# Utilization of Expert Systems in the Work Place: Performing Project Software Cost Estimation on Training Systems

Henry A. Marshall
*University of Central Florida*

# UTILIZATION OF EXPERT SYSTEMS IN THE WORK PLACE: PERFORMING PROJECT SOFTWARE COST ESTIMATIONS ON TRAINING SYSTEMS

BY

HENRY A. MARSHALL
B.S.E., University of Central Florida, 1979

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program of the College of Engineering
University of Central Florida
Orlando, Florida

Spring Term
1986

## ABSTRACT

This research report investigates the use of an expert system to aid project engineers at the Naval Training Systems Center in making decisions concerning the requirements of the computer systems used in simulators. For a prototype system domain, the author chose an expert system that would generate a software development cost estimate. This system questions the user about the features and options required on the training system. The expert system then analyzes the information to generate a "lines of code" estimate. A selected model will combine various factors to generate a value answer for the user. The capabilities and features of current expert system development tools are reviewed as to what features would best address this problem domain. EXSYS, a rule-based expert system shell that runs on both Zenith and IBM PCs, was selected to develop the prototype because of its capability to meet the requirements of the software cost estimation domain. The COCOMO estimation model was selected to generate the user answers. The technique of using a rule-based system in combination with other management decision tools, such as spreadsheets, holds a potential of being an excellent approach for providing a tool for storing and utilizing estimation data and heuristics.

TABLE OF CONTENTS

## LIST OF TABLES

LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

## What is an Expert System?

An expert system is a computer system with domain knowledge capable of aiding the user in making intelligent decisions within that domain. It provides advice based on both the answers the user gives and the knowledge the system possesses. An expert system programmer is referenced to as a knowledge engineer. His/her function is to gather the facts, rules of thumb and heuristics that domain experts use in making decisions. He/she then installs this information in the computer so it can allow users the benefit of expert knowledge in making decisions within the system's domain.

Most expert systems have two major parts: the knowledge base and the inference engine. The knowledge base contains the facts, rules and heuristics gathered from the domain experts. The inference engine uses the knowledge base in combination with the answers it gathers from the user to come up with a conclusion. The conclusion can take many forms, depending on the inference engine. An inference engine with an empty knowledge base is called an expert system shell. Expert system shells are commonly used by knowledge engineers because of the large amount of effort involved in programming an inference engine. However, very few of the shells will custom fit any given expert system requirement. Many of the expert system development tools allow for some flexibility of the inference engine to meet the requirements of various users (Forsyth 1984).

The expert system is designed on a much different software design premise than the traditional algorithmic system. The algorithmic system has its knowledge structured in the code with "go to" and "if-then" statements. This structure would make the expansion of a knowledge based system difficult. In an expert system, knowledge is separated from the inference engine which allows easier expansion of the knowledge base. Thus, the knowledge that drives the system is explicit and is easy to access.

The parts and functions of the expert system will be discussed in greater detail later in this report.

## Statement of the Problem

Embedded in almost every training simulator is a complex computer system. Most of the development effort on the simulator is involved with the software and computer hardware requirements. The knowledge on how to best meet these requirements is scarce and known by very few individuals.

Many of the requirements for the computer system contained within a trainer will be determined by the contractor who builds the trainer. However, numerous decisions must be made throughout the acquisition process by the project engineer/manager at the Naval Training Systems Center.

To aid the project engineer/manager in making decisions regarding computer systems, the author proposes to create an expert system whose domain knowledge will contain facts, rules and heuristics associated with trainer computer systems.

This expert system will increase the productivity of the project engineer/manager, as well as the software engineer he/she normally consults. The limited number of experts in this area are unable to review all of the trainer programs on a case-by-case basis. This system will free some of the software engineer's time, thus allowing him/her to be more productive. The system also will ensure that experience or lessons learned on previous trainer procurement will be considered in the advice given by the expert system to the user. The knowledge engineer will translate the information necessary to solve the new situations into the knowledge representation of the expert system being utilized. The system will utilize the new information in generating advice for future users. This will generate a significant cost savings by allowing engineers to be more productive and thorough in their jobs.

The knowledge engineering task this system proposes will be enormous. Gathering the facts, rules and heuristics associated with the computer systems will be difficult because of common disagreement on methods.

Since computer technology and government standards are constantly changing, the expert system will have to be changed on a periodic basis. This makes an expert system more difficult to implement. These revisions may force the system to reside on a common host versus numerous small PCs because it will be vital to control the configuration of the expert system. These updates will ensure that the system formulates its conclusion with the latest rules.

## Why an Expert System?

Most potential expert system users will not realize their application is a good choice for an expert system. Some of the basic evaluation factors are listed in Table 1.

TABLE 1

SUITABLE VERSUS UNSUITABLE EVALUATION FACTORS

| SUITABLE | UNSUITABLE |
|---|---|
| Heuristic | Algorithmic |
| No established theory | "Magic formula" exists |
| Human expertise scarce | Human experts are a dime a dozen |
| Data are unclear | Facts are known precisely |
| Task requires mainly cognitive skills | Task requires common sense decisions (very situation dependent) and/or skills acquired through practice |

SOURCE: Forsyth 1984, Waterman 1986

To solve any given problem, two possible alternatives exist. First, there is the algorithmic approach. This is a step-by-step procedure which guarantees that the right answer will be given if the inputs are correct. The heuristic approach is based on developing probable answers based on the various rules of thumb developed through experiences. Unlike algorithms, heuristics do not guarantee a correct solution (Forsyth 1984).

Expert systems are the branch of computer science that derives solutions with the heuristics that human experts use. Therefore, any application that could be expressed with an exact solution method should be implemented using the algorithmic approach. Areas where there is no known exact method for generating solutions may be good expert system candidates.

A major consideration in implementing an expert system is the cost. Implementing a very small system can easily cost hundreds of thousands of dollars. Therefore, the expertise in the chosen domain must be both rare and capable of significant cost savings. This will allow the recovery of the agency's funds expended on the system.

There are several possible applications for expert systems at the Naval Training Systems Center. The main goal at the center will be to allow the project engineer/manager to monitor more activities while simultaneously increasing the quality and quantity of the decisions they make, thus increasing productivity and ensuring cost savings. The areas reviewed are the various software engineering functions.

One possibility is an expert system that will facilitate determining and writing the "Proposal Requirements Documents" for a training system. This includes defining technical proposal and specification requirements. This expert system could present probable inputs to the project engineer. Most expert systems allow you to ask the expert system why it is asking a question. The system responds with the rule(s) it is trying to satisfy. This would provide a computer-aided instruction environment which would help junior or inexperienced personnel become proficient faster. It would also take a burden off senior engineers.

The next possibility is an estimator of computer coding costs. Although you would assume this function would best be performed by the algorithm approach, the author found that most estimation is performed based on heuristics. Most cost estimators have developed heuristics based on past projects that had certain requirements and circumstances. They combine these heuristics with algorithms to justify the costs they propose. An expert system to estimate project costs would greatly aid the program managers in cost planning. People with this expertise are rare, thus a tool to do cost estimating would be widely used.

Other areas for a possible application include an estimator for life cycle support requirements and an estimator for the computer size and type requirements for a given training device application.

## Selection of a Prototype System Domain

The intent of this research paper is to develop one of the possible applications of expert systems at the Naval Training Systems Center through the prototype phase. This will attempt to demonstrate the feasibility of using expert systems as a productivity tool to aid the project engineers/managers. It will also explore the development and tool selection process associated with developing expert systems. After reviewing the list of possible applications, the author selected the software cost estimation system because it generated the most interest. This application would be unique in that the system would have to support the mathematics involved with software cost estimation.

Several software costing models have been reviewed and used by estimators. Some examples of software cost estimating methods are the COCOMO and Taylor models. None of the models available generate a "lines of code" estimate. In fact, they require this as an assumed input to the model. The disadvantage with these methods is that a great deal of knowledge of the system being estimated is required. The knowledge is necessary to ensure that the values placed in the software costing models are correct and justifiable. This work is very time-consuming for the expert performing the estimate.

The author proposes that the expert's evaluation role in this process be replaced by an expert system. The expert system would ask the user about the features and options of the training system.

The expert system would evaluate the size and complexity of the software development effort based on program histories and other heuristics gathered and developed by estimators. The system would then place these values in the cost model to generate a value answer for the user. While performing this estimate, the system could also note information on the type and amount of computer equipment required in the past to perform the proposed task.

Limiting the scope of the prototype system domain is necessary to ensure that the problem can be kept within manageable bounds. Many of the early expert systems failed because the domains they covered were much too broad. An expert system to generate a software cost estimate for all types of trainers is much too broad a task. In fact, it would be unwise to select this as the final goal of the expert system. A general rule for selecting a prototype domain is to pick a domain that most people feel is too small. The author will restrict the system to fixed wing operational flight trainers. In addition, the author will abridge the number of features the system will address. The operational flight trainers make a good candidate because most personnel involved with training systems can identify with the parts and features of these trainers. In selecting this domain, the author will be able to best demonstrate the problem definition, and possible knowledge representation for the domain.

# CHAPTER II

# EXPERT SYSTEMS

## Overview

This chapter will explain the components of an expert system and the options available for each component. This is very important in making a decision as to what options would best serve the computer cost estimation domain. A later section will explore the tools and languages available for developing expert systems.

The basic components of most expert or knowledge-based systems are shown in Figure 1.

The knowledge base contains the facts and heuristics about the domain the system covers. The inference engine derives new facts and conclusions by using the knowledge base (Harmon 1985). It controls the questioning of the user to derive information to generate a conclusion. As the inference engine derives new facts, either from questioning the user or by inferring a logical truth in the knowledge base, it stores this information in the working memory.

The user interface generates a user-friendly environment by presenting the questions the inference engine wishes to ask in an understandable form. It often will allow the user to ask why it is asking a certain question or what line of reasoning the inference engine is following (Harmon and King 1985, Hayes-Roth 1985). The

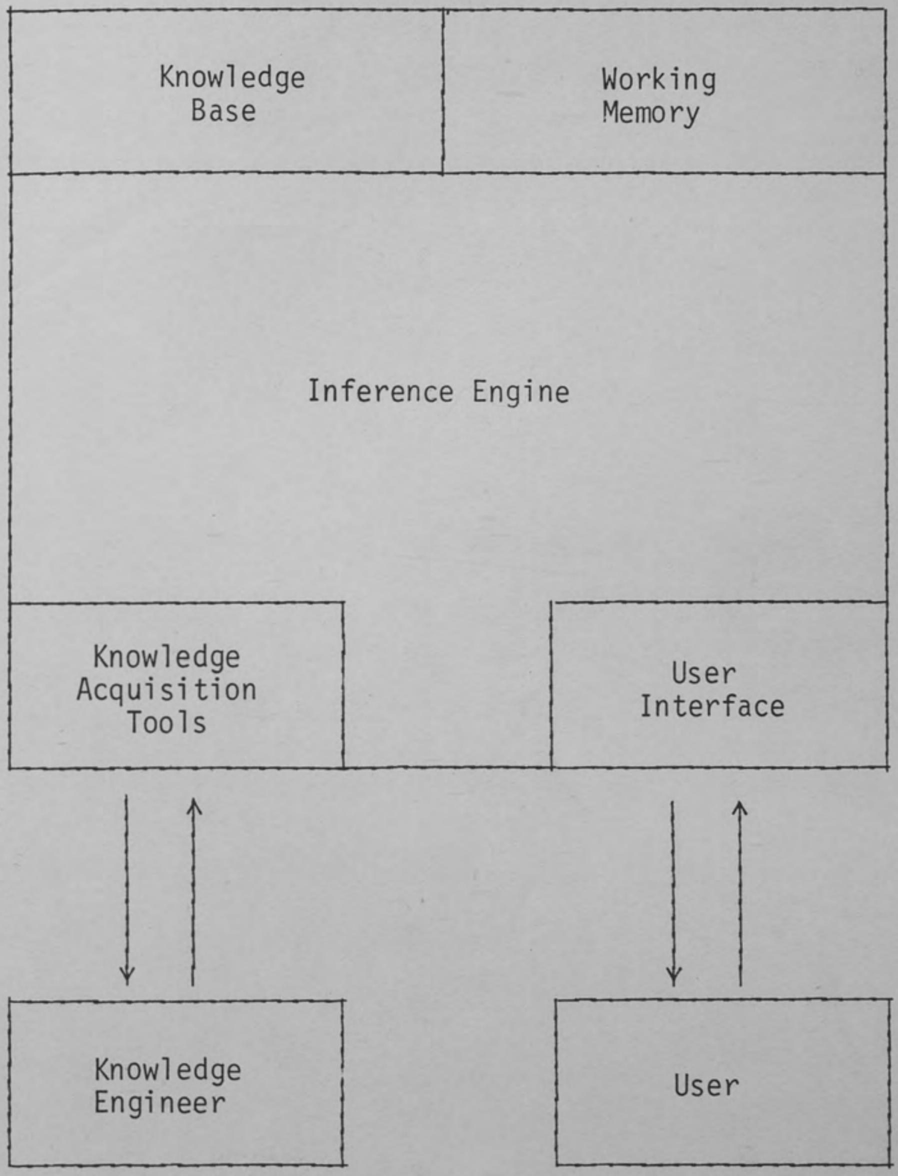

Figure 1. Basic Architecture of an Expert System.

