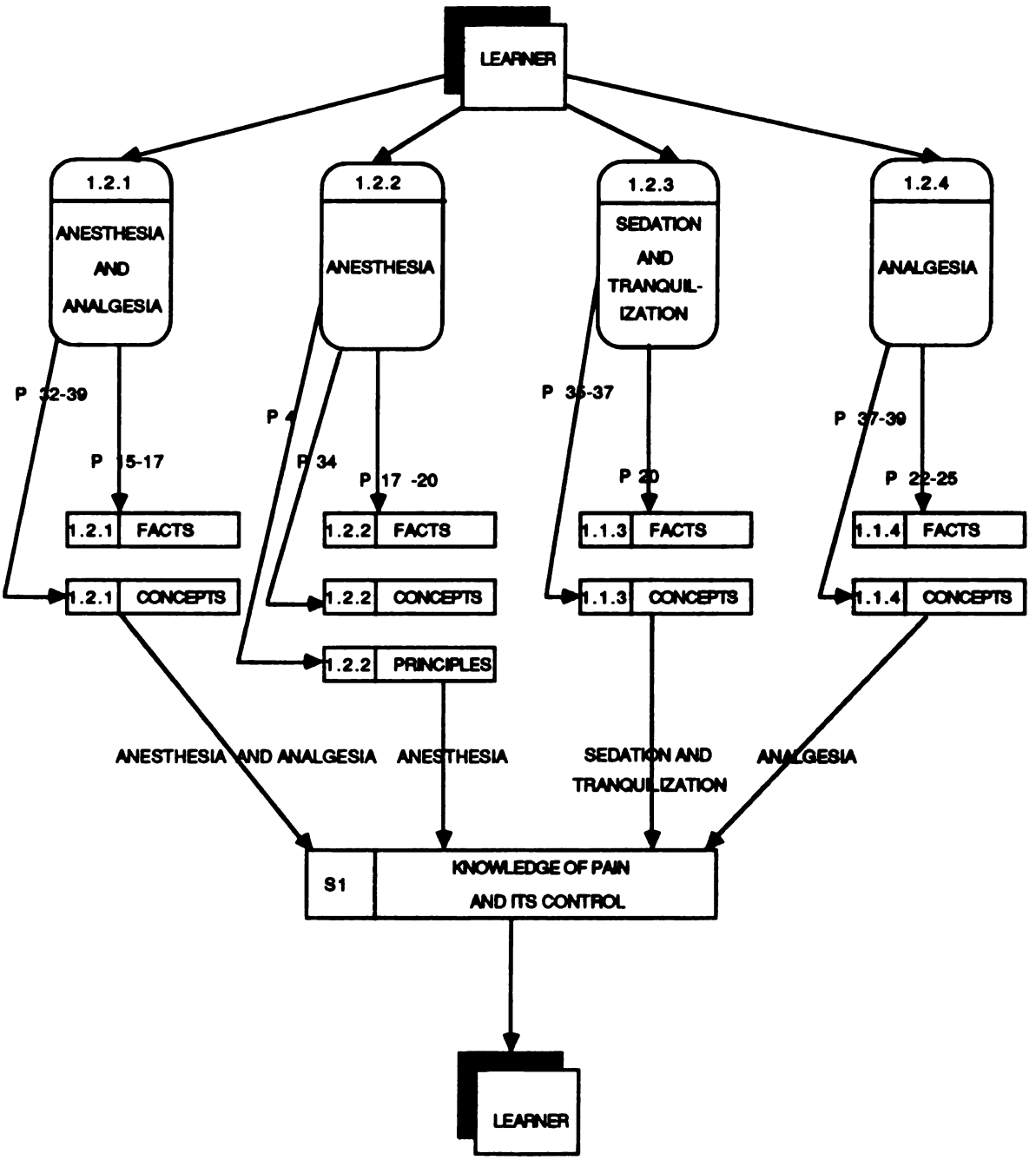


Figure 27: TYPES OF PAIN CONTROL 1.2

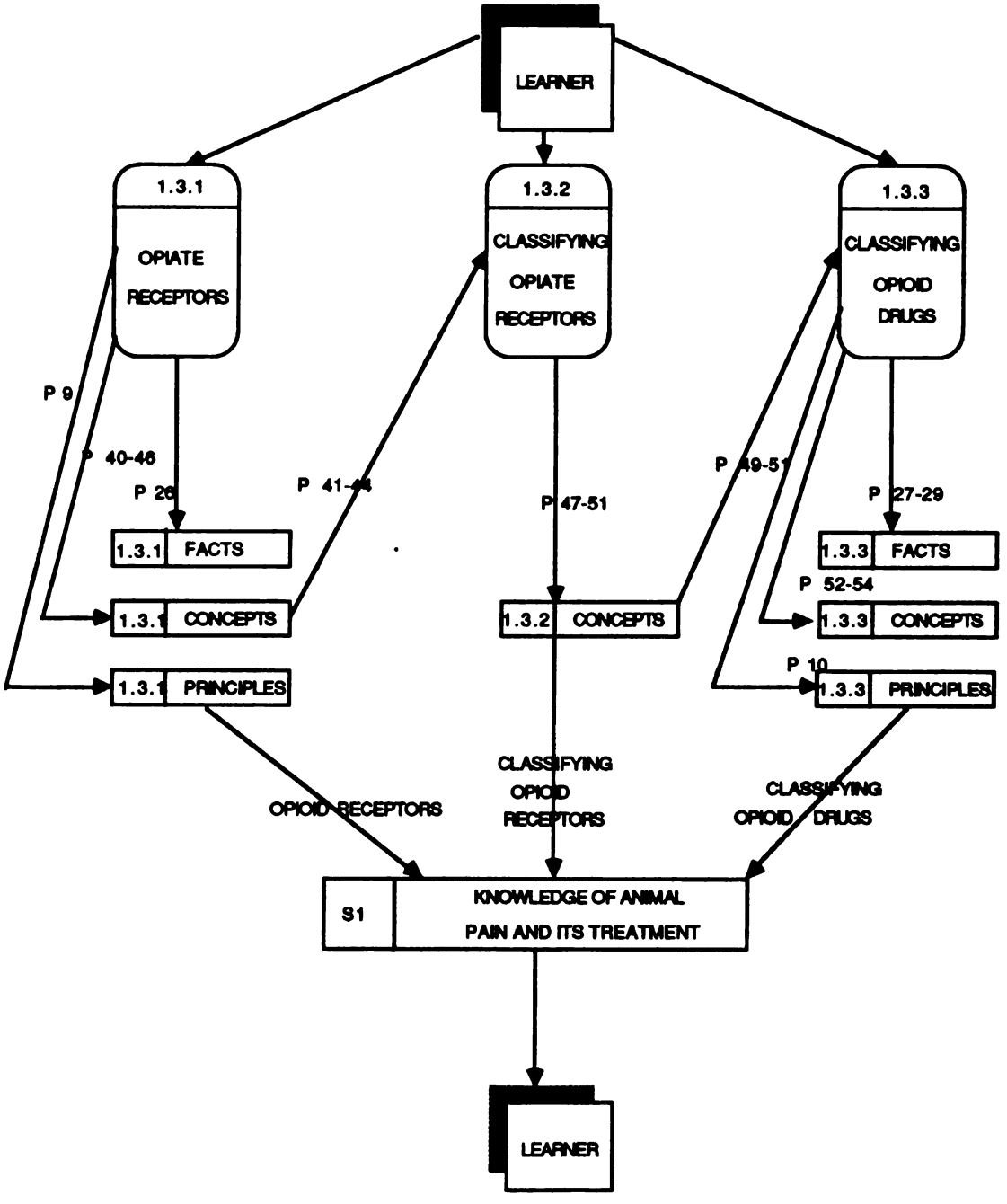


Figure 28: OPIATES AND OPIATE RECEPTORS 1.3

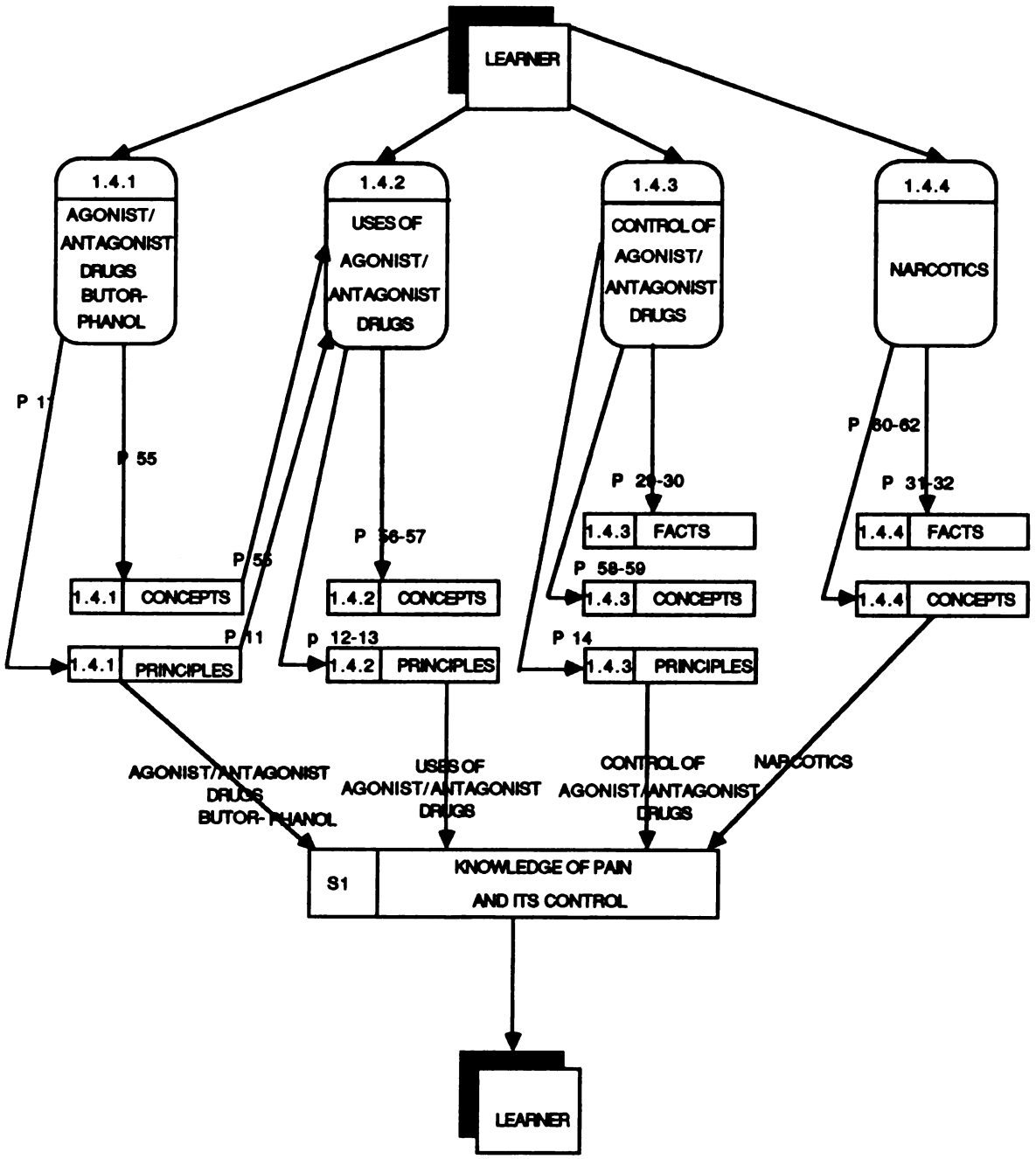


Figure 29: . AGONIST/ANTAGONIST AND NARCOTIC DRUGS 1.4

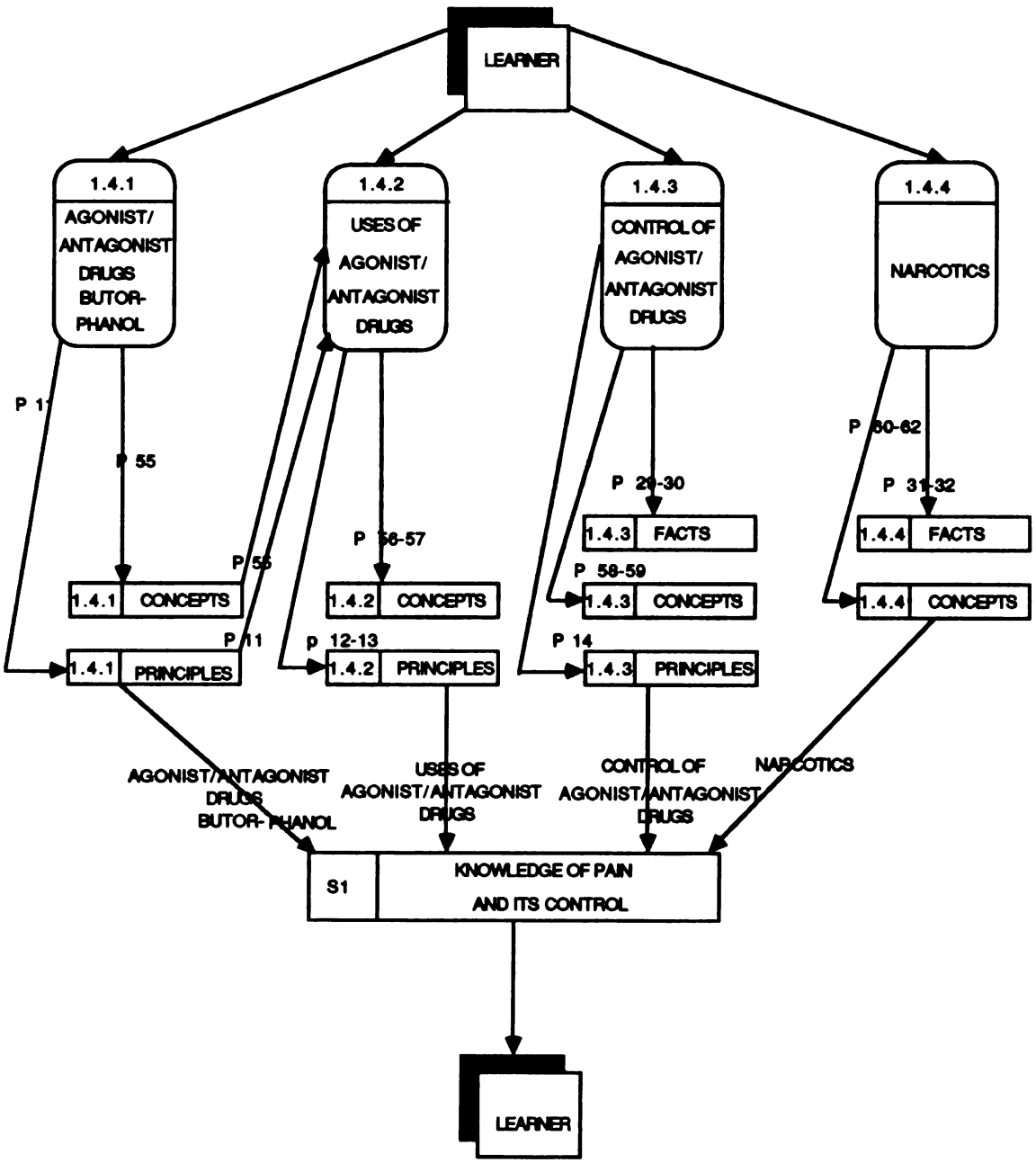


Figure 29: . AGONIST/ANTAGONIST AND NARCOTIC DRUGS 1.4

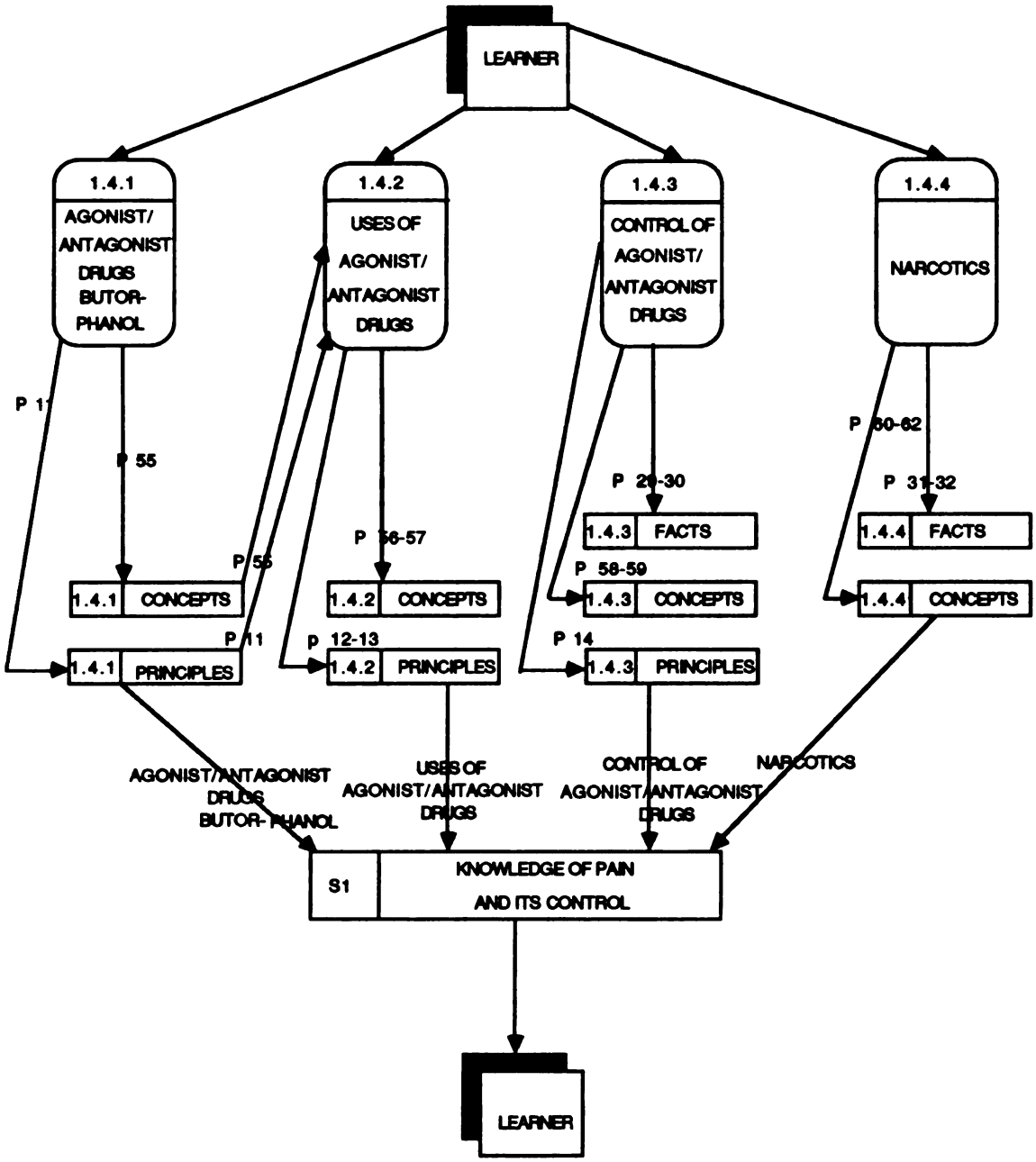


Figure 29: . AGONIST/ANTAGONIST AND NARCOTIC DRUGS 1.4

INSTRUCTIONAL DICTIONARY
for
Methods of Controling Pain in Veterinary Medicine

Section I: Pain

(num)(level)(sequence) (reference) type

Part A: 1.1.1 (Pain and the veterinarian's concern about pain)

(1) (42) (1) Pain is a complex physiological phenomenon that is difficult to identify and interpret in animals. (1)2

(2) (22) (2) No single area of the brain is specifically responsible for the perception of pain. (1)2

(1) (42) (3) Veterinarians should alleviate pain and suffering in animals. (1)1

(2) (42) (4) Veterinarians should suspect the presence of pain in animals if under the same circumstances humans experience pain. (1)1