knowledge acquisition tools are usually composed of debugging aids and a knowledge base editor. The debugging aids allow the user to trace the steps of the inference engine as it searches through the knowledge base. The knowledge base editors are used to create and make changes to the knowledge base. A common feature is a consistency checker to ensure that a new piece of knowledge does not conflict with an existing one. The following sections describe the options currently used in developing expert systems. These options are very important because they allow expert systems to support different types of knowledge domains.

## The Knowledge Base

As stated earlier, the knowledge base is the part of the expert system that contains the domain knowledge. To best represent this knowledge, several different types of knowledge representations have been developed. They are rules-based systems, frame-based systems and semantic nets. Each one of these methods has inherit advantages and disadvantages. Different types of applications are best performed with each type of knowledge representation. Some applications combine these methods to allow unique applications to be represented.

Rule-based knowledge centers on the use of the IF (condition), THEN (action/results) representation. An example of a rule is shown in Figure 2.

Rule-based or production systems constitute the most popular method for representing the problem-solving know how of human experts.

RULE #8:

IF: The birds are flying south

THEN: The season is fall

Figure 2. An Example of a Rule.

Experts tend to express most of their problem-solving techniques in terms of situation-action rules (Hayes-Roth 1985). This makes a rule-based system the suggested choice for decision intensive expert systems.

In a rule-based system, the domain knowledge is represented as a set of rules that are checked against a collection of facts. When the IF portion of a rule is satisfied by the facts, the action specified by the THEN portion is performed. When that happens, the rule is said to fire or execute. The new facts are stored in the working memory and the cycle repeats itself with the new facts.

To build the rules, attribute-value pairs or object-attribute-value pairs are used. In the object-attribute-value scheme, the objects may be either physical or conceptual entities. Attributes are generally characteristics or properties associated with objects (Harmon and King 1985). The value specifies the specific nature of an attribute in a particular situation. An example of an object-attribute-value pair is shown in Figure 3. The rule shown in Figure 2 is split into parts.

RULE #8:

| | Object | Attribute | Value |
|---|---|---|---|
| IF: | Birds | Flying | South |
| THEN: | Season | Identity | Fall |

Figure 3. Example of an Object-Attribute-Value Representation.

Many systems are built for single objects. In this case, the systems represent facts in terms of attribute-value pairs. Attribute-value pair systems differ from object-attribute-value pair systems in that the object-attribute must be combined to form the attribute (Harmon and King 1985). This gives the attribute-value pair system less flexibility in expressing factual information.

Rule-based systems perform judgemental knowledge quite well. However, factual knowledge and procedural knowledge cannot be easily expressed with a rule-based system. Factual knowledge represents assertions about objects and relationships between objects (Hayes-Roth 1985). Procedural knowledge represents sequences of problem-solving steps.

The semantic net knowledge representation is based on a network organization. The basic elements of the network are nodes and arcs (Harmon and King 1985). The nodes could be objects, concepts, object descriptors or events. Arcs are the network links that connect the nodes. They describe the relationship between the nodes they connect.

Examples of arcs commonly used are "is-a" and "has-a." An example of a semantic network is shown in Figure 4.

Based on the simple network in Figure 4, we can infer that an instrument flight trainer has flight dynamics, an instructional system and an instrument simulation module. The arcs establish an inheritance hierarchy within the net. This means items lower in the net can inherit properties from items higher up in the net (Waterman 1986, Winston 1984). Semantic nets are useful in representing knowledge in domains that use well-established classifications. The primary use of semantic nets is in natural language research, where they are used to analyze the meaning of a sentence. Because semantic nets by themselves lack the capability to make judgements and perform math functions, they are clearly inadequate for the cost estimation domain.

The frame-based knowledge representation uses a network representation with frames instead of nodes. A frame is a description of an object that contains slots for all of the information associated with the object. Slots may also contain default values, pointers to other frames, sets of rules or procedures by which values may be obtained. The types of procedures are if-added, if-removed and if-needed. The if-added procedure executes when new information is placed in the slot, the if-removed executes when information is deleted from the slot and the if-needed executes when another frame or a variable within the frame needs the value for the slot. The basic concept of

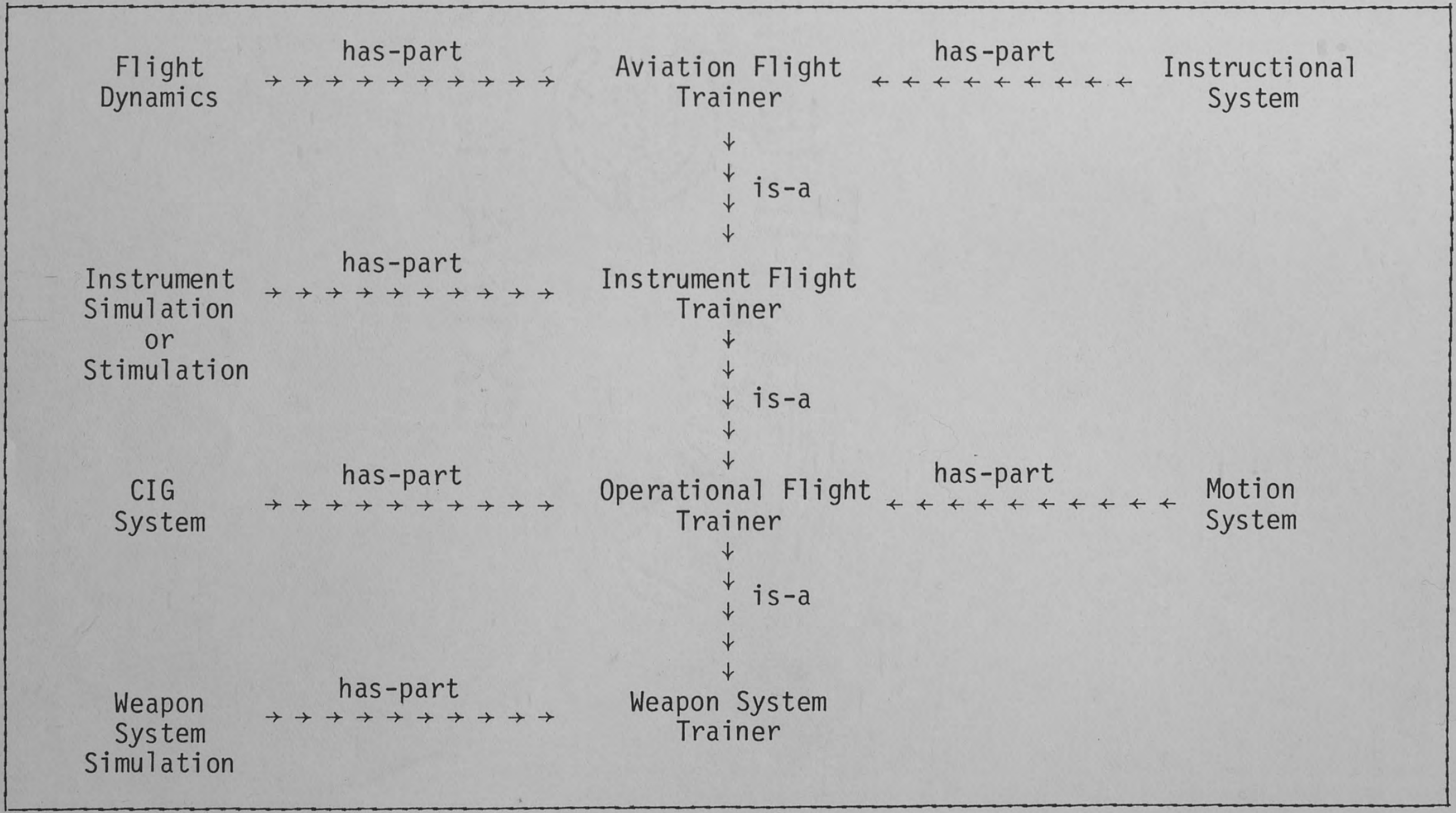

Figure 4. A Semantic Network Describing Some of the Breakdown Elements of Aviation Flight Trainers.

a frame is shown in Figure 5. As a possible frame representation for the coding effort on the computer image generation system in an operational flight trainer (Waterman 1986, Harmon and King 1985).

Computer Image Generation

| | | |
|---|---|---|
| SLOT 1 | Number of Channels | Attached Procedures and/or Rules |
| SLOT 2 | Speed of Device | |
| SLOT 3 | Size of Playing Area | ↓ |
| SLOT 4 | Degrees of Freedom | ↓ ↓ |
| SLOT 5 | Levels of Detail | ↓ ↓ |
| SLOT 6 | Estimated Number of Lines of Code to Support CIG System | ↓ ↓ |

Figure 5. An Example of What a Frame to Represent the Computer Image Generation Coding Effort Possibility Could Be.

The frames are joined together in the same manner as the semantic net. Frames have an inheritance hierarchy that allows frames to inherit values from frames higher in the net.

The final possibility is to develop a new knowledge representation specifically suited to the problem domain (Fikes and Kehler 1985). The ideal criteria for a knowledge representation is as follows: (1) the experts must be able to communicate their knowledge easily and effectively to the system, (2) the experts must be able to evaluate the knowledge representation and understand what the system knows and (3) the expert system must be able to use the representation effectively in generating advice for the user.

## The Inference Engine

The primary purpose of the inference engine is to act as a mediator between the user and the knowledge base. The two major tasks of the rule-based system's inference engine are to examine the knowledge base to determine new facts and conclusions, and to determine the order that rules are to be examined and the user questioned (Harmon and King 1985). These two factors are commonly called inference and control.

The basic inference strategy used by rule-based systems is the logical rule modus ponens. Modus ponens states that, "if A then B." Thus, if A is true, then we can conclude that B is true. The conditions listed in the "IF" portions of the rule are evaluated against the facts stored in the working memory. If the facts match the rule, it is said to fire or execute. The statements listed in the "THEN" portion of rule become facts for the next rule evaluation (Harmon and King 1985).