(1) (42) (5) The judicious use of drugs for relief of pain can significantly reduce animal suffering. (1)3

(2) (33) (6) Pain relievers should be selected for animals if
(a) the animal can, in fact, be made more comfortable.
(b) the pain-relieving drugs will not be detrimental. (1)3

Part B: 1.1.2 (Pain receptors and response levels)

(3) (33) (1) The modifier, PAIN, should not be applied to these neurological components because the PERCEPTION of pain does not necessarily occur with:
(a) stimuli
(b) impulses
(c) pathways
(d) reflexes (2)3

(3) (22) (2) Noxious describes stimuli which give rise to pain. (2)2

(4) (32) (3) A receptor is that portion of a nerve which responds to a stimulus and causes the transmission of an impulse to the brain. (2)2

(5) (32) (4) Nociceptors are receptors specifically responsive to noxious stimuli. (2)2

(6) (32) (5) Nocipceptive threshold is the intensity of stimulation of a nociceptor needed to generate nerve impulses. (2)2

(7) (33) (6) The pain detection threshold is the strength achieved by a noxious stimulus to cause the perception of pain. (2)2

(8) (33) (7) The pain tolerance threshold is the highest intensity of noxious stimuli a human subject will permit an experimenter to deliver. (2)2

(9) (32) (8) The strength of a noxious stimulus necessary to reach the nociceptive threshold varies little among humans and animals. (2)2

(10) (22) (9) The strength of a noxious stimulus needed to cross the pain detection threshold is variable. (2)2

(1) (33) (10) If individuals have experienced pain previously and repeatedly, the strength of a noxious stimulus necessary to cross the pain detection threshold is probably higher than for individuals who have not experienced pain. (2)4

(2) (32) (11) If pain is dull, it is more easily tolerated than sharp pain. (2)4

(4) (21) (12) A variety of different stimuli evoke pain: (2)3
(a) superficial (skin)
(b) mechanical (pressure)
(c) thermal (heat)
(d) chemical (burns)
(e) visceral (gut).

(5) (33) (13) There are many different types of pain (2)3
(a) the perceptual experience can change dramatically with variation of the intensity of the stimulation
(b) A needle puncture is not as painful as a skin incision.