To enhance the evaluation capability of rule-based systems, we can use uncertainty factors. The uncertainty factors allow users to convey their confidence to the questions the expert system asks them. One example of the use of confidence factors is the prospector system. This system is used to aid geologists in searching for ore deposits. When asked about the presence of a type of rock, the user responds using a scale from -5 (certain it is absent) to +5 (certain it is present) (Waterman 1986). Confidence factors also allow knowledge engineers to convey the confidence that the heuristic used to create the rule will generate proper advice for the user. An example of the use of uncertainty factors is shown in Figure 6. The need for uncertainty factors is a very important consideration in determining the best expert system approach to a given domain.

Confidence Range

| -1 | 0 | .3 | .6 | +1 |
|---|---|---|---|---|
| definitely not | ignored | slight evidence | probably confident | definite |

RULE: IF the birds are flying south

THEN probably cf(.6) the season is fall

QUESTION TO THE USER: Are the birds flying south?

USER RESPONDS: +1 (definite)

Then the inference engine combines (1) (.6)

∴ Therefore, the system concludes that probably (.6) the season is fall

Figure 6. Example of How Uncertainty Factors are Resolved.

The control mechanism of a rule-based system inference engine is responsible for providing the system's reasoning process (Harmon and King 1985). The two standard control strategies used by rule-based systems are backward and forward chaining.

Backward chaining is oriented towards proving or disproving a given goal or system conclusion. Backward chaining reduces a system conclusion into easier, simpler to achieve subgoals. For an example, see the two rules shown in Figure 7.

```
RULE #10:
     IF:     The application is real time
     THEN:   A large amount of speed is needed

RULE #33:
     IF:     A large amount of speed is needed
     AND:    A large amount of memory is needed
     THEN:   Select an XYZ Computer
```

Figure 7. Two Rules Used to Explain How Backward Chaining Works.

Using the two rules in Figure 7, the system would evaluate whether it should recommend that the user select an XYZ computer for the desired task. The system would establish if a large amount of speed and memory are needed as subgoals. If either of these subgoals have been proven false, the inference engine would disregard the rule and search for another rule which recommends the selection of an XYZ computer. Assuming both are unknown, the system evaluates the subgoal which asks if a large amount of speed is needed. The inference engine finds that rule 10 references the speed subgoal in the "then" (conclusion) part of the rule. The system establishes if the application is real time as the next subgoal. If the system is unable to find this subgoal referenced in the "then" portion of another rule, the system will ask the user a question to determine the solution (Harmon and King 1985, Hayes-Roth 1985). If any of the subgoals are proven false, the system disregards the conclusion. The advantage of this method is that the line of questioning generated by the inference device is towards proving a certain goal. This forces the user to maintain a logical line of reasoning. Backwards chaining is also thorough in that all possible conclusions are either proven or disproven.

While backward chaining is goal-directed, forward chaining is data-directed. In a forward chaining system, the objective is to find possible solutions based on the known facts. The user typically enters information which is stored as facts in the working memory (Waterman 1986, Harmon and King 1985). The system proceeds down the list of rules looking for a possible match. When a match occurs, the rule fires and new factual information is stored in working memory. The system cycles until it makes a complete pass through all the rules without any rules firing. Many forward chaining systems perform user questioning by using rules that ask the user for information if certain facts are present. Since user questioning is generated based on rule order, the questioning generated by a forward chaining system is random in nature. This is inappropriate for many expert system domains since the questioning may tend to confuse the user. An example of an appropriate usage of a forward chaining system is the XCON system used by DEC to configure computer systems. The user inputs a computer order and the system outputs the desired configuration.

Within XCON, there are a large number of possible computer configurations. These configurations cannot be narrowed to a few possible configurations by asking just a few questions. If a backwards chaining system was used, the amount of questioning generated by the system to try to verify every possible configuration would be enormous. A forward chaining system eliminates the unnecessary questioning, making it much faster for a user to configure a computer system.

The frame-based system operates on a combination of the procedures attached to each slot and the inheritance hierarchy set up by the semantic net connecting the frames. The procedures are used to find or determine the slot's value for the user. Figure 8 shows a possible sequence of frames to evaluate vehicle XYZ. Using the frames and the attached evaluation procedures as the knowledge

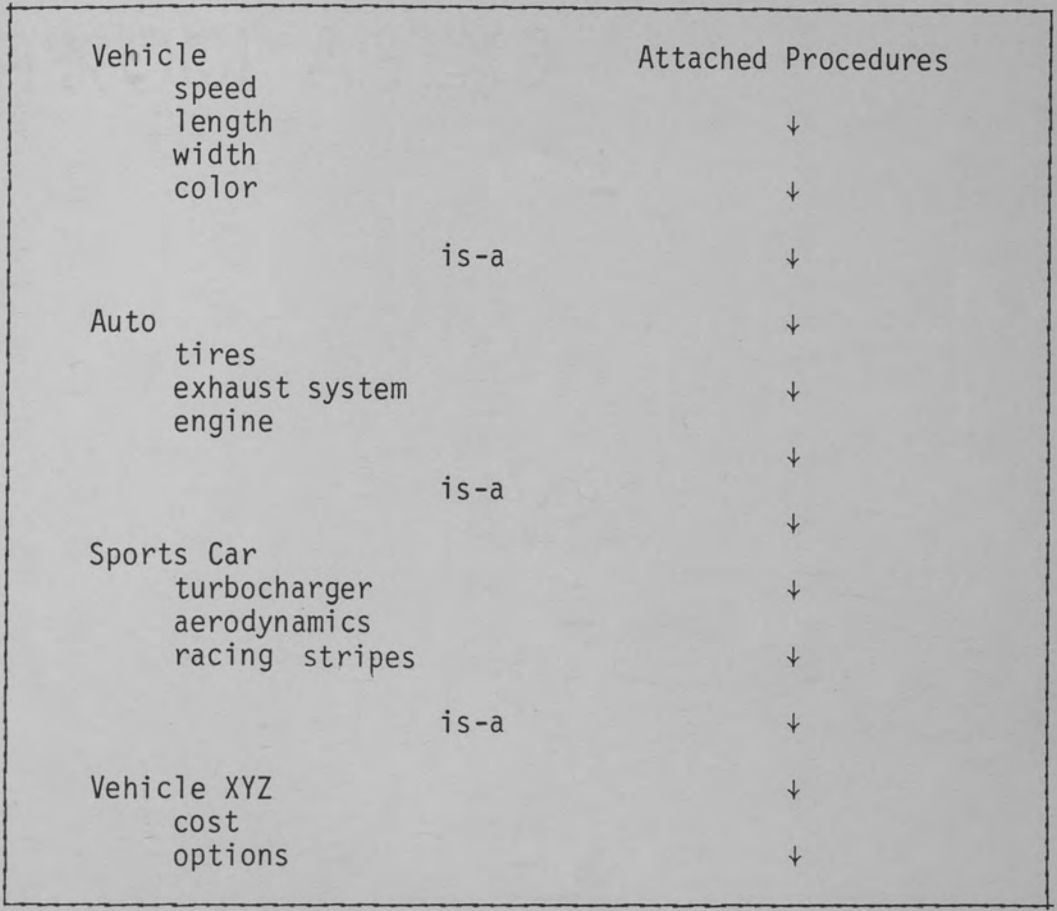

Figure 8. Frame System to Evaluate Vehicle XYZ.

base, the frame system would evaluate a given user request. Suppose the user asks for the speed of vehicle XYZ. The system would move through the hierarchy to the vehicle frame, there the system would invoke the necessary procedures to find the speed value (Waterman 1986, Winston 1984). This, in turn, could invoke other procedures in other frames where information to determine the speed is held.

## Expert System Development Tools

This section will explore the methods by which expert systems are currently being developed. Each method has inherent trade-offs and advantages that makes the method the best selection for different application domains. Defining the correct problem scope and picking the right rool for building the expert system are the two most difficult decisions to make in building an expert system.

The variety of current development methods are shown in Figure 9. On the left-hand side of the spectrum, we have the high level procedural languages. These languages are the development method for most of the expert system shells. Selecting to prototype your system with a high level language allows you to develop a shell with very few constraints, but remains a tremendous programming effort. Most of the early experimental systems were designed by using a high level language. Recently, numerous development tools have entered the software market. Most of the new expert systems have been developed with these tools because it allows the knowledge engineer to spend the majority of the time performing knowledge acquisition rather than programming. One should develop the expert system with a high level language only if none of the available development tools is able to address the requirements of the cost estimation domain.

Next are the expert system programming languages. These differ from the high level languages in that the system has an inference engine to evaluate the knowledge data. PROLOG, which stands for

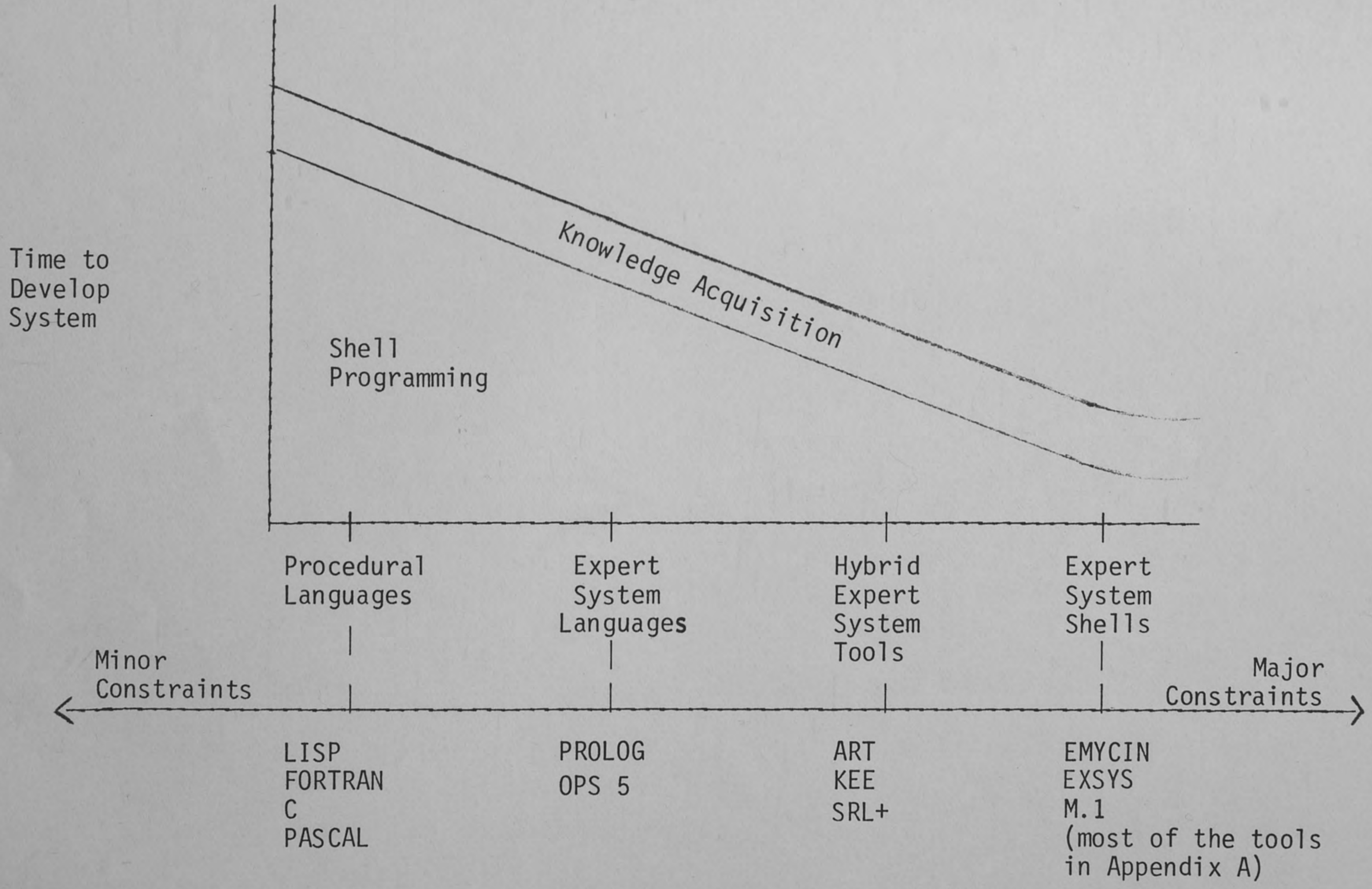

Figure 9. The Spectrum of Expert System Development Tools.

programming in logic, was designed to manipulate and evaluate logical expressions (Harmon and King 1985). The OPS 5 language is a forward chaining rule-based system language. Expert system programming languages differ from the expert system shells in that the knowledge acquisition and user interface must be designed by the programmer.