(3) (32) (14) If animals are stimulated at about the same intensity that human subjects first report the detection of pain, they will begin to escape stimulation. (2a)4

(3) (42) (15) Veterinarians should not ignore the perception of pain in animals. (2a)1

Part C: 1.1.3 (Animal Response to Pain)

(4) (32) (1) If an animal exhibits behavioral changes, the animal may be experiencing pain. (3)4

(6) (41) (2) Behavioral changes indicative of pain range from vigorous activity to relative lethargy. (3)3
(a) moaning, groaning, crying, whimpering, bellowing
(b) looking at a specific area
(c) licking, biting
(d) decrease in physical activity
(e) poor appetite

(5) (41) (3) If acute or sharp pain is identified by palpitation of an affected area, the animal may attempt to escape or attack. (3)4

(7) (31) (4) Clinical signs of the presence of pain: (3)3
(a) lameness
(b) shallow breathing
(c) moaning, crying

(8) (31) (5) Signs of localized pain: (3)3
(a) Belly: tense abdominal muscles
(b) Chest: shallow breathing or elbows away from the body in a distressed manner.

(11) (31) (6) Muscle guarding is tense abdominal muscles due to belly pain. (3)2

(6) (43 (7) If an animal has experienced pain for many days, the animal may tolerate the painful experience and become quiet or withdrawn with no overt behavioral evidence of suffering. (3)4

Part D: 1.1.4 (Specific responses to pain)

(9) (42) (1a) There can be a great deal of interspecies and individual variability in response to pain. (4)3
(a) Hounds seem less likely than poodles to be sensitive to pain.
(b) Cats may only show visible signs of pain when the stress becomes more pronounced.
(c) Grinding of teeth and head pressing are probable signs of pain in ruminants.
(d) Profuse sweating may indicate severe pain in a horse.

(10) (22) (1b) The presence of pain can be inferred from the observing changes in the animal's: (t1,4)3
(a) posture
(b) vocalizing
(c) temperament
(d) respiration
(e) other miscellaneous behaviors

(11) (41) (2a) General signs of behavior indicating pain or discomfort in the dog's posture: (t1,4)3
(a) anxious expression;
(b) tail between legs
(c) "hang dog" look
(d) shifts in position
(e) not comfortable

(12) (42) (2b) General signs of behavior indicating pain or discomfort in the dog's Vocalizing: (t1,4)3
(a) Howls
(b) Distinctive bark
(c) Moans
(d) Groans

(13) (32) (2c) General signs of behavior indicating pain or discomfort in the dog's Temperament: (t1,4)3
(a) Aggression
(b) Cringing
(c) Submissive
(d) Escapes
(e) Licks or bites area

(14) (32) (2d) General signs of behavior indicating pain or discomfort in the dog's Respiration: (t1,4)3
(a) Abnormal breathing pattern:
(b) rate and depth altered,
(c) panting, labored

(15) (42) (2e) General miscellaneous signs of behavior indicating pain or discomfort in the dog: (t1,4)3
(a) penile protrusion
(b) Frequent urination
(c) Shivering

(16) (42) (3a) General signs of behavior indicating pain or discomfort in the cat's posture: (t1,4)3
(a) Tucked in paws
(b) hunched head and neck
(c) Shifts position
(d) Tucked up abdomen
(e) not comfortable

(17) (32) (3b) General signs of behavior indicating pain or discomfort (t1,4)3
in the cat's Vocalizing:
(a) Distinctive cry
(b) Hissing
(c) Spitting

(18) (32) (3c) General signs of behavior indicating pain or discomfort (t1,4)3
in the cat's temperament:
(a) Ears flattened
(b) Fear of being handled
(c) May cringe

(19) (32) (3d) General signs of behavior indicating pain or discomfort (t1,4)3
in the cat's Respiration:
(a) Abnormal breathing pattern:
(b) panting, breathe heavy

(20) (32) (3e) General Miscellaneous signs of behavior indicating pain (t1,4)3
or discomfort in the cat:
(a) Unsteady gait
(b) limps
(c) Shivering

(21) (42) (4a) General signs of behavior indicating pain or discomfort (t1,4)3
in the cow's Posture:
(a) Head down - ears limp
(b) Tucked up abdomen
(c) Head pressing

(22) (32) (4b) General signs of behavior indicating pain or discomfort (t1,4)3
in the cow's Vocalizing:
(a) Grinding teeth

(23) (42) (4c) General signs of behavior indicating pain or discomfort (t1,4)3
in the cow's Temperament:
(a) Withdrawn
(b) Depressed
(c) Not active

(24) (32) (4d) General signs of behavior indicating pain or discomfort (t1,4)3
in the cow's Respiration:
(a) Grunting
(b) Dyspnea

(25) (32) (4e) General Miscellaneous signs of behavior indicating pain (t1,4)3
or discomfort in the cow:
(a) limp

(26) (42) (5a) General signs of behavior indicating pain or discomfort (t1,4)3
in the Horse's Posture:
(a) Rolls
(b) Stretches
(c) Looks at area
(d) Kick at area

(27) (42) (5b) General signs of behavior indicating pain or discomfort (t1,4)3
in the horse's Vocalizing:
(a) Grinding teeth

(28) (42) (5c) General signs of behavior indicating pain or discomfort (t1,4)3
in the horse's Temperament:
(a) Depressed
(b) Aggressive

(29) (42) (5d) General signs of behavior indicating pain or discomfort in the horse's Respiration: (t1,4)3
(a) Grunting
(b) Labored breathing

(30) (32) (5e) General miscellaneous signs of behavior indicating pain or discomfort in the horse: (t1,4)3
(a) limp
(b) Sweat

(12) (22) (6a) Even though there is a great deal of interspecies and individual variability in response to pain, there are some broad generalizations that can be made about pain that apply to many species. (4)2

(31) (43) (6b) Intense emotional reactions by a patient can be produced by (4)3
(a) Levels of pain
(b) above the tolerance threshold.

(13) (22) (6c) High levels of pain in animals can usually be observed through changes in the animal's behavior. (4)2

(14) (42) (6d) High levels of pain, when associated with human beings, are said to evoke suffering. (4)2

(7) (42) (6e) If pain is near-detection level, it (4)4
(a) can be tolerated
(b) is not necessarily disruptive
(c) does not produce a high degree of emotional reactivity.

Part E: 1.2.1 (Analgesia and Anesthesia)

(32) (43) (1) Drugs acting on the nervous system, particularly the brain, are the primary agents for the alleviation of pain: (5)3
(a) for use as a central control mechanism
(b) for use as a peripheral control mechanism.

(15) (42) (2) Analgesia is relief of pain without unconsciousness. (5)2

(16) (42) (3) General Anesthesia is the loss of both sensation to pain and consciousness. (5)2

(33) (42) (4) Sleep (5)3
(a) produces loss of consciousness
(b) increases the pain detection threshold
(c) does not eliminate the response to pain.

Part F: 1.2.2 (Anesthesia)

(34) (42) (1) The Principle purpose for the use of anesthetic and analgesic agents is to prevent response to pain (6)3
(a) induced from diagnostic procedures
(b) during surgery.

(17) (31) (2) Local anesthesia or local analgesia involves loss of sensation of a limited area of the body. (6)2

(18) (32) (3) Regional anesthesia involves loss of sensation of a large area of the body such as the hind legs. (6)2

(19) (32) (4) General anesthesia provides overall insensitivity of the whole body and unconsciousness. (6)2

(20) (22) (5) Amnesia is the inability to remember past experiences or a loss of memory for a given time period. (6)2

(8) (42) (6) If an individual is given a surgical level of general anesthesia, the individual will experience unconsciousness, analgesia, absence of muscle movement following a noxious stimulus, and amnesia. (6)4

Part G: 1.2.3 (Sedation and tranquilizers)

(21) (41) (1) Sedatives and tranquilizers produce a mild degree of central nervous system depression for which the patient is conscious, but calm. (7)2

(35) (32) (2) Sedatives and tranquilizers (7)3
(a) do not relieve pain
(b) may dull conscious perception of pain
(c) relieve tension and anxiety.