Hybrid development tools are very similar to expert system languages, except the hybrid tools are designed to support a variety of knowledge representations and inference methods (Harmon and King 1985, Waterman 1986). Hybrid systems are designed to provide a great deal of flexibility in designing the shell. Some of the disadvantages are these tools are very expensive (60K-80K dollars) and require a Symbolic 3600 or VAX computer. The vendors that sell hybrid systems provide a training course to familiarize the user with how to perform knowledge engineering with the tool. In conclusion, hybrid systems offer an excellent alternative to designing your shell with a high level language.

The last development possibility is the expert system shell. A shell has all of the elements of the expert system designed, the only thing that needs to be created is the knowledge base that allows the system to derive a solution. In selecting a shell, you have made a major design commitment. A general rule is that every shell has a task perfectly suited to it, unfortunately, if the shell does not fit the requirements of the application domain, then you have wasted a great deal of time and money. Thus, a shell is the preferred method of designing an expert system because it limits the design effort to mainly knowledge acquisition; however, the features of the shell must be thoroughly reviewed against the needs of the application domain. Appendix A shows (lists) the expert system development tools that are currently available for the IBM PC. Their features and capabilities are listed so they can be reviewed against the requirements of any given domain. The next section of this report analyzes the software cost estimation domain to determine the best method to design the demo prototype.

# CHAPTER III

# SELECTING AN EXPERT SYSTEM DEVELOPMENT TOOL

## Overview

This section of the report will perform an analysis of the procedures used by software cost estimators in estimating the costs and level of effort involved with a given training system. The different types of expert system features will be reviewed as to how they could address this application. Based on an analysis of the possible development methods, a method will be selected for developing the demo prototype. The general methods used in selecting a tool for this domain will be applicable for other expert system applications.

## The Cost Estimation Domain

Before deciding on a tool and a general knowledge engineering approach, it is necessary to thoroughly analyze the cognitive methods used by experts in deriving a solution. Hopefully, an expert system that performs with the same level of competence as the current domain experts can be developed.

The given input to the cost estimation cycle is usually a specification of the training system. The specification defines the scope and performance criteria of the training simulator. Sometimes a request to generate a software cost estimate has only several vague concepts as the requirement. This is usually because a specification has not been written and the military agency is trying to determine the scope and cost for the trainer. This will allow the agency to determine if the trainer is within their funding and budgeting constraints.

Depending on the thoroughness of the description, the engineer performing the estimation may have to perform some system level design work to define the hardware and software features of the trainer. To derive a preliminary estimate, the features of the trainer being estimated are compared to other trainers that have already been designed. Once trainers that have similar operational features have been identified, the documentation that was generated during the system's development is used to aid in developing the estimate.

The most useful document for estimating the level of coding effort for each function pertaining to the trainer is the Program Performance Specification (PPS). The PPS will translate and allocate the trainer system functional requirements specified in the trainer specification to software functional requirements. This document segments the complete computer program into computer program components (CPC) which are a functionally distinct part of the computer program. Each computer program component (CPC) is made up of one or more computer program modules. A computer program module is a unit of software which performs a sub-function of the computer program component. Because of rules set forth by software standard MIL-STD-1644, each program module may contain no more than 200 lines of code. Most modules average around 100 lines of code. By counting the number of modules associated with a given functional requirement and multiplying it by an average module size, an estimate of the number of lines of code could be generated. The PPS and associated documents provide a work breakdown structure for the coding effort. Once a line of code estimate is derived, the total "lines of code" is phased into a costing model that considers programmer and engineer productivity and programmer/engineer hour costs. Some models also consider the costing for certain contractors with different levels of software development experience. At this time, the author will limit the scope of the cost model to a general estimate versus a model that would evaluate the capabilities of the personnel and organization performing the coding.

The basic duty of the cost estimator is to generate a work breakdown structure from the requirements of the specification. The ideal expert system would use the following evaluation scheme: the inputs to the system would be the various features, options and performance criteria for the training system. Time sensitive cost information, like the cost per engineering personnel hour, will be external. From the inputs, the expert system will evaluate the scope and complexity of a training system to meet the given criteria. The system performs a work breakdown structure to estimate the types and numbers of software modules required. A line of code estimate for each module would be generated either by assigning a direct numerical value or by an algorithm. It may also be desirable to present the module estimate as a statistical value. The values of each module in the work breakdown structure are summed to generate a total lines of code estimate. It is also desirable that the system explain the work breakdown structure generated and the past training devices used as references.

## Selection of a Knowledge Representation

The possible selections for a knowledge representation are the frame-based system, the rule-based system or a custom or tailored knowledge representation fit to the cost estimation domain. Unfortunately, cost and time are both factors which enter into the selection criteria. If the custom knowledge representation is selected, the time and effort to develop the shell alone would be enormous. Many of the shells on the software market cost upwards of $10,000, far too much money for developing a prototype system. As a secondary criterion, the knowledge representation selected should compliment your own knowledge engineering capabilities and be developed within a reasonable amount of time (Waterman 1986). The development tools should be within the development budget.

The frame-based system offers the most implicit form of knowledge representation because the frames of the system can be used to represent the elements of the work breakdown structure. An example frame might be the software to support the computer image generation equipment. The slots for the computer image generation frame would contain attributes such as the number of channels and the image complexity. Other sub-module frames would be attached to the computer image generator frame via a semantic network. This hierarchical setup would allow the user to see and understand the work breakdown structure the system is using to derive a solution. Other hierarchies could be set up among common elements in the work breakdown structure; for example, if the software controlling a radar has simularities to the software controlling other instruments, then parts of each software element in the work breakdown structure could be represented by a general frame for commonly held features. Also, the hierarchical features among trainers themselves could be represented. For example, a weapon systems trainer is an operational flight trainer with weapon systems added. Each frame might have procedures attached to each slot to gather the information necessary to determine the slot's value. To use the system, the user would ask for the total software costs. This inquiry would trigger the if-needed procedure for the total software cost slot. The procedure would then ask for information from other slots, which trigger other procedures, which ask the user for needed information to fill in values where needed. The disadvantages of a frame-based system is that the procedures to compile the information to determine the value results would be quite complex. In addition, none of the shells or hybrid systems currently on the market support the math necessary for the cost estimation domain. Even if they could support the required math, the 60-80K price of the frame-based tools currently make them a financially impossible choice. The author's recommendation is to reconsider the frame-based system when the available tools support the required math functions for determining slot values and become lower in price.

The next knowledge representation reviewed is the rule-based system. The rule-based system has to its advantage a very large assortment of shells and tools. Rule-based systems are by far the most mature knowledge representation systems. In reviewing the use of a rule-based system, the control and inference methods of the various development tools must be compared with the requirements of the software cost estimation domain.

The primary disadvantage in selecting the rule-based knowledge representation is that the work breakdown structure used by the system will be much less implicit than a frame-based system. The user must sort through the different rules to determine how they are related to each other, and how the work breakdown structure is analyzed.

The rules will have to be written such that they are made to fit only certain hierarchies within the work breakdown structure. This requires that the knowledge engineer place the proper number of "IF" conditions on each rule.

One consideration is using a rule-based system is to determine if uncertainty factors are used when the expert system is trying to evaluate evidence. This is useful in domains where a simple yes-no answer is insufficient. In the cost estimation domain, the user will have to know exactly what is needed. For example, it is unlikely that a user would "maybe" desire a motion platform or graphics system. In order to generate a good estimate, the features and operational criteria of the training system would have to be well-defined in the user's mind. The best application for an uncertainty factor would be in the numerical value assigned to represent the line of code estimate for every element in the software work breakdown structure. These values would be combined to generate an overall value for cost and total lines of code. Many of the software costing models use statistics to convey to the user the uncertainty involved with generating a cost estimate. Some studies indicate that if given the exact same coding job, the number of lines of code generated by different programmers can vary by more than 30%. A major drawback to using statistical values is the amount of data that would have to be gathered on each work breakdown element to generate a proper value. In addition, the users of the system may have difficulty in properly utilizing and understanding the generated values. None of the tools on the market support statistical values. These values can be created by using multiple variables for every estimate or by calling external programs.

The two standard control strategies, forward and backward chaining, must be reviewed as to how they could be utilized in the software cost estimation domain. Both of these control methods were explained in detail in an earlier section.

In a forward chaining approach to the cost estimation domain, the user enters requirements and functional information about the trainer. The system scans the rule-base looking for a match based on the information entered. To derive a solution, the system would need enough information to generate an accurate guess for each of the elements in the work breakdown structure. A forward chaining system would need to combine both forward chaining and some backward chaining. The backward chaining would allow the system to ask additional questions in areas where the user inputs were deficient. Also, the system would need to realize if its knowledge base was insufficient to generate a cost estimate. An example would be for a user to ask for a feature on the training system that is not supported by the rule-base. In general, the software cost estimation domain does not lend itself to a forward chaining system.

Within the rules, numerous algorithms for generating a line of code estimation will be specified. From these algorithms, a backward chaining system would have to ask the user questions to determine a value for the variables used in the algorithms. The major advantage of a backward chaining system is that all of the elements in the work breakdown structure known to the expert system will be tested.

Potential users of these systems usually prefer the system ask for the information it needs, instead of the user inputting the trainer requirements. Although the number of possible elements within the work breakdown structure is huge, the elements can be quickly narrowed or eliminated by asking just a few questions, thereby eliminating the major reason for selecting a forward chaining system.

The major disadvantage is that the user would have to recognize if certain features in the trainer's requirements were not asked for by the expert system. Since the main purpose of this system is to act as a co-worker, with an engineer doing the cost estimating, this may not be a major problem. The rule-based expert system would have to support backwards chaining on selected variables instead of the usual symbolic choices. Only a few rule-based expert systems on the market support this feature,which makes selection very limited and difficult.

The last possibility is to develop a unique or a variation of one of the current types of knowledge representations. To select this path would require a major time commitment. Many of the knowledge engineering projects that have chosen the custom shell course had to undergo several prototype changes before the knowledge acquisition process began. In the author's opinion, it would be more time efficient to perform the demo with an existing tool and use the knowledge engineering experience in building the demo to better analyze the requirements of the chosen domain.

Therefore, the most reasonable selection for a knowledge representation would be a rule-based system that supports math and backward chaining on variables.

## Selection of the Development Tool

The ideal selection would be to find an expert system shell that resides on an IBM PC that would be suitable for the cost estimation domain. The advantages of selecting a shell that can reside on the IBM PC are the lower initial costs of purchasing the software and faster development due to the common availability of the PC in the work place. The two possibilities for a development computer are the VAX 11/780 and the Zenith PC because of their availability at the Naval Training Systems Center. As a general rule, the software costs for a comparable shell or language is approximately ten times greater for the VAX than the PC. As the system grows, it may be necessary to place the system on the VAX; however, the PC offers the most attractive choice for developing the prototype.

Shown in Appendix A is a list of the development tools currently available for the IBM PC. This list was compiled by a committee at the International Artificial Intelligence Conference held in Los Angeles, in the summer of 1985. Out of the list, two tools seemed to be able to address the domain requirements. These tools are EXSYS and M1 by Teknowledge. Both tools support the required math, variables and backward chaining on variables to find their values.

The expert system shell, M1, supported many features involved with confidence levels which are totally unnecessary. Also, since the shell is written in PROLOG, the execution speed is quite slow. All of the symbolic languages, like LISP and PROLOG, have slow execution speeds because many of the systems run on interpreters, and the factual information is managed by creating a giant list of attribute-value pairs that have been proven true. The process of checking the attribute-value pairs in the rules against this list is a very slow process for an IBM PC. Most of the shells written for the PC are now being done in an algorithmic language like C. In fact, Teknowledge's M1 is now in the process of being rewritten in C. The major advantage of an algorithmic language in performing a rule-based system is numeric values can be assigned to each value and an attribute can be a certain location in an array. When the system determines the proper value for the attribute, it can be placed in the reserved array location. This method makes comparisons and searches much faster.

Based on the $10,000 cost, and since it is implemented in PROLOG, the author decided to reject M1 in favor of EXSYS. EXSYS is very user-friendly. It supports a knowledge base editor and a consistency checker. The system supports all of the major math functions and uses backward chaining to determine values for every variable that is going to be displayed at the end of the user session. EXSYS is written in C which permits greater operational speed and allows more rules to be stored with less memory, permitting large expert systems to reside on the PC.

# CHAPTER IV

# THE DEMO PROTOTYPE

## Overview

The major function of the demonstration program is to show how a complex software estimation system could be implemented using expert system techniques. The function of the expert system will be to provide an implicit format for capturing and perfecting software cost estimation heuristics. It is impossible for the author to attempt to provide a verifiable model for generating a cost estimate. Such a model or models will require a long-term research, data collection and validation effort. It is important to note that the effort in estimating a trainer's cost is different at the proposal phase than at the later development phase, where the product definition is much better. For this reason, different expert systems supporting different models would be required for estimation at different stages in the development effort. The system the author will demonstrate will show how the system could support decision making in the early phases of a project. In the conclusion of the report, the author will suggest additional features to add to the system.

The rules in the cost estimation expert system will be directed towards analyzing one of the two major functions. The first function is determining cost, time and personnel required to complete the project. Different models should be used for different types of software development efforts. Many advanced models use adjustment factors to reflect project difficulty and personnel capability. Rules will be written for each model describing the conditions under which a given model should be utilized. Other rules can define the condition under which different adjustment factors should be utilized.

As an input to the model, a "line-of-code" estimate will be required. Thus, the second function will be to attempt to estimate the total number of "lines-of-code" to implement a training system with given functional requirements. The rules in this section will be oriented toward breaking the effort down into subfunctions and estimating the "lines-of-code" required to implement the subfunctions. The condition part of each rule will state the circumstances under which a given subfunction would be required. The estimates of all of the desired subfunctions will be summed to generate an overall estimate for the project.

The following sections will discuss the cost estimation model and the generation of the "line-of-code" estimate. The last section will discuss the operation of the demo.

## The Software Estimation Model

The software estimation model which is the most thoroughly documented and accepted is COCOMO (Cost Constructive Model). The COCOMO model has three versions: the basic, intermediate and detailed models. Each model can serve different cost estimation requirements. The scope of these models will be discussed later. For the demonstration program, the author will implement the COCOMO basic model. The basic model will provide a sufficient demonstration of the feasibility of using expert systems to perform software costs estimating. The advanced COCOMO models follow the basic model, except that a more detailed analysis of the project scope is required. The COCOMO model is thoroughly described in the book, Software Engineering Economics, by Barry Boehm (1981). This text makes an excellent reference for any person attempting to perform software cost estimation.

The COCOMO model generates estimates based on the number of thousands of delivered source instructions (KDSI) in the software project. A source instruction includes all program instructions created by project personnel that are processed into machine code. It also includes job control language, format statements and data declarations. Excluded are comment statements and unmodified utility software. It is important to realize that source instructions are not a uniform commodity. Yet, most models prove mathematically that the number of source instructions is the most reliable variable in generating an accurate estimate (Boehm 1981). To reflect the difference in the basic effort involved with different types of projects, COCOMO splits efforts into three different groups: organic, semi-detached and embedded.

The simplest mode of development is the organic mode. An organic object would require little new hardware integration, innovative data processing architectures or algorithms (Boehm 1981). The project would place a low requirement on an early completion. Due to the generally familiar nature of organic projects, most project personnel can contribute in the early stages and throughout the development process. This makes for higher productivity in developing the project.

On the other end of the spectrum is the embedded mode. The embedded mode project must operate within very tight operational and reliability requirements (Boehm 1981). Due to the complexity of the development effort, longer design and testing phases are required. Since most personnel can be utilized for limited functions within the development cycle, higher peaks in the personnel curve occur.

Last, is the semi-detached mode. This mode has a mixture of both organic and embedded mode characteristics. Based on the examples presented in the Software Engineering Economics text (Boehm 1981), the author can conclude that the software development effort for most training simulators will fall under the semi-detached mode.

The basic COCOMO effort and schedule equations for all three modes are shown in Table 2. These equations estimate the number of man-months (MM) and the time to develop the software (TDEV).

The basic COCOMO equations will provide gradually increasing programmer productivity with larger size programs. The model assumes a man-month consists of 152 hours of working time (Boehm 1981). The

TABLE 2

BASIC COCOMO EFFORT AND SCHEDULE EQUATIONS

| MODE | EFFORT | SCHEDULE |
|---|---|---|
| Organic | $MM = 2.4(KDSI)^{1.05}$ | $TDEV = 2.5(MM)^{0.38}$ |
| Semi-detached | $MM = 3.0(KDSI)^{1.12}$ | $TDEV = 2.5(MM)^{0.35}$ |
| Embedded | $MM = 3.6(KDSI)^{1.20}$ | $TDEV = 2.5(MM)^{0.32}$ |

SOURCE: Boehm 1981

development phases used by the COCOMO model are highlighted in Table 3 (Boehm 1981). The development and maintenance phases are add-on phases that are not estimated by the COCOMO model, but by a separate model. These two phases are the equivalent to life cycle support efforts. Each phase can be broken down into COCOMO functions. This is done by using the tables in the Software Engineering Economics textbook (Boehm 1981) for the mode of development being estimated. Table 3 also presents the phases used by the military standards for developing trainer software to permit a comparison to the COCOMO development phases.

The basic COCOMO model provides a level of accuracy useful in the rough early stages of software product definition. These stages occur before any actual development work begins. The intermediate and advanced models are more suitable for cost estimation

TABLE 3

A BREAKDOWN OF THE FUNCTIONS AND PHASES OF THE COCOMO MODEL AND THE PHASES OF THE STANDARDS USED TO DEVELOP TRAINING SYSTEMS SOFTWARE

| COCOMO MODEL FUNCTIONS | COCOMO PHASES | DOD-STD-2167 PHASES | MIL-STD-1644 PHASES |
|---|---|---|---|
| Requirements analysis | Plans and requirements | Pre-software | Planning phase |
| Product design | Product design | Software requirements analysis | Analysis phase |
| Programming | Programming-detailed design-code and unit test | Preliminary design | Design phase |
| Test planning | Integration and test | Detailed design | Production phase |
| Verification and validation | Development/ Maintenance | Coding and unit testing | Integration phase |
| Project office functions |  | Computer system component testing | Acceptance phase |
| Configuration management and quality assurance |  | Computer system configuration item testing |  |
| Manuals |  | System integration and testing/operational testing and evaluation |  |

SOURCE: Boehm 1981

in the more detailed stages of software product definition (Boehm 1981). The intermediate model uses an additional fifteen adjustment variables which provide greater estimation accuracy. Some of the adjustment values include required software reliability, programmer capability and required development schedule. Although the author chose not to implement the advanced models, the rule-based system would provide an excellent tool for implementing the advanced models. Rules could be written for every adjustment value describing the conditions under which the adjustment variable should be given a certain numeric value.

Shown in Figure 10 is a rule and a qualifier. Qualifiers are the basic elements used by EXSYS to create rules and to ask user questions (Huntington 1985). The qualifier can be split into two parts: the attribute and its values. Possible values for the attribute are presented in a menu form. The symbolic statements in the rules are created combining a qualifier with a selected value, such an example can be in the "IF" portion of rule 1. To create this statement, qualifier #1 with value one would be specified.

The major reason for selecting EXSYS to develop a software cost estimation system was its capability to support mathematical formulations and perform backwards chaining on selected variables.

While in the rule-base editor, the system will ask if a certain variable should be displayed at the end of the user session. If the answer is "yes," then the system treats the variable as a goal and

```
Qualifier #1

     The basic COCOMO model to select is the:

     1.  organic mode

     2.  semi-detached mode

     3.  embedded mode


Rule #1:

     IF:  The basic COCOMO model to select is the
          organic mode

   THEN:  [MM] is given the value (2.4*EXP(1.05*(LOG([KDSI]))))
                                         and
          [TDEV] is given the value (2.5*EXP(.38*(LOG([MM]))))

NOTE:  Basic COCOMO model for effort and schedule for organic
       mode
```

Figure 10. An Example of a Qualifier and a Rule to Implement the Basic COCOMO Model (Boehm 1981).

will attempt via the rule-base to find a value for it. If the system cannot find a value for the variable by inferring the rules and/or asking user questions, EXSYS will ask the user to enter a value for the variable. This will be necessary for such items as labor costs, which are very time sensitive.

In the "THEN" portion of rule #1 in Figure 10, two COCOMO formulas for determining man-months and the time to develop an organic mode project are stated. The variables, MM and TDEV, will be displayed at the end of the user session. Because of this, EXSYS will attempt to find a value for qualifier #1 to determine if rule #1 is true. First, EXSYS will examine the rule-base for any rules that use qualifier #1 in the "THEN" portion of the rule. The knowledge engineer may wish to create rules that would infer the conditions under which certain modes should be utilized. Assuming no such rules exist, the system would ask the user to select a value for qualifier #1. Assuming a one is selected, the two formulas would become factual information. The unknown variable, KDIS (thousands of deliverable source instruction), will become a goal for the system to determine. To find the value for the total number of source instructions, the rules to generate a "line-of-code" estimate will be utilized. Other rules in this section can state conditions for which various formulas for manpower, cost and work breakdown be utilized.

## Generating a "Line-of-Code" Estimate

Software cost and size estimates are typically based on historical data. Therefore, data must be collected during current projects in order to estimate effort and schedule for future projects. The experts in the cost estimation field suggest that organizations develop procedures for software cost data collection throughout the life cycle of a software development process. In Software Engineering Economics (Boehm 1981), the suggested data collection forms and procedures are presented.

The purpose of this section is to examine how a rule-based expert system could aid in storing the sizing data that has been collected.

For planning purposes, it is useful to organize project activity elements into a hierarchical structure called a work breakdown structure (WBS). There are two major hierarchies to generate a WBS (Boehm 1981). There is an activity hierarchy and a product hierarchy. An activity hierarchy indicates the functions which may deal with the software development effort. An example function could be programming, quality assurance or configuration management. The activity hierarchy is useful for generating man-month estimation models, but not for estimating the number of "lines-of-code." The product hierarchy indicates how the various software components fit into the overall software system. The product hierarchy has already been discussed to a limited degree in an earlier section examining the cost estimation domain. An example of the basic structure of a product hierarchy WBS is shown in Figure 11. The general feeling among most estimators is that the smaller elements the product hierarchy is broken down into, the less the possibility exists for making a large estimation error.