(36) (42) (3) Some drugs provide analgesia as well as mild sedation in the dog and human, but do not readily produce sleep: (7)3
(a) Morphine
(b) meperidine
(c) oxymorphone

(37) (42) (4) Tranquilizers are administered to animals (7)3
(a) to produce a calming effect
(b) providing a chemical restraint.

Part H: 1.2.4 (analgesia)

(22) (32) (1) Opiates are a group of naturally occurring opium compounds and their chemically related derivatives such as morphine. (8)2

(23) (21) (2) An exogenous substance is a substance outside the body. (8)2

(24) (32) (3) An Opioid is an exogenous substance which binds a specific type of nociceptor called opiate receptors. (8)2

(38) (33) (4) Opioids (8)3
(a) attach to the opiate receptors.
(b) At least partially block the sensation of painful stimulus

(25) (32) (5) Nonopioid antinfalammatory agents such as flunixin (banamine) and analgesics such as xylazine (Rompun) have no affinity for opiate receptors. (8)2

(39) (32) (6) Opioids produce their major effects on the central nervous system: (8)3
(a) analgesia
(b) sedation, with variable side effects of respiratory depression
(c) decreased gastrointestinal motility
(d) nausea and vomiting

Part I: 1.3.1 (Opiate receptors, Enkephalins and beta-endorphins)

(40) (33) (1) The relative analgesic potency of opioids appears to be related to their attraction to receptors in the body. (9)3

(41) (33) (2) Densities of opiate receptors are high in anatomical areas associated with physiologic functions that are altered by opioids. (9)3

(42) (33) (3) There appears to be a correlation between the site of action and the opioid effect. (9)3

(26) (31) (4) Endogenous substances are produced inside the body. (9)2

(43) (33) (5) Opiate receptors appear to be associated with specific areas in the brain that function as sites of action for a number of endogenous substances. (9)3

(44) (33) (6) Enkephalins and beta-endorphins: (9)3
(a) are endogenous opioid-like compounds which
(b) appear to function as a built-in protective mechanism of the body to
(c) help relieve pain.

(45) (42) (7) It is assumed that exogenous opioids produce their effects by mimicking some of the actions of enkephalins and beta-endorphins (9)3

(46) (43) (8) Some injured animals may not appear to be in pain because of the production of endorphins. (9)3

(9) (44) (9) If phenothiazine tranquilizer is administered to a recently wounded animal, the treatment might actually enhance pain by inhibiting the endorphin system-even if the animal looks pain-free. (9)4

Part J: 1.3.2 (Classification of Opiate Receptors)

(47) (32) (1) The three major opiate receptors are Mu, Kappa, and Sigma. The Mu receptors mediate analgesia and euphoria and: (10)3
(a) respiratory depression
(b) dilation of the pupil
(c) sedation
(d) physical dependence
(e) abuse potential

(48) (32) (2) The three major opiate receptors are Mu, Kappa, and Sigma. Kappa receptors are primarily responsible for: (10)3
(a) analgesia
(b) pupillary constriction
(c) a modest degree of sedation
(d) a mild respiratory depression

(49) (32) (3) The three major opiate receptors are Mu, Kappa, and Sigma. Sigma receptors cause: (10)3
(a) restlessness
(b) mental anxiety
(c) hallucinations
(d) respiratory and circulatory stimulatory effects.

(50) (33) (4) Two receptor types found in selected tissue: (10)3
(a) Delta receptors which appear to be selective for enkephalins.
(b) Epsilon receptors which have a high selectivity for -endorphins but lack affinity for enkephalins.

(51) (42) (5) Most opioids (10)3
(a) have different affinities for different receptors
(b) are called drugs of the attracted receptor (i.e. kappa drugs or mu drugs).

Part K: 1.3.3 (Classifying Opioid Drugs)

(52) (33) (1) Opioid drugs are categorized as agonists, antagonists, (11)3
or mixed agonist-antagonists depending on their actions at the receptor sites. Opioid agonist acting at mu, kappa, and sigma receptors:
(a) morphine
(b) meperidine
(c) fentanyl
(d) oxymorphone

(53) (44) (2) Opioid drugs are categorized as agonists, antagonists, (11)3
or mixed agonist-antagonists depending on their actions at the receptor sites. Opioid antagonists, which exert little or no physiological affect on their own:
(a) act at opiate receptors
(b) block the effects of the opioid agonist by its displacement from the receptors.

(27) (32) (3) Drugs are referred to as kappa agonists or mu agonists (11)2
if they activate either kappa or mu receptors respectively.

(28) (32) (4) Drugs are antagonists if they reverse the effects of the(11)2
opioid agonist by its displacement from the receptors.

(10) (43) (5) If opioid agonists that produce their effects at only (11)4
kappa receptors are used, analgesia can be achieved without the side effects seen when mu receptors are also triggered.

(54) (41) (6) Naloxone (11)3
(a) is a pure opioid antagonist which
(b) attaches to all opiate receptors.

Part L: 1.4.1 (Agonist/Antagonist Drugs: Butorphanol)

(55) (43) (1) Drugs classified as agonist/antagonist analgesics may (12)3
bind to the mu receptors and either exert no action or have only limited effects while exerting agonist action at the kappa receptors to provide analgesia and varying degrees of sedation.
(a) Butorphanol (stronger agonist than nalbuphine, weaker antagonist)
(b) Nalorphine (weak agonist - strong antagonist)
(c) levallorphan (weak agonist - strong antagonist)
(d) Nalbuphine (weak agonist but stronger antagonist)
(e) Pentazocine

(11) (32) (2) If a drug has a selective nature as a kappa agonists, (12)4
fewer side effects are associated with its use.

Part M: 1.4.2 (Uses of Agonist/antagonist Drugs: Butrophanol)

(56) (42) (1) Agonist/antagonist analgesics may be used alone to (13)3
(a) provide analgesia
(b) produce the effects of an opioid agonist.

(57) (43) (2) Agonist/antagonist analgesics may be used after the use (13)3
of a pure opioid to
(a) return the patient to consciousness
(b) relieve the opioid induced respiratory depression
(c) and to enhance analgesia.

(12) (43) (3) If a pure opioid such as fentanyl or oxymorphone has been used as a component of an anesthetic procedure and analgesia is desired for postsurgical pain, an agonist/antagonist analgesic such as butorphanol may be given to return the patient to consciousness and relieve the opioid induced respiratory depression while enhancing analgesia. (13)4

(13) (42) (4) If all the effects of an agonist/antagonist analgesic are to be reversed, a pure antagonist such as naloxone may be given. (13)4

Part N: 1.4.3 (Control of Agonist/antagonist Drugs)

(29) (42) (1) Agonist/antagonist opioids have strong analgesic properties but have little propensity for physical dependence or abuse. (14)2

(58) (32) (2) Butorphanol and nalbuphine are not listed as controlled substances by the Federal Drug Enforcement Agency since:
(a) little propensity for physical dependence,
(b) however, they have a strong analgesic property. (14)3

(30) (32) (3) Pentazocine is included as a class IV drug. (14)2

(59) (32) (4) The addictive properties of opioids is of little direct importance in the treatment of animals
(a) because veerinary patients are not usually given opportunity to develop drug dependence
(b) but are often restricted because of human dependence. (14)3

(14) (31) (5) If a drug such as the opioid agonists has a liability of addiction or physical dependence by humans, drug control regulations may restrict veterinarians from using that drug; especially for practices without good security and record keeping systems. (14)4

Part O: 1.4.4 (Narcotics)

(31) (21) (1) The term narcotic, which originally designated drugs which stupify, now designates any substance which can cause physical dependence or abuse. (15)2

(60) (21) (2) Narcotic no longer denotes a pharmacological class, but designates any substance which can cause physical dependence or abuse. (15)3

(61) (32) (3) A narcotic may be an analgesic, but an analgesic may not be a narcotic, even if it affects opiate receptors. (15)3

(62) (32) (4) Butorphanol is a agonist/antagonist with low potential for physical dependence because:
(a) even though symptoms resembling those of opiate withdrawal were observed,
(b) addictive behavior indicating a psychological need was not exhibited. (15)3

(32) (32) (5) Butorphanol is not a Controlled substance and therefore is NOT classified as a narcotic drug with abuse potential. (15)2

ATTITUDES: 3 FACTS: 32 CONCEPTS: 62 PRINCIPLES: 14

APPENDIX F

APPENDIX F

PROCEDURES FOR DESIGNING AND DEVELOPING INTERACTIVE VIDEO INSTRUCTION

1). Determine that there is sufficient time to take on a new project.
   a). It there is, check that there are not other projects to which you have future commitment or other projects more worthy to do.
   b). If there is not time available, check whether any of the current projects that are less worthy can be temporarily set aside to work on this project. If so, reschedule them, do not just forget them.
   c). Do not undertake a project, no matter how worthy, if there will not be sufficient time to do a thorough job with the project.