The rules in the "line-of-code" estimation section would be based on data collected from past programs. In Figure 11, in the very last function block, are listed "unsupported functions" as a product. It is important to realize that every new software development project will have requirements not performed by past projects. This function will create a variable that will allow the user to estimate the number of lines of code to support these functions. When data is collected on the unsupported functions of past programs,

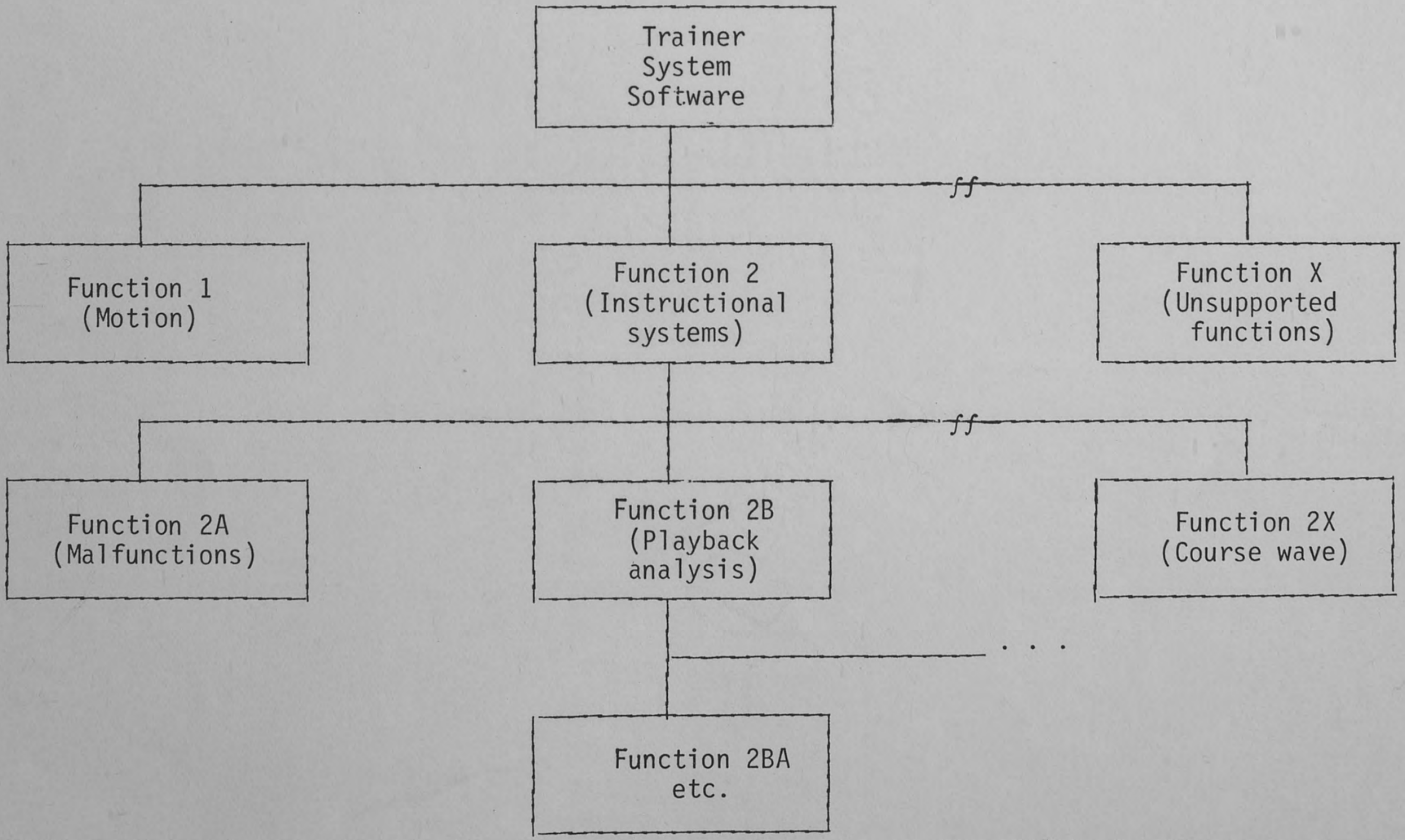
Trainer System Software
Function 1 (Motion)
Function 2 (Instructional systems)
Function X (Unsupported functions)
Function 2A (Malfunctions)
Function 2B (Playback analysis)
Function 2X (Course wave)
Function 2BA etc.
. . .

Figure 11. An Examole of a Product Hierarchy WBS.

the rule-base can be expanded to cover a wider range of development efforts. The control mechanism in EXSYS allows the user to follow the steps the system traces through the rule-base. This utility will allow the user to realize the scope of the knowledge base and determine what the deficient functions are.

In Figure 12, three example rules are shown to aid in the understanding of how a "line-of-code" estimate could be generated from a rule-based system.

```
RULE NUMBER 1:

     IF:     The type of trainer is an XYZ system

     THEN:   [DSI] is given the value [function 1] +
             [function 2] + [unsupported functions]

     NOTE:   Data from trainer system XYZ

RULE NUMBER 2:

     IF:     Function 1 is desired

     THEN:   [Function 1] is given the value [function 1A]
             + [function 1B]

RULE NUMBER 3:

     IF:     Function 1 is not desired

     THEN:   [Function 1] is given the value 0
```

Figure 12. Some Rules to Aid in the Understanding of How a "Line-of-Code" Estimate Could be Generated by a Rule-Based System.

In rule 1 in Figure 12, the major functions of an XYZ system are summed to generate a deliverable source instructions (DSI) estimate. The only variable that the system desires to display at the end of the user session is the DSI variable. Therefore, to find a value for DSI will become a goal for the system. Assuming that the trainer is an XYZ system, the formula for solving DSI will become factual information. In the process, all of the other variables in the formula will become sub-goals for the system. To find a value for "function 1," the system will search the rule-base for rules that define "function 1" in the "THEN" portion of the rule. The system will then create goals out of the conditions in the "IF" portion of the rule. In Figure 12, rule #2, a further breakdown of the functions of "function 1" are specified. This breakdown would continue to the point at which a "line-of-code" estimate could be generated. Structuring rules in this fashion allows the knowledge engineer to easily expand the WBS of a given training system when new data is collected.

## The Demo Program

In attempting to demonstrate the feasibility of using an expert system as a decision support tool for engineers performing cost estimation, the author found the data required to generate and validate custom modules is not available. Therefore, the thrust of the study must be explaining how an expert system environment could best support the implementation of a decision support tool that will evolve in the future.

In Appendix B, a rule listing of the small demonstration program is presented. When creating a new expert system, EXSYS asks the user for the subject and the system's author. EXSYS uses this information to generate an introduction to the system. Following the introduction, the "starting text" is displayed. This text can explain the scope and purpose of the system. The ending text, which is displayed at the end of the user session, provides the user with guidelines for interpreting the advice or information generated by the system.

The body of the rules are split into three major parts. Rules numbered 1 through 7 are designed to aid the user in selecting the proper COCOMO basic model. Rule number 1 selects the semi-detached mode for a flight trainer based on examples in Software Engineering Economics (Boehm 1981). Rule number 1 also presents an example partial product WBS for a flight trainer. In this situation, only the navigation and the fuel system are presented. In an actual analysis, the product WBS at the functional level would include over twenty elements. Based on the author's brief study, the elements which will require the most "lines of code" would be the program executive, computer image generation and any weapon or tactics simulation. The rest of the elements of the WBS would be generated in a manner similar to the methods presented for the navigation and fuel systems. The variable, "KDSI," in rule 1 is one of the variables to be displayed at the end of the user's session; therefore, the system will attempt to generate a value for the variable using backward chaining. Rules 2 through 7 define the criteria for selecting a COCOMO model. These rules will be utilized by EXSYS's control mechanism if the device is not a fixed wing trainer.

The next major section is rules 8 through 10. These three rules define the basic formulas for the three COCOMO modes of development. In these rules, the variables, "MM" (man-months) and "TDEV" (time to develop the project in man-months), are defined. Both of these variables are required to be displayed at the end of the user session.

The rules which are numbered 11 through 26 are designed to show how a rule-based system could be used to generate a basic line of code estimate. The usefulness of the "line of code" estimate is during the early stages of the procurement cycle where the product definition is very limited. As the project matures, the emphasis of the expert system should shift to a more detailed model that analyzes programming team capability, types of tools and languages utilized and other factors that will affect delivery time and cost.

The system's function must be broken into the smallest possible elements. An example of this is rule number 11 which splits navigation systems into a list of navigation instruments on past trainer systems.

In rules 12 through 14, an estimate for the "lines of code" to support the doppler radar simulation is generated. The qualifier that generated these three rules is qualifier #4 in Appendix B. Each value associated with the qualifier is assigned a code estimate by one of the three rules. The last value, called "take your own GUESS," is not supported by any of the rules; thus, the system will be forced to ask the user for the value. Another possible method is to use multiple qualifiers like in rules 19 through 23 which attempt to generate an estimate for the radar altimeter function. Two qualifiers are very important in finding a value for the function, these are numbers 6 and 8. The rule-base must address every possible combination of these qualifiers or the user will be forced to generate an estimate for the combination, but supported by the rule-base. In rules 20 through 23, every combination is given an estimate.

Now that a "line of code" estimate and a model have been selected by the system, the two are combined to generate a value for the key variables in the COCOMO model. At this point, the system needs to complete the estimation by generating an activity WBS for the project. In COCOMO, the phases are split into percentages of the total man-month estimate based on the development mode and the project size in deliverable source instructions. To perform this breakdown in a rule format would be an undesirable task. The most implicit format for this data would be in a spreadsheet format. A major inadequacy of version 2.3 of EXSYS is its inability to create data files for external program analysis. Version 3.0 (the newest version which the author has not been able to obtain) allows the system to create ".PRN" files containing key variables. This file could be utilized by either external programs that figure calculation factors like inflation or by spreadsheets like Lotus "123." In "123," the ".PRN" file can be loaded in by using the "/file import" command. In the conclusion, a final configuration for the system is suggested.

# CHAPTER V

# CONCLUSION

In 1980, approximately 2% of the gross national product was spent on software. Growth in software costs is considerably greater than the rest of the economy. In the area of training systems, software costs have become the lowest cost item in any training device procurement. Therefore, methods to guide managers in making budgetary decisions regarding software development costs have become increasingly important. It should be obvious that any organization heavily involved with either software procurement or development should place an increased emphasis on building a software cost data base and developing estimation models. To perfect estimation models and generate data useful to the development managers, a data analysis system to meet the requirement of all possibilities within the domain should be developed.

In expert system prototype development, the knowledge engineer usually must develop several prototype approaches before an approach which is suitable to the user is developed. Version 2 of the EXSYS program is lacking several utilities that would make the implementation of an expert system much easier and complete. In version 3 of EXSYS, numerous new utilities have been added to increase capability of the system to support the cost estimation domain. Some of the new features include "IF-THEN-ELSE" rules. This type of rule would allow the user to combine many parts of the product WBS analysis. An example would be rules 2 and 3 of Figure 12. The new rule would be assigned to the function's variable, else the value would be zero. Other new features include a built-in report generator and more flexibility in exchanging information with external programs. Because the types and number of expert system tools are presently very limited in scope and few in number, any person attempting to develop an expert system in the future should re-survey the market for expert system tools that may be useful for the selected problem domain.