2). Meet with the potential client
   a). Determine his perceived needs, target audience, and project expectations.
   b). Explain the CAI groups role (if the project is to be undertaken) and explain, as much as possible, the CAI groups expectations of the client.
   c). Collect copies of any relevant resource materials the client may have available for the project (or a working outline of what the course materials will be).

3). Review and organize the data from the client.
   a). If the need for the project is not readily apparent, do an needs analysis, otherwise merely state the need for the project in instructional format.
   b). Reviewing notes from the meeting with the client and at least skimming the materials the client presented, determine the course, major section, and unit goals.
   c). Prepare an instructional needs report and an unit level instructional goals outline.
   d). Develop a unit level data flow diagram WITHOUT INSTRUCTIONAL EVENT INFORMATION.

4). Meet with the potential client
   a). Verify that the materials developed so far are correct.
   b). Provide a rough estimate of the amount of time involved with the project.
   c). Establish deadline for a go/no go project decision.

5). Using the materials given by the client or meeting with the client or associate, determine the instructional events for each unit. Enter into a word processor with 10/70 margins. Save in ASCII (to end each line with a line feed) with extension .seq. (see Sawyer.seq as an example).
   a). Course Title
   b). Section heading
   c). Unit information (repeat for each unit)
      1). Part name (repeat each change in subpart)
      2). Subpart name (if applicable)
         A. 1st Instructional event
            1). (sequence number) First line
            2). indent, rest of lines
         B. 2nd Instructional event
            1). (sequence number) First line
            2). indent, rest of the lines
         C. Etc.

6). Check the instructional events against the source materials. Make sure all were entered. Determine the reference of the instructional event. Determine the type of each instructional event:
   1. Attitude
   2. Fact
   3. Concept
   4. Principle

7). Reload the instructional event file with margins 10/80. Add the reference to the first line of each instructional event so that it takes a maximum of 5 spaces and ends at 77. Add the number for the type of instructional event so that it ends at 79. Save in ASCII with extension .iev.

8). Change the names of the files Run Instrdic.bas to create the instructional dictionary. The output should be an ASCII file ending with the extension .dic.

9). Using the instructional dictionary as a guide,
   (a) complete the Data Flow Diagrams at least to the Part level. Complete a subpart level data flow diagram if:
      (1) any part appears to have natural subdivisions or
      (2) any part has more than 15 events and the amount of information presented needs grouping to aid in both comprehension and recall
   (b) Indicate instructional events introduced in one section and used in another on the data flow diagram

(c) Recheck the instructional dictionary and data flow diagram with the original source of instructional material
  (1) making any necessary changes
    (a) omissions of materials
    (b) sequencing errors
    (c) misread materials
  (2) prepare suggestions for the client for changes where materials
    (a) seem to be missing for good instructional flow
    (b) seem to be redundant
    (c) seem to be extraneous

10). Meet with the client to
  (a) verify the work completed:
    (1) data flow diagram is correct at all desired levels
    (2) the instructional dictionary is complete
    (3) the suggestions for client changes need to be made
  (b) ask client to
    (1) determine the level of importance and difficulty for each instructional event (leveldir.wp)
    (2) fill out course and section worksheets (Course.out).
  (c) Provide followup support for activities in (b) where necessary.

11). When client has completed (b) from 10)
  (a) Enter the two digit level importance/difficulty number into the data dictionary and save it in ASCII.
  (b) After changing the names, Run INSTRWRK.BAS. The result will be:
    (1) filename0.wrk containing all the low level instructional events
    (2) filenames.wrk containing the medium level instructional events
    (3) filename1.wrk containing the high level instructional events.
  (c) Using the importance/difficulty number as a guide, give the client an estimate of the amount of time the project will take for each of the interested parties.

12). While the client is filling out the instructional events worksheets the authoring should begin for the courseware
  (a) create Menus from the Data Flow Diagrams

(1) Each process symbol should be a menu item
(2) Use models from the authoring system Library.

(b) set up course introductions (If the video is not developed yet, merely access dummy points on any available disk and change frame numbers when the video is available)
(c) program the low level worksheets
(d) If possible, start filming video segments from the course and section outlines.

13). Encourage the client to "turn in" the worksheets after each part is completed (or at most, each section).
(a) suggest improvements to the client if the materials on the worksheets seem in ANY WAY inadequate.
(b) Program complete parts whenever possible.
(1) Have a member of the staff check it.
(2) Make corrections/modifications
(c) Have the client "look over" each part (if possible) as soon as it is developed. (This will not only allow for formative feedback as to how the course is progressing, but will also help to keep the client enthusiastic as he sees concrete progress being made.)
(d) make modifications the client thinks important. Verify!
(e) Have students from the target audience check each part upon completion for formative feedback. Check any MAJOR recommendations by the student evaluators with the client before programming the "improvements".

14). Check on the video production (if not completed earlier).
(a) Check that the storyboard has all necessary elements for the instruction
(b) Check that the storyboard is arranged and organized efficiently
(c) Check that all instructional elements were actually taped. If not, schedule taping OR consult with client to modify the instruction
(d) Check editing of master video tape
(e) Review the final version with the client
(f) Repeat (c)-(e) for mastered video disk

15). When all the course programming is completed,
(a) completely test the course with CAI staff member with little contact with the development of the course.
(b) Modify any problem areas. Repeat (a) & (b) as needed.
(c) have the client test the program completely at the CAI offices, if possible.

(1) Solicit feedback NON-JUDGMENTALLY. Provide information when requested ONLY.
(2) Watch for non verbal feedback as the client works through the complete system. Solicit feedback on any areas that appear to be problem areas for the client.

16). When the client is satisfied with the program, two or three students from the target student population should run the program while being observed at the CAI offices.
(a) Watch for nonverbal feedback as the students work through the complete system.
(b) Solicit feedback on any areas that appear to be problem areas for the students AFTER they have finished the complete course.
(c) students should also fill out formative evaluation forms that help determine areas that may be improved in the courseware.

17). When the improvements suggested by the students are incorporated in the course (if they are considered improvements by the instructional designer and the client),
(a) the course should be tested by four or five students from the target student population in the setting that will be used for the target population.
(b) Students should fill out formative evaluation forms that help determine areas that may be improved in the courseware
(c) students should be debriefed, if possible, to see if there are other possible areas of improvement.

18). When the improvements suggested by the students are incorporated in the course (if they are considered improvements by the instructional designer and the client),
(a) the course should be ready for target student use
(b) continue to monitor the course carefully checking
(1) formative evaluation forms
(2) summative evaluation forms
(3) final tests of student learning.

19). Make any modifications necessary if 17) indicates the need.

20). Ask client to evaluate the performance of your group and the product produced. Use results to improve performance on next project.

21). Continue modifications 17 and 18 as long as the course is being used on at least a periodic check schedule.

LIST OF REFERENCES

List of References

Abdolmohammadi, M.J. (1987, Spring). Decision support and expert systems in auditing: A review and research directions. Accounting and business research. 173-185.

Adams, D.M. & Hamm, M. (1987). Artificial intelligence and instruction: Thinking tools for education. T.H.E. Journal, 15(1), 59-62.

Adams, R. & Imhof, H. (1987). Computers in the sciences. The Computing Teacher, 15(4), 20.

Anderson, J. R. (1985). Cognitive psychology and its implications (2nd ed.). New York:W.H. Freeman and Company.

Andrews, D.H. and Goodson, L.A. (1980). A comparative analysis of models of instructional design. Journal of instructional development, 3(4), 2-16.

Aronson, D.T., & Briggs, L.J. (1983). Contributions of Gagne and Briggs to a prescriptive model of instruction. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 75-100). New Jersey: Lawrence Erlbaum Associates.

Atkinson, R.C. (1976). Adaptive instructional systems: Some attempts to optimize the learning process. In D. Klahr (ed.), Cognition and Instruction, (pp. 81-108). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Ausubel, D.P. (1968). Educational psychology: A cognitive view. New York: Holt, Rinehart and Winston.

Baker, L., & Brown, A.L. (1984). Metacognitive skills and reading. In D. Pearson (ed.), Handbook of reading research (pp. 353-394). New York: Longman.

Banathy, B.H. (1968). Instructional systems. Palo Alto, Ca.: Fearon Publishers.

Bereiter, C. (1985). Toward a solution of the learning paradox. Review of Educational Research. 55, 201-226.

Blaisdell, J. H. (1985, October). Knowledge engineering software supporting delivery of an expert system within the CIS curriculum. In a paper presented to Information Systems Education Conference, Huston, Texas.

Bloom, B.S., & Broder, L.J. (1950). Problem-solving processes of college students. An exploratory investigation. Chicago: University of Chicago Press.

Boose, J. H. (1986). Expertise transfer for expert system design. New York:Elsevier.

Briggs, L.J. & Wagner, W.W. (1981). Handbook of procedures for the design of instruction (2nd ed.). New Jersey:Educational Technology Publications.

Brown, J.S., & Burton, R.R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. In D.F. Walker & R.D. Hess, Instructional software. Belmont, Ca.: Wadsworth Publishing Company.

Bruner, J.S. (1960). The process of education. New York: Random House.

Buchanan, B.G., & Shortliffe, E.H. (1984). Rule-Based expert systems: The MYCIN experiments of the Stanford heuristic programming project. Menlo Park, California:Addison-Wesley.

Carroll, J. B. (1976). Promoting language skills: The role of instruction. In D. Klahr (ed.), Cognition and Instruction, (pp. 3-22). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Chaturvedi, A.R. (1987, July). Situation-action framework for expert system problem selection. Paper presented at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Collins, A., & Stevens, A.L. (1983). A cognitive theory of inquiry teaching. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 4-36). New Jersey: Lawrence Erlbaum Associates.

Data Processing Curriculum Guide. (1983). produced by MSU Curriculum Resource Team. Michigan State University. East Lansing, Michigan: Michigan Department of Education.

Davis, G. B., & Olson, M. H. (1985). Management Information Systems: Conceptual Foundations, Structure, and Development. N.Y.: McGraw-Hill Book Company.

Davis, L. M. (1987, June). Seeding expert system technology in industrial settings. Paper presented at the National Educational Computing Conference, Philadelphia, Pennsylvania.

Davis, R.H., Alexander, L.T., & Yelon, S.L. (1974). Learning System Design An Approact To The Improvement of Instruction. New York: McGraw-Hill Book Company.

Derry, S. J. (1984, April). Strategy Training: An Incidental Learning Model for CAI. Paper presented at Annual Meeting, American Educational Research Association, New Orleans, LA.

Gable, A, & Page, C.V. (1980). The use of artificial intelligence techniques in computer-assisted instruction: an overview. In D.F. Walker & R.D. Hess, Instructional software. Belmont, Ca.: Wadsworth Publishing Company.

Gagne, E.D. (1985). The cognitive Psychology of school learning. Boston: Little, Brown and Company.

Gagne, R.M. (1977). The conditions of learning (3rd ed.). New york: Holt, Rienhart, and Winston.

Gagne, R.M. (1985). The conditions of learning and theory of instruction. (4th ed.) New York: Holt, Rinehart and Winston.

Gagne, R. M. & Briggs, L.J. (1979). Principles of instructional design (2nd ed.). New York: Holt, Rinehart and Winston.

Gagne, R.M. & White, R.T. (1978). Memory structures and learning outcomes. Review of Educational Research, 48. 187-222.

Gane, C., & Sarson, T. (1979). Structured Systems Analysis: tools & techniques. N.Y.: Improved System Technologies Inc.

Glaser, R. (1976). Cognitive psychology and instructional design. In D. Klahr (ed.), Cognition and instruction,(pp. 303-315). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Glaser, R. (1984). Education and thinking: The role of knowledge. American Psychologist, 39(2), 93-104.

Glaser, R., & Resnick, L.B. (1972). Instructional psychology. Annual review of psychology, 23, 207-276.

Gropper, G.L. (1983). A metatheory of instruction: A framework for analyzing and evaluating instructional theories and models. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 37-53). New Jersey: Lawrence Erlbaum Associates.

Hamreus, D.G. (1968). The Systems Approach to Instructional Development. In Teaching Research, A Division of the Oregon State System of Higher Education. The Contribution of Behavioral Science to Instructional Technology.

Hayes-Roth, B. (1985). A blackboard architecture for control. Artificial Intelligence, 26, 251-321.

Hayes-Roth, F. (1985). Rule-Based systems. Communications of the ACM, 28, 9.

Hayes-Roth, F., Waterman, D.A., & Lenat, D.B. (Eds). (1983). Building expert systems. Reading, Ma: Addison-Wesley.

Hayes-Roth, F., Waterman, D.A. & Lenat, D.B. (1983). An overview of expert systems, in F. Hayes-Roth, D. A. Waterman & D.B. Lenat (Eds.). Building expert systems. Reading, Mas.:Addison-Wesley.

Hayes-Roth, F., Klahr, P., & Mostow, D.J. (1986). Knowledge acquisition, knowledge programming, and knowledge refinement. in P. Klahr & D.A. Waterman (Eds.). Expert systems techniques, tools, and applications. Reading, Massachusetts: Addison-Wesley.

Insurance embraces expert systems. (1988, April). Modern Office Technology, 33(4), 39.

Keravnou, E.T.,& Johnson, L. (1986). Competent expert systems. London: Koglan Page.

Klahr, P., & Waterman, D.A. (1986). Artificial intelligence: a Rand perspective. in P. Klahr & D.A. Waterman (Eds.). Expert systems techniques, tools, and applications. Reading, Massachusetts: Addison-Wesley Publishing Company.

Knirk, F. G. & Gustafson, K.L. (1986). Instructional technology: A systematic Approach to Education. New York: Holt, Rinehart and Winston.

Landa, L.N. (1982). The algo-heuristic theory of instruction. In C. M. Reigeluth (Ed.). Instructional design theories and models: an overview of their current status (pp. 163-211). New Jersey: Lawrence Erlbaum Associates.

Larkin, J., McDermott, J., Simon, D.P., & Simon, H.A. (1980). Expert and novice performance in solving physics problems. Science, 208, 185-193.