In implementing an expert system that will be useful to both the user and knowledge engineer, it is necessary to expand the capabilities of the rule-based system in communicating results to the user. The best methods of performing this is to utilize spreadsheet packages and external programs in addition to the rule-based system. In Figure 13, a suggested implementation of a cost estimation expert system is presented. One of the major elements in the configuration is a spreadsheet program. The spreadsheet provides a useful way for the cost estimation engineer to enter a percentage breakdown by phase and function. In COCOMO, each development mode has its own project activity distribution by phase. The phase percentages are referenced by program size. Clearly, the most implicit and understandable method to present the percentages is in a table form. Thus, storing the information in a table format

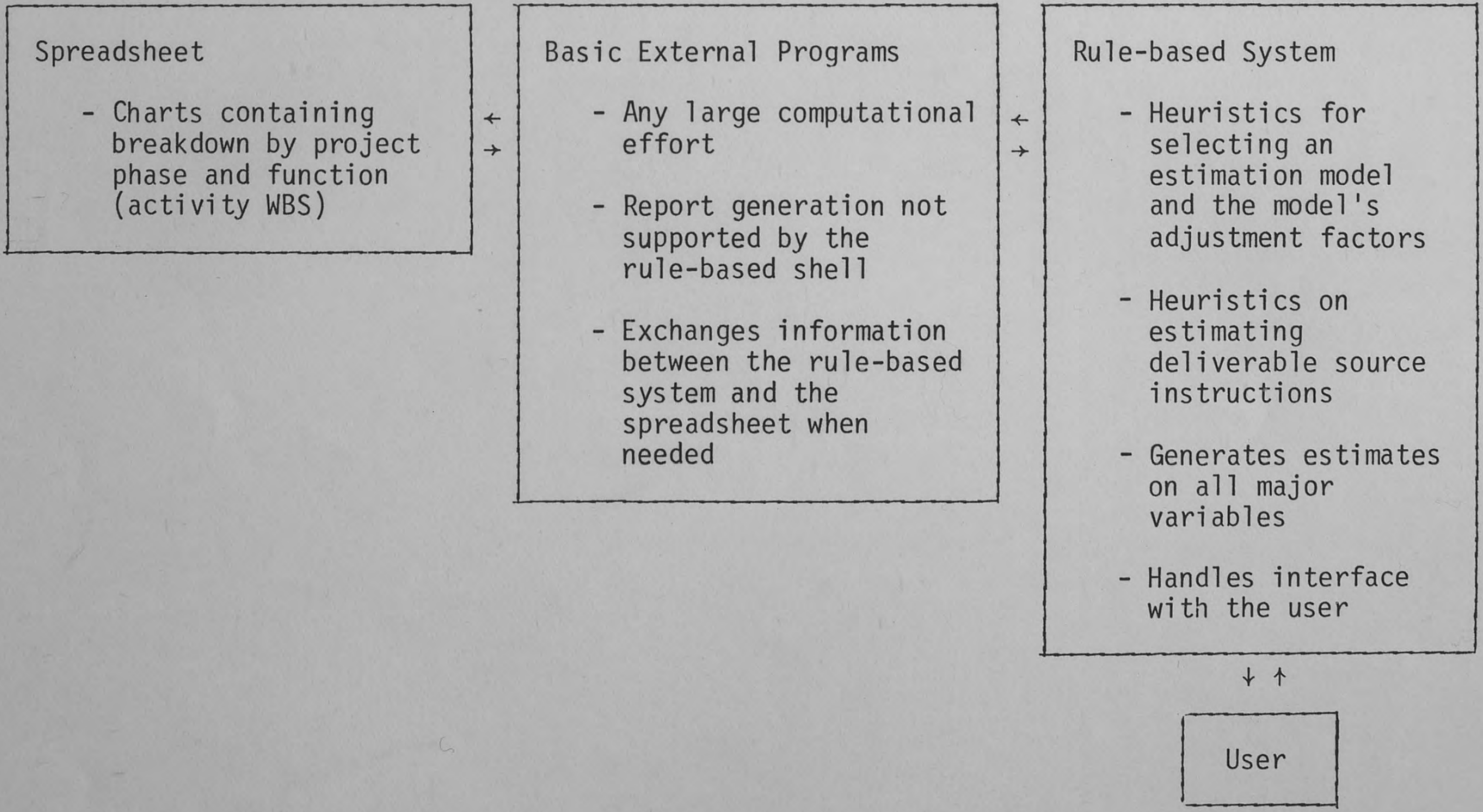
Spreadsheet
- Charts containing breakdown by project phase and function (activity WBS)
Basic External Programs
- Any large computational effort
- Report generation not supported by the rule-based shell
- Exchanges information between the rule-based system and the spreadsheet when needed
Rule-based System
- Heuristics for selecting an estimation model and the model's adjustment factors
- Heuristics on estimating deliverable source instructions
- Generates estimates on all major variables
- Handles interface with the user
User

Figure 13. The Suggested Implementation of a Software Cost Estimation Expert System.

would be more superior than trying to store the data in a rule format. It is possible to have a basic program to utilize the spreadsheet's data file to generate a project breakdown, or to have another spreadsheet to generate the breakdown, based on certain inputs by the user.

The external BASIC program provides a method for allowing the system to support complex computational analysis of the expert system's results. An example would be the effect of inflation, project overhead and contractor profit. Regardless of the decisions made by the rule-base, the basic computation of these variables to achieve a cost breakdown will remain the same. Therefore, having a separate program (or programs) to provide a breakdown of expenses in a format that can be utilized by management is a desirable approach.

The brain of the whole system outlined in Figure 13 is the rule-based system. The system contains the rules by which decisions concerning the software cost estimation variables are made. The system will generate an estimate on all major variables based on questions asked to the user. In any area where a large computational effort or special report generation is involved, the system will exchange information via a data file and invoke an external program that can support the desired functions.

After all the commotion generated by expert systems, some users may wonder if you have to develop the heuristics that go into the expert system, why not just write a program in a procedural language like Fortran or Basic, or develop a spreadsheet using complex macros to implement the system. The author has developed several reasons why a rule-based system would be superior to the others mentioned. First, and perhaps the major reason, is the implicit presentation of a knowledge representation like a rule-based system presents to both the knowledge engineer and user. While working on this paper, the author discussed software cost estimation with another engineer who wrote a program to implement several models. It was very difficult to extract the decision processes that were embedded in the code. The "condition-results" format used in a rule-based system can be easily understood by both the domain expert and the knowledge engineer, allowing the team to spend their efforts on validation of the system, instead of programming it. Second, the rule-based system provides an easy method to add or subtract evaluation conditions based on circumstances. Third, the backward chaining control mechanism automatically generates user questioning based on conditions that cannot be satisfied by the rule-base. To generate the same user questioning system in a procedural language would be a huge effort. The inference engine uses the knowledge base to create a logical decision tree. The changing of one rule or its conditions could greatly alter the tree generated by the inference engine. If this decision tree was implemented in a procedural language, the changing of one decision parameter could require a major re-ordering of the decision process and user questioning. Fourth, most expert systems, including EXSYS, have trace capabilities to allow the user to follow the control mechanism of the expert system as it traces through the rule-base. Also, expert systems allow users to ask why the system is asking a certain question to which the system responds with the rule or conditions it is trying to satisfy. All of the conditions above combined make a rule-based system a worthwhile choice for this problem domain.

The disadvantages of implementing a rule-based system surface when the system grows in number of rules and the interrelationship between facts grow. At that point, generating new rules that correctly and logically integrate with the rest of the rules will become more difficult and improper relationships between rules could result.

In continuing this project, several critical questions or problems could occur. While lots of work has been done on developing models for generating cost and activity breakdowns based on a line of code input, almost no guidelines have been developed for generating the line of code estimate. In the simulator area, trainers such as aviation trainers have similar elements in the product WBS which can be associated with past programs. Other trainers, such as surface weapons trainers, often have uniquely functional requirements which cannot be associated with past development efforts. As additional data is collected, it is likely that better guidelines for developing lines of code estimates can be developed. Another major question is the suitability of rule-based systems to support the cost estimation domain. As data is collected and new models are developed, some of the new models may be awkward or impossible to implement on a shell such as EXSYS or any other rule-based system. Many new shells that support a wide range of capabilities are entering the market at a rapid rate. Anyone attempting to support a cost estimation system should keep informed of the new products which may be more suitable to support the requirements of the system.

Another major question is on what computer system will the final expert system reside. The EXSYS shell can support 3000-5000 rules on a PC with 640K of memory. This should allow the system to begin development using a PC. The author envisions that the final system will be a combination of expert systems providing analysis of different types of trainers and different systems to estimate costs at different stages of product definition/development.

The software engineering development cycle forms a neat step-by-step development sequence. An example of this sequence can be seen in COCOMO's development phases. The knowledge engineering development cycle involves a constant cycle of prototyping, criticizing and refining program heuristics. This cycle will be an ongoing activity as the technology and software procurement standards for training systems change. If any organization is to provide reasonably accurate software cost estimates, a comprehensive project data collection system should be instigated. To support the collection system, software management tools to store, analyze and provide other users with the capability using the heuristics learned on past programs to analyze a current development effort is a necessity. The author feels the type of system suggested by this paper deserves serious consideration by any organization involved with either software development or procurement.

## APPENDICES

# APPENDIX A

# IBM PC EXPERT SYSTEM DEVELOPMENT TOOLS

| COMPANY | PRODUCT NAME | PRICE ($) | WRITTEN IN | MAXIMUM RULES | COMMENTS |
|---|---|---|---|---|---|
| Artelligence, Inc.<br>1402 Preston Road<br>Dallas, TX 75240 | OPS5+ | 3,000.00 | C | 1500 | Implementation of OPS 5, a forward chaining system. Requires a mouse. |
| California Intelligence<br>912 Powerll Street<br>San Francisco, CA 94103 | XSYS | 1,000.00 | IQ LISP | Systems can be linked | Forward and backward chaining on an opportunistic basis. Supports uncertainty, math and direct LISP programming. Rule-based. Requires IQ LISP. |
| Digitalk, Inc.<br>5200 W. Century Blvd.<br>Los Angeles, CA 90045 | Methods | 250.00 | Assembler and Basic | Systems can be linked | Implementation of Smalltalk. An object oriented programming language. Supports forward and backward chaining, math and confidence levels. |
| Dynamic Master Systems<br>P.O. Box 566456<br>Atlanta, GA 30356 | TOPSI | 75.00 | Turbo Pascal | 5,000 systems can be linked | Implementation of OPS 5, a forward chaining system. |
| Expert Systems Int'l.<br>1150 First Avenue<br>King of Prussia, PA 19406 | ES/P Advisor | 1,895.00 | PROLOG | 400-systems can be linked | Forward and backward chaining, is best used with their PROLOG. Can be compiled |
| EXSYS, Inc.<br>P.O. Box 75158<br>Albuquerque, NM 87194 | EXSYS | 295.00 | C | 5,000 | Rule-based language supports math and confidence levels. Backward chaining. |
| General Research, Inc.<br>7655 Old Springhouse Road<br>McLean, VA 22102 | TIMM | 9,500.00 | Fortran 77 | 500 | Induction extraction tool, can generate its own examples. Generated rules can be deleted. Supports confidence levels. |
| Human Edge Software, Inc.<br>2445 Farber Place<br>Palo Alto, CA 94303 | Expert Ease | 695.00 | UCSD Pascal | 300-systems can be linked | Induction extraction tool, forward chaining. Supports confidence levels. |
| Level 5 Research, Inc.<br>4980 S-A1A<br>Melbourne Beach, FL 32951 | Insight 1 | 95.00 | Turbo Pascal | 2000 | Rule-based language, supports confidence levels. Backward chaining with limited forward chaining ability. |

| COMPANY | PRODUCT NAME | PRICE ($) | WRITTEN IN | MAXIMUM RULES | COMMENTS |
|---|---|---|---|---|---|
| PPE, Inc.<br>P.O. Box 2027<br>Gathersburg, MD 20879 | Expert System | 20.00 | Basic | 5000 | Rule-based system, uses internal data base system for rule entry, it support confidence levels and math. Backward chaining. This is a freeware program. |
| Radian<br>8501 MO-Pac Blvd.<br>Austin, TX 78766 | Rule Master | 5,000.00 | C | 200-systems can be linked | Induction extraction tool. Rules can edited. Supports math and confidence levels. |
| Software A&E, Inc.<br>1500 Wilson Blvd.<br>Arlington, VA 22209 | KES | 4,000.00 | IQ LISP | Systems can be linked | Supports multiple objects, inheritance procedural control and Bayesian probabilities. Includes IQ LISP and suppor direct LISP programming. |
| SRI International<br>333 Ravenswood Ave.<br>Menlo Park, CA 94025 | SeriesPC | 15,000.00 | IQ LISP | 300-systems can be linked | Rule-based language. Requires IQ LISP license and supports direct LISP programming. Backward chaining. |
| Teknowledge<br>525 University Ave.<br>Palo Alto, CA 94301 | MI<br>MIA | 10,000.00<br>2,500.00 | PROLOG | 300-systems can be linked | Rule-based language, supports confiden levels, variables, math and cycles. Backward chaining. |
| Texas Instruments<br>P.O. Box 2909<br>Austin, TX 78769 | Personal Consultant | 3,000.00 | IQ LISP | 400-systems can be linked | Rule-based language will also support direct LISP programming. Includes IQ LISP. Backward chaining with multiple context structure, inheritance and confidence levels. |