Lee, M.C. (1987). Expert-system based instructional systems. In W.C. Ryan (Ed.), Proceedings National Educational Computing Conference (pp. 139-142). Eugene, Or.: International Council on Computers in Education.

Lord, K.W., Jr. (1983). CDP Review Manual A data Processing Handbook. New York: Van Nostrand Reinhold Company.

Mayer, R.E. (1983). Thinking, problem solving, cognition. New York: W.H. Freeman and Company.

McCorduck, P. (1987, July). Telling the Truth, Mainly. Paper presented at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Merrill, M.D. (1983) Component display theory. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 279-333). New Jersey: Lawrence Erlbaum Associates.

Merrill, M.D., Kowallis, T., & Wilson, B.G. (1981). Instructional Design in Transition. In F. H. Farley & N.J. Gordon (Eds.), Psychology and Education: The state of the Union (pp. 298-348). Berkley, Ca.: McCutchan Publishing Corporation.

Merrill, P. F. (1978). Hierarchical and informational processing task analysis: A comparison. Journal of instructional development, 1(2), 35-40.

Newell, A., & Simon, H.. (1972). Human problem solving. Englewood Cliffs, N.J.: Prentice-Hall.

Norman, D.A. (1980). What goes on in the mind of the learner. In W. J. McKeachie (Ed.), Learning, cognition, and college teaching. New Directions for Teaching and Learning (No. 2). San Francisco: Jossey-Bass.

Olson, D.R. (1973) What is worth knowing and what can be taught. School Review, 82, 27-43.

O'Shea, T. & Self, J. (1983). Learning and teaching with computers. Harvester Press.

Palincsar, A.S., Brown, A.S., & Martin, S. M. (1987). Peer interaction in reading comprehension instruction. Educational Psychologist, 231-253.

Perelman, L.J. (1987). Technology and transformation of schools. An ITTE technology leadership network special report from the institute for the transfer of technology to education of the National School Boards Association.

Polhemus, N. (1987, July). Interactive Computer Graphics for Exploratory Data Analysis. Presentation at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Ray, R. (1987, Fall Term). In handout for course: Developing vocational curricula in the two year college. Michigan: Michigan State University.

Reigeluth, C. M. (1983). Instructional design: what is it and why is it?. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 4-36). New Jersey: Lawrence Erlbaum Associates.

Reigeluth, C.M., & Stein, F.S. (1983) The elaboration theory of instruction. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 335-382). New Jersey: Lawrence Erlbaum Associates.

Resnick, L.B. (1963). Programmed instruction and the teaching of complex intellectual skills: Problems and prospects. Harvard Educational Review, 33, 439-471.

Resnick, L. B. (1973). Hierarchies in children's learning: a symposium. Instructional Science, 2, 311-323.

Resnick, L.B. (1976). Task analysis in Instructional design: Some cases from mathematics. In D. Klahr (ed.), Cognition and Instruction, (pp. 51-80). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Publishers.

Resnick, L. B. (1985). Cognition and instruction: Recent theories of human competence. In B.L. Hammonds (Ed.), Master lecture series: Vol. 4. Psychology and learning (pp. 123-186), Washington, DC: American Psychological Association.

Resnick, L.B. (1987). Education and learning to think. Washington: National Academy Press.

Richey, Rita, (1986). The theoretical and conceptual bases of instructional design. New York:London/Nichols Publishing.

Roblyer, M.D. & Hall, K.A. (1985). Systematic instructional design of computer courseware: A workshop handbook. Tallahassee, F.: Florida A&M University.

Ruth, S.R. (1987, July). Teaching expert systems in business schools--lessons learned. Paper presented at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Scandura, J.M. (1983). Instructional strategies based on the structural learning theory. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 213-246). New Jersey: Lawrence Erlbaum Associates.

Sebrechts, M., Schooler, L.J., LaClaire, L., & Soloway, E. (1987). Computer-based interpretation of students' statistical errors: a preliminary empirical analysis of GIDE. In W.C. Ryan (Ed.), Proceedings National Educational Computing Comference (pp. 143-148). Eugene, Or.: International Council on Computers in Education.

Simon, H. (1981). The sciences of the artificial (2nd ed.). Cambridge, Ma.: MIT Press.

Simon, H.A. & Newell, A. (1972). Human problem solving. Englewood Cliffs, New Jersey:Prentice-Hall.

Sleeman, D. & Brown, J.S. (1982). Introduction: Intelligent tutoring systems. In D. Sleeman & J. S. Brown (Eds.). Intelligent tutoring systems. N.Y.: Academic Press.

Snelbecker, G.E. (1983). Is instructional theory alive and well?. In C. M. Reigeluth (Ed.). Instructional-design theories and models: an overview of their current status (pp. 437-472). New Jersey: Lawrence Erlbaum Associates.

Suppes, P. (1979). Observations about the application of artificial intelligence research to education. In D.F. Walker & R.D. Hess, Instructional software. Belmont, Ca.: Wadsworth Publishing Company.

Tanimoto, S. L. (1987). The elements of artificial intelligence. Rockville, Maryland: Computer Science Press.

Waterman, D.A, (1986). A guide to expert systems. Reading, M.A.: Addison-Wesley.

Waterman, D. A. & Jenkins, B.M. (1986). Developing expert systems to combat international terrorism. In P. Klahr, & D.A. Waterman (Eds.). Expert systems techniques, tools, and applications. Reading, Massachusetts: Addison-Wesley Publishing Company.

Wensley, A. (1987, July). Research issues in expert systems. Paper presented at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Yazdani, M. (1987). Intelligent tutoring systems: an overview. In R. W. Lawler & M. Yazdani (Eds.). Artificial intelligence and education (vol. 1). Norwood, New Jersey: Ablex Publishing.

General References

American Psychological Association. (1983). Publication manual (3rd ed.). Menasha, Wisconsin: Banta Company.

Jonassen, D. H. (ed.). (1988). Instructional designs for microcomputer courseware. New Jersey: Lawrence Erlbaum Associates.

Hunt, M. (1982). The universe within: A new science explores the human mind. N.Y.: Simon and Schuster.

Kulik, J.A, Bangert, R.L., & Williams, G.W. (1983). Effects of computer-based teaching on secondary school students. In D.F. Walker & R.D. Hess, Instructional software. Belmont, Ca.: Wadsworth Publishing Company.

Lenat, D. B. (1984). Computer software for intelligent systems. DPMA Magazine, 67, 24-25.

McCorduck, P. (1985). The universal machine. N.Y.: McGraw-Hill.

Mayer, R. E. (1987). Educational psychology: A cognitive approach. Boston: Little-Brown.

Michigan State University. (1986). The graduate school guide to the preparation of master's theses and doctoral dissertations. E. Lansing, Mi.: Michigan State University.

Minsky, M. (1986). Society of mind. N.Y.: Simon and Schuster.

Quinlan, J. R. (1986). FERNO: A cautious approach to uncertain inference. In P. Klahr & D.A. Waterman (Eds.). Expert systems techniques, tools, and applications. Reading, Massachusetts: Addison-Wesley Publishing Company.

Rankin, S. C., & Hughes, C.S. (1987). The Rankin-Hughes framework. In C. Canning & K. Bunting (Eds.). Developing thinking skills across the curriculum and how computing can help. Westland, Michigan: MACUL.

Umar, A. (1987, July). Software support environments. Paper presented at the North American Conference of the International Business Schools Computer Users Group, Flint, Michigan.

Whimbey, A., & Lochhead, J. (1980). Problem solving and comprehension (2nd ed.). Phila.: The Franklin Institute Press.

Yen, D. (1987, July). An investigation on management applications of expert systems. Paper presented at the North American Conference of the Inter national Business Schools Computer Users Group, Flint, Michigan.