SOURCE: Schwartz 1985

# APPENDIX B

# DEMO PROGRAM RULE-BASE

This is a demonstration system to examine the possibility of using an expert system to aid in estimating software costs. This system is not complete and also has not been verified. This system shows how the shell (EXSYS) could handle this problem domain. It also examines possible approaches to generating a software cost estimate. The expert system performs backwards chaining on the variables in the COCOMO model. The knowledge in the rules is used in combination with user answers to derive the proper values to be placed in the model. If a value is not derivable from the rule-base, the system will ask the user to determine the proper value.

Again note, this system is not complete and has not been verified. The estimate generated by the system is for demonstration only.

RULE NUMBER 1:

IF: The type of operational flight trainer is fixed wing

THEN: The suggested COCOMO development mode is the semi-detached mode and [KDSI] is given the value [navigation system] + [fuel system] + [unsupported elements]

NOTE: This rule selects the COCOMO basic formula for flight trainer training devices and presents an example product WBS.

---

RULE NUMBER 2:

IF: Concurrent development of associated new hardware and operational procedures:some and need for innovative data processing architectures, algorithms:minimal

THEN: The suggested COCOMO development mode is the organic mode

NOTE: From Table 6-3 in Software Engineering Economics, p. 81 (Boehm 1981)

---

RULE NUMBER 3:

IF: Concurrent development of associated new hardware and operational procedures:moderate and need for innovative data processing architectures, algorithms: some or :considerable

THEN: The suggested COCOMO development mode is the semi-detached mode

NOTE: From Table 6-3 in Software Engineering Economics, p. 81 (Boehm 1981)

---

RULE NUMBER 4:

IF: Need for innovative data processing architectures, algorithms:considerable and concurrent development of associated new hardware and operational procedures: extensive

THEN: The suggested COCOMO development mode is the embedded mode

NOTE: From Table 6-3 in Software Engineering Economics, p. 81 (Boehm 1981)

---

RULE NUMBER 5:

IF: Concurrent development of associated new hardware and operational procedures:some and need for innovative data processing architectures, algorithms: some or :considerable

THEN: The suggested COCOMO development mode is the semi-detached mode

NOTE: This assumes the semi-detached mode is a mixture of characteristics

---

RULE NUMBER 6:

IF: Need for innovative data processing architectures, algorithms:some and concurrent development of associated new hardware and operational procedures: extensive

THEN: The suggested COCOMO development mode is the semi-detached mode

NOTE: Based on a mixture of characteristics

---

RULE NUMBER 7:

IF: Concurrent development of associated new hardware and operational procedures:moderate or :extensive and need for innovative data processing architectures, algorithms:minimal

THEN: The suggested COCOMO development mode is the semi-detached mode

NOTE: Assumes semi-detached is a mixture of organic and embedded characteristics

---

RULE NUMBER 8:

IF: The suggested COCOMO development mode is the organic mode

THEN: [MM] is given the value (2.4*EXP(1.05*(LOG([KDSI])))) and [TDEV] is given the value (2.5*EXP(.38*(LOG([MM])))) and the organic COCOMO development mode was selected. Utilize the organic spreadsheet to generate an activity WBS.

NOTE: From Table 6-1 in Software Engineering Economics, p. 75 (Boehm 1981)

---

RULE NUMBER 9:

IF: The suggested COCOMO development mode is the semi-detached mode

THEN: [MM] is given the value (3.0*EXP(1.12*(LOG[KDSI])))) and [TDEV] is given the value (2.5*EXP(.35*(LOG([MM])))) and the COCOMO semi-detached mode was selected. Utilize the semi-detached spreadsheet to generate an activity WBS.

---

RULE NUMBER 10:

IF: The suggested COCOMO development mode is the embedded mode

THEN: [MM] is given the value (3.6*EXP(1.2*(LOG[KDSI]))))) and [TDEV] is given the value (2.5*EXP(.32*(LOG ([MM])))) and the COCOMO embedded mode was selected. Utilize the embedded spreadsheet to generate an activity WBS.

NOTE: COCOMO model for embedded mode, for complex development projects. From Table 6-1 in Software Engineering Economics, p. 75 (Boehm 1981)

-----------------------------------------------------------------

RULE NUMBER 11:

IF: Does this trainer simulate the navigation systems inside the airplane:no

THEN: [navigation system] is given the value 0

-----------------------------------------------------------------

RULE NUMBER 12:

IF: Does this trainer simulate the navigation systems inside the airplane:yes

THEN: [navigation system] is given the value [Doppler radar] + [inertial navigation system] + [radar altimeter]

NOTE: This is a product WBS of some common navigation instruments

-----------------------------------------------------------------

RULE NUMBER 13:

IF: Which of the following best describes the Doppler radar system in the simulator you are estimating: no system on trainer

THEN: [Doppler radar] is given the value 0

----------------------------------------------------------------

RULE NUMBER 14:

IF: Which of the following best describes the Doppler radar system in the simulator you are estimating: system with no installed malfunctions

THEN: [Doppler radar] is given the value 3

NOTE: Based on A-6 simulation system

----------------------------------------------------------------

RULE NUMBER 15:

IF: Which of the following best describes the Doppler radar system in the simulator you are estimating: system with instructor installed failures

THEN: [Doppler radar] is given the value 7

NOTE: From estimate on A-6 trainer

----------------------------------------------------------------

RULE NUMBER 16:

IF: The statement which best describes the inertial navigation system is:no system

THEN: [inertial navigation] is given the value 0

----------------------------------------------------------------

RULE NUMBER 17:

IF: The statement which best describes the inertial navigation system is:major instructor installed failures

THEN: [inertial navigation] is given the value 2

NOTE: Estimate from A-6 simulator

---

RULE NUMBER 18:

IF: The statement which best describes the inertial navigation system is:normal operation with no failures

THEN: [inertial navigation] is given the value .9

NOTE: Estimated from A-6 module that does the control simulation alone

---

RULE NUMBER 19:

IF: A radar altimeter is desired:no

THEN: [radar altimeter] is given the value 0

NOTE: Murphy's Law

---

RULE NUMBER 20:

IF: A radar altimeter is desired:yes; and the radar altimeter will simulate a malfunction:no; and the terrain the aircraft will be flying over is flat, like an ocean

THEN: [radar altimeter] is given the value .7

NOTE: Based on A-6 modules

---

RULE NUMBER 21:

IF: A radar altimeter is desired:yes; and the radar altimeter will simulate a malfunction:yes; and the terrain the aircraft will be flying over is flat, like an ocean

THEN: [radar altimeter] is given the value 1

---

RULE NUMBER 22:

IF: A radar altimeter is desired:yes; and the radar altimeter will simulate a malfunction:no; and the terrain the aircraft will be flying over is of varying elevation

THEN: [radar altimeter] is given the value 1.2

NOTE: Based on an estimate of a breakdown of A-6 simulator components by function

---

RULE NUMBER 23:

IF: A radar altimeter is desired:yes; and the terrain the aircraft will be flying over is of varying elevation; and the radar altimeter will simulate a malfunction:yes

THEN: [radar altimeter] is given the value 1.4

NOTE: From A-6 program module breakdown

---

RULE NUMBER 24:

IF: The statement which best describes the fuel system is:no system

THEN: [fuel system] is given the value 0

NOTE: Murphy's Law

---

RULE NUMBER 25:

IF: The statement which best describes the fuel system is:normal operation with no failures, except the effects of running out of fuel

THEN: [fuel system] is given the value 1

NOTE: Based on the A-6 trainer

---------------------------------------------------------------------

RULE NUMBER 26:

IF: The statement which best describes the fuel system is:system which simulates the effects of the loading of fuel tanks on the plane's center of gravity and the effects of running out of fuel

THEN: [fuel system] is given the value 1.5

NOTE: Based on A-6 trainer

---------------------------------------------------------------------

## Qualifiers

1 The type of operational flight trainer is

Other trainer type
Fixed wing

Used in rule(s): 1

2 The suggested COCOMO development mode is the

Organic mode
Semi-detached mode
Embedded mode

Used in rule(s): 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

3 Does this trainer simulate the navigation systems inside the airplane

:yes
:no

Used in rule(s): 11, 12

4 Which of the following best describes the Doppler radar system in the simulator you are estimating

:no system on trainer
:system with no installed malfunctions
:system with instructor installed failures
TAKE YOUR OWN GUESS

Used in rule(s): 13, 14, 15

5 The statement which best describes the inertial navigation system is

:no system
:major instructor installed failures
:normal operation with no failures

Used in rule(s): 16, 17, 18

6 The terrain the aircraft will be flying over is

:flat, like an ocean
:of varying elevation

Used in rule(s): 20, 21, 22, 23

7 A radar altimeter is desired

:yes
:no

Used in rule(s): 19, 20, 21, 22, 23

8 The radar altimeter will simulate a malfunction

:yes
:no

Used in rule(s): 20, 21, 22, 23

9 The statement which best describes the fuel system is

:no system
:normal operation with no failures, except the effects of running out of fuel
:system which simulates the effects of the loading of fuel tanks on the plane's center of gravity and the effects of running out of fuel

Used in rule(s): 24, 25, 26

10 Concurrent development of associated new hardware and operational procedures

:some
:moderate
:extensive

Used in rule(s): 2, 3, 4, 5, 6, 7

11 Need for innovative data processing architectures, algorithms

:minimal
:some
:considerable

Used in rule(s): 2, 3, 4, 5, 6, 7

## REFERENCES

Boehm, Barry W. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

Fikes, Richard, and Kehler, Tom. "The Role of Frame-Based Representation in Reasoning." Communications of the ACM (September 1985): 904-920.

Forsyth, Richard. Expert Systems. New York: Chapman and Hall, 1984.

Harmon, Paul, and King, David. Expert Systems. New York: John Wiley and Sons, Inc., 1985.

Hayes-Roth, Frederick. "Rule-Based Systems." Communications of the ACM (September 1985): 921-932.

Huntington, Dustin. EXSYS User's Guide. Albuquerque, NM: EXSYS, Inc., 1985.

Schwartz, Tom. "Artificial Intelligence in the Personal Computer Environment, Today and Tomorrow." Proceedings of the Ninth International Joint Conference on Artificial Intelligence (1985): 1261-1266.

Waterman, D.A. A Guide to Expert Systems. New York: Addison-Wesley Publishing Company, 1986.

Winston, Patrick Henry. Artificial Intelligence. New York: Addison-Wesley Publishing Company, 1984.